NUMERICAL METHODS FOR THE EVOLUTION OF FIELDS WITH APPLICATIONS TO PLASMAS

By

William A. Sands

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computational Mathematics, Science, and Engineering – Doctor of Philosophy

2022

ABSTRACT

NUMERICAL METHODS FOR THE EVOLUTION OF FIELDS WITH APPLICATIONS TO PLASMAS

By

William A. Sands

In this dissertation, we present a collection of algorithms for evolving fields in plasmas with applications to the Vlasov-Maxwell system. Maxwell's equations are reformulated in terms of the Lorenz and Coulomb gauge conditions to obtain systems involving wave equations. These wave equations are solved using the methods developed in this thesis and are combined with a particle-in-cell method to simulate plasmas. The particle-in-cell methods developed in this work treat particles using several approaches, including the standard Newton-Lorenz equations, as well as a generalized momentum formulation that eliminates the need to compute time derivatives of the field data.

In the first part of this thesis, we develop and extend some earlier methods for scalar wave equations, which are used to update the potentials in these formulations. Our developments are based on a class of algorithms known as the MOL^T, which combines a dimensional splitting technique with a one-dimensional integral equation method. This results in methods that are unconditionally stable, can address geometry, and are $O(N)$, where N is the number of mesh points. Our work contributes methods to construct spatial derivatives of the potentials for this class of dimensionally-split algorithms, which are used to evolve particles.

The second part of this thesis considers core algorithms used in the MOL^T and the related class of successive convolution methods in the context of high-performance computing environments. We developed a novel domain decomposition approach that ultimately allows the method to be used on distributed memory computing platforms. Shared memory algorithms were developed using the Kokkos performance portability library, which permits a user to write a single code that can be executed on various computing devices with the architecture-dependent details being managed by the library. We optimized predominant loop structures in the code and developed a blocking

pattern that prescribes parallelism at multiple levels and is also more cache-friendly. Moreover, the proposed iteration pattern is flexible enough to work with shared memory features available on GPU systems.

The final part of this thesis presents the particle-in-cell method for the Vlasov-Maxwell system, which leverages the methods for fields and derivatives developed in this work. The proposed methods are applied to several test problems involving beams. Our results are generally encouraging and demonstrate the capabilities of the proposed field solvers in simulating basic plasma phenomena. Additionally, our results serve to validate the generalized momentum formulation, which will be the foundation of our future work.

To my family, friends, and educators that have always brought out the best in me.

ACKNOWLEDGEMENTS

Many thanks and acknowledgements are in order, as this work was influenced by many people in a collaborative environment. First, I would like to express my gratitude to my advisor Professor Andrew Christlieb who supported me as a graduate student over the past few years, encouraged me to take on challenging problems, and who “believed in me" at times when I did not. I am, however, most grateful for the warm friendship he has provided me and for the compassion and emotional support he provided at an incredibly difficult junction in my personal life.

I am also grateful for my committee members, who took the time to write generous recommendation letters on my behalf during my time as a student. I would also like to thank Dr. John Luginsland for his insightful suggestions and comments regarding the plasma examples presented in this work and for always putting a humorous spin on everything! Discussions with Dr. Eric Wolf were also helpful in this regard, as he was always willing to take time to answer my questions. Additionally, I would like to thank Professor Michael Murillo and Dr. Jeffrey Haack, who provided me with an opportunity to work at Los Alamos National Laboratory in the summer of 2019. This experience significantly contributed to my growth, both as a person and as a scientist.

The Christlieb group, informally SPECTRE, (yep, like the James Bond movie) has been my home for the past couple of years. Even though I am changing groups for my next position, I hope to continue working with many of you as you assemble your own research portfolios. I have had the great fortune of being in a position to share my experiences and knowledge with many of you, whose curiosity inspires my own work.

Lastly, but certainly not least, are my parents, who have constantly encouraged me to pursue the things I enjoy and always made themselves available at times when I needed help. We will see if the late night phone calls still happen after grad school... I am also grateful for the valuable friendships forged in the CMSE program. This includes the secretaries who were always incredibly friendly and helpful; they are the unsung heroes! While many of us are moving on to different stages in our professional lives, we will always be connected by the time we spent together.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SCHEMES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

Plasmas are ubiquitous in nature. In fact, it is well-known that they comprise a majority of the visible universe, including the electronic devices that we regularly use as part of our daily lives, or the sun, which sustains life on Earth. Consequently, the properties of plasmas span an enormous range of space and time scales, often orders of magnitude in size. These multi-scale features pose a significant challenge for model developers and computational scientists, as many of these models used in the general interrogation of such systems are computationally intractable, even with the existing capabilities offered by supercomputers. This necessitates the development of new algorithms for plasmas which can successfully address the issue of scales and fit within the design constraints posed by new computational hardware.

Mathematically, plasmas are systems of charged particles, which can be conveniently described in terms of probability distribution functions that characterize the probability of finding a particular charged particle at some point of phase space. The background electromagnetic fields, which are described by Maxwell's equations, adapt to the motion of these charges, inducing changes in the probability distribution functions. This results in a complex system of partial differential equations known as the Vlasov-Maxwell system. A defining characteristic of Vlasov plasmas is that effects due to collisions arise only through the electromagnetic fields, so they are often called "collisionless" plasmas in an effort to distinguish them from more complex collision models such as the Boltzmann equation.

Building on the concept of a plasma, the next section provides a review of the literature concerning techniques for evolving the electromagnetic fields in the context of particle-in-cell methods for the Vlasov-Maxwell system. Then, we discuss the specific details concerning the mathematical models adopted in this work as well as the non-dimensionalization process. Lastly, we provide an overview of the work presented in this thesis, providing a delineation with some of the earlier work.

1.1 Background and Literature Review

In this section, we provide a review of the literature pertaining to algorithms employed in the simulation of Vlasov-type plasmas. An emphasis is placed on the particle-in-cell (PIC) method, which is the plasma simulation technique adopted in this thesis. A comprehensive review of the literature for PIC methods up to 2005 can be found in the review article [1]. Much of the work highlighted by this reference is now largely considered standard, so, instead, we discuss articles from more recent years that are more aligned with the developments in this thesis.

PIC methods [2, 3] have been extensively applied in the numerical simulation of plasmas and are an important class of techniques used in the design of experimental devices including lasers, pulsed power systems, particle accelerators, among others. The earliest work involving these methods began in the 1950s and 1960s, and it remains an active area of research to this day. At its core, a PIC method combines an Eulerian approach for the electromagnetic fields with a Lagrangian method that evolves collections of samples taken from general distribution functions in phase space. In other words, the fields are evolved using a mesh, while the distribution function is evolved using particles whose equations of motion are set according to characteristics of the PDEs for the distribution functions. Lastly, to combine the two approaches, an interpolation method is used to map data between the mesh and the particles. These maps are typically taken to be linear splines with the multi-dimensional interpolation performed by taking tensor products of one-dimensional interpolants. While PIC methods are capable of simulating complex nonlinear processes in plasmas, it is worth mentioning their weaknesses. Firstly, a natural consequence of the statistical element in PIC is that “bulk" processes in plasmas will be well represented, while the tails of the distribution will be largely underresolved even with good sampling methods. Another well-known consequence of the statistical element is the large number of simulation particles that are required in more systematic refinement studies due to certain numerical fluctuations. This consequence is critical to the computational efficiency of the method, which, in most applications, warrants the use of supercomputers to perform the simulations. The goal of this work is to supply new algorithms which aim to improve the computational efficiency of existing simulation tools,

such as PIC, in the numerical investigation of plasmas.

Most PIC methods evolve the simulation particles explicitly using some form of leapfrog time integration along with the Boris rotation method [4] in the case of electromagnetic plasmas. The exploration of semi- and fully-implicit particle treatments in PIC methods began in the 1980s [5, 6, 7]. These approaches suffered from a number of unattractive features, including issues with numerical heating and cooling [8], slow nonlinear convergence, and inconsistencies between the fluid moments and particle data. Consequently, these approaches were abandoned in favor of explicit formulations which were more aligned with the computational hardware available at the time. In recent years, implicit PIC methods have seen a resurgence, beginning with [9], which addressed many of these issues in the case of the Vlasov-Poisson system. Nonlinear convergence and self-consistency were enforced using a Jacobian-free Newton-Krylov method [10] with a fluid preconditioner that enforced the continuity equation. The resulting solver demonstrated remarkable savings compared to explicit methods because they eliminated the need to resolve the charge separation in the plasma, allowing for a coarser mesh to be used. These techniques were later extended to curved geometries through the use of smooth grid mappings [11]. Recently, an effort has been made to extend these techniques to the full Vlasov-Maxwell system [12] to avoid the highly restrictive CFL condition posed by the gyrofrequency. While these contributions are significant in their own right, there are many opportunities for improvement. Many of these methods are fundamentally stuck at second-order accuracy in both space and time and may greatly benefit from more accurate field solvers. Additionally, applications of interest involve complex geometries which introduce additional complications with stability and are often poorly resolved with uniform Cartesian meshes. Lastly, there is the concern of scalability. Krylov subspace methods pose a massive challenge for scalability on large machines due to the various collective operations used in the algorithms. It seems that the scalability of these methods could be significantly improved if similar implicit methods could be developed which eliminate the inner Krylov solve altogether, though this is beyond the scope of the present work.

A challenge associated with developing any solver for Maxwell's equations is the enforcement

of the involutions for the fields, namely $\nabla \cdot \mathbf{E} = \rho/\epsilon_0$ and $\nabla \cdot \mathbf{B} = 0$. In the case of a structured Cartesian grid, Maxwell's equations can be discretized using a staggered grid technique introduced by Yee [13]. The use of a staggered mesh yields a structure-preserving discrete analogue of the integral form of Maxwell's equations that automatically enforces the involutions for $\mathbf{E}$ and $\mathbf{B}$ without additional treatment. This is the basis of the well-known finite-difference time-domain method (FD-TD) [14]. While the staggering in both space an time used in the FD-TD method is second-order accurate, there exists a fourth-order extension of the spatial discretization that was developed as a way of dealing with certain dispersion errors known as Cerenkov radiation [15]. While the use of a staggered mesh with finite-differences is quite effective for Cartesian grids, issues arise in problems defined with geometry, such as curved surfaces, in which one resorts to stair step boundaries [16]. To mitigate the effect of stair step boundaries in explicit methods, the mesh resolution is increased, resulting in a highly restrictive time step to meet the CFL stability criterion. Recently, an approach for dealing with geometry in the Yee scheme, which avoids the stair stepping along the boundary, was developed for two-dimensional problems [17]. The grid cells along the boundary in the method are replaced with cut-cells which use generalized finite-difference updates that account for different intersections with the boundary. While this scheme was shown to be energy conserving and, perhaps more remarkably, preserved the CFL condition of the Yee scheme, the convergence rates demonstrated by the method are suboptimal. The theory in this article established half-order accuracy, yet demonstrated first-order accuracy in numerical experiments.

While many electromagnetic PIC methods solve Maxwell's equations on Cartesian meshes through the FDTD method, other methods have been developed specifically for addressing issues posed by geometries through the use of unstructured meshes. In [18], a finite-element method (FEM) was coupled with PIC to solve the Darwin model in which the fields move significantly faster than the plasma. Explicit finite-volume methods (FVM), which can address geometry, were considered in [19], which also developed divergence cleaning methods suitable for applications to PIC simulations of the Vlasov-Maxwell system. Discontinuous Galerkin (DG) methods have also

been used to develop high-order PIC methods with elliptic [20] and hyperbolic [21] divergence cleaning methods being employed to enforce Gauss' law. Other work in this area has explored more generalized FEM discretizations to enforce charge conservation on arbitrary grids [22] as well as certain structure preserving discretizations [23, 24, 25]. In particular, [24, 25, 26] employed so called Whitney basis sets taken from the de Rham sequences to automatically enforce involutions for the electric and magnetic fields. Despite the advantages afforded by the use of such bases, these implicit field solvers rely on the solutions of large linear systems which need to be solved using GMRES [27]. Even with preconditioning, such methods can be slow and difficult to achieve scalability. In the case of explicit solvers, such as FVMs and DG methods, other challenges exist. The basic FVM, without additional reconstructions, is first-order in space. These methods can, of course, be improved to second-order accuracy by performing reconstructions based on a collection of cells. Beyond second-order accuracy, the reconstruction process becomes quite complicated due to the size of the interpolation stencils. DG methods, on the other hand, store cell-wise expansions in a basis, which eliminates the issue encountered in the FVM, typically at the cost of a highly restrictive condition on the size of a time step. Additionally, the significant amount of local work in DG methods makes them appealing for newer hardware, yet the restriction on the time step size is often left unaddressed. Notable exceptions to this restriction exist for the two-way wave equation including staggered formulations [28] and Hermite methods [29], which allow for a much larger time step. It will be interesting to see the performance of such methods in plasma problems, especially in problems with intricate geometric features.

Other methods for Maxwell's equations have been developed with unconditional stability for the time discretization. The first of these methods is the ADI-FDTD method [30, 31], which combined an ADI approach with a two-stage splitting to achieve an unconditionally stable solver. Time stepping in these methods was later generalized using a Crank-Nicolson splitting and several techniques for enhancing the temporal accuracy were proposed [32]. Of particular significance to this work are methods based on successive convolution, also known as the method-of-lines-transpose (MOL^T) [33, 34]. These methods are unconditionally stable in time and can be obtained

by reversing the typical order in which discretization is performed. By first discretizing in time, one can solve a resulting boundary-value problem by formally inverting the differential operator using a Green's function in conjunction with a fast summation method. Mesh-free methods [35, 36] for Maxwell's equations have also been explored in the Darwin limit under the Coulomb gauge. These formulations are in some ways similar to PIC in that they evolve particles with shapes, except no mesh is used in the simulation. The elliptic equations are solved using a Green's function on an unbounded domain and a fast summation method is used for efficiency. Green's function methods have also been used to develop asymptotic preserving schemes [37]. This article utilized a boundary integral formulation with a multi-dimensional Green's function, to obtain a method that recovers the Darwin limit under appropriate conditions. The methods considered in this work incorporate dimensional splittings, which results in algorithms with unconditional stability, high-order accuracy [38], parallel scalability [39] and geometric flexibility [40]. In [41], a PIC method was developed based on the MOL^T discretization combined with a staggered grid formulation of the Vlasov-Maxwell system. In this approach, the field equations were cast in terms of the Lorenz gauge, producing wave equations for the scalar and vector potentials. Additionally, the wave equation for the scalar potential was replaced with an elliptic equation to control errors in the gauge condition. Since the particle equations were written in terms of $\mathbf{E}$ and $\mathbf{B}$, additional finite-difference derivatives were required to compute the electric and magnetic fields from the potentials. While the contributions of this work differ significantly from the methods in [41], we consider the latter work as a baseline to focus our efforts. A fairly detailed outline of the contributions of this thesis can be found in section 1.4 at the end of this chapter.

1.2 Mathematical Models

In this section, we provide relevant details of the mathematical models employed for the plasma applications considered in this work. We begin with a discussion of the Vlasov-Maxwell system, which is the most general model considered in this work, in section 1.2.1. Then, once we have finished the discussion of the model, we discuss the mathematical formulation in 1.2.2, which

expresses Maxwell's equations in terms of potentials through the use of gauge conditions. More specifically, we discuss two formulations: one which employs the Lorenz gauge, and another, which uses the Coulomb gauge. With the exception of the expanding beam problems in section 4.6, the numerical examples shall exclusively work with the Lorenz gauge. Since we have adopted a gauge formulation of Maxwell's equations, we also introduce the equations of motion for the particles in section 1.2.2, which are written entirely in terms of the potentials used for Maxwell's equations. Once we have presented the formulations, we discuss the non-dimensionalized systems used in the numerical experiments presented in section 4.6. Finally, we conclude the chapter with a summary of the original contributions presented in this thesis, which can be found in section 1.4.

1.2.1 Vlasov-Maxwell System

In this work, we develop numerical algorithms for plasmas described by the Vlasov-Maxwell (VM) system, which in SI units, reads as

$$\partial_t f_s + \mathbf{v} \cdot \nabla_x f_s + \frac{q_s}{m_s} (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_v f_s = 0, \tag{1.1}$$

$$\nabla \times \mathbf{E} = -\partial_t \mathbf{B}, \tag{1.2}$$

$$\nabla \times \mathbf{B} = \mu_0 (\mathbf{J} + \epsilon_0 \partial_t \mathbf{E}), \tag{1.3}$$

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}, \tag{1.4}$$

$$\nabla \cdot \mathbf{B} = 0. \tag{1.5}$$

The first equation (1.1) is the Vlasov equation which describes the evolution of a probability distribution function $f_s(\mathbf{x}, \mathbf{v}, t)$ for particles of species s in phase space which have mass m_s and charge q_s. More specifically, it describes the probability of finding a particle of species s at the position $\mathbf{x}$, with a velocity $\mathbf{v}$, at a given time t. Since the position and velocity data are vectors with 3 components, the distribution function is a scalar function of 6 dimensions plus time. While the equation itself has fairly simple structure, the primary challenge in numerically solving this equation is its high dimensionality. This growth in the dimensionality has posed tremendous difficulties for grid-based discretization methods, where one often needs to use many grid points to resolve scales

in the problem. The use of grids for problems involving beams poses additional challenges due to excessive dissipation along the edge of the beams. This difficulty is further compounded by the fact that many plasmas of interest contain multiple species. Despite the lack of a collision operator on the right-hand side of (1.1), collisions do exist in a certain mean-field sense, through the electric and magnetic fields which appear as coefficients of the velocity gradient.

Equations (1.2) - (1.5) are Maxwell's equations, which describe the evolution of the background electric and magnetic fields. Since the plasma is a collection of moving charges, any changes in the distribution function for each species will be reflected in the charge density $\rho(\mathbf{x}, t)$, as well as the current density $\mathbf{J}(\mathbf{x}, t)$, which, respectively, are the source terms for Gauss' law 1.4 and Ampère's law (1.3). For N_s species, the total charge density and current density are defined by summing over the species

$$\rho(\mathbf{x}, t) = \sum_{s=1}^{N_s} \rho_s(\mathbf{x}, t), \quad \mathbf{J}(\mathbf{x}, t) = \sum_{s=1}^{N_s} \mathbf{J}_s(\mathbf{x}, t), \tag{1.6}$$

where the species charge and current densities are defined through moments of the distribution function f_s according to

$$\rho_s(\mathbf{x}, t) = q_s \int_{\Omega_v} f_s(\mathbf{x}, \mathbf{v}, t)\, d\mathbf{v}, \quad \mathbf{J}_s(\mathbf{x}, t) = q_s \int_{\Omega_v} \mathbf{v} f_s(\mathbf{x}, \mathbf{v}, t)\, d\mathbf{v}. \tag{1.7}$$

Here, the integrals are taken over the velocity components of phase space, which we have denoted by Ω_v. The remaining parameters ϵ_0 and μ_0 describe the permittivity and permeability of the media in which the fields propagate, which we take to be free-space. In free-space, Maxwell's equations move at the speed of light c, which leads to the useful relation $c^2 = (\mu_0 \epsilon_0)^{-1}$. The last two equations (1.4) and (1.5) are constraints placed on the fields to maintain charge conservation and prevent the appearance of so-called "magnetic monopoles". It is imperative that numerical schemes for Maxwell's equations satisfy these conditions. This requirement is one of the reasons we adopt a formulation for Maxwell's equations in potential form, which is the subject of the next section.

1.2.2 Problem Formulation

In this work, we adopt a particle formulation of the Vlasov equation (1.1) and use a gauge formulation of Maxwell's equations. Here we discuss the models that form the basis of the numerical methods presented in this work.

1.2.2.1 Maxwell's Equations with the Lorenz Gauge

Under the Lorenz gauge, Maxwell's equations transform to a system of decoupled wave equations of the form

$$\frac{1}{c^2}\frac{\partial^2 \psi}{\partial t^2} - \Delta \psi = \frac{1}{\epsilon_0}\rho, \tag{1.8}$$

$$\frac{1}{c^2}\frac{\partial^2 \mathbf{A}}{\partial t^2} - \Delta \mathbf{A} = \mu_0 \mathbf{J}, \tag{1.9}$$

$$\nabla \cdot \mathbf{A} + \frac{1}{c^2}\frac{\partial \psi}{\partial t} = 0, \tag{1.10}$$

where c is the speed of light, ϵ_0 and μ_0 represent, respectively, the permittivity and permeability of free-space. Further, we have used ψ to denote the scalar potential and $\mathbf{A}$ is the vector potential. In fact, under any choice of gauge condition, given ψ and $\mathbf{A}$, one can recover $\mathbf{E}$ and $\mathbf{B}$ using the definitions

$$\mathbf{E} = -\nabla \psi - \frac{\partial \mathbf{A}}{\partial t}, \quad \mathbf{B} = \nabla \times \mathbf{A}, \tag{1.11}$$

where "$\times$" denotes the vector cross product. The structure of equations (1.8) and (1.9) is appealing because the system, modulo the gauge condition (1.10), is essentially four "decoupled" scalar wave equations. Since this system is over-determined, the coupling manifests itself through the gauge condition which should be thought of as a constraint. Moreover, Maxwell's equations (1.2) - (1.5) are equivalent to (1.8) and (1.9) as long as the Lorenz gauge condition (1.10) is satisfied by ψ and $\mathbf{A}$. This formulation is appealing for several reasons. First, this system is purely hyperbolic, so it evolves in a local sense. Computationally, this means that a localized method can be used to evolve the system, which will be more efficient for parallel computers. Another attractive feature is that

many of the methods developed for scalar wave equations, e.g., [38] and [34] can be applied to the system in a straightforward manner.

1.2.2.2 Maxwell's Equations with the Coulomb Gauge

If instead, we impose the Coulomb gauge in Maxwell's equations, we obtain a coupled, mixed-type system

$$-\Delta\psi = \frac{1}{\epsilon_0}\rho, \tag{1.12}$$

$$\frac{1}{c^2}\frac{\partial^2 \mathbf{A}}{\partial t^2} - \Delta\mathbf{A} = \mu_0 \mathbf{J} - \frac{1}{c^2}\frac{\partial\left(\nabla\psi\right)}{\partial t}, \tag{1.13}$$

$$\nabla \cdot \mathbf{A} = 0, \tag{1.14}$$

where, again, c is the speed of light, ϵ_0 and μ_0 represent, respectively, the permittivity and permeability of free-space and ψ and $\mathbf{A}$ to denote the scalar potential and vector potential, respectively. There are several noticeable differences in the system above, when compared to the one obtained with the Lorenz gauge (1.8)-(1.10). The equations are no longer decoupled in the same way as in the case of the Lorenz gauge. Moreover, the equation for the scalar potential (1.12) is an elliptic equation, rather than a hyperbolic equation. This requires elliptic solvers, which are more difficult to scale on parallel computers than hyperbolic solvers due to their global properties. As a consequence, additional parallel communications are required to coordinate the solves. Additionally, the Coulomb gauge introduces a somewhat unusual time derivative $\partial_t \nabla\psi$, which is connected to the steady state equation (1.12).

In our implementation of the solver with the Coulomb gauge, we use a Helmholtz decomposition of the vector fields $\mathbf{A}$ and $\mathbf{J}$ that separates the rotational and irrotational components according to

$$\mathbf{A} = \mathbf{A}_{\text{rot}} + \mathbf{A}_{\text{irrot}} \equiv \mathbf{A}_{\text{rot}} - \nabla\xi, \tag{1.15}$$

$$\mathbf{J} = \mathbf{J}_{\text{rot}} + \mathbf{J}_{\text{irrot}} \equiv \mathbf{J}_{\text{rot}} - \nabla\eta. \tag{1.16}$$

Here, ξ and η are scalar functions and by definition, $\nabla \cdot \mathbf{A}_{\text{rot}} = 0$ and $\nabla \cdot \mathbf{J}_{\text{rot}} = 0$. Substituting this

decomposition into equation (1.13), and separating the equations (by linearity), we obtain

$$\frac{1}{c^2}\frac{\partial^2 \mathbf{A}_{\text{rot}}}{\partial t^2} - \Delta \mathbf{A}_{\text{rot}} = \mu_0 \mathbf{J}_{\text{rot}}, \tag{1.17}$$

$$\frac{1}{c^2}\frac{\partial^2 \mathbf{A}_{\text{irrot}}}{\partial t^2} - \Delta \mathbf{A}_{\text{irrot}} = \mu_0 \mathbf{J}_{\text{irrot}} - \frac{1}{c^2}\frac{\partial\left(\nabla\psi\right)}{\partial t}. \tag{1.18}$$

The second equation is connected to the continuity equation. If we assume that the Coulomb gauge is satisfied, then it follows that $\nabla \cdot \mathbf{A}_{\text{irrot}} = -\Delta\xi = 0$. With this in mind, if we now take the divergence of equation (1.18), using (1.12) and (1.16), we find that

$$\begin{aligned} 0 &= -\frac{1}{c^2}\frac{\partial\left(\Delta\psi\right)}{\partial t} + \mu_0 \nabla \cdot \mathbf{J}_{\text{irrot}}, \\ &= \frac{1}{c^2\epsilon_0}\frac{\partial \rho}{\partial t} + \mu_0 \nabla \cdot \left(\mathbf{J} - \mathbf{J}_{\text{rot}}\right), \\ &= \mu_0 \left(\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{J}\right), \end{aligned}$$

which is the continuity equation. In order to solve (1.17), which avoids the term involving the strange time derivative, we need to identify $\mathbf{J}_{\text{rot}}$ from $\mathbf{J}$. One way of doing this is by appealing to equation (1.16). Taking its divergence and rearranging the terms, we obtain the elliptic equation

$$-\Delta\eta = \nabla \cdot \mathbf{J}. \tag{1.19}$$

Once η is identified, we can compute $\nabla\eta$ and set

$$\mathbf{J}_{\text{rot}} = \mathbf{J} + \nabla\eta. \tag{1.20}$$

Our implementation works off of equations (1.12) and (1.17), in addition to (1.19) and (1.20). The issue of enforcing the gauge condition will be addressed in section 4.3, where we discuss methods for controlling errors in the gauges.

1.2.2.3 Formulation for the Particles

A particle formulation of the Vlasov equation (1.1) can be developed by writing the species distribution function f_s as a collection of Dirac delta distributions over phase space

$$f_s\left(\mathbf{x}, \mathbf{v}, t\right) = \sum_{p=1}^{N_{ps}} \delta(\mathbf{x} - \mathbf{x}_p)\delta(\mathbf{v} - \mathbf{v}_p). \tag{1.21}$$

Here, N_{p_s} symbolizes the number of particles of species s. Notice that we also have the relation

$$\int_{\Omega_x}\int_{\Omega_v} f_s\left(\mathbf{x},\mathbf{v},t\right)\, d\mathbf{v}\, d\mathbf{x} = N_{p_s},$$

which holds at any time t. Furthermore, f_s can be converted into a proper distribution by including a normalization factor of $1/N_{p_s}$. By combining the ansatz (1.21) with the definitions (1.6) and (1.7), we obtain the following definitions of charge density and current density for a collection of N_p simulation particles:

$$\rho(\mathbf{x}) = \sum_{p=1}^{N_p} q_p \delta(\mathbf{x} - \mathbf{x}_p), \tag{1.22}$$

$$\mathbf{J}(\mathbf{x}) = \sum_{p=1}^{N_p} q_p \mathbf{v}_p \delta(\mathbf{x} - \mathbf{x}_p). \tag{1.23}$$

In the above equators, q_p, $\mathbf{x}_p$, and $\mathbf{v}_p$ denote the charge, position, and velocity, respectively, of a particle whose label is p. In defining things this way, we have dropped the reference to the species altogether, since each particle can be thought of as its own entity. These particles move along characteristics of the equation (1.1), which are given by the system of ordinary differential equations

$$\begin{aligned}\dot{\mathbf{x}}_i &= \mathbf{v}_i \\ \dot{\mathbf{v}}_i &= \frac{1}{m_i}\mathbf{F}(\mathbf{x}_i).\end{aligned}$$

The vector field $\mathbf{F}$ is the Lorentz force that acts on particles and is defined by

$$\mathbf{F} = q\Big(\mathbf{E} + \mathbf{v}\times\mathbf{B}\Big), \tag{1.24}$$

where we have removed the subscript that refers to a specific particle for simplicity. Next, we write the fields in terms of their potentials. Using (1.11), we can obtain the equivalent expression

$$\mathbf{F} = q\Big(-\nabla\psi - \frac{\partial\mathbf{A}}{\partial t} + \mathbf{v}\times(\nabla\times\mathbf{A})\Big).$$

This expression can be simplified with the aid of the vector identity

$$\nabla\left(\mathbf{a}\cdot\mathbf{b}\right) = \mathbf{a}\times\nabla\times\mathbf{b} + \mathbf{b}\times\nabla\times\mathbf{a} + \left(\mathbf{a}\cdot\nabla\right)\mathbf{b} + \left(\mathbf{b}\cdot\nabla\right)\mathbf{a}. \tag{1.25}$$

Using $\mathbf{a} \equiv \mathbf{A}$ and $\mathbf{b} \equiv \mathbf{v}$, along with the fact that the velocity $\mathbf{v}$ does not depend on $\mathbf{x}$, we obtain the relation

$$\mathbf{v} \times (\nabla \times \mathbf{A}) = \nabla (\mathbf{A} \cdot \mathbf{v}) - (\mathbf{v} \cdot \nabla) \mathbf{A}.$$

Inserting this expression into the force yields

$$\mathbf{F} = q\Big(- \nabla\psi - \frac{\partial \mathbf{A}}{\partial t} + \nabla (\mathbf{A} \cdot \mathbf{v}) - (\mathbf{v} \cdot \nabla) \mathbf{A}\Big).$$

Then we can use the definition of the total (convective) derivative to write

$$\frac{d\mathbf{A}}{dt} = \frac{\partial \mathbf{A}}{\partial t} + (\mathbf{v} \cdot \nabla) \mathbf{A},$$

which means the force is equivalent to

$$\mathbf{F} = q\Big(- \nabla\psi - \frac{d\mathbf{A}}{dt} + \nabla (\mathbf{A} \cdot \mathbf{v}) \Big).$$

If we let $\mathbf{p}$ denote the classical momentum $\mathbf{p} = m\mathbf{v}$, so that $\frac{d\mathbf{p}}{dt} = \mathbf{F}$, then we can move the time derivative to the left side of the equation, which gives

$$\frac{d}{dt}\Big(\mathbf{p} + q\mathbf{A}\Big) = q\Big(- \nabla\psi + \nabla (\mathbf{A} \cdot \mathbf{v}) \Big).$$

The expression on the left contains the canonical (generalized) momentum

$$\mathbf{P} := \mathbf{p} + q\mathbf{A}, \tag{1.26}$$

and the right side can be expressed as $-\nabla U$, if we let $U = q\,(\psi - \mathbf{A} \cdot \mathbf{v})$. Therefore, we obtain the canonical momentum equation

$$\frac{d\mathbf{P}}{dt} = q\Big(- \nabla\psi + \nabla (\mathbf{A} \cdot \mathbf{v}) \Big).$$

This is the penultimate form of the equation we seek. Instead, we want the derivatives to appear on the vector potential $\mathbf{A}$ rather than $\mathbf{A} \cdot \mathbf{v}$. For this, we can use an equivalent form of the identity (1.25), namely

$$\nabla (\mathbf{a} \cdot \mathbf{b}) = (\nabla \mathbf{b}) \cdot \mathbf{a} + (\nabla \mathbf{a}) \cdot \mathbf{b}.$$

Again, we select $a \equiv \mathbf{A}$ and $b \equiv \mathbf{v}$, which shows that

$$\nabla\left(\mathbf{A}\cdot\mathbf{v}\right) = \left(\nabla\mathbf{v}\right)\cdot\mathbf{A} + \left(\nabla\mathbf{A}\right)\cdot\mathbf{v}.$$

Equation (1.26) provides the connection between the linear momentum $\mathbf{p} = m\mathbf{v}$ and the canonical momentum $\mathbf{P}$, so that the velocity is given by

$$\mathbf{v} \equiv \frac{1}{m}\Big(\mathbf{P} - q\mathbf{A}\Big). \tag{1.27}$$

Since $\mathbf{v}$ is a function only of time, we have that

$$\left(\nabla\mathbf{v}\right)\cdot\mathbf{A} = 0.$$

Combining this with the other term yields an expanded form for the canonical momentum update

$$\frac{d\mathbf{P}}{dt} = q\Big(-\nabla\psi + \frac{1}{m}\left(\nabla\mathbf{A}\right)\cdot\left(\mathbf{P} - q\mathbf{A}\right)\Big). \tag{1.28}$$

Note that since $\mathbf{A}$ is a vector, taking the gradient increases its rank by 1, which means that $\nabla\mathbf{A}$ is a dyad. In component form, we can write $\nabla\left(\mathbf{a}\cdot\mathbf{b}\right)$ as

$$\partial_i\left(a_j b_j\right) = \left(\partial_i a_j\right) b_j + \left(\partial_i b_j\right) a_j = \frac{\partial a^{(j)}}{\partial x_i} b_j + \frac{\partial b^{(j)}}{\partial x_i} a_j, \tag{1.29}$$

where the summation convention over repeated indices has been used. For our case, the vector $\mathbf{b}$ in the above calculation does not depend on space. Therefore, one only requires computing the entries of the dyad

$$\frac{\partial A^{(j)}}{\partial x_i} \equiv J_{\mathbf{A}}^{T} \tag{1.30}$$

where $J_{\mathbf{A}}$ is the Jacobian matrix associated with $\mathbf{A}$.

Remark 1.2.1. Another way to see that the (1.29) and (1.30) are the correct expressions for (1.28) is to use the fact that $\mathbf{A}\cdot\mathbf{v}$ is a scalar, then apply the usual gradient operator for scalar functions. In other words,

$$\begin{aligned}\nabla\left(\mathbf{A}\cdot\mathbf{v}\right) &= \nabla\left(A^{(1)}v^{(1)} + A^{(2)}v^{(2)} + A^{(3)}v^{(3)}\right),\\ &= \begin{bmatrix}\partial_x\left(A^{(1)}v^{(1)} + A^{(2)}v^{(2)} + A^{(3)}v^{(3)}\right)\\ \partial_y\left(A^{(1)}v^{(1)} + A^{(2)}v^{(2)} + A^{(3)}v^{(3)}\right)\\ \partial_z\left(A^{(1)}v^{(1)} + A^{(2)}v^{(2)} + A^{(3)}v^{(3)}\right)\end{bmatrix}.\end{aligned}$$

The result follows by distributing the derivatives in each row, using the fact that the components of the velocity do not depend on space, followed by use of the definition (1.27).

To obtain the position equation, we note that $\frac{d\mathbf{x}}{dt} = \mathbf{v}$ and that the classical momentum is given by $\mathbf{p} = m\mathbf{v}$. This implies that

$$\mathbf{P} = m\frac{d\mathbf{x}}{dt} + q\mathbf{A},$$

which can be arranged to obtain

$$\frac{d\mathbf{x}}{dt} = \frac{1}{m}\Big(\mathbf{P} - q\mathbf{A}\Big). \tag{1.31}$$

Since the transformed equations of motion given by (1.28) and (1.31) will be identical in structure among the particles (differing only by labels for the particles) this results in the system

$$\frac{d\mathbf{x}_i}{dt} = \frac{1}{m_i}\Big(\mathbf{P}_i - q_i\mathbf{A}\Big), \tag{1.32}$$

$$\frac{d\mathbf{P}_i}{dt} = q_i\Big(- \nabla\psi + \frac{1}{m_i}\left(\nabla\mathbf{A}\right)\cdot\left(\mathbf{P}_i - q_i\mathbf{A}\right)\Big). \tag{1.33}$$

The complete formulation consists of evolving the fields with (1.8)-(1.10) and (1.32) - (1.33) for the particles. In the next section, we provide details concerning the non-dimensionalization used for the models.

1.3 Non-dimensionalization

In this section, we discuss the scalings used to non-dimensionalize the models explored in this work. Our choice in exploring the normalized form of these models is simply to reduce the number of floating point operations with small or large numbers. We first non-dimensionalize the models for particles, then focus our efforts on the field equations obtained with both the Lorenz and Coulomb gauges. To minimize repetition, we shall illustrate the process for parts of the formulation and simply state the results for those that follow an identical pattern.

The setup for the non-dimensionalization process used in this section considers the following

substitutions:

$$\mathbf{x} \to L\tilde{\mathbf{x}}, \quad \mathbf{P} \to P\tilde{\mathbf{P}}, \quad t \to T\tilde{t},$$
$$n \to \bar{n}\tilde{n}, \quad \psi \to \psi_0\tilde{\psi}, \quad \rho \to Q\bar{n}\tilde{\rho},$$
$$\mathbf{A} \to A_0\tilde{\mathbf{A}}, \quad \mathbf{J} \to Q\bar{n}V\tilde{\mathbf{J}} \equiv \frac{Q\bar{n}L}{T}\tilde{\mathbf{J}}.$$

Here, we use $\bar{n}$ to denote a reference number density [m^{-3}], Q is the scale for charge in [C], and we also introduce M, which represents the scale for mass [kg]. The values for Q and M are set according to the electrons, so that $Q = |q_e|$ and $M = m_e$. Other scales, such as ψ_0 and A_0 shall be specified later. A natural choice of the scales for L, and T are the Debye length and angular plasma period, which are defined, respectively, by

$$L = \lambda_D = \sqrt{\frac{\epsilon_0 k_B \bar{T}}{\bar{n} q_e^2}} \quad [\mathrm{m}], \quad T = \omega_{pe}^{-1} = \sqrt{\frac{m_e \epsilon_0}{\bar{n} q_e^2}} \quad [\mathrm{s/rad}],$$

where k_B is the Boltzmann constant, m_e is the electron mass, q_e is the electron charge, and $\bar{T}$ is an average macroscopic temperature for the plasma. We choose to select these scales for all test problems considered in section 4.6 with the exception of the expanding beam problems, which are the last two test problems in that section. For these problems, the length scale L corresponds to the longest side of the simulation domain and T is the crossing time for a particle that is injected into the domain. Generally, the user will need to provide a macroscopic temperature $\bar{T}$ [K] in addition to the reference number density $\bar{n}$ to compute λ_D and ω_{pe}^{-1}. Note that in some cases we shall refer to the plasma period, which can be obtained from the angular plasma period T by multiplying with 2π. Having introduced the definitions for the normalized variables, we proceed to non-dimensionalize the models, beginning with the equations for particles before addressing the field equations in both the Lorenz and Coulomb gauges.

1.3.1 Equations of Motion in E-B form

In our development of new field solvers with applications to PIC, it is helpful to benchmark the proposed methods against standard approaches for integrating particles which use the Newton-

Lorenz equations

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i, \tag{1.34}$$

$$\frac{d\mathbf{v}_i}{dt} = \frac{q_i}{m_i}\Big(\mathbf{E}(\mathbf{x}_i) + \mathbf{v}_i \times \mathbf{B}(\mathbf{x}_i)\Big). \tag{1.35}$$

After inserting the scales into the position equation (1.34), we obtain the equivalent non-dimensional form

$$\frac{d\tilde{\mathbf{x}}_i}{d\tilde{t}} = \tilde{\mathbf{v}}_i.$$

Following the same process for the velocity equation, after some rearrangement, we obtain

$$\frac{d\tilde{\mathbf{v}}_i}{d\tilde{t}} = \frac{\tilde{q}_i}{r_i}\Big(\frac{QE_0T}{MV}\tilde{\mathbf{E}} + \frac{QB_0T}{M}\tilde{\mathbf{v}}_i \times \tilde{\mathbf{B}}\Big).$$

Here we have introduced the non-dimensional electric and magnetic fields $\tilde{\mathbf{E}}$ and $\tilde{\mathbf{B}}$, which are normalized by E_0 and B_0, respectively, and $r_i = m_i/M$ is a mass ratio . From equation (1.11), we can express these scales in terms of ψ_0 and A_0 as

$$E_0 = \frac{\psi_0}{L}, \quad B_0 = \frac{A_0}{L}.$$

Therefore, the non-dimensionalized equation for the velocity can be expressed in terms of these scales as

$$\begin{aligned}\frac{d\tilde{\mathbf{v}}_i}{d\tilde{t}} &= \frac{\tilde{q}_i}{r_i}\Big(\frac{Q\psi_0T^2}{ML^2}\tilde{\mathbf{E}} + \frac{QA_0T}{ML}\tilde{\mathbf{v}}_i \times \tilde{\mathbf{B}}\Big),\\ &\equiv \frac{\tilde{q}_i}{r_i}\Big(\alpha_1\tilde{\mathbf{E}} + \alpha_2\tilde{\mathbf{v}}_i \times \tilde{\mathbf{B}}\Big),\end{aligned}$$

where we have introduced the parameters

$$\alpha_1 = \frac{Q\psi_0T^2}{ML^2}, \quad \alpha_2 = \frac{QA_0T}{ML}.$$

We then select ψ_0 and A_0 so that $\alpha_1 = \alpha_2 = 1$, i.e.,

$$\psi_0 = \frac{ML^2}{QT^2}, \quad A_0 = \frac{ML}{QT}, \tag{1.36}$$

which results in the non-dimensional system

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i, \tag{1.37}$$

$$\frac{d\mathbf{v}_i}{dt} = \frac{q_i}{r_i}\Big(\mathbf{E} + \mathbf{v}_i \times \mathbf{B}\Big). \tag{1.38}$$

Note that we have dropped the tildes for brevity. The next section provides the non-dimensionalized form for the analogous equations that evolve particles using the potentials ψ and $\mathbf{A}$ in the generalized momentum framework.

1.3.2 Equations of Motion for the Generalized Hamiltonian

Here, we non-dimensionalize the generalized momentum model for the particles, which is given by equations (1.32) and (1.33). For convenience, the system is given by

$$\frac{d\mathbf{x}_i}{dt} = \frac{1}{m_i}\Big(\mathbf{P}_i - q_i\mathbf{A}\Big),$$
$$\frac{d\mathbf{P}_i}{dt} = q_i\Big(- \nabla\psi + \frac{1}{m_i}\left(\nabla\mathbf{A}\right)\cdot\left(\mathbf{P}_i - q_i\mathbf{A}\right)\Big).$$

Substituting the scales introduced at the beginning of the section into the position equation, and rearranging terms, we obtain

$$\frac{d\tilde{\mathbf{x}}_i}{d\tilde{t}} = \frac{1}{r_i}\Big(\frac{TP}{ML}\tilde{\mathbf{P}}_i - \frac{QTA_0}{ML}\tilde{q}_i\tilde{\mathbf{A}}\Big) \equiv \tilde{\mathbf{v}}_i.$$

This equation can be simplified further noting that A_0 is chosen according to (1.36) and the scale for momentum is $P = MLT^{-1}$. Therefore, we obtain the non-dimensionalized position equation

$$\frac{d\tilde{\mathbf{x}}_i}{d\tilde{t}} = \frac{1}{r_i}\Big(\tilde{\mathbf{P}}_i - \tilde{q}_i\tilde{\mathbf{A}}\Big) \equiv \tilde{\mathbf{v}}_i.$$

Following an identical treatment for the generalized momentum equation, using the scales set according to (1.36), we obtain

$$\frac{d\tilde{\mathbf{P}}_i}{d\tilde{t}} = \tilde{q}_i\Big(- \tilde{\nabla}\tilde{\psi} + \frac{1}{r_i}\left(\tilde{\nabla}\tilde{\mathbf{A}}\right)\cdot\left(\tilde{\mathbf{P}}_i - \tilde{q}_i\tilde{\mathbf{A}}\right)\Big).$$

Therefore, the non-dimensional system is (again, dropping tildes for simplicity)

$$\frac{d\mathbf{x}_i}{dt} = \frac{1}{r_i}\Big(\mathbf{P}_i - q_i\mathbf{A}\Big) \equiv \mathbf{v}_i, \tag{1.39}$$

$$\frac{d\mathbf{P}_i}{dt} = q_i\Big(-\nabla\psi + \frac{1}{r_i}\,(\nabla\mathbf{A})\cdot(\mathbf{P}_i - q_i\mathbf{A})\Big). \tag{1.40}$$

The next sections are focused on the non-dimensionalized form of the field equations cast under the Lorenz and Coulomb gauge conditions.

1.3.3 Maxwell's Equations in the Lorenz Gauge

Substituting scales introduced at the beginning of the section into the equations (1.8) - (1.10), we find that

$$\frac{1}{c^2}\frac{\psi_0}{T^2}\frac{\partial^2\tilde{\psi}}{\partial\tilde{t}^2} - \frac{\psi_0}{L^2}\tilde{\Delta}\tilde{\psi} = \frac{Q\bar{n}}{\epsilon_0}\tilde{\rho},$$
$$\frac{1}{c^2}\frac{A_0}{T^2}\frac{\partial^2\tilde{\mathbf{A}}}{\partial\tilde{t}^2} - \frac{A_0}{L^2}\tilde{\Delta}\tilde{\mathbf{A}} = \frac{\mu_0 Q\bar{n}L}{T}\tilde{\mathbf{J}},$$
$$\frac{A_0}{L}\tilde{\nabla}\cdot\tilde{\mathbf{A}} + \frac{1}{c^2}\frac{\psi_0}{T}\frac{\partial\tilde{\psi}}{\partial\tilde{t}} = 0.$$

The first equation can be rearranged to obtain

$$\frac{L^2}{c^2T^2}\frac{\partial^2\tilde{\psi}}{\partial\tilde{t}^2} - \tilde{\Delta}\tilde{\psi} = \frac{L^2Q\bar{n}}{\epsilon_0\psi_0}\tilde{\rho}.$$

Similarly, with the second equation we obtain

$$\frac{L^2}{c^2T^2}\frac{\partial^2\tilde{\mathbf{A}}}{\partial\tilde{t}^2} - \tilde{\Delta}\tilde{\mathbf{A}} = \frac{Q\bar{n}VL^2}{c^2\epsilon_0A_0}\tilde{\mathbf{J}},$$

where we have used $V = LT^{-1}$ as well as the fact that $c^2 = (\mu_0\epsilon_0)^{-1}$. Finally, the gauge condition becomes

$$\tilde{\nabla}\cdot\tilde{\mathbf{A}} + \frac{\psi_0V}{c^2A_0}\frac{\partial\tilde{\psi}}{\partial\tilde{t}} = 0.$$

Introducing the normalized speed of light $\kappa = c/V$, and selecting ψ_0 and A_0 from (1.36), we find that the above equations simplify to (dropping the tildes)

$$\frac{1}{\kappa^2}\frac{\partial^2\psi}{\partial t^2} - \Delta\psi = \frac{1}{\sigma_1}\rho, \tag{1.41}$$

$$\frac{1}{\kappa^2}\frac{\partial^2\mathbf{A}}{\partial t^2} - \Delta\mathbf{A} = \sigma_2\mathbf{J}, \tag{1.42}$$

$$\nabla\cdot\mathbf{A} + \frac{1}{\kappa^2}\frac{\partial\psi}{\partial t} = 0, \tag{1.43}$$

where we have introduced the new parameters

$$\sigma_1 = \frac{M\epsilon_0}{QT\bar{n}}, \quad \sigma_2 = \frac{Q^2L^2\bar{n}\mu_0}{M}. \tag{1.44}$$

These are nothing more than normalized versions of the permittivity and permeability constants in the original equations.

1.3.4 Maxwell's Equations in the Coulomb Gauge

Following an approach identical to the one used for the Lorenz gauge in the previous section, one finds that the non-dimensional form of Maxwell's equations in the Coulomb gauge is given by

$$-\Delta\psi = \frac{1}{\sigma_1}\rho, \tag{1.45}$$

$$\frac{1}{\kappa^2}\frac{\partial^2\mathbf{A}}{\partial t^2} - \Delta\mathbf{A} = \sigma_2\mathbf{J} - \frac{1}{\kappa^2}\frac{\partial\left(\nabla\psi\right)}{\partial t}, \tag{1.46}$$

$$\nabla\cdot\mathbf{A} = 0, \tag{1.47}$$

where σ_1 and σ_2 are defined according to (1.44) and $\kappa = c/V$ is the normalized speed of light.

1.4 Contributions of This Thesis

The remainder of this thesis is devoted to the design of algorithms for solving the VM system using a PIC method. The particles will be advanced in the proposed methods according to either the Newton-Lorentz equations (1.37)-(1.38) or the generalized formulation (1.39)-(1.40). In the latter approach, particles are evolved using the smooth potentials ψ and $\mathbf{A}$, as well as their derivatives,

which has the advantage of eliminating time derivatives from the gauge formulation. The former approach for moving particles is largely standard and serves not only as a useful benchmark, but also demonstrates ways in which our methods can be incorporated into existing methodologies. While the methods developed in this work are more aligned with the Lorenz gauge formulation of Maxwell's equations (1.41)-(1.43), we also include some results obtained with the formulation (1.45)-(1.47) based on the Coulomb gauge.

Chapter 2 provides a discussion of the methods used to evolve the wave equations that appear in the gauge formulations considered in this work. The core algorithms for the fields are quite similar to those presented in the thesis [42], as well as the article [34]. The latter article analyzed stability and characterized other properties of the second-order solvers, including dissipation and dispersion. This work extends these ideas by introducing new methods for the construction of analytical derivatives that can be obtained directly from the base solver with no degradation in the rate of convergence. This provides a more general approach for the calculation of derivatives, which can be leveraged in problems with non-uniform meshes and non-trivial geometries [40] and costs the same as the base solver. In contrast, the particle methods developed in [41] used centered finite-differences on staggered Cartesian grids, which reduce the accuracy of the fields by one order in space. We also revisit outflow boundary conditions, focusing primarily on the multi-dimensional setting and propose a form of outflow that is based on extrapolation. Additional details concerning these approaches is provided to clarify the ambiguity in some of the earlier presentations. Time and space refinement experiments are used to demonstrate the effectiveness of the proposed approaches.

In chapter 3, we discuss parallel algorithms for successive convolution methods (see, e.g., [38, 43, 44, 45]). Our paper [39] developed a nearest-neighbor domain decomposition algorithm by leveraging certain decay properties of the methods, which ultimately allowed the algorithms to run on distributed memory systems. Using the Kokkos performance portability library [46], we assessed the performance of several different strategies of dispatching threads in the shared memory space, focusing on optimizing common looping patterns in the algorithms. Weak and strong scaling experiments were performed on both linear and nonlinear problems to study the efficiency of the

algorithms.

The focus of chapter 4 concerns the development of new PIC methods for the simulation of plasmas. We first introduce the time stepping methods used to evolve particles in the simulations, including leapfrog integration as well as the popular Boris rotation method [4]. These methods are used to forge a comparison with the generalized momentum formulation (1.39)-(1.40), which is evolved using algorithms for non-separable Hamiltonian systems [36, 47]. Several techniques are proposed in an effort to control errors in the gauge conditions and enforce charge conservation. We demonstrate the performance of the methods on several test problems, beginning with a single particle test in fixed fields, before moving to more challenging beam problems.

The last chapter of this thesis provides a high-level summary of the results. We also discuss several ideas for future work. This includes outlining possible improvements to the methods presented here, as well as other interesting ideas and test problems for which there was not enough time to explore.

CHAPTER 2

NUMERICAL METHODS FOR THE FIELD EQUATIONS

In this chapter, we describe algorithms used for wave propagation, which will be used in the formulations of Maxwell's equations discussed in the previous chapter. We begin with a general discussion of Green's function methods and integral equations in section 2.1, which are the foundation of the approaches considered in this work. The discussion of the solvers is primarily focused on two types of second-order accurate (time) methods, which are presented in section 2.2 in their corresponding semi-discrete form. Using a dimensional splitting technique, which is presented at the end of the same section, we formulate the solution in terms of one-dimensional operators that can be inverted using the methods discussed in section 2.3. We then discuss the application of boundary conditions in section 2.4, which also includes caveats for multi-dimensional problems. Additionally, we show how to construct derivatives of the fields analytically, which retains the convergence rate of the original method. This section also discusses outflow boundary conditions, which have presented a challenge for this class of methods. Following [38], we briefly discuss the extensions of these methods to higher-order accuracy in time and take care to address complications for boundary conditions in these methods. We present some numerical results in section 2.6, which demonstrate the accuracy of the proposed methods. Lastly, in section 2.7, we provide a brief summary of the contributions in this work.

2.1 Integral Equation Methods and Green's Functions

Integral equation methods or, more generally, Green's function methods, are a powerful class of techniques used in the solution of boundary value problems that occur in a range of applications, including mechanics, fluid dynamics, and electromagnetism. Such methods allow one to write an explicit solution of an elliptic PDE in terms of a fundamental solution or Green's function. While explicit, this solution can be difficult or impossible to evaluate, so numerical quadrature is used to evaluate these terms. So-called layer potentials can then be introduced in the form of

surface integrals, which are used to adjust the solution to satisfy the prescribed boundary data. This framework allows one to solve problems in complicated domains without resorting to the use of a mesh. We illustrate the basics of the method with an example.

Suppose that we are solving the following modified Helmholtz equation

$$\left(\mathcal{I} - \frac{1}{\alpha^2}\Delta\right) u(\mathbf{x}) = S(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{2.1}$$

where $\Omega \subset \mathbb{R}^n$ and $\mathcal{I}$ is the identity operator, Δ is the Laplacian operator in $\mathbb{R}^n$, S is a source term, and $\alpha \in \mathbb{R}$ is a parameter. While this method can be broadly applied to other elliptic PDEs, equation (2.1) is of interest to us because it can be obtained from the time discretization of a parabolic or hyperbolic PDE. In this case, the source function would include additional time levels of u and the parameter $\alpha = \alpha(\Delta t)$ would be connected to the time discretization of the original problem being solved. We shall not prescribe boundary conditions for this equation, and instead consider the most general solution.

To apply a Green's function method to equation (2.1), one first needs to identify a function $G(\mathbf{x}, \mathbf{y})$ that solves the equation

$$\left(\mathcal{I} - \frac{1}{\alpha^2}\Delta\right) G(\mathbf{x}, \mathbf{y}) = \delta\left(\mathbf{x} - \mathbf{y}\right), \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \tag{2.2}$$

over free-space, with $\delta\left(\mathbf{x} - \mathbf{y}\right)$ being the Dirac delta distribution. There are many ways to approach solving this equation. For example, it is common to take advantage of radial symmetry so that the problem reduces to a single variable, which can be solved using a Fourier transform. Green's functions have been tabulated for many different operators, including the modified Helmholz operator. Therefore, we shall not elaborate on this further and assume that the fundamental solution $G(\mathbf{x}, \mathbf{y})$ is readily available.

We now show the connection between fundamental solution $G(\mathbf{x}, \mathbf{y})$, which is defined on free-space and solves (2.2), and the original problem (2.1). First, let u be a solution of the problem

(2.1). If we multiply the equation (2.2) by u and integrate over Ω:

$$\int_\Omega \left[\left(\mathcal{I} - \frac{1}{\alpha^2}\Delta\right) G(\mathbf{x},\mathbf{y})\right] u(\mathbf{y})\, dV_\mathbf{y} = \int_\Omega \delta\left(\mathbf{x}-\mathbf{y}\right) u(\mathbf{y})\, dV_\mathbf{y}, \tag{2.3}$$
$$= u(\mathbf{x}),$$

using properties of the delta distribution. The left side of this equation can be addressed using integration by parts. First, we split the left side into two terms:

$$\int_\Omega \left[\left(\mathcal{I} - \frac{1}{\alpha^2}\Delta\right) G(\mathbf{x},\mathbf{y})\right] u(\mathbf{y})\, dV_\mathbf{y} = \int_\Omega G(\mathbf{x},\mathbf{y}) u(\mathbf{y})\, dV_\mathbf{y} - \frac{1}{\alpha^2}\int_\Omega \Delta G(\mathbf{x},\mathbf{y}) u(\mathbf{y})\, dV_\mathbf{y}. \tag{2.4}$$

Using integration by parts and the divergence theorem, we find that the second integral can be expressed as

$$\begin{aligned}\int_\Omega \Delta G(\mathbf{x},\mathbf{y}) u(\mathbf{y})\, dV_\mathbf{y} &= \int_\Omega \nabla\cdot\left(\nabla G(\mathbf{x},\mathbf{y}) u(\mathbf{y})\right)\, dV_\mathbf{y} - \int_\Omega \nabla G(\mathbf{x},\mathbf{y})\cdot\nabla u(\mathbf{y})\, dV_\mathbf{y},\\ &= \int_{\partial\Omega} u(\mathbf{y})\frac{\partial G}{\partial \mathbf{n}}\, dS_\mathbf{y} - \int_\Omega \nabla G(\mathbf{x},\mathbf{y})\cdot\nabla u(\mathbf{y})\, dV_\mathbf{y}.\end{aligned}$$

Here we have used $\partial/\partial\mathbf{n}$ to denote the normal derivative. If we, again, apply integration by parts along with the divergence theorem to the second integral, we find that

$$\begin{aligned}\int_\Omega \nabla G(\mathbf{x},\mathbf{y})\cdot\nabla u(\mathbf{y})\, dV_\mathbf{y} &= \int_\Omega \nabla\cdot\left(G(\mathbf{x},\mathbf{y})\nabla u(\mathbf{y})\right)\, dV_\mathbf{y} - \int_\Omega G(\mathbf{x},\mathbf{y})\Delta u(\mathbf{y})\, dV_\mathbf{y},\\ &= \int_{\partial\Omega} G(\mathbf{x},\mathbf{y})\frac{\partial u}{\partial \mathbf{n}}\, dS_\mathbf{y} - \int_\Omega G(\mathbf{x},\mathbf{y})\Delta u(\mathbf{y})\, dV_\mathbf{y}.\end{aligned}$$

Combining each of these results with the relation (2.4), and using this in place of the left side of (2.3), we obtain, after some simplifications, the equation

$$u(\mathbf{x}) = \int_\Omega G(\mathbf{x},\mathbf{y})\left(\mathcal{I} - \frac{1}{\alpha^2}\Delta\right) u(\mathbf{x})\, dV_\mathbf{y} + \frac{1}{\alpha^2}\int_{\partial\Omega}\left(G(\mathbf{x},\mathbf{y})\frac{\partial u}{\partial\mathbf{n}} - u(\mathbf{y})\frac{\partial G}{\partial\mathbf{n}}\right)\, dS_\mathbf{y}.$$

Finally, since u solves the PDE (2.1), the above equation is equivalent to

$$u(\mathbf{x}) = \int_\Omega G(\mathbf{x},\mathbf{y}) S(\mathbf{y})\, dV_\mathbf{y} + \frac{1}{\alpha^2}\int_{\partial\Omega}\left(G(\mathbf{x},\mathbf{y})\frac{\partial u}{\partial\mathbf{n}} - u(\mathbf{y})\frac{\partial G}{\partial\mathbf{n}}\right)\, dS_\mathbf{y}. \tag{2.5}$$

Since the volume integral term does not enforce boundary conditions, the surface integral contributions involving u are replaced with layer potentials to ensure that these conditions are satisfied.

Therefore, the general solution, shown above, is replaced with the ansatz

$$u(\mathbf{x}) = \int_{\Omega} G(\mathbf{x}, \mathbf{y}) S(\mathbf{y}) \, dV_{\mathbf{y}} + \int_{\partial\Omega} \left(\sigma(\mathbf{y}) G(\mathbf{x}, \mathbf{y}) + \gamma(\mathbf{y}) \frac{\partial G}{\partial \mathbf{n}} \right) dS_{\mathbf{y}}, \tag{2.6}$$

where $\sigma(\mathbf{y})$ is the single-layer potential and $\gamma(\mathbf{y})$ is the double-layer potential, which must now be determined. The names reflect the behavior of the Green's function associated with each of the terms. The Green's function itself is continuous, but its derivative will have a jump. Based on the boundary conditions, one selects either a single or double layer form as the ansatz for the solution. The single layer form is used in the Neumann problem, while the double layer form is chosen for the Dirichlet problem.

Numerically evaluating the solution (2.6) first requires a discretization of the integrals using quadrature. The evaluation proceeds by computing the volume integral term, which is the particular solution to the problem (2.1). Using the particular solution, one obtains the homogeneous solution with modified boundary data that accounts for the particular solution's contributions along the boundary. This step for the homogeneous solution requires the identification of $\sigma(\mathbf{y})$ or $\gamma(\mathbf{y})$ at the quadrature points taken along the domain boundary, which results in a dense linear system. In contrast to other classes of solvers, e.g, finite-element or finite-difference schemes, these linear systems are usually well-conditioned, so the application of an iterative solver, such as the GMRES method [27], converges with only a few iterations and is independent of the number of quadrature points. For efficiency, the fast-multipole method (FMM) [48] can be used to reduce the evaluation time required in the GMRES iterations [49].

The algorithms presented in the subsequent sections are essentially a one-dimensional analogue of these methods. Rather than invert the multi-dimensional operator corresponding to (2.6), the methods presented here, instead, factor the Laplacian and invert one-dimensional operators, dimension-by-dimension, using the one-dimensional form of (2.6). We will see that the resulting methods solve for something that looks like a layer potential, with the key difference being that the linear system is now a 2×2 matrix that can be inverted by hand rather than with an iterative method. Similarly, the particular solution along a given line segment can be rapidly computed with a lightweight, recursive, fast summation method, rather than a more complicated method, such as the

FMM. Moreover, these methods retain the geometric flexibility since the domain can be represented using one-dimensional line segments with termination points specified by the geometry.

2.2 Semi-discrete Schemes for the Wave Equation

Here we provide a description of the second-order accurate wave solvers based on backwards-difference (BDF) and time-centered discretizations. We show how to derive the semi-discrete equations associated with each of the methods, which take the form of modified Helmholtz equations (2.1). Then, we discuss the splitting technique that is used for multi-dimensional problems.

2.2.1 The BDF Scheme

To derive the BDF form of the wave solver, we start with the equation

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \Delta u = S(\mathbf{x}, t), \tag{2.7}$$

where c is the wave speed and S is a source function. Then, using the notation $u(\mathbf{x}, t^n) = u^n$, we can apply a second-order accurate backwards finite-difference stencil for the second derivative

$$\frac{\partial^2 u}{\partial t^2}\Big|_{t=t^{n+1}} = \frac{2u^{n+1} - 5u^n + 4u^{n-1} - u^{n-2}}{\Delta t^2} + O(\Delta t^2),$$

where $\Delta t = t^k - t^{k-1}$, for any k, is the grid spacing in time. Evaluating the remaining terms in equation (2.7) at time level t^{n+1}, and inserting the above difference approximation, we obtain

$$\frac{1}{c^2\Delta t^2}\left(2u^{n+1} - 5u^n + 4u^{n-1} - u^{n-2}\right) - \Delta u^{n+1} = S^{n+1}(\mathbf{x}) + O\left(\frac{1}{\alpha^2}\right),$$

which can be rearranged to obtain the semi-discrete equation

$$\left(\mathcal{I} - \frac{1}{\alpha^2}\Delta\right)u^{n+1} = \frac{1}{2}\left(5u^n - 4u^{n-1} + u^{n-2}\right) + \frac{1}{\alpha^2}S^{n+1}(\mathbf{x}) + O\left(\frac{1}{\alpha^4}\right), \tag{2.8}$$

where we have introduced the parameter $\alpha = \sqrt{2}/(c\Delta t)$. We note that the source term is treated implicitly in this method, which creates additional complications if the source function S depends on u. This necessitates some form of iteration, which increases the cost of the method.

Stability properties for the semi-discrete equation (2.8) were presented in the thesis [42], which showed that the method, above, is purely diffusive and unconditionally stable. Higher-order BDF methods can be obtained simply by using a wider finite-difference stencil to approximate the time derivative $\partial_{tt}u$. While moving to higher-order reduces the (overly) diffusive feature of the second-order method, there are other concerns surrounding the stability of such methods.

2.2.2 Time-centered Scheme

In the semi-discrete form of the second-order BDF method, we saw that the source term was treated implicitly, which, in some cases, requires some form of iteration. If the iteration procedure converges slowly, this can increase the cost of the method significantly. Another approach to deal with this problem is to use a time-centered method, in which the source is treated explicitly. In this case, the second time derivative is approximated with a second-order centered difference

$$\frac{\partial^2 u}{\partial t^2}\Big|_{t=t^n} = \frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} + O(\Delta t^2),$$

where, again, $\Delta t = t^k - t^{k-1}$, for any k, is the grid spacing in time. Evaluating the equation (2.7) at time level t^n, and using this difference approximation, we obtain

$$\frac{1}{c^2\Delta t^2}\left(u^{n+1} - 2u^n + u^{n-1}\right) - \Delta u^n = S^n(\mathbf{x}) + O(\Delta t^2). \tag{2.9}$$

To obtain a semi-discrete equation of the form (2.1) for the data u^{n+1}, the Laplacian term is made implicit through the introduction of the term

$$\Delta\left(\frac{u^{n+1} - 2u^n + u^{n-1}}{\beta^2}\right), \quad \beta \in \mathbb{R},$$

which is added and subtracted from both sides of (2.9) to obtain (after some rearrangement)

$$\frac{\beta^2}{c^2\Delta t^2}\left(u^{n+1} - 2u^n + u^{n-1}\right) - \Delta\left(u^{n+1} - (2-\beta^2)u^n + u^{n-1}\right) \\ = \beta^2 S^n(\mathbf{x}) - \Delta\left(u^{n+1} - 2u^n + u^{n-1}\right) + O\left(\beta^2\Delta t^2\right).$$

To make the semi-discrete equation take the form (2.1), we add $\frac{\beta^4}{c^2\Delta t^2}u^n$ to both sides of the equation, so that

$$\left(\frac{\beta^2}{c^2\Delta t^2} - \Delta\right)\left[u^{n+1} - (2-\beta^2)u^n + u^{n-1}\right] = \frac{\beta^4}{c^2\Delta t^2}u^n + \beta^2 S^n(\mathbf{x}) \\ - \Delta\left(u^{n+1} - 2u^n + u^{n-1}\right) + O\left(\beta^2\Delta t^2\right).$$

If we now write $\alpha = \beta/(c\Delta t)$, and multiply through by $1/\alpha^2$, this equation can be written as

$$\left(\mathcal{I} - \frac{1}{\alpha^2}\Delta\right)\left[u^{n+1} - (2-\beta^2)u^n + u^{n-1}\right] = \beta^2 u^n + \frac{\beta^2}{\alpha^2}S^n(\mathbf{x}) + O\left(\frac{\beta^2}{\alpha^4}\right), \tag{2.10}$$

where we have used the fact that the Laplacian term on the right side satisfies

$$\Delta\left(u^{n+1} - 2u^n + u^{n-1}\right) = \Delta\left(\partial_{tt}u^n\right)\Delta t^2 + O(\Delta t^4) = O(\Delta t^2) \equiv O\left(\frac{1}{\alpha^2}\right).$$

In contrast to the semi-discrete equation (2.8), it was shown that the time-centered update (2.10) is purely dispersive [42]. Through stability analysis, it was shown that an unconditionally stable scheme could be obtained as long as $0 < \beta \leq 2$.

2.2.3 Splitting Method Used for Multi-dimensional Problems

The semi-discrete equations (2.8) and (2.10) are both modified Helmholtz equations of the form (2.1). Rather than appealing to (2.6), which formally inverts the multi-dimensional modified Helmholtz operator, we apply a factorization into a product of one dimensional operators. For example, in two-spatial dimensions the factorization writes

$$\begin{aligned}\mathcal{I} - \frac{1}{\alpha^2}\Delta &= \left(\mathcal{I} - \frac{1}{\alpha^2}\partial_{xx}\right)\left(\mathcal{I} - \frac{1}{\alpha^2}\partial_{yy}\right) + \frac{1}{\alpha^4}\partial_{xx}\partial_{yy}, \\ &\equiv \mathcal{L}_x\mathcal{L}_y + \frac{1}{\alpha^4}\partial_{xx}\partial_{yy},\end{aligned}$$

where $\mathcal{L}_x$ and $\mathcal{L}_y$ are one-dimensional operators and the last term represents the splitting error associated with the factorization step. For second-order accuracy in time, the coefficient of the splitting error is $1/\alpha^4 = O(\Delta t^4)$, so we shall ignore this term. Therefore, the semi-discrete equation (2.8) and (2.10) can be written more compactly (dropping error terms) as

$$\mathcal{L}_x\mathcal{L}_y u^{n+1} = \frac{1}{2}\left(5u^n - 4u^{n-1} + u^{n-2}\right) + \frac{1}{\alpha^2}S^{n+1}(\mathbf{x}), \quad \alpha := \frac{\sqrt{2}}{c\Delta t}, \tag{2.11}$$

and

$$\mathcal{L}_x\mathcal{L}_y\left[u^{n+1} - (2-\beta^2)u^n + u^{n-1}\right] = \beta^2 u^n + \frac{\beta^2}{\alpha^2}S^n(\mathbf{x}), \quad \alpha := \frac{\beta}{c\Delta t}, \quad 0 < \beta \leq 2, \tag{2.12}$$

respectively.

2.3 Inverting One-dimensional Operators

The choice of factoring the multi-dimensional modified Helmholtz operator means we now have to solve a sequence of one-dimensional boundary value problems (BVPs) of the form

$$\left(\mathcal{I} - \frac{1}{\alpha^2}\partial_{xx}\right) w(x) = f(x), \quad x \in [a,b], \tag{2.13}$$

where $[a,b]$ is a one-dimensional line and f is a new source term that can be used to represent a time history or an intermediate variable constructed from the inversion of an operator along another direction. We also point out that the parameter α depends on the choice of the semi-discrete scheme employed to solve the problem. For the BDF scheme $\alpha = \sqrt{2}/(c\Delta t)$, while the centered scheme uses $\alpha = \beta/(c\Delta t)$, with $0 < \beta \leq 2$. We will show the process by which one obtains the general solution to the problem (2.13), deferring the application of boundary conditions to section 2.4.

2.3.1 Integral Solution

Since the BVP (2.13) is linear, its general solution can be expressed using the one dimensional analogue of equation (2.5):

$$w(x) = \int_a^b G(x,y)f(y)\,dy + \frac{1}{\alpha^2}\left[G(x,y)\partial_y u(y) - u(y)\partial_y G(x,y)\right]\Big|_{y=a}^{y=b}. \tag{2.14}$$

A simple way to obtain the free-space Green's function for this problem is to use a Fourier transform. In Fourier space, this equation reads

$$\left(1 + \frac{k^2}{\alpha^2}\right)\widehat{G} = 1 \implies \widehat{G} = \frac{\alpha^2}{\alpha^2 + k^2}.$$

A closely related Fourier transform is obtained with the function

$$\mathcal{F}\left[e^{-\lambda|x|}\right] = \frac{2\lambda}{\lambda^2 + k^2},$$

from which, it follows that

$$\mathcal{F}\left[\frac{\lambda}{2}e^{-\lambda|x|}\right] = \frac{\lambda^2}{\lambda^2 + k^2}.$$

Therefore, matching transforms, it follows that the free-space Green's function in one-dimension is

$$G(x, y) = \frac{\alpha}{2}e^{-\alpha|x-y|}. \tag{2.15}$$

To use the relation (2.14), we need to compute the derivatives in the Green's function. We note that

$$\partial_y G(x, y) = \begin{cases} \alpha e^{-\alpha(x-y)}, & x \geq y, \\ -\alpha e^{-\alpha(y-x)}, & x < y. \end{cases}$$

Taking limits, we find that

$$\lim_{y\to a} \partial_y G(x, y) = \alpha e^{-\alpha(x-a)},$$

$$\lim_{y\to b} \partial_y G(x, y) = -\alpha e^{-\alpha(b-x)}.$$

Combining these limits with (2.15) and (2.14), we obtain the general solution

$$w(x) = \frac{\alpha}{2}\int_a^b e^{-\alpha|x-y|} f(y)\, dy + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}, \tag{2.16}$$

where A and B are constants that are determined by boundary conditions. Comparing with (2.6), these terms serve the same purpose as the layer potentials. Further, we identify the general solution (2.16) as the inverse of the one-dimensional modified Helmholtz operator. In other words, we define $\mathcal{L}_x^{-1}$ so that

$$w(x) = \mathcal{L}_x^{-1}\left[f\right](x), \tag{2.17}$$

$$\equiv \frac{\alpha}{2}\int_a^b e^{-\alpha|x-y|} f(y)\, dy + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}, \tag{2.18}$$

$$\equiv \mathcal{I}_x[f](x) + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}. \tag{2.19}$$

Section 2.4 will make repeated use of (2.17)-(2.19) to illustrate the application of boundary conditions. Additionally, the result (2.16) is general enough that it can be used for both the BDF and time-centered schemes.

2.3.2 Fast Summation Method

In order to compute the inverse operators according to (2.17)-(2.19), suppose we have discretized the one-dimensional computational domain $[a, b]$ into a mesh consisting of $N + 1$ grid points:

$$a = x_0 < x_1 < \cdots < x_N = b,$$

with the spacing defined by

$$\Delta x_j = x_j - x_{j-1}, \quad j = 1, \cdots N.$$

If we directly evaluate the function $w(x)$ at each of the mesh points, according to (2.18), we obtain

$$w(x_i) = \frac{\alpha}{2} \int_a^b e^{-\alpha|x_i - y|} f(y)\, dy + Ae^{-\alpha(x_i - a)} + Be^{-\alpha(b - x_i)}, \quad i = 0, \cdots, N + 1.$$

Since the evaluation of the integral term in the variable y requires with quadrature requires $O(N)$ operations, this direct approach requires a total of $O(N^2)$ operations.

With the aid of a recursive fast summation algorithm, the cost of evaluating these terms can be reduced from $O(N^2)$ to $O(N)$. To this end, it is helpful to introduce the operators

$$\mathcal{I}_x^R[f](x) \equiv \alpha \int_a^x e^{-\alpha(x-y)} f(y)\, dy, \tag{2.20}$$

$$\mathcal{I}_x^L[f](x) \equiv \alpha \int_x^b e^{-\alpha(y-x)} f(y)\, dy, \tag{2.21}$$

so that the total integral over $[a, b]$ can be expressed as the average of these operators

$$\mathcal{I}_x[f](x) = \frac{1}{2} \left(\mathcal{I}_x^R[f](x) + \mathcal{I}_x^L[f](x) \right). \tag{2.22}$$

Then, the task now relies on computing the integrals (2.20) and (2.21) in an efficient manner. To develop a recursive expression, consider evaluating the integral (2.20) at a grid point x_i. Then,

it follows that

$$\begin{aligned}
\mathcal{I}_x^R[f](x_i) &= \alpha \int_a^{x_i} e^{-\alpha(x_i-y)} f(y)\, dy, \\
&= \alpha \int_a^{x_{i-1}} e^{-\alpha(x_i-y)} f(y)\, dy + \alpha \int_{x_{i-1}}^{x_i} e^{-\alpha(x_i-y)} f(y)\, dy, \\
&= \alpha \int_a^{x_{i-1}} e^{-\alpha(x_i-x_{i-1}+x_{i-1}-y)} f(y)\, dy + \alpha \int_{x_{i-1}}^{x_i} e^{-\alpha(x_i-y)} f(y)\, dy, \\
&= \alpha \int_a^{x_{i-1}} e^{-\alpha(\Delta x_i+x_{i-1}-y)} f(y)\, dy + \alpha \int_{x_{i-1}}^{x_i} e^{-\alpha(x_i-y)} f(y)\, dy, \\
&= e^{-\alpha\Delta x_i} \left(\alpha \int_a^{x_{i-1}} e^{-\alpha(x_{i-1}-y)} f(y)\, dy \right) + \alpha \int_{x_{i-1}}^{x_i} e^{-\alpha(x_i-y)} f(y)\, dy, \\
&\equiv e^{-\alpha\Delta x_i} \mathcal{I}_x^R[f](x_{i-1}) + \mathcal{J}_x^R[f](x_i).
\end{aligned}$$

In the last line, we have introduced the local integral

$$\mathcal{J}_x^R[f](x_i) = \alpha \int_{x_{i-1}}^{x_i} e^{-\alpha(x_i-y)} f(y)\, dy.$$

We see that the integral (2.20) can be expressed through a recursive weighting of its previous values plus an additional term that is localized in space. In the next chapter, we use a variation of this method to develop a domain decomposition algorithm that allows the method to scale on parallel computers. To initialize the recursion, we set $\mathcal{I}_x^R[f](x_0) = 0$, which follows directly from its definition (2.20). Since the calculation of the local integrals $\mathcal{J}_x^R[f](x_i)$ employs quadrature over a collection of M points, the cost of computing (2.20) is now of the form $O(MN)$. Further more, since the number of localized quadrature points M is independent of the mesh size N, and we additionally select $M \ll N$, the resulting approach scales as $O(N)$. A similar argument is made for the second integral (2.21).

In summary, the fast summation method computes the integrals (2.20) and (2.21) according to

$$\mathcal{I}_x^R[f](x_i) = e^{-\alpha\Delta x_i} \mathcal{I}_x^R[f](x_{i-1}) + \mathcal{J}_x^R[f](x_i), \quad \mathcal{I}_x^R[f](x_0) = 0, \quad i = 1, \cdots N, \tag{2.23}$$

$$\mathcal{I}_x^L[f](x_i) = e^{-\alpha\Delta x_{i+1}} \mathcal{I}_x^L[f](x_{i+1}) + \mathcal{J}_x^L[f](x_i), \quad \mathcal{I}_x^L[f](x_N) = 0, \quad i = 0, \cdots N-1, \tag{2.24}$$

where the local integrals are defined by

$$\mathcal{J}_x^R[f](x_i) = \alpha \int_{x_{i-1}}^{x_i} e^{-\alpha(x_i - y)} f(y)\, dy, \quad i = 1, \cdots N, \tag{2.25}$$

$$\mathcal{J}_x^L[f](x_i) = \alpha \int_{x_i}^{x_{i+1}} e^{-\alpha(y - x_i)} f(y)\, dy, \quad i = 0, \cdots N-1. \tag{2.26}$$

Next, we discuss the approximations used in the evaluation of the local integrals.

2.3.3 Approximating the Local Integrals

Here, we present the general process used to obtain quadrature rules for the local integrals defined by (2.25) and (2.26), in the case of a uniform grid, i.e.,

$$\Delta x = x_j - x_{j-1}, \quad j = 1, \cdots, N.$$

Rather than use numerical quadrature rules, e.g., Gaussian quadrature or Newton-Cotes formulas, for purposes of stability, it was discovered that a certain form of analytical integration was required [50]. In this approach, the operand $f(x)$ is approximated by an interpolating function, which is then analytically integrated against the kernel. We provide a sketch of the approach to illustrate the idea. Specific details can be found in a number of papers, e.g., [34, 43, 44].

First, it is helpful to transform the integrals (2.25) and (2.26) using a change of variable. Consider the integral (2.25) and let

$$y = (x_j - x_{j-1})\tau + x_{j-1} \equiv \Delta x \tau + x_{j-1}, \quad \tau \in [0, 1].$$

Then we can write

$$\mathcal{J}_x^R[f](x_i) = \alpha \Delta x e^{-\alpha \Delta x} \int_0^1 e^{\alpha \tau \Delta x} f(\tau \Delta x + x_{i-1})\, d\tau. \tag{2.27}$$

Next, we approximate $f(x)$ in (2.27) using interpolation of some desired order of accuracy. As an example, suppose that we want to use linear interpolation with the data $\{f_{i-1}, f_i\}$ using the basis $\{1, x - x_{i-1}\}$, which is shifted for convenience to cancel with the shift in (2.27). A direct calculation shows that the interpolating polynomial is

$$p(x) = f_{i-1} + \frac{f_i - f_{i-1}}{\Delta x} (x - x_{i-1}).$$

By replacing f in (2.27) with the above interpolant, and integrating the result analytically, we find that

$$\begin{aligned}\mathcal{J}_x^R[f](x_i) &\approx \alpha\Delta x e^{-\alpha\Delta x}\int_0^1 e^{\alpha\tau\Delta x}\Big(f_{i-1} + (f_i - f_{i-1})\,\tau\Big)\,d\tau,\\ &= w_0 v_{i-1} + w_1 v_i,\end{aligned}$$

where the weights for integration are

$$\begin{aligned}w_0 &= \frac{1 - e^{-\alpha\Delta x} - \alpha\Delta x e^{-\alpha\Delta x}}{\alpha\Delta x},\\ w_1 &= \frac{(\alpha\Delta x - 1) + e^{-\alpha\Delta x}}{\alpha\Delta x}.\end{aligned}$$

Modifications of the above can be made to accommodate additional interpolation points, as well as techniques for shock capturing. In the latter case, methods have been devised following the idea of WENO reconstruction [51] to create quadrature methods that can address non-smooth features including shocks and cusps [44, 45, 52]. Additional details on the WENO quadrature, including the reconstruction stencils can be found in chapter 3. In [45], we developed a quadrature rule using the exponential polynomial basis, which offers additional flexibility in capturing localized features through a "shape" parameter introduced in the basis. These tools offer a promising approach to addressing problems with discontinuities in the material properties as well as more complex domains with non-smooth boundaries. Despite the notable differences in the type of approximating function used for the operand, the process is essentially identical to the example shown here. We also wish to point out that certain issues may arise when $\alpha \gg 1$ (i.e., $\Delta t \ll 1$). In such circumstances, when the weights are computed on-the-fly, the kernel function can be replaced with a Taylor expansion [38]. Otherwise, this results in a "narrow" Green's function that is vastly under-resolved by the mesh, which causes wave phenomena to remain stagnant. Our experience has found this situation to be quite rare, but it is something to be aware of when a small CFL number is used in a simulation.

2.4 Applying Boundary Conditions

In this section, we discuss the application of boundary conditions for the schemes based on BDF and centered time discretizations. Boundary conditions are presented for these methods from the perspective of one-dimensional problems in sections 2.4.1 and 2.4.2. Lastly, in section 2.4.3, we describe how these conditions can be used in multi-dimensional problems.

2.4.1 BDF Method

The update for second order BDF method, in one-spatial dimension, can be obtained by combining (2.16) with the semi-discrete equation (2.8). Defining the operand

$$R(x) = \frac{1}{2}\left[5u^n - 4u^{n-1} + u^{n-2}\right](x) + \frac{1}{\alpha^2}S^{n+1}(x),$$

we obtain

$$u^{n+1}(x) = \frac{\alpha}{2}\int_a^b e^{-\alpha|x-y|}R(y)\,dy + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}, \tag{2.28}$$

$$\equiv \mathcal{I}_x[R](x) + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}, \tag{2.29}$$

where we have used $\mathcal{I}_x[\cdot]$ to denote the term involving the convolution integral which is not to be confused with the identity operator. Applying different boundary conditions amounts to determining the values of A and B used in (2.29). In the description of boundary conditions for the method, we shall assume that the boundary conditions at the ends of the one-dimensional domain are the same. Using slight variations of the methods illustrated below, one can mix the boundary conditions at the ends of the line segments.

In order to enforce conditions on the derivatives of the solution, we will also need to compute a derivative of the update (2.28) (equivalently (2.29)). For this, we observe that the dependency for x appears only on analytical functions, i.e., the Green's function (kernel) and the exponential functions in the boundary terms. To differentiate (2.28) we start with the definition (2.22), which splits the integral at the point $y = x$ and makes the kernel easier to manipulate. Then, using the

fundamental theorem of calculus, we can calculate derivatives of (2.20) and (2.21) to find that

$$\frac{d}{dx}\left(\mathcal{I}_x^R[f](x)\right) = \frac{d}{dx}\left(\alpha \int_a^x e^{-\alpha(x-y)} f(y)\, dy\right) = -\alpha \mathcal{I}_x^R[f](x) + f(x), \tag{2.30}$$

$$\frac{d}{dx}\left(\mathcal{I}_x^L[f](x)\right) = \frac{d}{dx}\left(\alpha \int_x^b e^{-\alpha(y-x)} f(y)\, dy\right) = \alpha \mathcal{I}_x^L[f](x) - f(x). \tag{2.31}$$

These results can be combined according to (2.22), which provides an expression for the derivative of the convolution term:

$$\frac{d}{dx}\Big(\mathcal{I}_x[f](x)\Big) = \frac{\alpha}{2}\left(-\mathcal{I}_x^R[f](x) + \mathcal{I}_x^L[f](x)\right). \tag{2.32}$$

Additionally, by evaluating this equation at the ends of the interval, we obtain the identities

$$\frac{d}{dx}\Big(\mathcal{I}_x[f](a)\Big) = \alpha \mathcal{I}_x[f](a), \tag{2.33}$$

$$\frac{d}{dx}\Big(\mathcal{I}_x[f](b)\Big) = -\alpha \mathcal{I}_x[f](b), \tag{2.34}$$

which are helpful in enforcing the boundary conditions.

The relation (2.32) can be used to obtain a derivative for the solution at the new time level. From the update (2.29), a direct computation reveals that

$$\frac{du^{n+1}}{dx} = \frac{\alpha}{2}\left(-\mathcal{I}_x^R[R](x) + \mathcal{I}_x^L[R](x)\right) - \alpha A e^{-\alpha(x-a)} + \alpha B e^{-\alpha(b-x)}. \tag{2.35}$$

Notice that no additional approximations have been made beyond what is needed to compute $\mathcal{I}_x^R$ and $\mathcal{I}_x^L$, which are needed in the base method. For this reason, we think of equation (2.35) as an analytical derivative. The boundary coefficients A and B appearing in (2.35) will be calculated in the same way as the update (2.29), which are discussed in the remaining subsections. This treatment ensures that the discrete derivative will be consistent with the conditions imposed on the solution variable.

2.4.1.1 Dirichlet Boundary Conditions

Suppose we are given the function values along the boundary, which is represented by the data

$$u^{n+1}(a) = g_a\left(t^{n+1}\right), \quad u^{n+1}(b) = g_b\left(t^{n+1}\right).$$

If we evaluate the BDF-2 update (2.29) at the ends of the interval, we obtain the conditions

$$g_a\left(t^{n+1}\right) = \mathcal{I}_x[R](a) + A + \mu B,$$
$$g_b\left(t^{n+1}\right) = \mathcal{I}_x[R](b) + \mu A + B,$$

where we have defined $\mu = e^{-\alpha(b-a)}$. This is a simple linear system for the boundary coefficients A and B, which can be inverted by hand. Proceeding, we find that

$$A = \frac{g_a\left(t^{n+1}\right) - \mathcal{I}_x[R](a) - \mu\left(g_b\left(t^{n+1}\right) - \mathcal{I}_x[R](b)\right)}{1-\mu^2},$$
$$B = \frac{g_b\left(t^{n+1}\right) - \mathcal{I}_x[R](b) - \mu\left(g_a\left(t^{n+1}\right) - \mathcal{I}_x[R](a)\right)}{1-\mu^2}.$$

2.4.1.2 Neumann Boundary Conditions

We can also enforce conditions on the derivatives at the end of the domain. Given the Neumann data

$$\frac{du^{n+1}(a)}{dx} = h_a\left(t^{n+1}\right), \quad \frac{du^{n+1}(b)}{dx} = h_b\left(t^{n+1}\right),$$

we can evaluate the derivative formula for the update (2.35) and use the identities (2.33) and (2.34). Performing these evaluations, we obtain the system of equations

$$-A + \mu B = \frac{1}{\alpha} h_a\left(t^{n+1}\right) - \mathcal{I}_x[R](a),$$
$$-\mu A + B = \frac{1}{\alpha} h_b\left(t^{n+1}\right) + \mathcal{I}_x[R](b),$$

where, again, $\mu = e^{-\alpha(b-a)}$. Solving this system, we find that

$$A = -\frac{\frac{1}{\alpha}h_a\left(t^{n+1}\right) - \mathcal{I}_x[R](a) - \mu\left(\frac{1}{\alpha}h_b\left(t^{n+1}\right) + \mathcal{I}_x[R](b)\right)}{1-\mu^2},$$
$$B = -\frac{\mu\left(\frac{1}{\alpha}h_a\left(t^{n+1}\right) - \mathcal{I}_x[R](a)\right) - \left(\frac{1}{\alpha}h_b\left(t^{n+1}\right) + \mathcal{I}_x[R](b)\right)}{1-\mu^2}.$$

We note that Robin boundary conditions, which combine Dirichlet and Neumann conditions can be enforced in a nearly identical way.

2.4.1.3 Periodic Boundary Conditions

Periodic boundary conditions are enforced by taking

$$u^{n+1}(a) = u^{n+1}(b), \quad \partial_x u^{n+1}(a) = \partial_x u^{n+1}(b).$$

Enforcing these conditions through the update (2.29) and its derivative (2.35), using the identities (2.33)-(2.34), leads to the system of equations

$$(1-\mu)A + (\mu-1)B = \mathcal{I}_x[R](b) - \mathcal{I}_x[R](a),$$
$$(\mu-1)A + (\mu-1)B = -\mathcal{I}_x[R](b) - \mathcal{I}_x[R](a),$$

with $\mu = e^{-\alpha(b-a)}$. The solution of this system, after some simplifications is given by

$$A = \frac{\mathcal{I}_x[R](b)}{1-\mu},$$
$$B = \frac{\mathcal{I}_x[R](a)}{1-\mu}.$$

2.4.1.4 Outflow Boundary Conditions

In problems defined over free-space, we must allow for waves to exit the computational domain. Additionally, as the waves exit the domain, we would like to minimize the number of reflections, which are non-physical, along this boundary. Exit conditions can be formulated in one spatial dimension in the sense of characteristics, by requiring that

$$\frac{\partial u}{\partial t} - c\frac{\partial u}{\partial x} = 0, \quad x = a, \tag{2.36}$$
$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \quad x = b, \tag{2.37}$$

where $c > 0$ is a wave speed.

To formulate the boundary conditions for the BDF-2 update (2.28), we follow the approach used in [34], which developed outflow boundary conditions for the central method, discussed in the next section. We start from the free-space solution that is defined over the real line:

$$u^{n+1}(x) = \frac{\alpha}{2}\int_{-\infty}^{\infty} e^{-\alpha|x-y|}R(y)\,dy.$$

Here, $R(x)$ is the operand for the BDF-2 method and is defined as

$$R(x) = \frac{1}{2}\left(5u^n - 4u^{n-1} + u^{n-2}\right).$$

The free-space solution can then be separated to isolate the computational domain over $[a, b]$, which we write as

$$u^{n+1}(x) = \mathcal{I}_x[R](x) + \frac{\alpha}{2}\int_{-\infty}^{a} e^{-\alpha|x-y|}R(y)\,dy + \frac{\alpha}{2}\int_{b}^{\infty} e^{-\alpha|x-y|}R(y)\,dy.$$

If we use the definitions

$$A = \frac{\alpha}{2}\int_{-\infty}^{a} e^{-\alpha(a-y)}R(y)\,dy, \tag{2.38}$$

$$B = \frac{\alpha}{2}\int_{b}^{\infty} e^{-\alpha(y-b)}R(y)\,dy, \tag{2.39}$$

then the decomposed free-space solution can be written in the familiar form (2.29):

$$u^{n+1}(x) = \mathcal{I}_x[R](x) + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}.$$

Now assume that the support of $u(x, 0)$ and $R(x)$ is entirely contained within the domain $[a, b]$ initially. Using the finite speed of propagation c, it follows that the region of support at time t^n extends to $[a - ct^n, b + ct^n]$, which means that the integrals (2.38) and (2.39) can be simplified to

$$A^n = \frac{\alpha}{2}\int_{a-ct^n}^{a} e^{-\alpha(a-y)}R(y)\,dy, \tag{2.40}$$

$$B^n = \frac{\alpha}{2}\int_{b}^{b+ct^n} e^{-\alpha(y-b)}R(y)\,dy. \tag{2.41}$$

Since these integrals are defined over regions of space outside of the computational domain, the idea is now to exchange space with time, using the characteristics of equations (2.36) and (2.37).

Along the right boundary, the waves propagate to the right so that for any $x > b$, the solution to (2.37) is $u(x, t) = u(x - ct)$. Tracing the ray backwards in time, we see that

$$u(b + y, t) = u\left(b, t - \frac{y}{c}\right), \quad y > 0. \tag{2.42}$$

Note that on a space-time plot, the characteristic has a slope of c^{-1}. Similarly, for along the left boundary, we have that $x < a$, and the solution to (2.36) is given by $u(x, t) = u(x + ct)$. Here, the

characteristic has slope $-c^{-1}$, so we obtain

$$u(a-y,t) = u\left(a, t - \frac{y}{c}\right), \quad y > 0. \tag{2.43}$$

In other words, since the data remains constant along characteristics, it follows that any point outside of the computational domain corresponds to a boundary point at some earlier time.

We now show the illustrate the process for converting space integrals to recursive time integrals by considering the integral (2.41). The argument for (2.40) is identical and is therefore omitted. To use the ray tracing formula (2.42), we shift the integration variable so that

$$\begin{aligned}
B^n &= \frac{\alpha}{2}\int_b^{b+ct^n} e^{-\alpha(y-b)} R(y)\, dy, \\
&= \frac{\alpha}{2}\int_0^{ct^n} e^{-\alpha y} R(b+y)\, dy, \\
&\equiv \frac{\alpha}{2}\int_0^{ct^n} e^{-\alpha y}\left[\frac{1}{2}\left(5u^n(b+y) - 4u^{n-1}(b+y) + u^{n-2}(b+y)\right)\right] dy, \\
&= \frac{\alpha}{2}\int_0^{ct^n} e^{-\alpha y}\left[\frac{1}{2}\left(5u\left(b, t^n - y/c\right) - 4u\left(b, t^{n-1} - y/c\right) + u\left(b, t^{n-2} - y/c\right)\right)\right] dy,
\end{aligned}$$

where the last line applies (2.42) to the time history. If we now apply another transformation

$$y/c \mapsto y,$$

then the integral becomes

$$\begin{aligned}
B^n &= \frac{\alpha c}{2}\int_0^{t^n} e^{-\alpha c y}\left[\frac{1}{2}\left(5u\left(b, t^n - y\right) - 4u\left(b, t^{n-1} - y\right) + u\left(b, t^{n-2} - y\right)\right)\right] dy, \\
&= \frac{\alpha c}{2}\int_0^{\Delta t} e^{-\alpha c y}\left[\frac{1}{2}\left(5u\left(b, t^n - y\right) - 4u\left(b, t^{n-1} - y\right) + u\left(b, t^{n-2} - y\right)\right)\right] dy \\
&\quad + \frac{\alpha c}{2}\int_{\Delta t}^{t^n} e^{-\alpha c y}\left[\frac{1}{2}\left(5u\left(b, t^n - y\right) - 4u\left(b, t^{n-1} - y\right) + u\left(b, t^{n-2} - y\right)\right)\right] dy, \\
&\equiv I_1 + I_2.
\end{aligned}$$

The integral I_2 exhibits a recursive form. If we shift the integration bounds by $-\Delta t$, then it

follows that I_2 is now given by

$$\begin{aligned} I_2 &= \frac{\alpha c}{2} \int_0^{t^{n-1}} e^{-\alpha c(y+\Delta t)} \left[\frac{1}{2} \left(5u \left(b, t^{n-1} - y \right) - 4u \left(b, t^{n-2} - y \right) + u \left(b, t^{n-3} - y \right) \right) \right] dy, \\ &= e^{-\alpha c \Delta t} \left(\frac{\alpha c}{2} \int_0^{t^{n-1}} e^{-\alpha c y} \left[\frac{1}{2} \left(5u \left(b, t^{n-1} - y \right) - 4u \left(b, t^{n-2} - y \right) + u \left(b, t^{n-3} - y \right) \right) \right] dy \right), \\ &\equiv e^{-\alpha c \Delta t} B^{n-1}. \end{aligned} \tag{2.44}$$

The integral term I_1, which is defined over $[0, \Delta t]$, can be mapped to the interval $[0, 1]$ using the transformation

$$y = z\Delta t, \quad z \in [0, 1].$$

Then it follows that

$$I_1 = \frac{\alpha c \Delta t}{2} \int_0^1 e^{-\alpha c \Delta t z} \left[\frac{1}{2} \left(5u \left(b, t^n - z\Delta t \right) - 4u \left(b, t^{n-1} - z\Delta t \right) + u \left(b, t^{n-2} - z\Delta t \right) \right) \right] dz. \tag{2.45}$$

To evaluate this local integral (in time), we store use a set of time history along the boundary point to construct an interpolating function. After inserting this approximation into (2.45), the integration can be performed analytically to generate the weights for the time history data. This is done in the same spirit as the approach for computing quadrature weights to approximate the local integrals, which was discussed in section 2.3.3. We demonstrate this process using two approaches, which differ in the stencil used to interpolate the time history.

A simple way to approximate the integral (2.45) is to build an "explicit" interpolating function based on the time history $\left\{ u^{n-2}(b), u^{n-1}(b), u^{n-1}(b) \right\}$. Using the algebraic polynomial basis and performing the integration of the resulting interpolant analytically, we obtain an approximation of the form

$$I_1 \approx \gamma_0 u^n(b) + \gamma_1 u^{n-1}(b) + \gamma_2 u^{n-2}(b),$$

where the outflow weights are

$$\begin{aligned} \gamma_0 &= -\frac{5\nu + 2e^{-\nu} + \nu^2 e^{-\nu} - 5\nu^2 - 3\nu e^{-\nu} - 2}{4\nu^2}, \\ \gamma_1 &= -\frac{\nu^2 e^{-\nu} - 2e^{-\nu} - 4\nu + 2\nu^2 + 2\nu e^{-\nu} + 2}{2\nu^2}, \\ \gamma_2 &= \frac{e^{-\nu}(-1+\nu)(\nu - 2e^{\nu} + \nu e^{\nu} + 2)}{4\nu^2}, \end{aligned}$$

with $\nu = \sqrt{2}$. Combining this result with (2.44), we obtain the explicit update formula

$$B^n = e^{-\alpha c \Delta t} B^{n-1} + \gamma_0 u^n(b) + \gamma_1 u^{n-1}(b) + \gamma_2 u^{n-2}(b). \tag{2.46}$$

In the case of outflow boundary conditions for the second-order, time-centered update, which was presented in [34], the authors mentioned that including $u^{n+1}(b)$ in the interpolation stencil is necessary for a convergent outflow method (see Remark 3 in [34]). Numerical experiments, which we present in section 2.6 seem to suggest otherwise. This includes the BDF-2 method with the explicit form of outflow given by (2.46).

An "implicit" form of the outflow procedure can be obtained by including time level $n+1$ data in the interpolation stencil used to approximate the integral (2.45). Repeating the steps shown above, we obtain a corresponding implicit formula

$$I_1 \approx \gamma_0 u^{n+1}(b) + \gamma_1 u^n(b) + \gamma_2 u^{n-1}(b) + \gamma_3 u^{n-2}(b),$$

where the outflow weights are

$$\begin{aligned}
\gamma_0 &= -\frac{\nu^2 e^{-\nu} - 6e^{-\nu} - 12\nu - 3\nu^3 e^{-\nu} + 8\nu^2 + 6\nu e^{-\nu} + 6}{12\nu^3}, \\
\gamma_1 &= \frac{4\nu^2 e^{-\nu} - 6e^{-\nu} - 10\nu - 4\nu^3 e^{-\nu} + 3\nu^2 + 5\nu^3 + 4\nu e^{-\nu} + 6}{4\nu^3}, \\
\gamma_2 &= -\frac{5\nu^2 e^{-\nu} - 8\nu - \nu^3 e^{-\nu} + 4\nu^3 + 2\nu e^{-\nu} + 6}{4\nu^3}, \\
\gamma_3 &= -\frac{6\nu + 6e^{-\nu} - 4\nu^2 e^{-\nu} + \nu^2 - 3\nu^3 - 6}{12\nu^3},
\end{aligned}$$

again, with $\nu = \sqrt{2}$. To deal with the appearance of $u^{n+1}(b)$ in the approximation of (2.45), we appeal to the update for the BDF-2 scheme (2.29). Assuming that outflow is applied to the left boundary point, as well, we can repeat the above process to obtain a linear system, as with the other types of boundary conditions discussed earlier. This results in the linear system

$$\begin{aligned}
(1-\gamma_0)A^n - \gamma_0 \mu B^n &= e^{-\alpha c \Delta t} A^{n-1} + \gamma_0 \mathcal{I}_x[R](a) + \gamma_1 u^n(a) + \gamma_2 u^{n-1}(a) + \gamma_3 u^{n-2}(a) \equiv f_a, \\
-\gamma_0 \mu A^n + (1-\gamma_0)B^n &= e^{-\alpha c \Delta t} B^{n-1} + \gamma_0 \mathcal{I}_x[R](b) + \gamma_1 u^n(b) + \gamma_2 u^{n-1}(b) + \gamma_3 u^{n-2}(b) \equiv f_b,
\end{aligned}$$

where $\mu = e^{-\alpha(b-a)}$ and we have used f_a and f_b to indicate the right side of the system for brevity. This system can be analytically inverted and leads to the solution

$$A^n = \frac{(1-\gamma_0)\, f_a + \gamma_0 \mu f_b}{(1-\gamma_0)^2 - (\gamma_0 \mu)^2},$$
$$B^n = \frac{\gamma_0 \mu f_a + (1-\gamma_0)\, f_b}{(1-\gamma_0)^2 - (\gamma_0 \mu)^2}.$$

This finishes the introduction for outflow boundary conditions. In section 2.4.3, we mention some of the caveats in applying the aforementioned boundary conditions to problems in a multi-dimensional setting. Next, in section 2.4.2, we present the boundary conditions for the second-order, time-centered scheme.

2.4.2 Centered Method

The update for second order time centered method, in one spatial dimension, can be obtained by combining (2.16) with the semi-discrete equation (2.10) to obtain

$$u^{n+1}(x) = (2-\beta^2)u^n - u^{n-1} + \beta^2 \left(\frac{\alpha}{2} \int_a^b e^{-\alpha|x-y|} \left[u^n + \frac{1}{\alpha^2} S^n \right] dy \right) + A e^{-\alpha(x-a)} + B e^{-\alpha(b-x)}, \tag{2.47}$$

$$\equiv (2-\beta^2)u^n - u^{n-1} + \beta^2 \mathcal{I}_x \left[u^n + \frac{1}{\alpha^2} S^n \right](x) + A e^{-\alpha(x-a)} + B e^{-\alpha(b-x)}, \tag{2.48}$$

where we have, again, used $\mathcal{I}_x[\cdot]$ to denote the convolution integral term. By specifying different conditions on the boundary data for the solution u, we can identify the corresponding values of A and B in the update (2.48). As with the BDF methods discussed in the previous sections, we will assume that the boundary conditions along the line are the same, though this can be generalized to mixed boundary conditions with minimal modifications. The techniques used to obtain boundary conditions are nearly identical to the BDF-2 method shown in the previous section, so we skip the details and simply state the results.

As with the BDF method, we can obtain a method for constructing derivatives by directly differentiating the update (2.48). Making use of the identity (2.32), the derivative at the new time

level is found to be

$$\frac{du^{n+1}}{dx} = (2-\beta^2)\frac{du^n}{dx} - \frac{du^{n-1}}{dx} + \frac{\beta^2\alpha}{2}\left(-\mathcal{I}_x^R\left[u^n + \frac{1}{\alpha^2}S^n\right](x) + \mathcal{I}_x^L\left[u^n + \frac{1}{\alpha^2}S^n\right](x)\right) - \alpha A e^{-\alpha(x-a)} + \alpha B e^{-\alpha(b-x)} \quad (2.49)$$

In contrast to the derivative (2.35) obtained with the BDF-2 method, we can see that the derivative of the time-centered method includes a time history for the derivative itself. We have not shown that computing derivatives with a recursive approach of this form is a stable process. Otherwise, no additional approximations have been made beyond what is needed to compute $\mathcal{I}_x^R$ and $\mathcal{I}_x^L$. As with the BDF method, the boundary coefficients A and B appearing in (2.49) will be calculated in the same way as the update (2.48).

2.4.2.1 Dirichlet Boundary Conditions

Dirichlet boundary conditions for which we are given the data

$$u^{n+1}(a) = g_a\left(t^{n+1}\right), \quad u^{n+1}(b) = g_b\left(t^{n+1}\right),$$

can be enforced by solving the linear system that results from evaluating (2.48) at the ends of the domain. The solution to this system is

$$A = -\frac{f_a - \mu f_b}{1-\mu^2},$$
$$B = -\frac{f_b - \mu f_a}{1-\mu^2},$$

where we have used $\mu = e^{-\alpha(b-a)}$ and

$$f_a = \mathcal{I}_x\left[u^n + \frac{1}{\alpha^2}S^n\right](a) - g_a(t^n) - \frac{g_a\left(t^{n+1}\right) - 2g_a(t^n) + g_a\left(t^{n-1}\right)}{\beta^2},$$
$$f_b = \mathcal{I}_x\left[u^n + \frac{1}{\alpha^2}S^n\right](b) - g_b(t^n) - \frac{g_b\left(t^{n+1}\right) - 2g_b(t^n) + g_b\left(t^{n-1}\right)}{\beta^2},$$

for brevity.

2.4.2.2 Neumann Boundary Conditions

The boundary coefficients associated with Neumann boundary conditions

$$\frac{du^{n+1}(a)}{dx} = h_a\left(t^{n+1}\right), \quad \frac{du^{n+1}(b)}{dx} = h_b\left(t^{n+1}\right),$$

can be determined using the derivative (2.49) with the aid of the identities (2.33) and (2.34). The resulting linear system has the solution

$$A = \frac{w_a - \mu w_b}{1-\mu^2},$$
$$B = \frac{w_a - \mu w_b}{1-\mu^2},$$

where we have used the definitions

$$w_a = \mathcal{I}_x\left[u^n + \frac{1}{\alpha^2}S^n\right](a) - \frac{1}{\alpha}h_a\left(t^n\right) - \frac{h_a\left(t^{n+1}\right) - 2h_a\left(t^n\right) + h_a\left(t^{n-1}\right)}{\alpha\beta^2},$$
$$w_b = \mathcal{I}_x\left[u^n + \frac{1}{\alpha^2}S^n\right](b) + \frac{1}{\alpha}h_b\left(t^n\right) - \frac{h_b\left(t^{n+1}\right) - 2h_b\left(t^n\right) + h_b\left(t^{n-1}\right)}{\alpha\beta^2},$$

and $\mu = e^{-\alpha(b-a)}$. An approach for Robin boundary conditions follows a nearly identical path.

2.4.2.3 Periodic Boundary Conditions

For periodic boundary conditions, we assume that for any time level $n > 0$

$$u^{n+1}(a) = u^{n+1}(b), \quad \partial_x u^{n+1}(a) = \partial_x u^{n+1}(b).$$

To enforce these conditions, we can appeal to the update (2.48), its derivative (2.49), and the identities (2.33) and (2.34). By solving the linear system obtained from the evaluation of these updates at the ends of the domain, we obtain the coefficients

$$A = \frac{\mathcal{I}_x\left[u^n + \frac{1}{\alpha^2}S^n\right](b)}{1-\mu},$$
$$B = \frac{\mathcal{I}_x\left[u^n + \frac{1}{\alpha^2}S^n\right](a)}{1-\mu},$$

where, again, $\mu = e^{-\alpha(b-a)}$.

2.4.2.4 Outflow Boundary Conditions

The procedure used to derive outflow boundary coefficients for the time-centered method was originally presented in [34], so we shall skip many of the details here. In fact, the developments for the time-centered method can be treated as a simplification of the process used in section 2.4.1.4, which developed the outflow coefficients for the BDF-2 method.

Repeating the steps shown in section 2.4.1.4, with the central scheme, one finds that

$$\begin{aligned} B^n &= \frac{\alpha c}{2} \int_0^{t^n} e^{-\alpha c y} u\left(b, t^n - y\right)\, dy, \\ &= \frac{\alpha c}{2} \int_0^{\Delta t} e^{-\alpha c y} u\left(b, t^n - y\right)\, dy \\ &\quad + \frac{\alpha c}{2} \int_{\Delta t}^{t^n} e^{-\alpha c y} u\left(b, t^n - y\right)\, dy, \\ &\equiv I_1 + I_2, \end{aligned}$$

where the second integral I_2 can be written in the form

$$\begin{aligned} I_2 &= \frac{\alpha c}{2} \int_0^{t^{n-1}} e^{-\alpha c (y + \Delta t)} u\left(b, t^{n-1} - y\right)\, dy, \\ &= e^{-\beta} \left(\frac{\alpha c}{2} \int_0^{t^{n-1}} e^{-\alpha c y} u\left(b, t^{n-1} - y\right)\, dy \right), \\ &\equiv e^{-\beta} B^{n-1}. \end{aligned} \tag{2.50}$$

Note that the simplifications shown above use the definition $\alpha = \beta/(c\Delta t)$ for the central scheme.

Likewise, the first integral I_1, can be expressed as

$$I_1 = \frac{\beta}{2} \int_0^1 e^{-\beta z} u\left(b, t^n - z\Delta t\right)\, dz, \tag{2.51}$$

using the definition $\alpha = \beta/(c\Delta t)$. As with the BDF-2 method, this integral can be approximated in an explicit or implicit manner depending on the choice of points used to create the interpolating function for the function $u\left(b, t^n - z\Delta t\right)$.

An explicit outflow method can be obtained if we approximate the function $u\left(b, t^n - z\Delta t\right)$ using algebraic polynomials with the stencil $\left\{u^{n-2}(b), u^{n-1}(b), u^n(b)\right\}$. Integrating this analytically, and

combining the result with (2.50), we obtain the explicit outflow method

$$B^n = e^{-\beta} B^{n-1} + \gamma_0 u^n(b) + \gamma_1 u^{n-1}(b) + \gamma_2 u^{n-2}(b),$$

where the integration weights for the second-order, time-centered method are given by

$$\begin{aligned}
\gamma_0 &= \frac{2e^{-\beta} - \beta + 2\beta^2 + 3\beta e^{-\beta} - 2}{4\beta^2}, \\
\gamma_1 &= -\frac{2e^{-\beta} - 2\beta + 3\beta^2 e^{-\beta} + 4\beta e^{-\beta} - 2}{6\beta^2}, \\
\gamma_2 &= -\frac{\beta + 2e^{-\beta} + \beta e^{-\beta} - 2}{12\beta^2}.
\end{aligned}$$

An implicit form of the outflow method for the central-2 scheme can be obtained using the stencil $\left\{u^{n-1}(b), u^n(b), u^{n+1}(b)\right\}$ to approximation of $u\left(b, t^n - z\Delta t\right)$. Using the update (2.48) eliminates the variable $u^{n+1}(b)$ from the interpolation formula, which is not yet available. Repeating the steps for the other boundary point, we obtain the linear system

$$\begin{aligned}
\left(1 + \beta^2\gamma_0\right) A^n + \mu\beta^2\gamma_0 B^n &= f_a, \\
\mu\beta^2\gamma_0 A^n + (1 + \beta^2\gamma_0) B^n &= f_b,
\end{aligned}$$

with

$$\begin{aligned}
f_a &= e^{-\beta} A^{n-1} + \gamma_0\beta^2 \mathcal{I}_x \left[u^n + \frac{1}{\alpha^2} S^n\right](a) + \left(\left(2 - \beta^2\right)\gamma_0 + \gamma_1\right) u^n(a) + (\gamma_2 - \gamma_0) u^{n-1}(a), \\
f_b &= e^{-\beta} B^{n-1} + \gamma_0\beta^2 \mathcal{I}_x \left[u^n + \frac{1}{\alpha^2} S^n\right](b) + \left(\left(2 - \beta^2\right)\gamma_0 + \gamma_1\right) u^n(b) + (\gamma_2 - \gamma_0) u^{n-1}(b),
\end{aligned}$$

and $\mu = e^{-\alpha(b-a)}$. The solution to this linear system is given by

$$\begin{aligned}
A^n &= \frac{\left(1 - \beta^2\gamma_0\right) f_a + \mu\beta^2\gamma_0 f_b}{\left(1 - \beta^2\gamma_0\right)^2 - \left(\mu\beta^2\gamma_0\right)^2}, \\
B^n &= \frac{\mu\beta^2\gamma_0 f_a + \left(1 - \beta^2\gamma_0\right) f_b}{\left(1 - \beta^2\gamma_0\right)^2 - \left(\mu\beta^2\gamma_0\right)^2},
\end{aligned}$$

where the corresponding integration weights are defined as

$$\begin{aligned}
\gamma_0 &= -\frac{\beta + 2e^{-\beta} + \beta e^{-\beta} - 2}{4\beta^2}, \\
\gamma_1 &= \frac{2e^{-\beta} + \beta^2 + 2\beta e^{-\beta} - 2}{2\beta^2}, \\
\gamma_2 &= -\frac{2e^{-\beta} - \beta + 2\beta^2 e^{-\beta} + 3\beta e^{-\beta} - 2}{4\beta^2}.
\end{aligned}$$

2.4.3 Some Remarks for Multi-dimensional Problems

In this section we briefly discuss some of the issues concerning the application of boundary conditions for the multi-dimensional updates given by (2.11) for the BDF-2 method and (2.12) for the time-centered method. For convenience, these are given, respectively, by

$$\mathcal{L}_x\mathcal{L}_y u^{n+1} = \frac{1}{2}\left(5u^n - 4u^{n-1} + u^{n-2}\right) + \frac{1}{\alpha^2}S^{n+1}(\mathbf{x}), \quad \alpha := \frac{\sqrt{2}}{c\Delta t},$$

and

$$\mathcal{L}_x\mathcal{L}_y\left[u^{n+1} - (2-\beta^2)u^n + u^{n-1}\right](x,y) = \beta^2 u^n + \frac{\beta^2}{\alpha^2}S^n(x,y), \quad \alpha := \frac{\beta}{c\Delta t}, \quad 0 < \beta \leq 2.$$

By inverting the factored operator one direction at a time using the techniques presented in 2.3, it follows that the solutions to these two equations are given respectively by

$$u^{n+1} = \mathcal{L}_x^{-1}\left[\mathcal{L}_y^{-1}\left[\frac{1}{2}\left(5u^n - 4u^{n-1} + u^{n-2}\right) + \frac{1}{\alpha^2}S^{n+1}\right]\right], \quad \alpha := \frac{\sqrt{2}}{c\Delta t},$$

and

$$u^{n+1} = (2-\beta^2)u^n - u^{n-1} + \beta^2\mathcal{L}_x^{-1}\left[\mathcal{L}_y^{-1}\left[u^n + \frac{1}{\alpha^2}S^n\right]\right], \quad \alpha := \frac{\beta}{c\Delta t}, \quad 0 < \beta \leq 2,$$

We wish to point out here that things are assumed to be smooth so that the ordering conventions used for operators are irrelevant, i.e.,

$$\mathcal{L}_x\mathcal{L}_y = \mathcal{L}_y\mathcal{L}_x.$$

2.4.3.1 Sweeping Patterns in Multi-dimensional Problems

In the two-dimensional case, we need to construct terms of the form

$$\mathcal{L}_y\mathcal{L}_x w = f \implies w = \mathcal{L}_x^{-1}\left[\mathcal{L}_y^{-1}\left[f\right]\right],$$

with boundary data being prescribed for the variable w. The construction is performed over two steps. The first step inverts the y operator, so we obtain

$$\mathcal{L}_x w = \mathcal{L}_y^{-1}\left[f\right]. \tag{2.52}$$

The step (2.52) requires boundary data for the intermediate variable $\mathcal{L}_x w$ when we are only given boundary data for w. From the definition of $\mathcal{L}_x$, we note that

$$\mathcal{L}_x w \equiv \left(\mathcal{I} - \frac{1}{\alpha^2} \partial_{xx} \right) w = w + O\left(\frac{1}{\alpha^2} \right), \tag{2.53}$$

In other words, boundary conditions for $\mathcal{L}_x w$ can be approximated to second-order in time by those of w; however, unless we are dealing with outflow boundary conditions, we do not need to sweep along the boundary of the domain, so the approximation (2.53) is not necessary. Proceeding further, the second step of the inversion process leads to the solution w

$$w = \mathcal{L}_x^{-1} \left[\mathcal{L}_y^{-1} \left[f \right] \right], \tag{2.54}$$

which simply enforces the known boundary data on w.

For purposes of clarity, we separate the types of boundary conditions. In each case, we summarize the changes associated with moving to multi-dimensional problems, which includes any changes necessitated by the proposed methods for calculating derivatives.

2.4.3.2 Periodic Boundary Conditions

Periodic boundary conditions in multi-dimensional problems can be enforced in a straightforward way by directly applying the one-dimensional approaches outlined in sections 2.4.1.3 and 2.4.2.3 to each dimension. No modifications are required for either the scheme or the proposed derivative methods.

2.4.3.3 Dirichlet Boundary Conditions

In the case of Dirichlet boundary conditions, the values of the function are known along the boundary. Therefore, we only need to update the grid points corresponding to the interior of the domain. As mentioned earlier, rather than approximating the boundary conditions for the intermediate sweep, e.g., (2.53), we simply avoid sweeping along the boundary points of the domain, since the values are known. The direction corresponding to the intermediate sweep will

now only use boundary data set by the solution, since the boundaries are left untouched. In the case of homogeneous Dirichlet conditions, the sweeps can be performed on the boundary with no effect.

When sweeping over different directions for the derivatives, we note that along the boundary, the derivative information is not known. Therefore, the sweeps should extend all the way to the boundary. Otherwise the derivative will not be available there. In this case, the boundary data for the intermediate data can be approximated according to (2.53).

2.4.3.4 Neumann Boundary Conditions

The treatment of Neumann boundary conditions in multi-dimensional problems is identical to the Dirichlet case discussed in the previous section for the case of Cartesian grids. This also applies to the proposed methods for computing derivatives. In problems defined on complex geometries with embedded boundaries, based on the theory presented in [53], it was discovered that dissipation was necessary to obtain stable numerical solutions [34]. While this is not a problem for the BDF-2 method, which is dissipative, problems occur for the centered scheme. This motivated the introduction of a tuneable dissipation term using the successive convolution framework for which we provide an overview in section 2.5.1. We also wish to note that Robin boundary conditions follow an identical approach.

2.4.3.5 Outflow Boundary Conditions

In the case of outflow boundary conditions, one should pay careful attention to the structure of the time history used in the convolution integral for a particular scheme. For example, the integrand for the second-order BDF method (2.11) uses data from three time levels $\{u^{n-2}, u^{n-1}, u^n\}$, while the time-centered method uses a single time level, i.e., u^n for this term. This led to two different outflow procedures, which were presented in sections 2.4.1.4 and 2.4.2.4. When dealing with outflow boundaries, the sweeps can be performed along the boundary, since these values are unspecified. We note that this includes derivatives in addition to the fields. Next, we focus on the

structure of the sweeping patterns used in the methods, beginning with the time-centered scheme and then moving to the BDF method.

In the time-centered approach (2.12), the structure of the time history used in the convolution integral follows a regular pattern in the sense that the first and second layers of sweeps defined by (2.52) and (2.54) operate on data from single time level. Assuming the sweeping pattern (2.54), we first need to evaluate the term

$$v^{(1)} = \mathcal{L}_y^{-1}\left[u^n + \frac{1}{\alpha^2}S^n\right],$$

which requires the a time history for the solution u at the x boundary points along the y-direction. The implicit approach to outflow for the time-centered method is constructed using the one-dimensional update (2.48), which is not consistent with what we have written here. Instead, we recommend that the explicit form of the outflow procedure be used for the calculation of the boundary data. For the second set of sweeps given by (2.54), we need to compute

$$v^{(2)} = \mathcal{L}_x^{-1}\left[v^{(1)}\right],$$

which similarly requires the time history for $v^{(1)}$, now along boundary the y boundary points in the x-direction. Since this is the last layer of sweeps, one is free to use either the implicit or explicit forms of outflow to construct the boundary terms. While we have proposed approaches for computing derivatives for the time-centered scheme, we find the structure of the BDF derivatives (2.35) more appealing. Therefore, we shall not discuss multi-dimensional aspects of the derivatives for the time-centered method any further. This will be clarified when we present numerical results in section 2.6.

The BDF-2 scheme (2.11), in contrast to the time-centered method has other nuances which should be addressed. Again, if we assume a sweeping pattern of the form (2.54), we first need to evaluate

$$v^{(1)} = \mathcal{L}_y^{-1}\left[\frac{1}{2}\left(5u^n - 4u^{n-1} + u^{n-2}\right) + \frac{1}{\alpha^2}S^{n+1}\right], \tag{2.55}$$

which requires the a time history for the solution u at the x boundary points along the y-direction. The construction of the boundary coefficients for this term follows the procedure presented in

section 2.4.1.4; however, when one changes directions to perform the second layer of sweeps following (2.54), we see that we now need to construct a term of the form

$$v^{(2)} = \mathcal{L}_x^{-1}\left[v^{(1)}\right],$$

which is identical in structure found with the central-2 scheme. Computing this term with outflow boundary conditions requires that we store a time history for $v^{(1)}$ along the y boundary points in the x-direction. This requires us to change the reconstruction methods based on the order in which sweeps are performed, which creates additional complications if we now wish to interchange the order in which sweeps are performed. Instead, for the multi-dimensional BDF-2 update, we Taylor expand the time history in the convolution integral (2.55) about time level n, so that it looks like the integrand for the time-centered scheme. Assuming the source is not present at the boundary, the integrand can be approximated as

$$\frac{1}{2}\left(5u^n - 4u^{n-1} + u^{n-2}\right) = u^n + O(\Delta t).$$

While this modification results in a loss of accuracy when the method is applied in outflow problems, it creates a regular structure that is easier to work with in the numerical implementation, since the outflow procedure for the time-centered method can now be used in both directions. A consequence of this decision is that this approach is only compatible with explicit outflow because the implicit form of outflow uses knowledge of the particular update, which would be inconsistent.

Lastly, we remark on the treatment of derivatives obtained with the BDF method in the multi-dimensional case. In section 2.4.1, we obtained an equation for spatial derivatives through direct calculations involving the one-dimensional BDF-2 method, which resulted in (2.35). We would like to apply these one-dimensional derivatives in multi-dimensional problems, as well. Since the directions over which sweeps occur are treated independently, the one-dimensional derivatives can also be applied in a dimensionally-split manner. When combined with outflow boundary conditions, care should be taken to ensure that the time history used in the reconstruction is consistent with the operand of a particular convolution integral. To illustrate, assuming a sweeping pattern of the form

(2.54), we find that the BDF-2 update in two spatial dimensions has the form

$$u^{n+1} = \mathcal{L}_x^{-1}\left[\mathcal{L}_y^{-1}\left[R\right]\right], \tag{2.56}$$

where we used

$$R(x,y) = \frac{1}{2}\left(5u^n - 4u^{n-1} + u^{n-2}\right) + \frac{1}{\alpha^2}S^{n+1}.$$

Alternatively, if we swapped the order in which sweeps are performed, we obtain a similar update

$$u^{n+1} = \mathcal{L}_y^{-1}\left[\mathcal{L}_x^{-1}\left[R\right]\right], \tag{2.57}$$

with R being unchanged. Now, consider a y-derivative of the schemes (2.56) and (2.57). Since the inverse operator $\mathcal{L}_x^{-1}$ varies only with respect to x and remains constant in y, we identify two options to compute y-derivatives, namely

$$\partial_y u^{n+1} = \mathcal{L}_x^{-1}\left[\partial_y \mathcal{L}_y^{-1}\left[R\right]\right], \tag{2.58}$$

$$\partial_y u^{n+1} = \partial_y \mathcal{L}_y^{-1}\left[\mathcal{L}_x^{-1}\left[R\right]\right]. \tag{2.59}$$

Both options are valid for computing the derivatives in the multi-dimensional case, but, in problems with outflow boundary conditions, the data that is stored in these approaches is different. In the first approach (2.56), the outflow boundary conditions require the time history for the derivative of the intermediate variable. The second approach (2.57) proceeds in a manner which is more closely related to the update of the solution, since the derivative (2.35) does not change A and B for a given line. For this reason, we prefer the second option (2.57) in our implementation to compute the y derivative. Similarly, the x derivative works with the pattern (2.56).

2.5 Extensions for High-order Accuracy

Here we provide a brief discussion regarding high-order extensions of the aforementioned methods in the context of the two-way wave equation. We include an overview of the successive convolution method for the wave equation, which is loosely taken from [38]. We also mention methods for increasing the accuracy of the BDF methods introduced in section 2.2, including ways to obtain

more accurate spatial derivatives. While we are primarily focused on the developments with the second-order solvers, the purpose of this section is to demonstrate paths to high-order solvers. Therefore, we provide fewer details concerning the implementation of these methods compared to earlier sections.

2.5.1 Successive Convolution Methods

The first work on high-order extensions of the solvers presented in the previous sections was presented in the 2014 paper [38]. Rather than increasing the width of the stencil to approximate the second time derivative, these methods introduced additional spatial derivatives to retain a compact stencil in time using the symmetries appearing in the truncation error for the time-centered method. To illustrate, suppose we are solving the homogeneous wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \Delta u.$$

If the second time derivative is approximated with a second-order centered finite-difference about time level n, then

$$\frac{\partial^2 u}{\partial t^2} = \frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} + \mathcal{O}(\Delta t^2).$$

Taylor expanding of the terms in the numerator of the above approximation yields an expansion containing only even order time derivatives, i.e.,

$$u^{n+1} - 2u^n + u^{n-1} = 2 \sum_{m=1}^{\infty} \frac{\Delta t^{2m}}{(2m)!} \frac{\partial^{2m} u^n}{\partial t^{2m}}.$$

The time derivatives in the above equation were then replaced with spatial derivatives using the PDE to obtain the expansion

$$u^{n+1} - 2u^n + u^{n-1} = 2 \sum_{m=1}^{\infty} \frac{\beta^{2m}}{(2m)!} \left(\frac{\Delta}{\alpha^2} \right)^m u^n, \quad \alpha := \frac{\beta}{c\Delta t}, \quad 0 < \beta \leq \beta_{\max}, \tag{2.60}$$

with the powers of the Laplacian being evaluated in a dimension-by-dimension fashion and the parameter β was introduced to tune the stability of the method. In [38], the powers of the Laplacian were constructed recursively using the convolution operators defined earlier in this chapter, so the

approach became known as successive convolution. To approximate the Laplacian as a convolution, the authors introduce the one-dimensional modified Helmholtz operators

$$\mathcal{L}_\gamma := \mathcal{I} - \frac{1}{\alpha^2}\partial_{\gamma\gamma}, \quad \gamma = x, y, \cdots, \tag{2.61}$$

and another operator

$$\mathcal{D}_\gamma := \mathcal{I} - \mathcal{L}_\gamma^{-1}, \quad \gamma = x, y, \cdots, \tag{2.62}$$

which can be combined to form the Laplacian operator through the relation

$$-\frac{1}{\alpha^2}\Delta = \mathcal{L}_x\mathcal{D}_x + \mathcal{L}_y\mathcal{D}_y + \cdots = -\frac{1}{\alpha^2}\partial_{xx} - \frac{1}{\alpha^2}\partial_{yy} - \cdots$$

The authors introduce a new operator C, which, in two-dimensions, is defined as

$$C_{xy} := \mathcal{L}_y^{-1}\mathcal{D}_x + \mathcal{L}_x^{-1}\mathcal{D}_y, \tag{2.63}$$

so that the Laplacian can be expressed in the factored form

$$-\frac{1}{\alpha^2}\Delta = \mathcal{L}_x\mathcal{L}_y\left[C_{xy}\right].$$

To remove the term $\mathcal{L}_x\mathcal{L}_y$, they introduced yet another operator

$$\mathcal{D}_{xy} := \mathcal{I} - \mathcal{L}_x^{-1}\mathcal{L}_y^{-1}, \tag{2.64}$$

which can be rearranged to obtain the identity

$$\mathcal{L}_x\mathcal{L}_y = \left(\mathcal{I} - \mathcal{D}_{xy}\right)^{-1}.$$

With this last identity, the Laplacian becomes

$$-\frac{1}{\alpha^2}\Delta = \left(\mathcal{I} - \mathcal{D}_{xy}\right)^{-1} C_{xy}.$$

Then, they expand the Laplacian into a power series to obtain the result

$$\left(\frac{\Delta}{\alpha^2}\right)^m = (-1)^m C_{xy}^m \sum_{p=m}^{\infty} \binom{p-1}{m-1} \mathcal{D}_{xy}^{p-m}. \tag{2.65}$$

Note that the expansion above are valid because $||\mathcal{D}_{xy}|| \leq 1$ in the sense of operator norms. This can be seen from the one dimensional analogue (2.62). In Fourier space, the operator $\mathcal{D}_x$ satisfies

$$\mathcal{F}\left[\mathcal{D}_x\right] = 1 - \mathcal{F}\left[\mathcal{L}_x^{-1}\right] = 1 - \frac{1}{1 + (k/\alpha)^2} = \frac{(k/\alpha)^2}{1 + (k/\alpha)^2} \leq 1.$$

With slight modifications, one can similarly show that $\mathcal{F}\left[\mathcal{D}_{xy}\right] \leq 1$ also holds.

By inserting the identity (2.65) into the error expansion (2.60), they obtained a family of methods defined by

$$u^{n+1} = 2u^n - u^{n-1} + 2\sum_{p=1}^{N}\sum_{m=1}^{p}(-1)^m \frac{\beta^{2m}}{(2m)!}\binom{p-1}{m-1} C_{xy}^m \mathcal{D}_{xy}^{p-m}\left[u^n\right], \quad \alpha := \frac{\beta}{c\Delta t}, \quad 0 < \beta \leq \beta_{\max}, \quad (2.66)$$

where the truncation of the outer sum to N terms led to a method of order $2N$ in time. The value of $\beta_{\max}$ in this expansion depends on the desired order of the scheme and decreases as the order of the scheme increases. For specific information on this particular topic, we refer the reader to the original paper [38], which analyzes the stability in great detail.

To include source terms, the expansion (2.60) is modified to account for space and time derivatives on the source itself. We wish to point out that there is a typographical error in the fourth-order scheme with sources presented in the paper [38]. The correct form of the scheme is given by

$$u^{n+1} = 2u^n - u^{n-1} - \beta^2 C_{xy}\left[u^n\right] - \left(\beta^2 \mathcal{D}_{xy} - \frac{\beta^4}{12} C_{xy}\right) C_{xy}\left[u^n\right] + \frac{\beta^2}{12\alpha^2}\left(S^{n+1} + 10S^n + S^{n-1}\right) - \frac{\beta^4}{12\alpha^2} C_{xy}\left[S^n\right]. \quad (2.67)$$

The above scheme contains an implicit source term that arises from an approximation of the second time derivative about time level n. It should be noted that a different stencil can be used to make the source explicit.

The approach used to apply boundary conditions in multi-dimensional problems with successive convolution is similar to the procedures presented for the second-order methods in section 2.4.3 with some caveats. Periodic boundary conditions can be applied level-by-level in the truncated

operator expansions directly following the procedure outlined in section 2.4.2.3 for the second-order time-centered scheme. Other boundary conditions such as Dirichlet and Neumann leverage the linearity of the wave equation being solved to enforce boundary conditions at the lowest level and using a homogeneous variant for the remaining levels. Higher-order outflow methods with successive convolution were introduced in the thesis [54], and were later published in the article [40]. We wish to mention that the outflow methods proposed in the paper [40] showed large errors along the boundaries in numerical experiments, despite being fourth-order. These methods are not be considered in this work, so we shall not elaborate on this any further. The subject of boundary conditions, especially high-order outflow, shall be the focus of future work, once this situation is better understood for the second-order methods.

2.5.2 BDF Methods

A straightforward way of increasing the accuracy of the BDF method can be achieved by increasing the size of the time history. In place of the second-order accurate approximation for $\partial_{tt} u^{n+1}$, we could instead use the third-order accurate difference given by

$$\partial_{tt} u^{n+1} = \frac{\frac{33}{12}u^{n+1} - \frac{26}{3}u^{n} + \frac{19}{2}u^{n-1} - \frac{14}{3}u^{n-2} + \frac{11}{12}u^{n-3}}{\Delta t^2} + \mathcal{O}(\Delta t^3), \tag{2.68}$$

or even

$$\partial_{tt} u^{n+1} = \frac{\frac{15}{4}u^{n+1} - \frac{77}{6}u^{n} + \frac{107}{6}u^{n-1} - 13u^{n-2} + \frac{61}{12}u^{n-3} - \frac{5}{6}u^{n-4}}{\Delta t^2} + \mathcal{O}(\Delta t^4), \tag{2.69}$$

which is fourth-order accurate. Then, to derive higher-order BDF methods, one can repeat the steps in section 2.2.1, replacing the second-order stencil with either the third-order stencil (2.68) or fourth-order stencil (2.69). This leads to the respective semi-discrete schemes

$$\left(\mathcal{I} - \frac{1}{\alpha^2}\Delta\right) u^{n+1} = \frac{12}{35}\left(\frac{26}{3}u^{n} - \frac{19}{2}u^{n-1} + \frac{14}{3}u^{n-2} - \frac{11}{12}u^{n-3}\right) + \frac{1}{\alpha^2} S^{n+1}(\mathbf{x}) + \mathcal{O}\left(\frac{1}{\alpha^5}\right), \quad \alpha := \frac{\sqrt{\frac{35}{12}}}{c\Delta t}, \tag{2.70}$$

and

$$\left(\mathcal{I} - \frac{1}{\alpha^2}\Delta\right) u^{n+1} = \frac{4}{15}\left(\frac{77}{6}u^n - \frac{107}{6}u^{n-1} + 13u^{n-2} - \frac{61}{12}u^{n-3} + \frac{5}{6}u^{n-4}\right) + \frac{1}{\alpha^2}S^{n+1}(\mathbf{x}) + O\left(\frac{1}{\alpha^6}\right), \quad \alpha := \frac{\sqrt{\frac{15}{4}}}{c\Delta t}. \quad (2.71)$$

The process for calculating derivatives is identical with the only differences from (2.35) being the operand of the convolution integral and the particular definition of the parameter α, so we shall not present these equations. Additionally, boundary conditions for this approach do not present any additional challenges beyond the base second-order scheme. This feature makes higher order BDF methods simple to implement.

Despite the advantages afforded by the BDF methods, there are two issues to address in this type of approach. First, there is the issue of stability. In the case of first-order ODEs, it is well known that BDF discretizations become unstable if the order is greater than 6. Similar issues will be encountered in these methods and we plan to address the topic of stability in our later work. The second issue, which is less of a concern, is the splitting error in multi-dimensional problems that arises from the factorization used for the modified Helmholtz operator; however, it is possible to eliminate the splitting error by essentially “subtracting" it from the scheme with the aid of an iterative method [43].

2.6 Numerical Examples

In this section, we present numerical results for the field solvers introduced in this chapter. Using a one-dimensional test problem, we first compare the performance of the second-order BDF and time-centered derivatives. Then, we assess the performance of the proposed methods in multi-dimensional problems, focusing on the types of boundary conditions required in the problems considered in chapter 4. In the case of outflow boundary conditions, we also demonstrate the complications that arise when moving from one- to two-dimensional problems.

2.6.1 BDF and Time-centered Derivatives in One Spatial Dimension

As a first test, we perform a refinement study to compare the derivatives obtained with the second-order BDF and time-centered methods, which are given, respectively, by equations (2.35) and (2.49). The test problem we consider is the homogeneous two-way scalar wave equation

$$\frac{1}{c^2}\partial_{tt}u - \partial_{xx}u = 0, \tag{2.72}$$

with $c = 1$ and subject to periodic boundary conditions on $[0, 2\pi]$. The solution is evolved to the final time $T = 1$ and we use the initial data

$$u(x, 0) = \sin(x), \quad \partial_t u(x, 0) = 0. \tag{2.73}$$

This equation has the solution

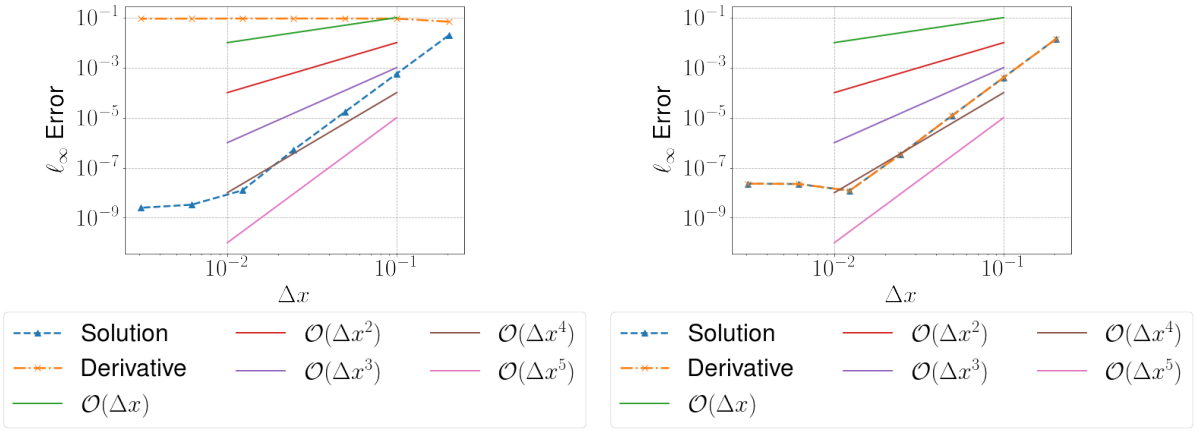
$$u(x, t) = \frac{\sin(x - t) + \sin(x + t)}{2}, \tag{2.74}$$

with the corresponding derivative

$$\partial_x u(x, t) = \frac{\cos(x - t) + \cos(x + t)}{2}. \tag{2.75}$$

Since these are multi-step methods, we need to supply values for the time history. While we can certainly use Taylor expansions to approximate this data, we use the analytical solution (2.74) and its corresponding spatial derivative (2.75). This allows us to avoid any potential errors associated with the initialization. We also wish to point out that the splitting error is not present in one-dimensional problems, so this eliminates another source of error.

Regarding the implementation, we wish to note that unlike the time-centered method (2.49), the BDF method (2.35) does not require additional time history for the derivative. In the experiment shown here, the derivatives for the time-centered method are stored in a time history, similar to the solution, which is updated in each time step of the simulation. A fifth-order spatial quadrature is used to compute the local integrals in both methods. In the time-centered method (2.48) and its derivative (2.49), we use $\beta = 2$, which is the largest allowable β that retains the stability of the method [34].



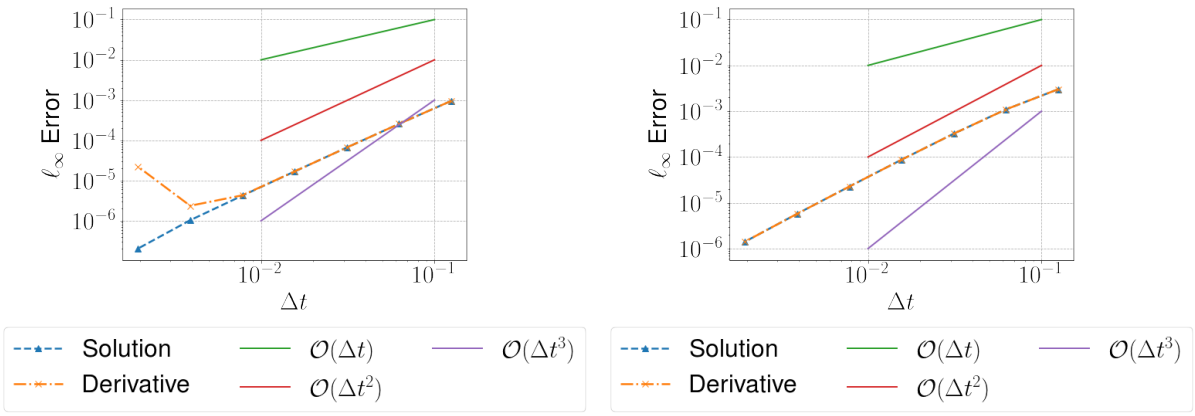
(a) Central-2

(b) BDF-2

Figure 2.1: Spatial refinement study for the solution and its derivative obtained with second-order methods for the periodic test problem in section 2.6.1. In Figure 2.1a, we plot the ℓ_∞ errors for both the numerical solution and the derivative obtained with the time-centered method. Similarly, in Figure 2.1b, we show the same quantities, which are instead computed using the BDF method. The derivative for the time-centered method fails to refine in space, while the BDF derivative is as accurate at the numerical solution itself.

In the refinement experiments for space, we run each case with $N_t = 2^{12}$ time steps. We successively double the number of mesh points beginning with $N_x = 32$ and finishing with $N_x = 2048$. In Figure 2.1 we compare the accuracy of the time-centered and BDF methods and their proposed derivatives, against analytical solutions. The results indicate that the proposed spatial derivatives computed via (2.35) refine at the same rate as the BDF method (2.29). In contrast, the proposed derivative based on the time-centered method fails to refine, even though the numerical solution demonstrates fifth-order accuracy in space.

The time refinement experiment uses a fixed spatial mesh consisting of $N_x = 256$ grid points and successively doubles the number of time steps from $N_t = 8$ to $N_t = 512$. In Figure 2.2, we show the results from the refinement study performed in time for the proposed methods. Based on Figure 2.2b, we can see that when fewer time steps are used in the time-centered method, the derivatives exhibit similar refinement properties of the numerical solution. As we use more time steps, the errors between the solution and its derivative grow, which suggests an issue with the accumulation of errors in time. The results obtained with the BDF method are displayed in Figure



(a) Central-2

(b) BDF-2

Figure 2.2: Time refinement study for the solution and its derivative obtained with second-order methods for the test problem in section 2.6.1. In Figure 2.2a, we plot the ℓ_∞ errors for both the numerical solution and the derivative obtained with the time-centered method. Similarly, in Figure 2.2b, we show the same quantities, which are instead computed using the BDF method. The derivative for the time-centered method initially converges together with the numerical solution, but at some point begins to diverge. In contrast, we can see that the errors for the derivatives obtained with the BDF method are aligned with those of the solution. Comparing the scales of the plots, we note that the BDF solution is slightly less accurate than the time-centered method.

2.2b and match the second-order accuracy of the base method.

The results of the space and time refinement experiments for the proposed methods suggest that we should use (2.35) to construct derivatives. While derivatives computed with the BDF method require a time history, we can think of this method as a one-step application in the sense that the derivatives at any given time level depend on the solution and not on derivatives at other time levels. Moreover, the time history in the BDF derivatives does not necessary have to come from the BDF method itself. In fact, as we will soon see, these methods work even if this time history is supplied by another solver, e.g., the time-centered method or perhaps a higher order method based on successive convolution.

2.6.2 Periodic Boundary Conditions

The next problem we consider is the two-dimensional in-homogeneous scalar wave equation

$$\frac{1}{c^2}\partial_{tt}u - \Delta u = S(x, y), \tag{2.76}$$

where $c = 1$ and

$$S(x, y) = 3e^{-t} \sin(x) \cos(y). \tag{2.77}$$

We apply two-way periodic boundary conditions on the domain $[0, 2\pi] \times [0, 2\pi]$ and use the initial data

$$u(x, y, 0) = \sin(x) \cos(y), \quad \partial_t u(x, y, 0) = -\sin(x) \cos(y). \tag{2.78}$$

The problem (2.76) is associated with the manufactured solution

$$u(x, y, t) = e^{-t} \sin(x) \cos(y), \tag{2.79}$$

and defines the source function (2.77). The partial derivatives of this solution are calculated to be

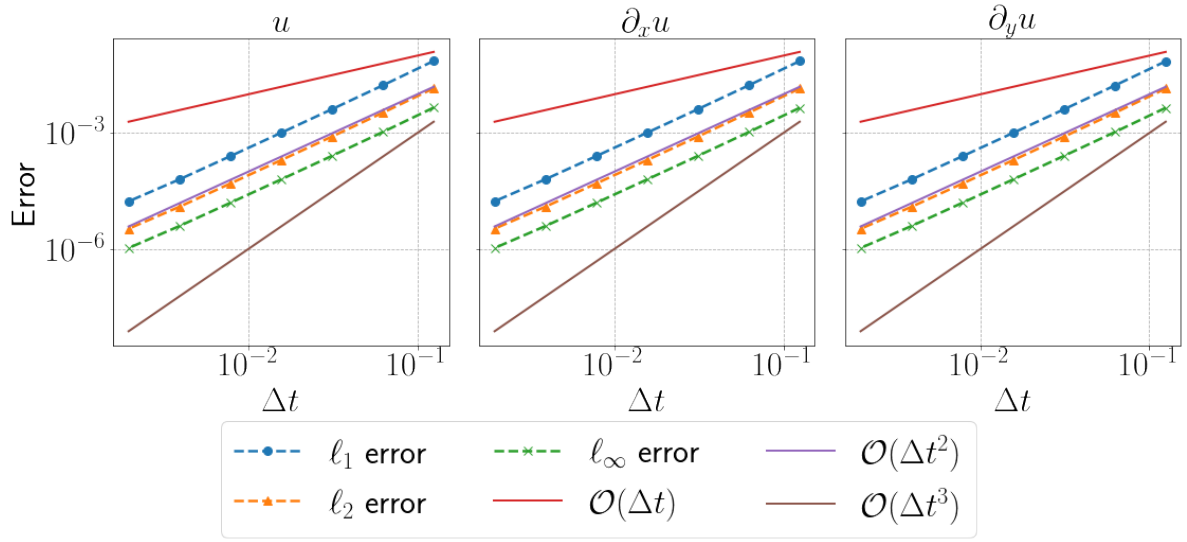
$$\partial_x u(x, y, t) = e^{-t} \cos(x) \cos(y), \tag{2.80}$$

$$\partial_y u(x, y, t) = -e^{-t} \sin(x) \sin(y). \tag{2.81}$$

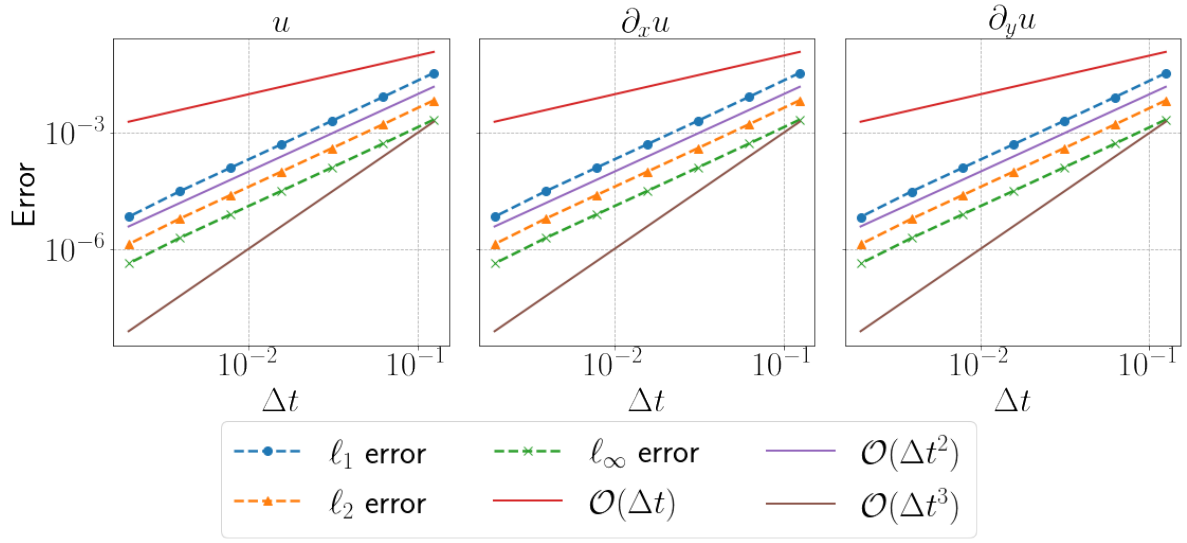
We performed temporal and spatial refinement studies using a mixed approach, as well as a pure BDF approach. The mixed approach uses the second-order time-centered method to evolve the solution u, and its partial derivatives in both variables are calculated using the second-order BDF scheme, which is denoted as "Central-2 + BDF-2" in the figures. Similarly, the pure BDF approach computes both the solution and its derivatives using the BDF scheme, i.e., "BDF-2 + BDF-2". In these experiments, we used a fifth-order spatial quadrature rule.

In the temporal refinement study, the solution is computed until a final time of $T = 1$ using a fixed 256×256 mesh in space. We successively double the number of time steps from $N_t = 8$ until $N_t = 512$. We use the analytical solution to initialize the method since it is available. The results of the temporal refinement study are presented in Figure 2.3, in which all methods, including those for the derivatives, display the expected second-order convergence rate in time.

For the space refinement experiment, we varied the spatial mesh in each direction from 16 points to 512 points. To keep the temporal error in the methods small during the refinement, we applied the methods for 1 time step using a step size of $\Delta t = 1 \times 10^{-4}$. Note that the disparity in the order between time and space (second-order versus fifth-order) necessitates a small time step



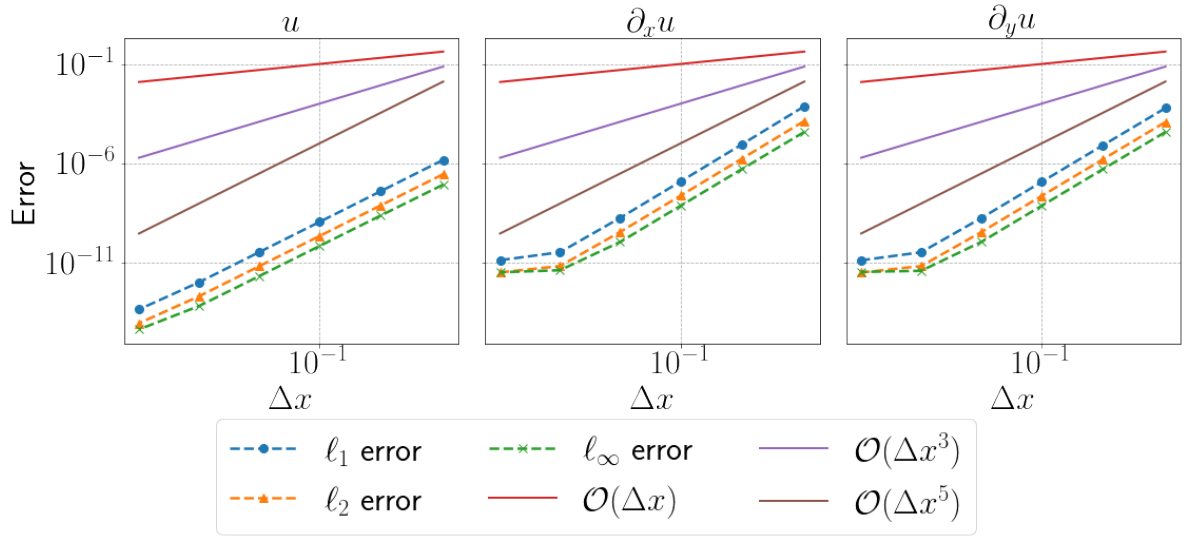
(a) Central-2 + BDF-2



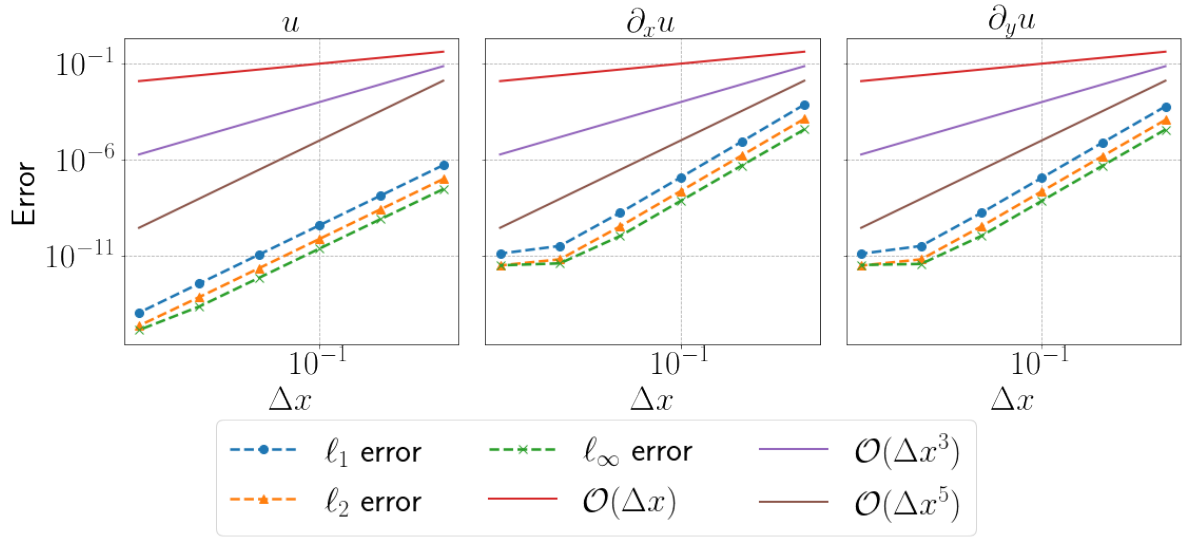
(b) BDF-2 + BDF-2

Figure 2.3: Time refinement study for the solution and its derivative for the two-dimensional periodic example 2.6.2 obtained with second-order methods. In Figure 2.3a, we plot errors for the numerical solution obtained with the central-2 method and the partial derivatives obtained with the BDF-2 method. Similarly, in Figure 2.3b, we show the same quantities, both of which are obtained using the BDF-2 method.

here; however, the waves may fail to propagate if too small a time step is used. This can be fixed using a Taylor expansion of the Green's function in the quadrature rule, as mentioned in section 2.3.3. The refinement plots in Figure 2.4 indicate fifth-order accuracy in space for all methods. We note that the derivatives in the methods begin to level-off as the error approaches 1×10^{-11}. This



(a) Central-2 + BDF-2



(b) BDF-2 + BDF-2

Figure 2.4: Space refinement of the solution and its derivative for the two-dimensional periodic example 2.6.2 obtained with second-order methods. In Figure 2.4a, we plot errors for the numerical solution obtained with the central-2 method and the partial derivatives obtained with the BDF-2 method. Similarly, in Figure 2.4b, we show the same quantities, both of which are obtained using the BDF-2 method.

is likely due to a different error coefficient in time, which arises from the differentiation process. A smaller time step would be necessary to remove this feature, but this requires some modification of the quadrature.

2.6.3 Dirichlet Boundary Conditions

For the Dirichlet problem, we, again, consider the two-dimensional in-homogeneous scalar wave equation

$$\frac{1}{c^2}\partial_{tt}u - \Delta u = S(x, y), \tag{2.82}$$

where $c = 1$ and

$$S(x, y) = 3e^{-t}\sin(x)\sin(y). \tag{2.83}$$

We apply homogeneous Dirichlet boundary conditions on the domain $[0, 2\pi] \times [0, 2\pi]$ and use the initial data

$$u(x, y, 0) = \sin(x)\sin(y), \quad \partial_t u(x, y, 0) = -\sin(x)\sin(y). \tag{2.84}$$

The problem (2.76) is associated with the manufactured solution

$$u(x, y, t) = e^{-t}\sin(x)\sin(y), \tag{2.85}$$

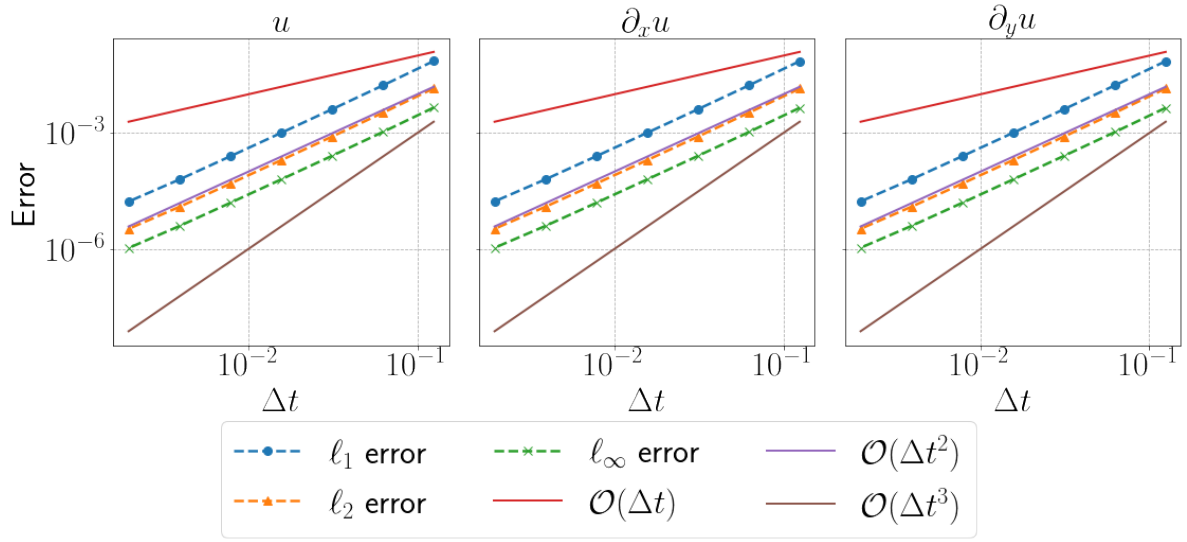
and defines the source function (2.83). The partial derivatives of this solution are calculated to be

$$\partial_x u(x, y, t) = e^{-t}\cos(x)\sin(y), \tag{2.86}$$

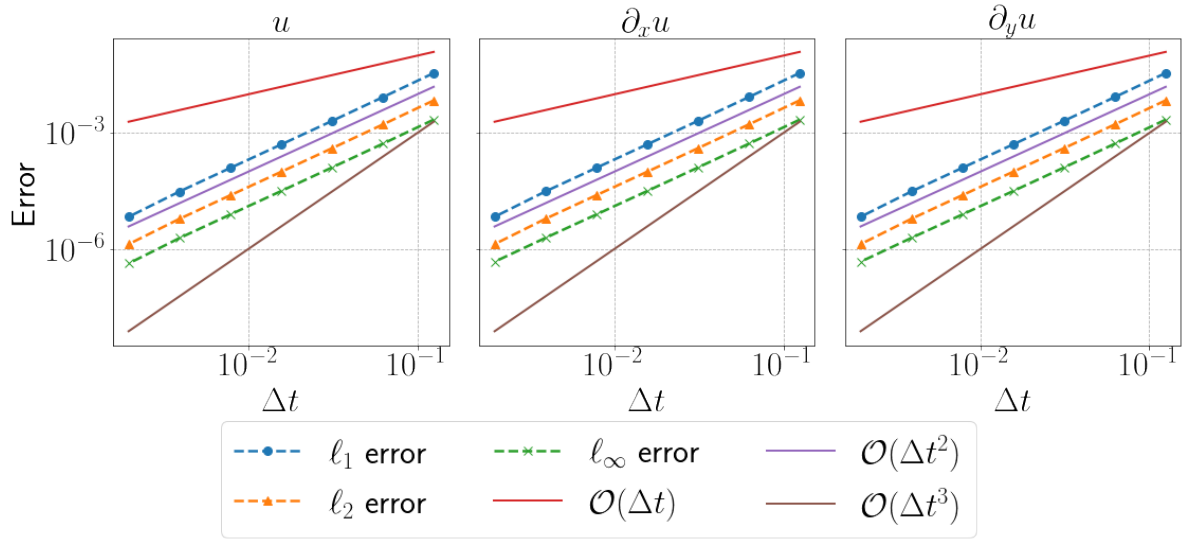
$$\partial_y u(x, y, t) = e^{-t}\sin(x)\cos(y). \tag{2.87}$$

We performed temporal and spatial refinement experiments using the same mixed approach and pure BDF approaches considered in the previous section 2.6.2. As a reminder, the mixed approach uses the second-order time-centered method to evolve the solution u, and its partial derivatives are calculated using the second-order BDF scheme, which is denoted as “Central-2 + BDF-2" in the figures. The pure BDF approach computes both the solution and its derivatives with the BDF scheme, which is similarly denoted as “BDF-2 + BDF-2" in the figures. We use the same fifth-order spatial quadrature rule as in the periodic test case to perform the runs.

In the temporal refinement study, the solution is computed until a final time of $T = 1$. We use a fixed 256×256 spatial mesh and the number of time steps in each case is successively doubled from $N_t = 8$ until $N_t = 512$. Errors can be directly measured with the analytical solution and its



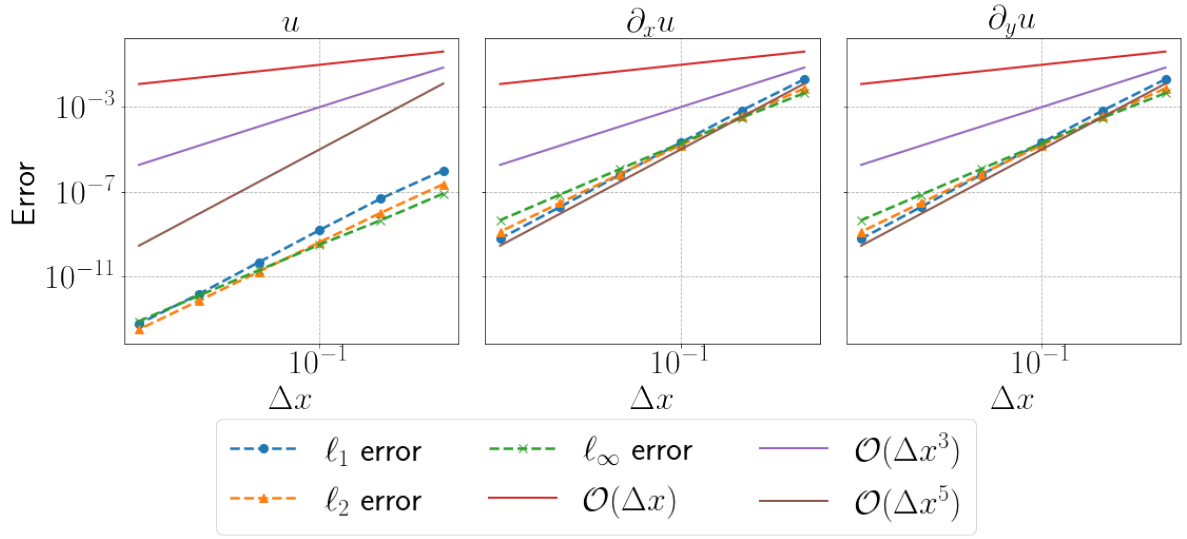
(a) Central-2 + BDF-2



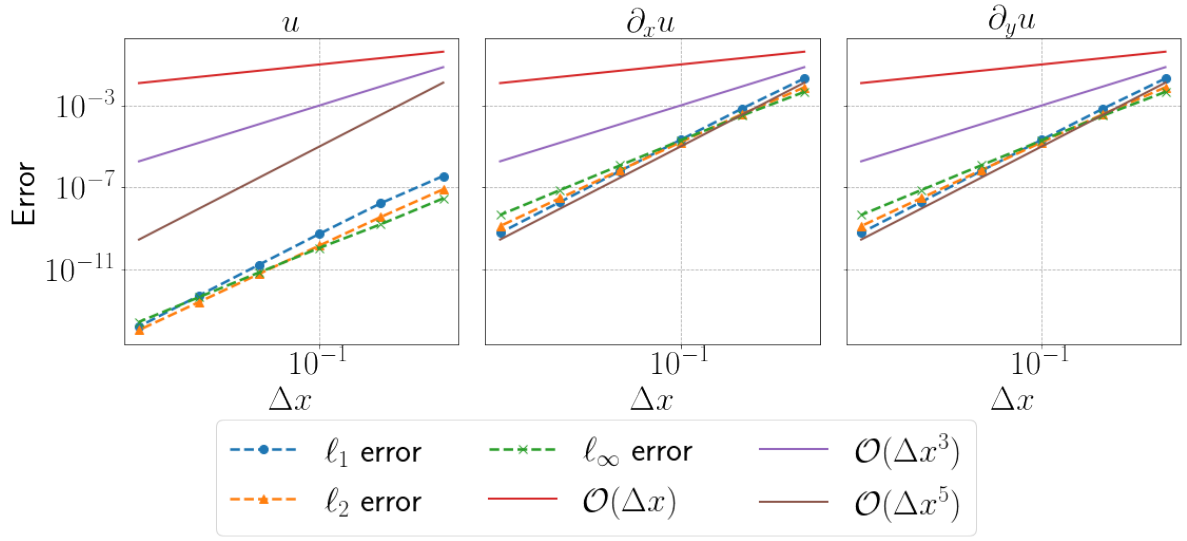
(b) BDF-2 + BDF-2

Figure 2.5: Time refinement study of the solution and its derivatives in the two-dimensional Dirichlet problem 2.6.3 obtained with second-order methods. In Figure 2.5a, we plot errors for the numerical solution obtained with the central-2 method and the partial derivatives obtained with the BDF-2 method. Similarly, in Figure 2.5b, we show the same quantities, both of which are obtained using the BDF-2 method.

derivatives. The results of the temporal refinement study are presented in Figure 2.5, in which all methods, including those for the derivatives, display the expected second-order convergence rate. The behavior is essentially identical to the results obtained for the periodic problem, which were presented in Figure 2.5



(a) Central-2 + BDF-2



(b) BDF-2 + BDF-2

Figure 2.6: Space refinement of the solution and its derivatives in the two-dimensional Dirichlet problem 2.6.3 obtained with second-order methods. In Figure 2.6a, we plot errors for the numerical solution obtained with the central-2 method and the partial derivatives obtained with the BDF-2 method. Similarly, in Figure 2.6b, we show the same quantities, both of which are obtained using the BDF-2 method.

We performed the spatial refinement study by varying the number of mesh points in each direction from 16 points to 512 points. Again, to keep the temporal error in the methods small while space is refinement, we applied the methods for only 1 time step with a step size of $\Delta t = 1 \times 10^{-4}$. The same remark about small time step sizes mentioned in the refinement experiment in space for

the periodic problem applies to this case, as well (see section 2.6.2). The refinement plots in Figure 2.6 indicate that the methods refine to fifth-order accuracy in space. In both the mixed and pure BDF approaches, the error in the derivatives behaves differently from what was observed in the periodic example. In particular, we do not observe a flattening of the error when the spacing Δx is small.

2.6.4 Outflow Boundary Conditions

To test outflow boundary conditions, we use the homogeneous scalar wave equation

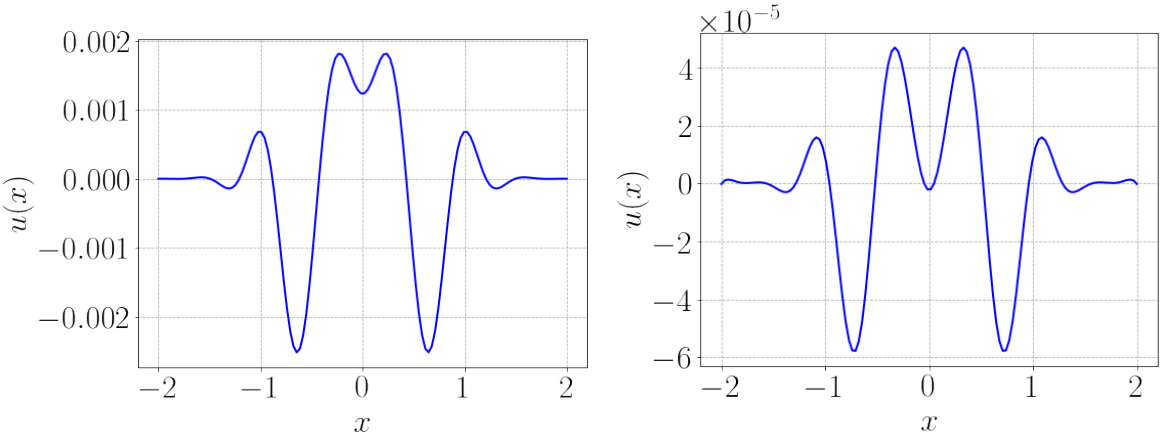
$$\frac{1}{c^2}\partial_{tt}u - \Delta u = 0, \tag{2.88}$$

where $c = 1$. We solve the problem (2.88) using the two-dimensional domain $[-2, 2] \times [-2, 2]$ and use the initial data

$$u(x, y, 0) = e^{-16(x^2+y^2)}, \quad \partial_t u(x, y, 0) = 0, \tag{2.89}$$

which is a Gaussian centered at the origin. Note that the width of the Gaussian is chosen so that the data is essentially machine zero at points along the boundary. If the initial data is not zero outside of the domain, then a specialized approach for initializing the boundary coefficients in the temporal recursion for outflow should be devised. This is not the case for this problem, so we can initialize the recursion for the boundary coefficients with zeroes.

The refinement experiments consider the same mixed and BDF approaches as in the previous sections. Since we do not have an analytical solution for the problem (2.88), we use a reference solution on a sufficiently fine temporal or spatial mesh. In all tests, the time history data is initialized using Taylor expansions in time, keeping terms up to fourth-order accuracy. Recall that we presented two forms of outflow in sections 2.4.1.4 and 2.4.2.4. We chose to use the explicit forms of outflow because they are simpler to implement in multi-dimensional problems. Moreover, using the one-dimensional analogue of problem (2.88), we found that explicit approaches were more effective at suppressing artificial reflections of the waves along the boundary. An example of this is shown in Figure 2.7, where we compared the implicit and explicit forms of outflow for



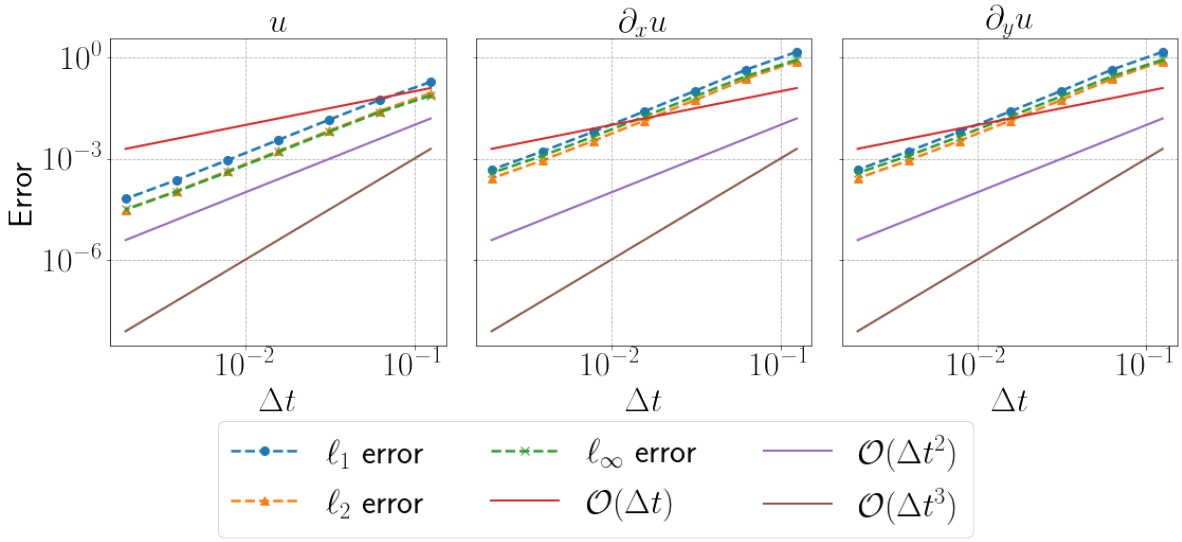
(a) BDF-2 with implicit weights (1-D)

(b) BDF-2 with explicit weights (1-D)

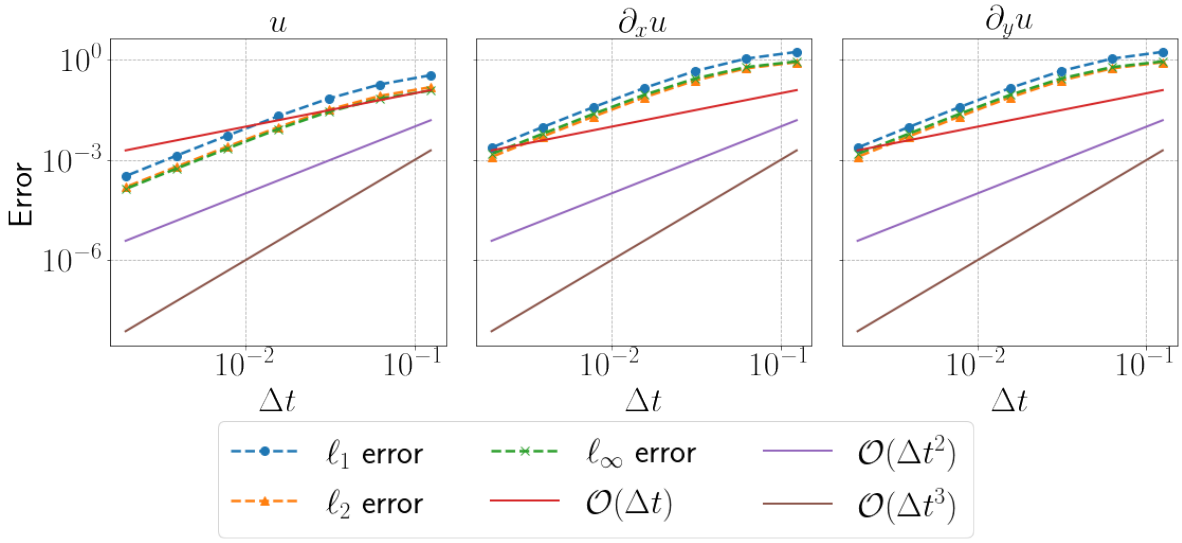
Figure 2.7: Here we show the reflection observed between the implicit and explicit forms of outflow boundary conditions for the second-order BDF method in a one-dimensional outflow problem. We run with the same Gaussian initial condition until the final time $T = 4$, at which point, the wave data should no longer be in the simulation. What is left is the reflection at the artificial boundaries of the domain. The plot shown on the left shows the results obtained with the proposed implicit form of the outflow weights developed for the BDF-2 method, while the plot on the right uses the explicit form of the weights. We find that the explicit form of the weights is more effective at suppressing the spurious reflections at the artificial boundaries.

the BDF method. At the time shown in the plots, what remains of the original wave is due to reflections.

In the refinement study for time, we evolve the solution on a fixed 512×512 mesh until the final time $T = 1.0$, which is before the wave leaves the domain. The number of time steps are successively doubled from $N_t = 8$ until $N_t = 512$. We note that the coarsest time discretization gives a CFL ≈ 16, which is much larger than we would typically use for the second-order schemes. The reference solution is obtained with the same spatial mesh with a total of $N_t = 2048$ time steps. Results obtained with the mixed and BDF approaches are presented in Figures 2.8a, respectively. The methods for the solution and the corresponding derivatives refine to second-order accuracy in time, with the mixed approach being the more accurate of the proposed methods. The overall error in these methods is notably larger than for the periodic and Dirichlet test problems. One reason for this is that periodic and Dirichlet conditions can be enforced exactly, while outflow conditions are only approximately enforced. This error is further amplified in the case of the derivatives



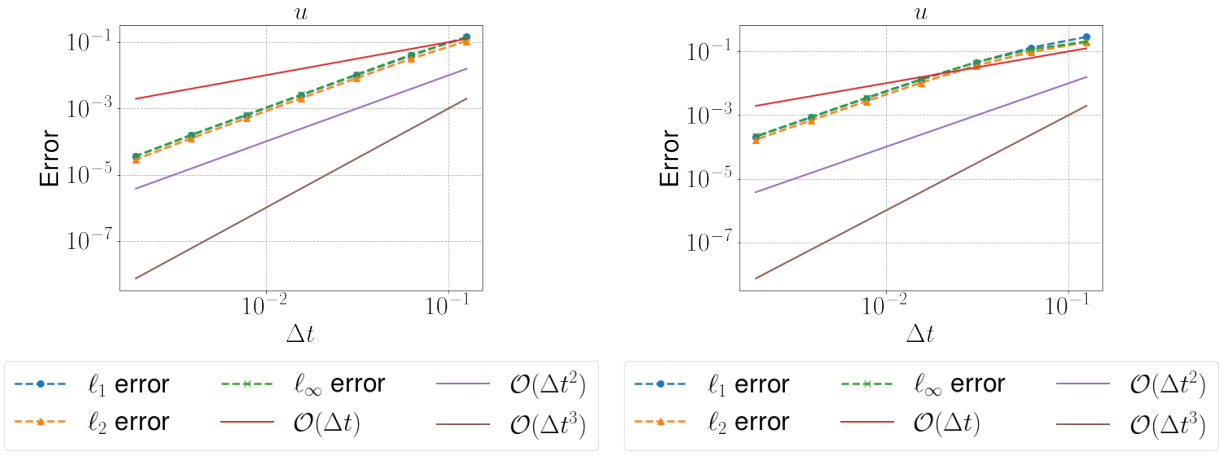
(a) Central-2 + BDF-2 (2-D)



(b) BDF-2 + BDF-2 (2-D)

Figure 2.8: Time refinement study of the solution and its derivatives in the two-dimensional outflow problem of section 2.6.4 obtained with second-order methods. In Figure 2.8a, we plot errors for the numerical solution obtained with the central-2 method and the partial derivatives obtained with the BDF-2 method. Similarly, in Figure 2.8b, we show the same quantities, both of which are obtained using the BDF-2 method.

which introduce an additional factor that is $O(1/\Delta t)$ which amplifies the overall size of the error. Similar error properties can be observed in the analogous one-dimensional problem. Using the same problem setup for the one-dimensional case, we applied the implicit forms of outflow for the time-centered and BDF methods, which we show in Figure 2.9. While the outflow methods



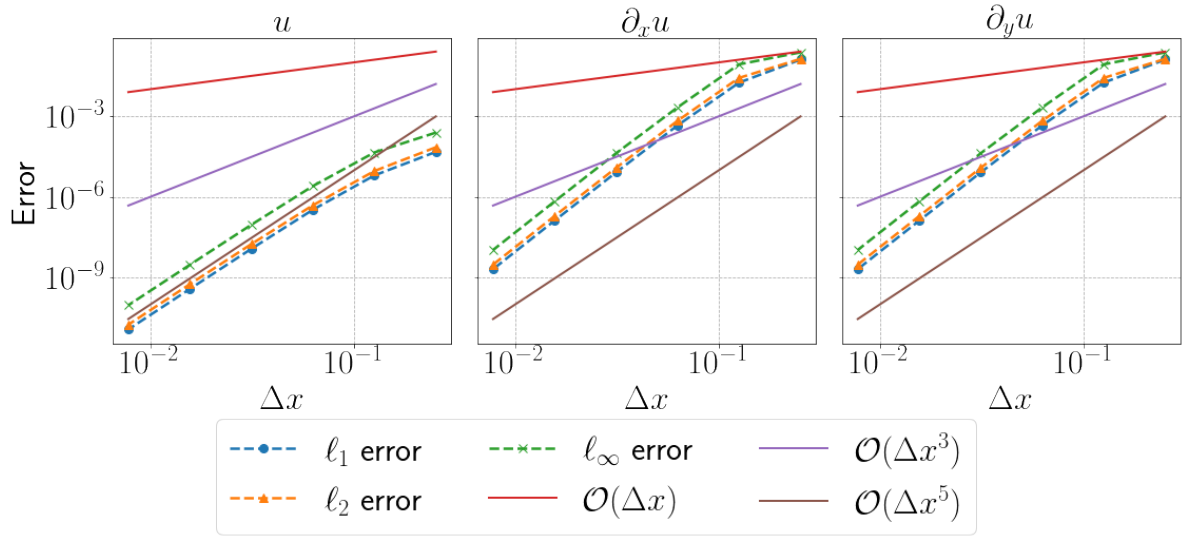
(a) Central-2 with implicit weights (1-D)

(b) BDF-2 with implicit weights (1-D)

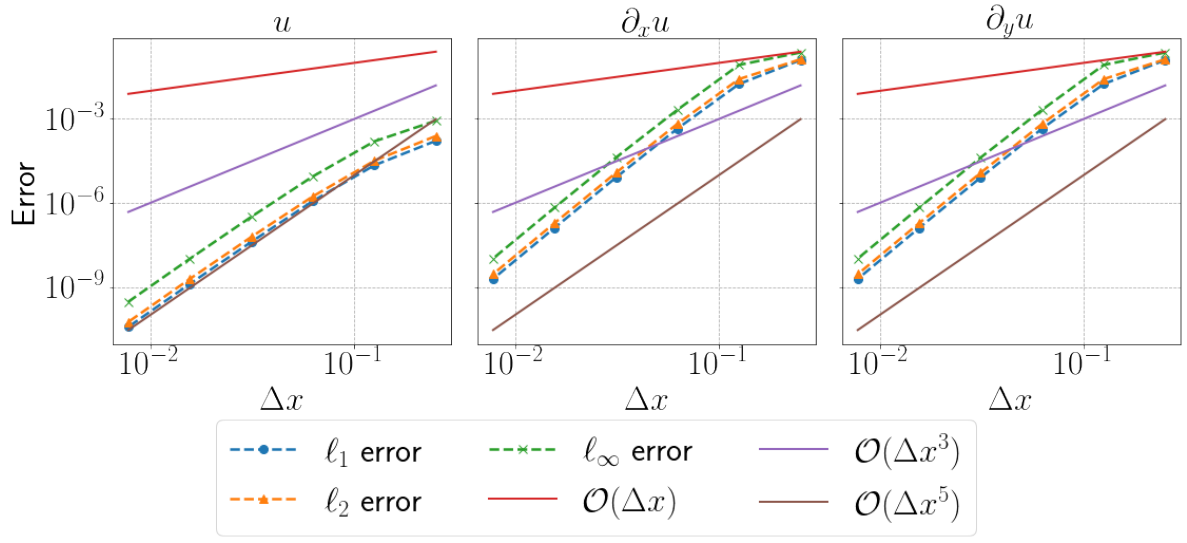
Figure 2.9: A comparison of the temporal refinement properties for the one-dimensional implicit methods. The weights for the time-centered method shown in Figure2.9a are taken from the paper [34]. We compare this to the proposed implicit approach to outflow, shown on the right, in Figure 2.9b.

themselves are different, we can see that similar refinement properties are observed here as well. This comparison informs us that the issue is likely not related to dimensionality or the form of the outflow weights (e.g., implicit or explicit). This issue seems to be more related to the propagation of errors in the methods when only approximate boundary data is available.

The spatial refinement study was performed by varying the number of mesh points in each direction from 17 points to 513 points. The methods were applied for only 1 time step with step size of $\Delta t = 1 \times 10^{-4}$ and the errors were measured against a reference solution computed on a 2049×2049 spatial mesh using $\Delta t = 1 \times 10^{-4}$, here, as well. The refinement plots in Figure 2.6 show that each of these methods is approximately first-order in space. As alluded to in the time refinement, this seems to be a consequence of using inexact boundary conditions. The particular form of the inverse operator (2.19), suggests that errors in A and B, which are used to enforce boundary conditions, can impact regions in the vicinity of the boundary. The particular size of these regions depends on the size of α, which, in turn, depends on the wave speed c and value of Δt. As with the time refinement, we can look at the analogous one-dimensional problem and compare these results with the two-dimensional problem. The one-dimensional refinement results, obtained



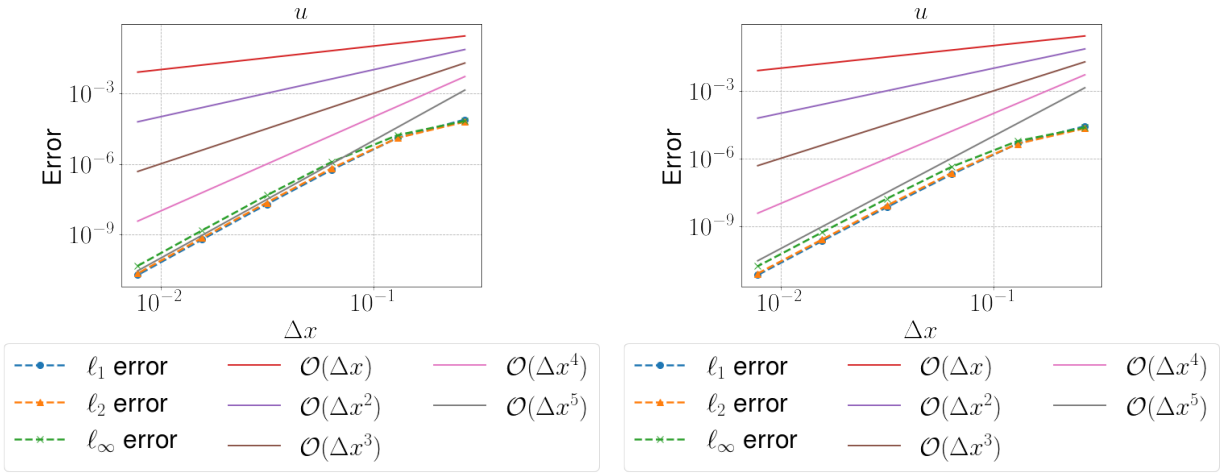
(a) Central-2 + BDF-2 (2-D)



(b) BDF-2 + BDF-2 (2-D)

Figure 2.10: Space refinement of the solution and its derivatives in the two-dimensional outflow problem of section 2.6.4 obtained with second-order methods. In Figure 2.10a, we plot errors for the numerical solution obtained with the central-2 method and the partial derivatives obtained with the BDF-2 method. Similarly, in Figure 2.10b, we show the same quantities, both of which are obtained using the BDF-2 method.

with the implicit form of the outflow weights, are presented in Figure 2.11. Again, we observe similar behavior in the refinement properties of the one- and two-dimensional test problems, which, again, hints at issues concerning the propagation of errors in the method.



(a) Central-2 with implicit weights (1-D)

(b) BDF-2 with implicit weights (1-D)

Figure 2.11: A comparison of the spatial refinement properties for the one-dimensional implicit methods. The weights for the time-centered method, shown in Figure2.11a, are taken from the paper [34]. We compare this to the proposed implicit approach to outflow, shown on the right, in Figure 2.11b.

2.7 Conclusion

In this work, we developed new approaches for computing fields and their derivatives with applications to scalar wave equations. Our contributions build on prior developments for a class of algorithms known as the MOL^T, which combines a dimensional splitting technique with a one-dimensional integral equation method to yield algorithms with unconditional stability, geometric flexibility, and have $O(N)$ complexity. The proposed methods for derivatives use data which is already available through the base method, so they naturally inherit the properties offered in the base method. We also presented a treatment of outflow boundary conditions for the BDF method. The accuracy of the proposed methods was evaluated by performing a series of refinement experiments in both space and time, using several types of boundary conditions. In particular, we established some refinement properties for outflow boundary conditions, which were not presented in earlier work.

CHAPTER 3

PARALLEL ALGORITHMS FOR SUCCESSIVE CONVOLUTION

3.1 Introduction

In this chapter, we develop parallel algorithms using novel approaches to represent derivative operators for linear and nonlinear time-dependent partial differential equations (PDEs). We chose to investigate algorithms for these representations due to the stability properties observed for a wide range of linear and nonlinear PDEs. The approach considered here uses expansions involving integral operators to approximate spatial derivatives. Here, we shall refer to this approach as the Method of Lines Transpose (MOL^T) though this can be more broadly categorized within a larger class of successive convolution methods. The name arises because the terms in the operator expansions, which we describe later, involve convolution integrals whose operand is recursively or successively defined. Despite the use of explicit data in these integral terms, the boundary data remains implicit, which contributes to both the speed and stability of the representations. The inclusion of more terms in these operator expansions, when combined with a high-order quadrature method, allow one to obtain a high-order discretization in both space and time. Another benefit of this approach is that extensions to multiple spatial dimensions are straightforward as operators can be treated in a line-by-line fashion. Moreover, the integral equations are amenable to fast-summation techniques, which reduce the overall computational complexity, along a given dimension, from $O(N^2)$ to $O(N)$, where N is the number of discrete grid points along a dimension.

High-order successive convolution algorithms have been developed to solve a range of time-dependent PDEs, including the wave equation [50], heat equation (e.g., Allen-Cahn [55] and Cahn-Hilliard equations [43]), Maxwell's equations [37], Vlasov equation [52], degenerate advection-diffusion (A-D) equation [44], and the Hamilton-Jacobi (H-J) equation [56, 45]. In contrast to these papers, this work focuses on the performance of the method in parallel computing environments, which is a largely unexplored area of research. Specifically, our work focuses on developing

effective domain decomposition strategies for distributed memory systems and building thread-scalable algorithms using the low-order schemes as a baseline. By leveraging the decay properties of the integral representation, we restrict the calculations to localized non-overlapping subsets of the spatial domain. The algorithms presented in this work consider dependencies between nearest-neighbors (N-N), but, as we will see, this restriction can be generalized to include additional information, at the cost of additional communication. Using a hybrid design that employs MPI and Kokkos [46] for the distributed and shared memory components of the algorithms, respectively, we show that our methods are efficient and can sustain an update rate $> 1 \times 10^8$ DOF/node/s. While experimentation on graphics processing units (GPUs) shall be left to future work, we believe choosing Kokkos will provide a path for a more performant and seamless integration of our algorithms with new computing hardware.

Recent developments in successive convolution methods have focused on extensions to solve more general nonlinear PDEs, for which an integral solution is generally not applicable. This work considers discretizations developed for degenerate advection-diffusion (A-D) equations [44], as well as the Hamilton-Jacobi (H-J) equations [56, 45]. The key idea of these papers exploited the linearity of a given *differential operator* rather than the underlying equations, allowing derivatives in nonlinear problems to be expressed using the same representations developed for linear problems. For linear problems, it was demonstrated that one could couple these representations for the derivative operators with an explicit time-stepping method, such as the strong-stability-preserving Runge-Kutta (SSP-RK) methods, [57] and still obtain schemes which maintain unconditional stability [56, 44]. To address shock-capturing and control non-physical oscillations, the latter two papers introduced quadratures that use WENO reconstructions, along with a nonlinear filter to further control oscillations. In [45], the schemes for the H-J equations were extended to enable calculations on mapped grids. This paper also proposed a new WENO quadrature method that uses a basis that consists of exponential polynomials that improves the shock capturing capabilities.

Our choice in discretizing time, first, before treating the spatial quantities, is not a new idea. A well-known approach is Rothe's method [58, 59] in which a finite difference approximation is

used for time derivatives and an integral equation solver is developed for the resulting sequence of elliptic PDEs (see e.g., [60, 61, 62, 49, 63, 64, 65]). The earlier developments for successive convolution methods, such as [50], are quite similar to Rothe's method in the treatment of the time derivatives. However, successive convolution methods differ from Rothe's method considerably in the treatment of spatial derivatives for nonlinear problems, such as those considered in more recent work on successive convolution (see e.g., [56, 44, 45]), as Newton iteration can be avoided on nonlinear terms. Additionally, these methods do not require solutions to linear systems. In contrast, Nyström methods, which are used to discretize the integral equations in Rothe's method, result in dense linear systems, which are typically solved using an iterative method such as GMRES [27]. Despite the fact that the linear systems are well-conditioned, the various collective operations that occur in distributed GMRES solves can become quite expensive on large computing platforms.

Similarly, in [66, 67], Bruno and Lyon introduced a spectral method, based on the FFT, for computing spatial derivatives of general, possibly non-periodic, functions known as Fourier-Continuation (FC). They combined this representation with the well-known Alternating-Direction (AD) methods, e.g., [68, 69, 70], dubbed FC-AD, to develop implicit solvers suitable for linear equations. This resulted in a method capable of computing spatial derivatives in a rapidly convergent and dispersionless fashion. A domain decomposition technique for the FC method is described in [71] and weak scaling was demonstrated to 256 processors, using 4 processors per node, but larger runs were not considered. Another related transform approach was developed to solve the linear wave equation [72]. This work introduced a windowed Fourier methodology and combined this with a frequency-domain boundary integral equation solver to simulate long-time, high-frequency wave propagation. While this particular work does not focus on parallel implementations, they suggest several generic strategies, including a trivially parallelizable approach that solves a collection of frequency-domain integral equations in parallel; however, a purely parallel-in-time approach may not be appropriate for massively parallel systems, especially if few frequencies are required across time. This issue may be further complicated by the parallel implementation of the frequency-domain integral equation solvers, which, as previously mentioned, require the solution of dense

linear systems. Therefore, it may be a rather difficult task to develop robust parallel algorithms, which are capable of achieving their peak performance.

This paper is organized as follows: In 3.2, we provide an overview of the numerical scheme used to formulate our algorithms. To this end, we first illustrate the connections among several characteristically different PDEs through the appearance of common operators in 3.2.1. Using these fundamental operators, we define the integral representation in 3.2.2, which is used to approximate spatial derivatives. Once the representations have been introduced, we briefly discuss the complications associated with boundary conditions and provide the relevant information, in 3.2.3, for implementing boundary conditions used in our numerical tests. Sections 3.2.4 and 3.2.5 briefly review the spatial discretization process (including the fast-summation method and the quadrature) and the coupling with time integration methods, respectively. We provide the details of our new domain decomposition algorithm in 3.3, beginning with the derivation of the so-called N-N conditions in 3.3.1. Using these conditions, we show how this can be used to enforce boundary conditions, locally, for first and second derivative operators (3.3.2 and 3.3.3, respectively). Details concerning the implementation of the parallel algorithms are contained entirely in 3.4. This includes the introduction of the shared memory programming model (3.4.1), the definition of a certain performance metric (3.4.2) used in both loop optimization experiments and scaling studies (3.4.3), the presentation of shared memory algorithms (3.4.4), and, lastly, implementation details concerning the distributed memory algorithms (3.4.5). 3.5 contains the core numerical results which confirm the convergence (3.5.1), as well as, the weak and strong scalability (3.5.2 and 3.5.3, respectively) of the proposed algorithms. In 3.5.4, we examine the impact of the restriction posed by the N-N conditions. Finally, we summarize our findings with a brief conclusion in 3.6.

3.2 Description of Numerical Methods

In this section, we outline the approach used to develop unconditionally stable solvers making use of knowledge for linear operators. We will start by demonstrating the connections between several different PDEs using operator notation, which will allow us to reuse, or combine, approximations

in several different ways. Once we have established these connections, we define an appropriate "inverse" operator and use this to develop the expansions used to represent derivatives. The representations we develop for derivative operators are motivated by the solution of simple 1-D problems. However, in multi-dimensional problems, these expressions are still valid approximations, in a certain sense, even though the kernels in the integral representation may not be solutions to the PDE in question. While these approximations can be made high-order in both time and space, the focus of this work is strictly on the scalability of the method, so we will limit ourselves to formulations which are first-order in time. Note that the approach described in 3.4, which considers first-order schemes, is quite general and can be easily extended for high-order representations. Once we have discussed our treatment of derivative terms, we describe the fast summation algorithm and quadrature method in 3.2.4. Despite the fact that this work only considers smooth test problems, we include the relevant modifications required for non-smooth problems for completeness. In 3.2.5, we illustrate how the representation of derivative operators can be used within a time stepping method to solve PDEs.

3.2.1 Connections Among Different PDEs

Before introducing the operators relevant to successive convolution algorithms, we establish the operator connections appearing in several linear PDE examples. This process helps identify key operators that can be represented with successive convolution. Specifically, we shall consider the following three prototypical linear PDEs:

- Linear advection equation: $(\partial_t - c\partial_x)u = 0$,
- Diffusion equation: $(\partial_t - \nu\partial_{xx})u = 0$,
- Wave equation: $(\partial_{tt} - c^2\partial_{xx})u = 0$.

Next, we apply an *implicit* time discretization to each of these problems. For discussion purposes, we shall consider lower-order time discretizations, i.e., backward-Euler for the $\partial_t u$ and a second-

order central difference for $\partial_{tt}u$. If we identify the current time as t^n, the new time level as t^{n+1}, and $\Delta t = t^{n+1} - t^n$, then we obtain the corresponding set of semi-discrete equations:

- Linear advection equation: $(\mathcal{I} - \Delta t c\partial_x)u^{n+1} = u^n$,
- Diffusion equation: $(\mathcal{I} - \Delta t \nu\partial_{xx})u^{n+1} = u^n$,
- Wave equation: $(\mathcal{I} - \Delta t^2 c^2\partial_{xx})u^{n+1} = 2u^n - u^{n-1}$.

Here, we use $\mathcal{I}$ to denote the identity operator, and, in all cases, each of the spatial derivatives are taken at time level t^{n+1} to keep the schemes implicit. The key observation is that the operator $(\mathcal{I} \pm \frac{1}{\alpha}\partial_x)$ arises in each of these examples. Notice that

$$(\mathcal{I} - \frac{1}{\alpha^2}\partial_{xx}) = (\mathcal{I} - \frac{1}{\alpha}\partial_x)(\mathcal{I} + \frac{1}{\alpha}\partial_x),$$

where α is a parameter that is selected according to the equation one wishes to solve. For example, in the case of diffusion, one selects

$$\alpha = \frac{\beta}{\sqrt{\nu\Delta t}},$$

while for the linear advection and wave equations, one selects

$$\alpha = \frac{\beta}{c\Delta t}.$$

The parameter β, which does not depend on Δt, is then used to tune the stability of the approximations. For test problems appearing in this paper, we always use $\beta = 1$. In 3.2.2, we demonstrate how the operator $(\mathcal{I} \pm \frac{1}{\alpha}\partial_x)$ can be used to approximate spatial derivatives. We remark that for second derivatives, one can also use $(\mathcal{I} - \frac{1}{\alpha^2}\partial_{xx})$ to obtain a representation for second order spatial derivatives, instead of factoring into “left" and “right" characteristics.

Next, we introduce the following definitions to simplify the notation:

$$\mathcal{L}_L \equiv \mathcal{I} - \frac{1}{\alpha}\partial_x, \quad \mathcal{L}_R \equiv \mathcal{I} + \frac{1}{\alpha}\partial_x. \tag{3.1}$$

Written in this manner, these definitions indicate the left and right-moving components of the characteristics, respectively, as the subscripts are associated with the direction of propagation. For

second derivative operators, which are not factored into first derivatives, we shall use

$$\mathcal{L}_0 \equiv \mathcal{I} - \frac{1}{\alpha^2}\partial_{xx}. \tag{3.2}$$

In order to connect these operators with suitable expressions for spatial derivatives, we need to define the corresponding "inverse" for each of these *linear* operators on a 1-D interval $[a, b]$. These definitions are given as

$$\begin{aligned}
\mathcal{L}_L^{-1}[\,\cdot\,;\alpha](x) &\equiv \alpha \int_x^b e^{-\alpha(s-x)}(\cdot)\, ds + Be^{-\alpha(b-x)}, &(3.3)\\
&\equiv I_L[\,\cdot\,;\alpha](x) + Be^{-\alpha(b-x)},\\
\mathcal{L}_R^{-1}[\,\cdot\,;\alpha](x) &\equiv \alpha \int_a^x e^{-\alpha(x-s)}(\cdot)\, ds + Ae^{-\alpha(x-a)}, &(3.4)\\
&\equiv I_R[\,\cdot\,;\alpha](x) + Ae^{-\alpha(x-a)}.
\end{aligned}$$

These definitions can be derived in a number of ways. In A.1, we demonstrate how these definitions can be derived for the linear advection equation using the integrating factor method. In these definitions, A and B are constants associated with the "homogeneous solution" of a corresponding semi-discrete problem and are used to satisfy the boundary conditions. In a similar way, one can compute the inverse operator for definition (3.2), which yields

$$\begin{aligned}
\mathcal{L}_0^{-1}[\,\cdot\,;\alpha](x) &\equiv \frac{\alpha}{2} \int_a^b e^{-\alpha|x-s|}(\cdot)\, ds + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}, &(3.5)\\
&\equiv I_0[\,\cdot\,;\alpha](x) + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}.
\end{aligned}$$

In these definitions, we refer to "$\cdot$" as the operand and, again, α is a parameter selected according to the problem being solved. Although it is a slight abuse of notation, when it is not necessary to explicitly indicate the parameter or the point of evaluation, we shall place the operand inside a pair of parenthesis.

If we connect these definitions to each of the linear semi-discrete equations mentioned earlier, we can determine the update equation through an *analytic inversion* of the corresponding linear operator(s):

- Linear advection equation: $u^{n+1} = \mathcal{L}_R^{-1}(u^n)$, or $u^{n+1} = \mathcal{L}_L^{-1}(u^n)$,

- Diffusion equation: $u^{n+1} = \mathcal{L}_L^{-1}(\mathcal{L}_R^{-1}(u^n))$, or $u^{n+1} = \mathcal{L}_0^{-1}(u^n)$,
- Wave equation: $u^{n+1} = \mathcal{L}_L^{-1}(\mathcal{L}_R^{-1}(2u^n - u^{n-1}))$, or $u^{n+1} = \mathcal{L}_0^{-1}(2u^n - u^{n-1})$,

with the appropriate choice of α for the problem being considered. We note that each of these methods can be made high-order following the work in [56, 44, 45, 55, 38], where it was demonstrated that these approaches lead to methods that are unconditionally stable to all orders of accuracy for these linear PDEs, even with variable wave speeds or diffusion coefficients. Since the process of analytic inversion yields an integral term, a fast-summation technique should be used to reduce the computational complexity of a naive implementation, which would otherwise scale as $O(N^2)$. Some details concerning the spatial discretization and the $O(N)$ fast-summation method are briefly summarized in 3.2.4 (for full details, please see [55]). Next we demonstrate how the operator $\mathcal{L}_*$ can be used to approximate spatial derivatives.

3.2.2 Representation of Derivatives

In the previous section, we observed that characteristically different PDEs can be described in-terms of a common set of operators. The focus of this section shall be on manipulating these approximations to obtain a high-order discretization in time through certain operator expansions. The process begins by introducing an operator related to $\mathcal{L}_*^{-1}$, namely,

$$\mathcal{D}_* \equiv \mathcal{I} - \mathcal{L}_*^{-1}, \tag{3.6}$$

where $*$ can be L, R, or 0. The motivation for these definitions will become clear soon. Additionally, we can derive an identity from the definitions (3.6). By manipulating the terms we quickly find that

$$\mathcal{L}_* \equiv (\mathcal{I} - \mathcal{D}_*)^{-1}, \tag{3.7}$$

again, where $*$ can be L, R, or 0. The purpose of the identity (3.7) is that it connects the spatial derivative to an expression involving integrals of the solution rather than derivatives. In other words, it allows us to avoid having to use a stencil operation for derivatives.

To obtain an approximation for the first derivative in space, we can use $\mathcal{L}_L$, $\mathcal{L}_R$, or both of them, which may occur as part of a monotone splitting. If we combine the definition of the left propagating first derivative operator in equation (3.1) with the definition (3.6) and identity (3.7), we can define the first derivative in terms of the $\mathcal{D}_L$ operator. Observe that

$$\begin{aligned}
\partial_x^+ &= \alpha \left(\mathcal{I} - \mathcal{L}_L \right), \\
&= \alpha \left(\mathcal{L}_L \mathcal{L}_L^{-1} - \mathcal{L}_L \right), \\
&= \alpha \mathcal{L}_L \left(\mathcal{L}_L^{-1} - \mathcal{I} \right), \\
&= -\alpha \mathcal{L}_L \left(\mathcal{I} - \mathcal{L}_L^{-1} \right), \\
&= -\alpha \left(\mathcal{I} - \mathcal{D}_L \right)^{-1} \mathcal{D}_L, \\
&= -\alpha \sum_{p=1}^{\infty} \mathcal{D}_L^p,
\end{aligned} \tag{3.8}$$

where, in the last step, we used the fact that the operator $\mathcal{D}_L$ is bounded by unity in an operator norm. We use the + convention to indicate that this is a right-sided derivative. Likewise, for the right propagating first derivative operator, we find that the complementary left-biased derivative is given by

$$\partial_x^- = \alpha \sum_{p=1}^{\infty} \mathcal{D}_R^p. \tag{3.9}$$

Additionally, the second derivative can be expressed as

$$\partial_{xx} = -\alpha^2 \sum_{p=1}^{\infty} \mathcal{D}_0^p. \tag{3.10}$$

As the name implies, each power of $\mathcal{D}_*$ is *successively* defined according to

$$\mathcal{D}_*^k \equiv \mathcal{D}_* \left(\mathcal{D}_*^{k-1} \right). \tag{3.11}$$

In previous work, [44], for periodic boundary conditions, it was established that the partial sums for the left and right-biased approximations to ∂_x satisfy

$$\partial_x^+ = -\alpha \left(\sum_{p=1}^{n} \mathcal{D}_L^p + \mathcal{O}\left(\frac{1}{\alpha^{n+1}} \right) \right), \quad \partial_x^- = \alpha \left(\sum_{p=1}^{n} \mathcal{D}_R^p + \mathcal{O}\left(\frac{1}{\alpha^{n+1}} \right) \right). \tag{3.12}$$

Similarly for second derivatives, with periodic boundaries, retaining n terms leads to a truncation error with the form

$$\partial_{xx} = -\alpha^2 \left(\sum_{p=1}^{n} \mathcal{D}_0^p + O\left(\frac{1}{\alpha^{2n+2}} \right) \right). \tag{3.13}$$

In both cases, the relations can be obtained through a repeated application of integration by parts with induction. These approximations are still exact, in space, but the integral operators nested in $\mathcal{D}_*$ will eventually be approximated with quadrature. From these relations, we can also observe the impact of α on the size of the error in time. In particular, if we select $\alpha = O\left(1/\Delta t\right)$ in (3.12) and $\alpha = O\left(1/\sqrt{\Delta t}\right)$ in (3.13), each of the approximations should have an error of the form $O(\Delta t^n)$. The results concerning the consistency and stability of these higher order approximations were established in [56, 44, 38].

As mentioned earlier, this work only considers approximations which are first-order with respect to time. Therefore, we shall restrict ourselves to the following operator representations:

$$\partial_x^+ \approx -\alpha \mathcal{D}_L, \quad \partial_x^- \approx \alpha \mathcal{D}_R, \quad \partial_{xx} \approx -\alpha^2 \mathcal{D}_0. \tag{3.14}$$

This is a consequence of retaining a single term from each of the partial sums in equations (3.8), (3.9), and (3.10). Consequently, computing higher powers of $\mathcal{D}_*$ is unnecessary, so the successive property (3.11) is not needed. However, it indicates, clearly, a possible path for higher-order extensions of the ideas which will be presented here. For the moment, we shall delay prescribing the choice of α used in the representations (3.14), in order to avoid a problem-dependent selection. As alluded to at the beginning of this section, an identical representation is used for multi-dimensional problems, where the $\mathcal{D}_*$ operators are now associated with a particular dimension of the problem. The operators along a particular dimension are constructed using data along that dimension of the domain, so that each of the directions remains uncoupled. This completes the discussion on the generic form of the representations used for spatial derivatives. Next, we provide some information regarding the treatment of boundary conditions, which determine the constants A and B appearing in the $\mathcal{D}_*$ operators.

3.2.3 Comment on Boundary Conditions

The process of prescribing values of A and B, inside equations (3.3),(3.4), and (3.5), which are required to construct $\mathcal{D}_*$, is highly dependent on the structure of the problem being solved. Previous work has shown how to prescribe a variety of boundary conditions for linear PDEs (see e.g., [50, 55, 43, 38]). For example, in linear problems, such as the wave equation, with either periodic or non-periodic boundary conditions, one can directly enforce the boundary conditions to determine the constants A and B. The situation can become much more complicated for problems which are both nonlinear and non-periodic. For approximations of at most third-order accuracy, with nonlinear PDEs, one can use the techniques from [56] for non-periodic problems. To achieve high-order time accuracy, the partial sums for this case were modified to eliminate certain low-order terms along the boundaries. We note that the development of high-order time discretizations, subject to non-trivial boundary conditions, for nonlinear operators, is still an open area of research for successive convolution methods.

As this paper concerns the scalability of the method, we shall consider test problems that involve periodic boundary conditions. For periodic problems defined on the line interval $[a, b]$, the constants associated with the boundary conditions for first derivatives are given by

$$A = \frac{I_R[v;\alpha](b)}{1-\mu}, \quad B = \frac{I_L[v;\alpha](a)}{1-\mu}, \tag{3.15}$$

where I_L and I_R were defined in equations (3.3) and (3.4). Similarly, for second derivatives, the constants can be determined to be

$$A = \frac{I_0[v;\alpha](b)}{1-\mu}, \quad B = \frac{I_0[v;\alpha](a)}{1-\mu}, \tag{3.16}$$

with the definition of I_0 coming from (3.5). In the expressions (3.15) and (3.16) provided above, we use

$$\mu \equiv e^{-\alpha(b-a)},$$

where α is the appropriately chosen parameter. Note that the function $v(x)$ denotes the generic operand of the operator $\mathcal{D}_*$. This helps reduce the complexity of the notation when several

applications of $\mathcal{D}_*$ are required, since they are recursively defined. As an example, suppose we wish to compute $\partial_{xx} h(u)$, where h is some known function. For this, we can use the first-order scheme for second derivatives (see (3.14)) and take $v = h(u)$ in the expressions (3.16) for the boundary terms.

3.2.4 Fast Convolution Algorithm and Spatial Discretization

To perform a spatial discretization over $[a, b]$, we first create a grid of $N + 1$ points:

$$x_i = a + i\Delta x_i, \quad i = 0, \cdots, N,$$

where

$$\Delta x_i = x_{i+1} - x_i.$$

A naive approach to computing the convolution integral would lead to method of complexity $O(N^2)$, where N is the number of grid points. However, using some algebra, we can write recurrence relations for the integral terms which comprise $\mathcal{D}_*$:

$$I_R[v;\alpha](x_i) = e^{-\alpha\Delta x_{i-1}} I_R[v;\alpha](x_{i-1}) + J_R[v;\alpha](x_i), \quad I_R[v;\alpha](x_0) = 0,$$
$$I_L[v;\alpha](x_i) = e^{-\alpha\Delta x_i} I_L[v;\alpha](x_{i+1}) + J_L[v;\alpha](x_i), \quad I_L[v;\alpha](x_N) = 0.$$

Here, we have defined the local integrals

$$J_R[v;\alpha](x_i) = \alpha \int_{x_{i-1}}^{x_i} e^{-\alpha(x_i - s)} v(s)\, ds, \tag{3.17}$$
$$J_L[v;\alpha](x_i) = \alpha \int_{x_i}^{x_{i+1}} e^{-\alpha(s - x_i)} v(s)\, ds. \tag{3.18}$$

By writing the convolution integrals this way, we obtain a summation method which has a complexity of $O(N)$. Note that the same algorithm can be applied to compute the convolution integral for the second derivative operator by splitting the integral at a point x. After applying the above algorithm to the left and right contributions, we can recombine them through “averaging" to recover the original integral. While a variety of quadrature methods have been proposed to compute the local integrals (3.18) and (3.17) (see e.g., [45, 50, 55, 43, 38, 33]), we shall consider sixth-order

quadrature methods introduced in [44], which use WENO interpolation to address both smooth and non-smooth problems. In what follows, we describe the procedure for $J_R[v;\alpha](x_i)$, since the reconstruction for $J_L[v;\alpha](x_i)$ is similar. This approximation uses a six point stencil given by

$$S(i) = \{x_{i-3}, \cdots, x_{i+2}\},$$

which is then divided into three smaller stencils, each of which contains four points, defined by $S_r = \{x_{i-3+r}, \cdots, x_{i+r}\}$ for $r = 0, 1, 2$. We associate r with the shift in the stencil. A graphical depiction of this stencil is provided in 3.1. The quadrature method is developed as follows:

1. On each of the small stencils $S_r(i)$., we use the approximation

$$J_R^{(r)}[v;\alpha](x_i) \approx \alpha \int_{x_{i-1}}^{x_i} e^{-\alpha(x_i-s)} p_r(s)\, ds = \sum_{j=0}^{3} c_{-3+r+j}^{(r)} v_{-3+r+j}, \tag{3.19}$$

   where $p_r(x)$ is the Lagrange interpolating polynomial formed from points in $S_r(i)$ and $c_\ell^{(r)}$ are the interpolation coefficients, which depend on the parameter α and the grid spacing, but not v.

2. In a similar way, on the large stencil $S(i)$ we obtain the approximation

$$J_R[v;\alpha](x_i) \approx \alpha \int_{x_{i-1}}^{x_i} e^{-\alpha(x_i-s)} p(s)\, ds. \tag{3.20}$$

3. When function $v(x)$ is smooth, we can combine the interpolants on the smaller stencils, so they are consistent with the high-order approximation obtained on the larger stencil, i.e.,

$$J_R[v;\alpha](x_i) = \sum_{r=0}^{2} d_r J_R^{(r)}[v;\alpha](x_i), \tag{3.21}$$

   where $d_r > 0$ are called the linear weights, which form a partition of unity. The problems we consider in this work involve smooth functions, so this is sufficient for the final approximation. For instances in which the solution is not smooth, the linear weights can be mapped to nonlinear weights using the notion of smoothness. We refer the interested reader to previous work [56, 44, 45] for details concerning non-smooth data sets.

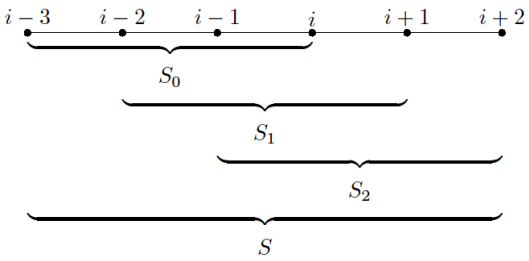


Figure 3.1: Stencils used to build the six point quadrature [56, 44].

In A.3, we provide the expressions used to compute coefficients $c_\ell^{(r)}$ and d_r for a uniform grid, although the non-uniform grid case can be done as well [73]. In the case of a non-uniform mesh, the linear weights d_r would become locally defined in the neighborhood of a given point and would need to be computed on-the-fly. Uniform grids eliminate this requirement as the linear weights, for a given direction, can be computed once per time step and reused in each of the lines pointing along that direction.

3.2.5 Coupling Approximations with Time Integration Methods

Here, we demonstrate how one can use this approach to solve a large class of PDEs by coupling the spatial discretizations in 3.2.4 with explicit time stepping methods. In what follows, we shall consider general PDEs of the form

$$\partial_t U = F(t, U),$$

where $F(t, U)$ is a collective term for spatial derivatives involving the solution variable U. Possible choices for F might include generic nonlinear advection and diffusion terms

$$F(t, U) = \partial_x g_1(U) + \partial_{xx} g_2(U),$$

or even components of the HJ equations

$$F(t, U) = H(U, \partial_x U).$$

To demonstrate how one can couple these approaches, we start by discretizing a PDE in time, but, rather than use backwards Euler, we use an s-stage explicit Runge-Kutta (RK) method, i.e.,

$$u_{n+1} = u_n + \sum_{i=1}^{s} b_i k_i,$$

where the various stages are given by

$$\begin{aligned} k_1 &= F(t^n, u^n), \\ k_2 &= F(t^n + c_2\Delta t, u^n + \Delta t a_{21} k_1), \\ &\vdots \\ k_s &= F\left(t^n + c_s\Delta t, u^n + \Delta t \sum_{j=1}^{s} a_{sj} k_j\right). \end{aligned}$$

As with a standard Method-of-Lines (MOL) discretization, we would need to reconstruct derivatives within each RK-stage. To illustrate, consider the nonlinear A-D equation,

$$F(t, u) = \partial_x g_1(u) + \partial_{xx} g_2(u).$$

For a term such as g_1, we would use a monotone Lax-Friedrichs flux splitting, i.e., $g_1 \sim \frac{1}{2}(g_1^+ + g_1^-)$, where $g_1^\pm = \frac{1}{2}(g_1(u) \pm ru)$ with $r = \max_u g_1'(u)$. Hence, a particular RK-stage can be approximated using

$$F(t, u) \approx -\frac{1}{2}\alpha \sum_{p=1}^{s} \mathcal{D}_L^p[g_1^+(u); \alpha] + \frac{1}{2}\alpha \sum_{p=1}^{s} \mathcal{D}_R^p[g_1^-(u); \alpha] + \alpha_\nu^2 \sum_{p=1}^{s} \mathcal{D}_0^p[g_2(u); \alpha_\nu].$$

The resulting approximation to the RK-stage can be shown to be $O(\Delta t^s)$ accurate. Another nonlinear PDE of interest to us is the H-J equation

$$F(t, u) = H(\partial_x u).$$

In a similar way, we would replace the Hamiltonian with a monotone numerical Hamiltonian, such as

$$\hat{H}(v^-, v^+) = H\left(\frac{v^- + v^+}{2}\right) + r(v^-, v^+)\frac{v^- - v^+}{2},$$

where $r(v^-, v^+) = \max_v H'(v)$. Then, the left and right derivative operators in the numerical Hamiltonian can be replaced with

$$\partial_x^- u = \alpha \sum_{p=1}^{s} \mathcal{D}_R^p[u;\alpha], \quad \partial_x^+ u = -\alpha \sum_{p=1}^{s} \mathcal{D}_L^p[u;\alpha],$$

which, again, yields an $O(\Delta t^s)$ approximation.

In previous work [56, 44], for linear forms of $F(t,u)$, it was shown that the resulting methods are unconditionally stable when coupled to explicit RK methods, up to order 3. Extensions beyond third-order are, indeed, possible, but were not considered. For general, nonlinear problems, we typically couple an s-stage RK method to a successive convolution approximation of the same time accuracy, so that the error in the resulting approximation is $O(\Delta t^s)$.

3.3 Nearest-Neighbor Domain Decomposition Algorithm

In this section, we provide the relevant mathematical definitions of our domain decomposition algorithm, which are derived from the key operators used in successive convolution. Our goal is to establish and exploit data locality in the method, so that certain reconstructions, which are nonlocal, can be independently completed on non-overlapping blocks of the domain. This is achieved in part by leveraging certain decay properties of the integral representations. Once we have established some useful definitions, we use them to derive conditions in 3.3.1, which restrict the communication pattern to N-Ns. Then in sections 3.3.2 and 3.3.3, we illustrate how this condition can be used to enforce boundary conditions, in a consistent manner, for first and second derivative operators, on each of the blocks. We then provide a brief summary of these findings along with additional comments in 3.3.4.

Maintaining a localized stencil is often advantageous in parallel computing applications. Code which is based on N-Ns is generally much easier to write and maintain. Additionally, messages used to exchange data owned by other blocks may not have to travel long distances within the network, provided that the blocks are mapped physically close together in hardware. Communication, even on modern computing systems, is far more expensive than computation. Therefore, an initial strategy for domain decomposition is to enforce N-N dependencies between the blocks. In order to

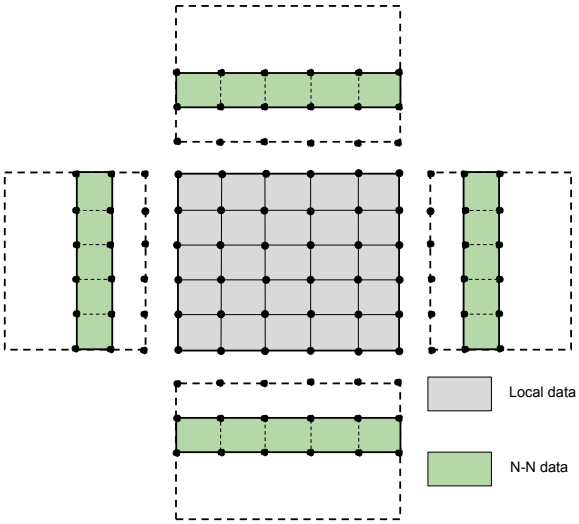


Figure 3.2: A six-point WENO quadrature stencil in 2-D.

decompose a problem into smaller, independent pieces, we separate the global domain into blocks that share borders with their nearest neighbors. For example, in the case of a 1-D problem defined on the interval $[a, b]$ we can form N blocks by writing

$$a = c_0 < c_1 < c_2 < \cdots < c_N = b,$$

with $\Delta c_i = c_{i+1} - c_i$ denoting the width of block i. Multidimensional problems can be addressed in a similar way by partitioning the domain along multiple directions. Solving a PDE on each of these blocks, independently, requires an understanding of the various data dependencies. First, we address the local integrals J_*. Depending on the quadrature method, the reconstruction algorithm might require data from neighboring blocks. Reconstructions based on previously described WENO-type quadratures require an extension of the grid in order to build the interpolant. This involves a “halo" region (see Figure 3.2), which is distributed amongst N-N blocks in the decomposition. On the other hand, more compact quadratures, such as Simpson’s method [33], do not require this data. In this case, the quadrature communication phase can be ignored.

The major task for this work involves efficiently communicating the data necessary to build each of the convolution integrals I_*. Once the local integrals J_* are constructed through quadrature, we

sweep across the lines of the domain to build the convolution integrals. It is this operation which couples the integrals across the blocks. To decompose this operation, we first rewrite the integral operators I_*, assuming we are in block i, as

$$\begin{aligned} I_L[v;\alpha](x) &= \alpha \int_x^b e^{-\alpha(s-x)} v(s)\, ds, \\ &= \alpha \int_x^{c_{i+1}} e^{-\alpha(s-x)} v(s)\, ds + \alpha \int_{c_{i+1}}^{c_N} e^{-\alpha(s-x)} v(s)\, ds, \end{aligned}$$

and

$$\begin{aligned} I_R[v;\alpha](x) &= \alpha \int_a^x e^{-\alpha(x-s)} v(s)\, ds, \\ &= \alpha \int_{c_0}^{c_i} e^{-\alpha(x-s)} v(s)\, ds + \alpha \int_{c_i}^x e^{-\alpha(x-s)} v(s)\, ds. \end{aligned}$$

These relations, which assume x is within the interval $[c_i, c_{i+1}]$, elucidate the local and non-local contributions to the convolution integrals within block i. Using simple algebraic manipulations, we can expand the non-local contributions to find that

$$\begin{aligned} \int_{c_{i+1}}^{c_N} e^{-\alpha(s-x)} v(s)\, ds &= \sum_{j=i+1}^{N-1} \int_{c_j}^{c_{j+1}} e^{-\alpha(s-x)} v(s)\, ds, \\ &= \sum_{j=i+1}^{N-1} e^{-\alpha(c_j-x)} \int_{c_j}^{c_{j+1}} e^{-\alpha(s-c_j)} v(s)\, ds, \end{aligned}$$

and

$$\begin{aligned} \int_{c_0}^{c_i} e^{-\alpha(x-s)} v(s)\, ds &= \sum_{j=0}^{i-1} \int_{c_j}^{c_{j+1}} e^{-\alpha(x-s)} v(s)\, ds, \\ &= \sum_{j=0}^{i-1} e^{-\alpha(x-c_{j+1})} \int_{c_j}^{c_{j+1}} e^{-\alpha(c_{j+1}-s)} v(s)\, ds, \end{aligned}$$

for the right and left-moving data, respectively. With these relations, each of the convolution integrals can be formed according to

$$I_L[v;\alpha](x) = \alpha \int_x^{c_{i+1}} e^{-\alpha(s-x)} v(s)\, ds + \alpha \left(\sum_{j=i+1}^{N-1} e^{-\alpha(c_j-x)} \int_{c_j}^{c_{j+1}} e^{-\alpha(s-c_j)} v(s)\, ds \right), \tag{3.22}$$

and

$$I_R[v;\alpha](x) = \alpha \left(\sum_{j=0}^{i-1} e^{-\alpha(x-c_{j+1})} \int_{c_j}^{c_{j+1}} e^{-\alpha(c_{j+1}-s)} v(s)\, ds \right) + \alpha \int_{c_i}^x e^{-\alpha(x-s)} v(s)\, ds. \tag{3.23}$$

From equations (3.22) and (3.23), we observe that both of the global convolution integrals can be split into a localized convolution with additional contributions coming from preceding or successive *global integrals* owned by other blocks in the decomposition. These global integrals contain exponential attenuation factors, the size of which depends on the respective distances between any pair of sub-domains. Next, we use this result to derive the restriction that facilitates N-N dependencies.

3.3.1 Nearest-Neighbor Criterion

Building a consistent block-decomposition for the convolution integral is non-trivial, since this operation globally couples unknowns along a *dimension* of the grid. Fortunately, the exponential kernel used in these reconstructions is pleasant in the sense that it automatically generates a region of compact support around a given block. Examining the exponential attenuation factors in (3.22) and (3.23), we see that contributions from blocks beyond N-Ns become small provided that (1) the distance between the blocks is large or (2) α is taken to be sufficiently large. Since we have less control over the block sizes e.g., Δc_i, we can enforce the latter criterion. That is, we constrain α so that

$$e^{-\alpha L_m} \leq \epsilon, \tag{3.24}$$

where $\epsilon \ll 1$ is some prescribed error tolerance, typically taken as 1×10^{-16}, and $L_m = \min_i \Delta c_i$ denotes the length smallest block. Taking logarithms of both sides and rearranging the inequality, we obtain the bound

$$-\alpha \leq \frac{\log(\epsilon)}{L_m}.$$

Our next step is to write this in terms of the time step Δt, using the choice of α. However, the bound on the time step depends on the choice of α. In 3.2.1, we presented two definitions for the parameter α, namely

$$\alpha \equiv \frac{\beta}{c_{\max}\Delta t}, \text{ or } \alpha \equiv \frac{\beta}{\sqrt{\nu \Delta t}}.$$

Using these definitions for α, we obtain two conditions depending on the choice of α. For the linear advection equation and the wave equation, we obtain the condition

$$-\frac{\beta}{c_{\max}\Delta t} \leq \frac{\log(\epsilon)}{L_m} \implies \Delta t \leq -\frac{\beta L_m}{c_{\max}\log(\epsilon)}. \tag{3.25}$$

Likewise, for the diffusion equation, the restriction is given by

$$-\frac{\beta}{\sqrt{\nu\Delta t}} \leq \frac{\log(\epsilon)}{L_m} \implies \Delta t \leq \frac{1}{\nu}\left(\frac{\beta L_m}{\log(\epsilon)}\right)^2. \tag{3.26}$$

Depending on the problem, if the condition (3.25) or (3.26) is not satisfied, then we use the maximally allowable time step for a given tolerance ϵ, which is given by the equality component of the relevant condition. If several different operators appear in a given problem and are to be approximated with successive convolution, then each operator will be associated with its own α. In such a case, we should bound the time step according to the condition that is more restrictive among (3.25) and (3.26), which can be accomplished through the choice

$$\Delta t \leq \min\left(-\frac{\beta L_m}{c_{\max}\log(\epsilon)}, \frac{1}{\nu}\left(\frac{\beta L_m}{\log(\epsilon)}\right)^2\right). \tag{3.27}$$

As before, when the condition is not met, then we use the equality in (3.27).

Restricting Δt according to (3.25), (3.26), or (3.27) ensures that contributions to the right and left-moving convolution integrals, beyond N-Ns, become negligible. This is important because it significantly reduces the amount of communication, at the expense of a potentially restrictive time step. Note that in 3.5.4, we analyze the limitations of such restrictions for the linear advection equation. In our future work, we shall consider generalizations of our approach, which do not require (3.25), (3.26), or (3.27). In sections 3.3.2 and 3.3.3, we demonstrate how to formulate block-wise definitions of the global $\mathcal{L}_*^{-1}$ operators using the derived conditions (3.25), (3.26), or (3.27).

3.3.2 Enforcing Boundary Conditions for ∂_x

In order to enforce the block-wise boundary conditions for the first derivative ∂_x, we recall our definitions (3.3) and (3.4) for the left and right-moving inverse operators:

$$\mathcal{L}_L^{-1}[v;\alpha](x) = I_L[v;\alpha](x) + Be^{-\alpha(b-x)}, \tag{3.28}$$

$$\mathcal{L}_R^{-1}[v;\alpha](x) = I_R[v;\alpha](x) + Ae^{-\alpha(x-a)}. \tag{3.29}$$

We can modify these definitions so that each block contains a pair of inverse operators given by

$$\mathcal{L}_{L,i}^{-1}[v;\alpha](x) = I_{L,i}[v;\alpha](x) + B_i e^{-\alpha(c_{i+1}-x)}, \tag{3.30}$$

$$\mathcal{L}_{R,i}^{-1}[v;\alpha](x) = I_{R,i}[v;\alpha](x) + A_i e^{-\alpha(x-c_i)}, \tag{3.31}$$

where $I_{*,i}$ are defined as

$$I_{L,i}[v;\alpha](x) = \alpha \int_x^{c_{i+1}} e^{-\alpha(s-x)} v(s)\, ds, \quad I_{R,i}[v;\alpha](x) = \alpha \int_{c_i}^{x} e^{-\alpha(x-s)} v(s)\, ds, \tag{3.32}$$

and the subscript i denotes the block in which the operator is defined. As before, this assumes $x \in [c_i, c_{i+1}]$. To address the boundary conditions, we need to determine expressions for the constants B_i and A_i on each of the blocks in the domain. First, substitute the definitions (3.22) and (3.23) into (3.28) and (3.29):

$$\begin{aligned}\mathcal{L}_L^{-1}[v;\alpha](x) = \alpha &\int_x^{c_{i+1}} e^{-\alpha(s-x)} v(s)\, ds \\ &+ \alpha \left(\sum_{j=i+1}^{N-1} e^{-\alpha(c_j - x)} \int_{c_j}^{c_{j+1}} e^{-\alpha(s-c_j)} v(s)\, ds \right) + Be^{-\alpha(c_N - x)},\end{aligned} \tag{3.33}$$

$$\begin{aligned}\mathcal{L}_R^{-1}[v;\alpha](x) = \alpha &\left(\sum_{j=0}^{i-1} e^{-\alpha(x-c_{j+1})} \int_{c_j}^{c_{j+1}} e^{-\alpha(c_{j+1}-s)} v(s)\, ds \right) \\ &+ \alpha \int_{c_i}^{x} e^{-\alpha(x-s)} v(s)\, ds + Ae^{-\alpha(x-c_0)}.\end{aligned} \tag{3.34}$$

Since we wish to maintain consistency with the true operator being inverted, we require that each of the block-wise operators satisfy

$$\mathcal{L}_L^{-1}[v;\alpha](x) = \mathcal{L}_{L,i}^{-1}[v;\alpha](x), \quad \mathcal{L}_R^{-1}[v;\alpha](x) = \mathcal{L}_{R,i}^{-1}[v;\alpha](x),$$

which can be explicitly written as

$$B_i e^{-\alpha(c_{i+1}-x)} = \left(\sum_{j=i+1}^{N-1} e^{-\alpha(c_j-x)} I_{L,j}[v;\alpha](c_j) \right) + B e^{-\alpha(c_N-x)}, \tag{3.35}$$

$$A_i e^{-\alpha(x-c_i)} = \left(\sum_{j=0}^{i-1} e^{-\alpha(x-c_{j+1})} I_{R,j}[v;\alpha](c_{j+1}) \right) + A e^{-\alpha(x-c_0)}. \tag{3.36}$$

Evaluating (3.35) at c_{i+1} and (3.36) at c_i, we obtain

$$B_i = \left(\sum_{j=i+1}^{N-1} e^{-\alpha(c_j-c_{i+1})} I_{L,j}[v;\alpha](c_j) \right) + B e^{-\alpha(c_N-c_{i+1})}, \tag{3.37}$$

$$A_i = \left(\sum_{j=0}^{i-1} e^{-\alpha(c_i-c_{j+1})} I_{R,j}[v;\alpha](c_{j+1}) \right) + A e^{-\alpha(c_i-c_0)}. \tag{3.38}$$

Modifying Δt according to either (3.25) or, if necessary (3.27), results in the communication stencil shown in Figure 3.3. More specifically, the terms representing the boundary contributions in each of the blocks are given by

$$B_i = \begin{cases} B, & i = N-1, \\ \\ I_{R,i+1}[v;\alpha](c_{i+1}), & i < N-1, \end{cases}$$

and

$$A_i = \begin{cases} A, & i = 0, \\ \\ I_{R,i-1}[v;\alpha](c_i), & 0 < i. \end{cases}$$

These relations generalize the various boundary conditions set by a problem. For example, with periodic problems, we can select

$$B = I_{L,0}[v;\alpha](c_0), \quad A = I_{R,N-1}[v;\alpha](c_N).$$

This is the relevant strategy employed by domain decomposition algorithms in this work.

3.3.3 Enforcing Boundary Conditions for ∂_{xx}

The enforcement of boundary conditions on blocks of the domain for the second derivative can be accomplished using an identical procedure to the one described in 3.3.2. First, we recall the inverse

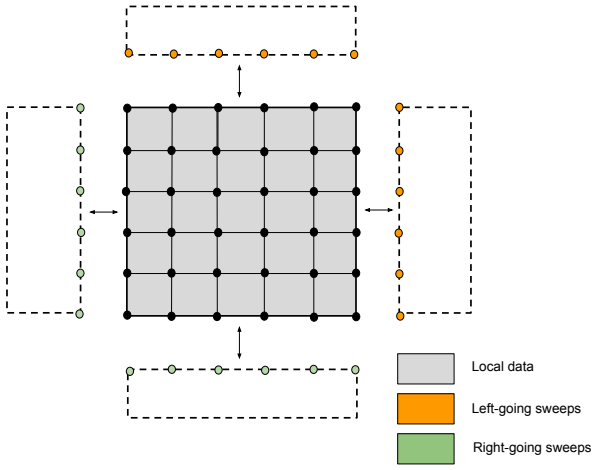


Figure 3.3: Fast convolution communication stencil in 2-D based on N-Ns.

operator associated with a second derivative (3.5):

$$\mathcal{L}_0^{-1}[v;\alpha](x) = I_0[v;\alpha](x) + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}, \tag{3.39}$$

and define an analogous block-wise definition of (3.39) as

$$\mathcal{L}_{0,i}^{-1}[v;\alpha](x) = I_{0,i}[v;\alpha](x) + A_i e^{-\alpha(x-c_0)} + B_i e^{-\alpha(c_N-x)},$$

with the localized convolution integral

$$I_{0,i}[v;\alpha](x) = \frac{\alpha}{2}\int_{c_i}^{c_{i+1}} e^{-\alpha|x-s|}v(s)\,ds.$$

Again, the subscript i denotes the block in which the operator is defined and we take $x \in [c_i, c_{i+1}]$. For the purposes of the fast summation algorithm, it is convenient to split this integral term into an average of left and right contributions, i.e.,

$$\mathcal{L}_{0,i}^{-1}[v;\alpha](x) = \frac{1}{2}\Big(I_{L,i}[v;\alpha](x) + I_{R,i}[v;\alpha](x)\Big) + A_i e^{-\alpha(x-c_i)} + B_i e^{-\alpha(c_{i+1}-x)}, \tag{3.40}$$

where $I_{*,i}$ are the same integral operators shown in equation (3.32) used to build the first derivative. As in the case of the first derivative, a condition connecting the boundary conditions on the blocks

to the non-local integrals can be derived, which, if evaluated at the ends of the block, results in the 2×2 linear system

$$\begin{aligned} A_i + B_i e^{-\alpha \Delta c_i} &= \frac{1}{2} \sum_{j=i+1}^{N-1} e^{-\alpha(c_j - c_i)} I_{L,j}[v;\alpha](c_j) \\ &\quad + \frac{1}{2} \sum_{j=0}^{i-1} e^{-\alpha(c_i - c_{j+1})} I_{R,j}[v;\alpha](c_{j+1}) \\ &\quad + A e^{-\alpha(c_i - c_0)} + B e^{-\alpha(c_N - c_i)}, \end{aligned} \tag{3.41}$$

$$\begin{aligned} A_i e^{-\alpha \Delta c_i} + B_i &= \frac{1}{2} \sum_{j=i+1}^{N-1} e^{-\alpha(c_j - c_{i+1})} I_{L,j}[v;\alpha](c_j) \\ &\quad + \frac{1}{2} \sum_{j=0}^{i-1} e^{-\alpha(c_{i+1} - c_{j+1})} I_{R,j}[v;\alpha](c_{j+1}) \\ &\quad + A e^{-\alpha(c_{i+1} - c_0)} + B e^{-\alpha(c_N - c_{i+1})}. \end{aligned} \tag{3.42}$$

Equations (3.41) and (3.42) can be solved analytically to find that

$$\begin{bmatrix} A_i \\ B_i \end{bmatrix} = \frac{1}{1 - e^{-2\alpha \Delta c_i}} \begin{bmatrix} 1 & -e^{-\alpha \Delta c_i} \\ -e^{-\alpha \Delta c_i} & 1 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \end{bmatrix},$$

where we have used the variables r_0 and r_1 to denote the terms appearing on the right-hand side of (3.41) and (3.42), respectively. Under the N-N constraints (3.26) or (3.27), many of the exponential terms can be neglected resulting in the compact expressions

$$A_i = \begin{cases} A, \quad i = 0, \\ \\ \frac{1}{2} I_{R,i-1}[v;\alpha](c_i) \equiv I_{0,i-1}[v;\alpha](c_i), \quad 0 < i, \end{cases}$$

and

$$B_i = \begin{cases} B, \quad i = N-1, \\ \\ \frac{1}{2} I_{L,i+1}[v;\alpha](c_{i+1}) \equiv I_{0,i+1}[v;\alpha](c_{i+1}), \quad i < N-1. \end{cases}$$

3.3.4 Additional Comments

In this section, we developed the mathematical framework behind our proposed domain decomposition algorithm. We derived a condition, which reduces the construction of a nonlocal operator to a N-N dependency by leveraging the decay properties of the exponential term within the convolution integrals. We wish to reiterate that this condition is not entirely necessary. One could remove this condition by including contributions beyond N-Ns at the expense of additional communication. This change would certainly result in a loss of speed per time step, but the additional expense could be amortized by the ability to use much a larger time step, which would reduce the overall time-to-solution. As a first pass, we shall ignore these additional contributions, which may limit the scope of problems we can study, but we plan to generalize these algorithms in our future work via an adaptive strategy. This approach would begin using data from N-Ns, then gradually include additional contributions using information about the decay from the exponential. In the next section, we shall discuss details regarding the implementation of our methods and particular design choices made in the construction of our algorithms.

3.4 Strategies for Efficient Implementation on Parallel Systems

In this section, we discuss strategies for constructing parallel algorithms to solve PDEs. We provide the details related to our work on *thread-scalable*, shared memory algorithms, as well as distributed memory algorithms, where the problem is decomposed into smaller, independent problems that communicate necessary information via message-passing. 3.4.1 introduces the core concepts in Kokkos performance portability library, which is used to develop our shared memory algorithms. Once we have introduced these ideas, we explore numerous loop-level optimizations for essential loop structures in 3.4.3, using the performance metrics discussed in 3.4.2. Building on the results of these loop experiments, we outline the structure of our shared memory algorithms in 3.4.4. We then discuss the implementation of the distributed memory component of our algorithms in 3.4.5, along with modifications which enable the use of an adaptive time stepping rule. Finally, we summarize the key findings and developments of the implementation which are used to conduct

our numerical experiments.

3.4.1 Selecting a Shared Memory Programming Model

Many programming models exist to address the aspects of shared memory paralellization, such as OpenMP, OpenACC, CUDA, and OpenCL. The question of which model to use often depends on the target architecture on which the code is to run. However, given the recent trend towards deploying more heterogeneous computing systems, e.g., ones in which a given node contains a variety of CPUs with one, or many, accelerators (typically GPUs), the choice becomes far more complicated. Developing codes which are performant across many computing architectures is a highly non-trivial task. Due to memory access patterns, code which is optimized to run on CPUs is often not optimal on GPUs, so these models address *portability* rather than *performance*. This introduces yet another concern related to code management and maintenance: As new architectures are deployed, code needs to be tuned or modified to take advantage of new features, which can be time consuming. Additionally, enabling these abstractions almost invariably results in either multiple versions of the code or rather complicated build systems.

In our work, we choose to adopt Kokkos [46], a *performance portable*, shared memory programming model. Kokkos tries to address the aforementioned problem posed by rapidly evolving architectures through template-metaprogramming abstractions. Their model provides abstractions for common parallel policies (i.e., for-loops, reductions, and scans), memory spaces, and execution spaces. The architecture specific details are hidden from the users through these abstractions, yet the setup allows the application programmer to take advantage of numerous performance-related features. Given basic knowledge in templates, operator overloads, as well as functors and lambdas, one can implement a variety of program designs. Also provided are the so-called views, which are powerful multi-dimensional array containers that allow Kokkos iterators to map data onto various architectures in a performant way. Additionally, the bodies of iterators become either a user-defined functor or a KOKKOS_LAMBDA, which is just a functor that is generated by the compiler. This allows users to maintain one version of the code which has the flexibility to run on various architec-

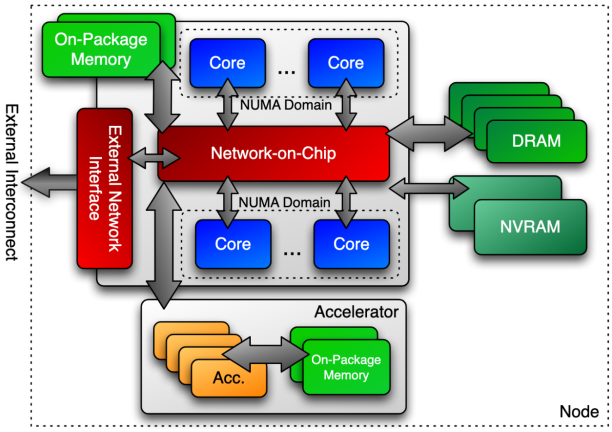


Figure 3.4: Heterogeneous platform targeted by Kokkos [46].

tures, such as the one depicted in 3.4. Other performance portability models, such as RAJA [74], work in a similar fashion as Kokkos, but they are less intrusive with regard to memory management. With RAJA, the user is responsible for implementing architecture dependent details such as array layouts and policies. In this sense, RAJA emphasizes portability, with the user being responsible for handling performance.

3.4.2 Comment on Performance Metrics

In order to benchmark the performance of the algorithms, we need a descriptive metric that accounts for varying workloads among problem sizes. In our numerical simulations we use a time stepping rule so that problems with a smaller mesh spacing require more time steps, i.e., $\Delta t \sim \Delta x$. Therefore, one can either time the code for a fixed number of steps or track the number of steps in the entire simulation $t \in (0, T]$ and compute the average time per time step. We adopt the former approach throughout this work. To account for the varying workloads attributed to varying cells/grid points, we define the update rate as Degrees-of-Freedom/node/s (DOF/node/s), which can be computed via

$$\text{DOF/node/s} = \frac{\text{total variables} \times N^d}{\text{nodes} \times \left(\frac{\text{total time (s)}}{\text{total steps}} \right)}, \tag{3.43}$$

where d is the number of spatial dimensions. This metric is a more general way of comparing the raw performance of the code, as it allows for simultaneous comparisons among linear or nonlinear

problems with varying degrees of dimensionality and number of components. It also allows for a comparison, in terms of speed, against other classes of methods, such as finite element methods, where the workload on a given cell is allowed to vary according to the number of basis elements [1]. In 3.4.3, we shall use this performance metric to benchmark a collection of techniques for prescribing parallelism across predominant loop structures in the algorithms for successive convolution.

3.4.3 Benchmarking Prototypical Loop Patterns

Often, when designing shared memory algorithms, one has to make design decisions prescribing the way threads are dispatched to the available data. However, there are often many ways of accomplishing a given task. Kokkos provides a variety of parallel iteration techniques — the selection of a particular pattern typically depends on the structure of the loop (perfectly or imperfectly nested) and the size of the loops. In [75], authors sought to optimize a recurring pattern, consisting of triple or quadruple nested for-loops, in the Athena++ MHD code [76]. Their strategy was to use a flexible loop macro to test various loop structures across a range of architectures. Athena++ was already optimized to run on Intel Xeon-Phi platforms, so they primarily focused on approaches for porting to GPUs which maintained this performance on CPUs. Our work differs in that we have not yet identified optimal loop patterns for CPUs or GPUs and algorithms used here contain at least two major prototypical loop patterns. At the moment, we are not focusing on optimizing for GPUs, but, we do our best to keep in mind possible performance-related issues associated with various parallelization techniques. Some examples of recurring loop structures, in successive convolution algorithms, for 3D problems, are provided in Scheme 3.1 and Scheme 3.2. Technically, there are left and right-moving operators associated with each direction, but, for simplicity, we will ignore this in the pseudo-code.

Another important note, we wish to make, concerns the storage of the operator data on the mesh. Since the operations are performed “line-by-line" on potentially large multidimensional arrays, we

[1] Note that the update frequency does not account for error in the numerical solution. Certainly, in order to compare the efficiency of various methods, especially those that belong to different classes, one must take into account the quality of the solution. This would be reflected in, for example, an error versus time-to-solution plot.

```
for(int ix = 0; ix < Nx; ix++){
  for(int iy = 0; iy < Ny; iy++){
    // Perform some intermediate calculations
    // ...
    // Apply 1-D algorithm to z-line data
    for(int iz = 0; iz < Nz; iz++){
        z_operator(ix,iy,iz) = ...
    }
  }
}
```

Scheme 3.1: Looping pattern used in the construction of local integrals, convolutions, and boundary steps.

```
// Looping pattern for the resolvent operators
for(int ix = 0; ix < Nx; ix++){
  for(int iy = 0; iy < Ny; iy++){
    for(int iz = 0; iz < Nz; iz++){
      z_operator(ix,iy,iz) = u(ix,iy,iz) - z_operator(ix,iy,iz);
      z_operator(ix,iy,iz) *= alpha_z;
    }
  }
}
```

Scheme 3.2: Another looping pattern used to build “resolvent" operators. With some modifications, this same pattern could be used for the integrator step. In several cases, this iteration pattern may require reading entries, which are separated by large distances (i.e., the data is strided), in memory.

choose to store the data in memory so that the sweeps are performed on the fastest changing loop variables. This allows us to avoid significant memory access penalties associated with reading and writing to arrays, as the entries of interest are now consecutive in memory [2]. For example, suppose we have an $N-$dimensional array with indices $x_1, x_2, \cdots, x_N$, and we wish to construct an operator in the x_1 direction. Then, we would store this operator in memory as operator$(x_2, \cdots, x_N, x_1)$. The loops appearing in Scheme 3.1 and Scheme 3.2 can then be permuted accordingly. Note that the solution variable u$(x_1, x_2, \cdots, x_N)$ is not transposed and is a read-only quantity during the construction of the operators.

In an effort to develop an efficient application, we follow the approach described in [75], to determine optimal loop iteration techniques for patterns, such as Scheme 3.1 and Scheme 3.2. Our

[2]This is true when the memory space is that of the CPU (host memory). In device memory, these entries will be “coalesced", which is the optimal layout for threading on GPUs. This mapping of indices, between memory spaces, is automatically handled by Kokkos.

simple 2-D and 3-D experiments tested numerous combinations of policies including naive, as well as more complex parallel iteration patterns using the OpenMP backend in Kokkos. Our goals were to quantify possible performance gains attainable through the following strategies:

1. auto-vectorization via #pragma statements or ThreadVectorRange (TVR)
2. improving data reuse and caching behavior with loop tiling/blocking
3. prescribing parallelism across combinations of team-type execution policies and team sizes

Vectorization can offer substantial performance improvements for data that is contiguous in memory; however, several performance critical operations in the algorithms involve reading data which is strided in memory. Therefore, it is not straightforward whether vectorization would offer any improvements. Additionally, for larger problems, the line operations along certain directions involve reading strided data, so that benefits of caching are lost. The performance penalty of operating on data with the wrong layout depends on the architecture, with penalties on GPUs typically being quite severe compared to CPUs. The use of a blocked iteration pattern, such as the one outlined in Scheme A.1 (see A.2), is a step toward minimizing such performance penalties. In order to see a performance benefit from this approach, the algorithms must be structured, in such a way, as to reuse the data that is read into caches, as much as possible. Naturally, one could prescribe one or more threads (of a team) to process blocks, so we chose to implement cache blocking using the hierarchical execution policies provided by Kokkos. From coarse-to-fine levels of granularity, these can be ordered as follows: TeamPolicy (TP), TeamThreadRange (TTR), and ThreadVectorRange (TVR). For perfectly nested loops, one can achieve similar behavior using MDRange and prescribing block sizes. During testing, we found that when block sizes are larger than or equal to the size of the view, a segmentation fault occurs, so this was avoided. The results of our loop experiments are provided in Figures 3.5 and A.1. For tests employing blocking, we used a block size of 256^2 in 2-D, while 3-D problems used a block size of 32^3. Information regarding various choices, such as compiler, optimization flags, etc., used to generate these results can be found in Table 3.1.

CPU Type	Intel Xeon Gold 6148
C++ Compiler	ICC 2019.03
Optimization Flags	-O3 -xCORE-AVX512 -qopt-zmm-usage=high -qno-opt-prefetch
Thread Bindings	OMP_PROC_BIND=close, OMP_PLACES=threads

Table 3.1: Architecture and code configuration for the loop experiments conducted on the Intel 18 cluster at Michigan State University's Institute for Cyber-Enabled Research. To leverage the wide vector registers, we encourage the compiler to use AVX-512 instructions. Hardware prefetching is not used, as initial experiments seem to indicate that it hindered performance. Initially, we used GCC 8.2.0-2.31.1 as our compiler, but we found through experimentation that using an Intel compiler improved the performance of our application by a factor of ~ 2 for this platform. Authors in [75] experienced similar behavior for their application and attribute this to a difference in auto-vectorization capabilities between compilers. An examination of the source code for loop execution policies in Kokkos reveals that certain decorators, e.g., #pragma ivdep are present, which help encourage auto-vectorization when Intel compilers are used. We are unsure if similar hints are provided for GCC.

As part of our blocking implementation, we stored block information in views, which could then be accessed by a team of threads. After the information about the block is obtained, we compute indices for the data within the block and use these to extract the relevant grid data. Then, one can either create subviews (shallow copies) of the block data or proceed directly with the line calculations of the block data. We refer to these as tiling with and without subviews, respectively. Intuitively, one would think that skipping the block subview creation step would be faster. Among the blocked or tiled experiments, those that created the subviews of the tile data were generally faster than those that did not. Using blocking for smaller problems typically resulted in a large number of idle threads, which significantly degraded the performance compared to non-blocked policies. In such situations, a user would need to take care to ensure that a sufficient number of blocks are used to generate enough work, i.e., each thread (or team) has at least one block to process. For larger problems, blocking was faster when compared to variants that did not use blocking. We observe that the performance of non-blocked policies begins to degrade once a problem becomes sufficiently large, whereas blocked policies maintained a consistent update rate, even as the problem size increased. By separating the key loop structures from the complexities of the application, we

were able to expedite the experimental process for identifying efficient loop execution techniques. In 3.4.4, we use the results of these experiments to inform choices regarding the design of the shared memory algorithms.

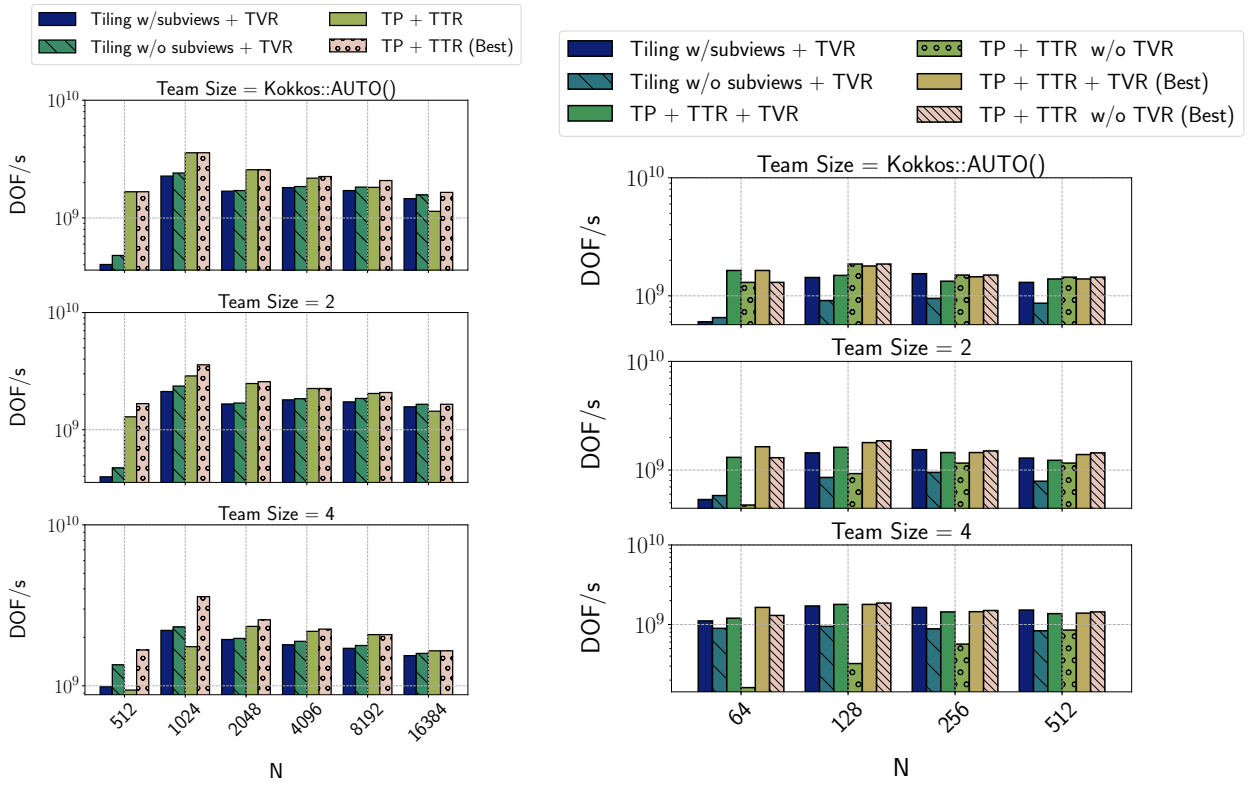


Figure 3.5: Plots comparing the performance of different parallel execution policies for the pattern in Scheme 3.1 using test cases in 2-D (left) and 3-D (right). Tests were conducted on a single node that consists of 40 cores using the code configuration outlined in 3.1. Each group consists of three plots, whose difference is the value selected for the team size. We note that hyperthreading is not enabled on our systems, so Kokkos::AUTO() defaults to a team size of 1. In each pane, we use “best" to refer to the best run for that configuration across different team sizes. Tile experiments used block sizes of 256^2, in 2-D problems, and 32^3 in 3-D. We observe that vectorized policies are generally faster than non-vectorized policies. Interestingly, among blocked/tiled policies, construction of subviews appears to be faster than those that skip the subview construction, despite the additional work. As the problem size increases, the performance of blocked policies improves substantially. This can be attributed to the large number of idle thread teams when the problem size does not produce enough blocks. In such cases, increasing the size of the team does offer an improvement, as it reduces the number of idle thread teams. For non-blocked policies, we observe that increasing the team-size generally results in minimal, if any, improvement in performance. In all cases, the use of blocking provides a more consistent update rate when enough work is introduced.

3.4.4 Shared Memory Algorithms

The line-by-line approach to operator reconstruction suggests that we employ a *hierarchical design*, which consists of thread teams. Rather than employ a fine-grained threading approach over loop indices, we use the coarse-grained, blocked iteration pattern devised in 3.4.3. In this approach, we divide the iteration space into blocks of nearly identical size, and assign one or more blocks to a team of threads. The threads within a given team are then dispatched to one (or more) lines, with vector instructions being used within the lines. As opposed to loop level parallelism, coarse-grained approaches allow one to exploit multiple levels of parallelism, common to many modern CPUs, and load balance the computation across blocks by adjusting the loop scheduling policy. In our implementation, we provide the flexibility of setting the number of threads per block with a macro, but, in general, we let Kokkos choose the appropriate team size using Kokkos::AUTO(). If running on the CPU, this sets the team size to be the number of hyperthreads (if supported) on a given core. For GPU architectures, the team size is the size of the warp.

A hierarchical design pattern is used because the loops in our algorithms are not perfectly nested, i.e., calculations are performed between adjoining loops. Information related to blocking can be precomputed to minimize the number of operations are required to manipulate blocks. The process of subview construction consists of shallow copies involving pointers to vertices of the blocks, so no additional memory is required. With a careful choice of a base block size, one can fit these blocks into high-bandwidth memory, so that accessing costs are reduced. Furthermore, a team-based, hierarchical pattern seems to provide a large degree of flexibility compared to standard loop-level parallelism. In particular, we can fuse adjacent kernels into a single parallel region, which reduces the effect of kernel launch overhead and minimizes the number of synchronization points. The use of a team-type execution policy also allows us to exploit features present on other architectures, such as CUDA's shared memory feature, through scratchpad constructs. Performing a stenciled operation on strided data is associated with an architecture-dependent penalty. On CPUs, while one wishes to operate in a contiguous or cached pattern, various compilers can hide these penalties through optimizations, such as prefetching. GPUs, on the other hand, prefer to operate in a

lock-step fashion. Therefore, if a kernel is not vectorizable, then one pays a significant performance penalty for poor data access patterns. Shared memory, while slower than register accesses, does not require coalesced accesses, so the cost can be significantly reduced. The advantage of using square-like blocks of a fixed size, as opposed to long pencils [3], is that one can adjust the dimensions of the blocks so that they fit into the constraints of the high-bandwidth memory. Moreover, these blocks can be loaded once and can be reused for additional directions, whereas pencils would require numerous transfers, as lines are processed along a given dimension. Such optimizations are not explored in this work, but algorithmic flexibility is something we must emphasize moving forwards.

The parallel nested loop structures, such as the one provided in A.2 (see Scheme A.1) are applied during reconstructions for the local integrals J_* and inverse operators $\mathcal{L}_*^{-1}$, as well as the integrator update. The current exception to this pattern is the convolution algorithm, shown in Scheme A.2, which is also provided in A.2. Here, each thread is responsible for constructing I_* on one or more lines of the grid. Therefore, within a line, each thread performs the convolution sweeps, in serial, using our $O(N)$ algorithm. Adopting the team-tiling approach for this operation requires that we modify our convolution algorithm considerably – this optimization is left to future work. Additionally the benefit of this optimizations is not large for CPUs, as profiling indicated that $< 5\%$ of the total time for a given run was spent inside this kernel. However, this will likely consume more time on GPUs, so this will need to be investigated.

If one wishes to use variable time stepping rules, where the time step is computed from a formula of the form

$$\Delta t = \text{CFL} \min\left(\frac{\Delta x}{c_x}, \frac{\Delta y}{c_y}, \cdots\right), \tag{3.44}$$

then one must supply parallel loop structures with simultaneous maximum reductions for each of the wave speeds c_i. This can be implemented as a custom functor, but the use of blocking/tiling introduces some complexities. More complex reducers that enable such calculations are not

[3] We refer to a pencil as a, generally, long rectangle (in 2-D) and a rectangular prism (in 3-D). The use of pencils, as opposed to square blocks, would require additional precomputing efforts and, possibly, restrictions on the problem size.

currently available. For this reason, problems that use time stepping rules, such as (3.44), are constructed with symmetry in the wave speeds, i.e., $c_x = c_y = \cdots$ to avoid an overly complex implementation with blocking/tiling. However, we plan to revisit this in later work as we begin targeting more general problems. Next, in 3.4.5, we discuss the distributed memory component of the implementation and the strategy used to employ an adaptive time stepping rule, such as (3.44).

3.4.5 Code Strategies for Domain Decomposition

One of the issues with distributed computing involves mapping the problem data in an intelligent way so that it best aligns with the physical hardware. Since the kernels used in our algorithms consume a relatively small amount of time, it is crucial that we minimize the time spent communicating data. Given that these schemes were designed to run on Cartesian meshes, we can use a "topology aware" virtual communicator supplied by MPI libraries. These constructs take a collection of ranks in a communicator (each of which manages a sub-domain) and, if permitted, attempt to reorganize them to best align with the physical hardware. This mapping might not be optimal, since it depends on a variety of factors related to the job allocation and the MPI implementation. Depending on the problem, these tools can greatly improve the performance of an application compared to a hand-coded implementation that uses the standard communicator. Additionally, MPI's Cartesian virtual communicator provides functionality to obtain neighbor references and derive additional communicator groups, say, along rows, columns, etc. The send and receive operations are performed with *persistent communications*, which remove some of the overhead for communication channel creation. Persistent communications require a regular communication channel, which, for our purposes, is simply N-Ns. For more general problems with irregular communication patterns, standard send and receive operations can be used.

One strategy to minimize exposed communications is to use *non-blocking* communications. This allows the programmer to overlap calculations with communications, and is especially beneficial if the application can be written in a staggered fashion. If certain data required for a later calculation is available, then communication can proceed while another calculation is being per-

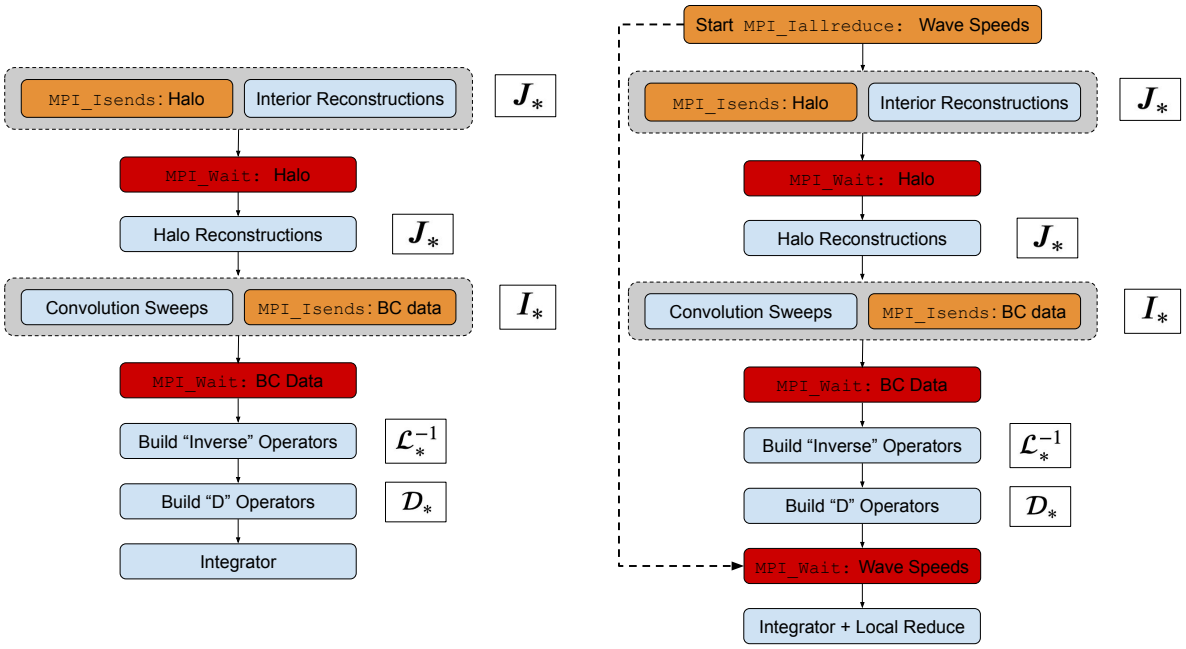


Figure 3.6: Task charts for the domain-decomposition algorithm under fixed (left) and adaptive (right) time stepping rules. The work overlap regions are indicated, laterally, using gray boxes. The work inside the overlap regions should be sufficiently large to hide the communications occuring in the background. To clarify, the overlap in calculations for I_* is achieved by changing the sweeping direction during an exchange of the boundary data. As indicated in the adaptive task chart, the reduction over the "lagged" wave speed data can be performed in the background while building the various operators. Note the use of MPI_WAIT prior to performing the integrator step. This is done to prevent certain overwrite issues during the local reductions in the subsequent integrator step.

formed. Once the data is needed, we can block progress until the message transfer is complete. However, we hope that the calculation done, in between, is sufficient to hide the time spent performing the communication. In the multi-dimensional setting, other operators may be needed, such as ∂_x and ∂_y. However, in our algorithms, directions are not coupled, which allows us to stagger the calculations. So, we can initialize the communications along a given direction and build pieces of other operators in the background.

A typical complication that arises in distributed implementations of PDE solvers concerns the use of various expensive collective operations, such as "all-to-one" and "one-to-all" communications. For implicit methods, these operations occur as part of the iterative method used for solving distributed linear systems. The method employed here is "matrix-free", which eliminates the need

to solve such distributed linear systems. For explicit methods, these operations arise when an adaptive time stepping rule, such as equation (3.44), is employed to ensure that the CFL restriction is satisfied for stability purposes. At each time step, each of the processors, or ranks, must know the maximum wave speeds across the entire simulation domain. On a distributed system, transferring this information requires the use of certain collective operations, which typically have an overall complexity of $O(\log N_p)$, where N_p is the number of processors. While the logarithmic complexity results in a massive reduction of the overall number of steps, these operations use a barrier, in which all progress stops until the operation is completed. This step cannot be avoided for explicit methods, as the most recent information from the solution is required to accurately compute the maximum wave speeds. In contrast, successive convolution methods, do not require this information. However, implementations for schemes developed in e.g., [56, 44, 45] considered "explicit" time stepping rules given by equation (3.44) because they improved the convergence of the approximations. By exploiting the stability properties associated with successive convolution methods, we can eliminate the need for accurate wave speed information, based on the current state, and, instead, use approximations obtained with "lagged" data from the previous time step. We present two, generic, distributed memory task charts in 3.6. The algorithm shown in the left-half of 3.6, which is based on a fixed time step, contains less overall communication, as the local and global reductions for certain information used to compute the time step are no longer necessary. The second version, shown in the right-half of 3.6, illustrates the key steps used in the implementation of an adaptive rule, which can be used for problems with more dynamic quantities (e.g., wave speeds and diffusivity). In contrast to distributed implementations of explicit methods, our adaptive approach allows us to overlap expensive global collective operations (approximately) with the construction of derivative operators, resulting in a more asynchronous algorithm (see Algorithm 3.1).

3.4.6 Some Remarks

In this section, we introduced key aspects that are necessary in developing a performant application. We began with a brief discussion on Kokkos, which is the programming model used for our

Goal: Approximate the global maximum wave speeds $c_x, c_y, \cdots$ using the corresponding "lagged" variables $\tilde{c}_x, \tilde{c}_y, \cdots$.

1: Initialize the c_i's and $\tilde{c}_i$'s via the initial condition (no lag has been introduced)
2: **while** timestepping **do**
3: Update the N-N condition (i.e., (3.25), (3.26), or (3.27)) using the "lagged" wave speeds $\tilde{c}_x, \tilde{c}_y, \cdots$
4: Compute Δt using the "lagged" wave speeds and check the N-N condition
5: Start the MPI_Iallreduce over the local wave speeds $c_x, c_y, \cdots$
6: Construct the spatial derivative operators of interest
7: Post the MPI_WAIT (in case the reductions have not completed)
8: Transfer the global wave speed information to the corresponding lagged variables:

$$\begin{aligned} \tilde{c}_x &\leftarrow c_x, \\ \tilde{c}_y &\leftarrow c_y, \\ &\vdots \end{aligned}$$

9: Perform the update step and computes the local wave speeds $c_x, c_y, \cdots$
10: Return to step 3 to begin the next time step

Algorithm 3.1: Distributed adaptive time stepping rule.

shared memory implementation. Then we introduced one of the metrics, namely (3.43), used to characterize the performance of our parallel algorithms. Using this performance metric, we analyzed a collection of techniques for parallelizing prototypical loop structures in our algorithms. These techniques considered several different approaches to the prescription of parallelism through both naive and complex execution policies. Informed by these results, we chose to adopt a coarse-grained, hierarchical approach that utilizes the extensive capabilities available on modern hardware. In consideration of our future work, this approach also offers a large degree of algorithmic flexibility, which will be essential for moving to GPUs. Finally, we provided some details concerning the implementation of the distributed memory components of the parallel algorithms. We introduced two different approaches: one based on a fixed time step, with minimal communication, and another, which exploits the stability properties of the representations and allows for adaptive time stepping rules. The next section provides numerical results, which demonstrate not only the performance

and scalability of these algorithms, but also their versatility in addressing different PDEs.

3.5 Numerical Results

This section provides the experimental results for our parallel algorithms using MPI and Kokkos, together, with the OpenMP backend. First, in 3.5.1, we define several test problems and verify the rates of convergence for the hybrid algorithms described in sections 3.3 and 3.4. Next, we provide both weak and strong scaling results obtained from each of the example problems discussed in 3.5.1. 3.5.4 provides some insight on issues faced by the distributed memory algorithms, in light of the N-N condition (3.25), which was derived in 3.3.1. Unless otherwise stated, the results presented in this section were obtained using the configurations outlined in 3.2. Timing data presented in Figures A.2, A.3, and A.4 was collected using 10 trials for each configuration (problem size and node count) with the update metric (3.43) being displayed relative to 10^9 DOF/node/s. Each of these trials, evolved the numerical solution over 10 time steps. Error bars, collected from data involving averages, were computed using the sample standard deviation.

3.5.1 Description of Test Problems and Convergence Experiments

Despite the fact that we are primarily focused on developing codes for high-performance applications, we must also ensure that the parallel algorithms produce reliable answers. Here, we demonstrate convergence of the 2-D hybrid parallel algorithms on several test problems, including a nonlinear example that employs the adaptive time stepping rule outlined in 3.1. The convergence results used 9 nodes, with 40 threads per node, assigning 1 MPI rank to each node, for a total of 360 threads. The quadrature method used to construct the local integrals is the fifth-order WENO-quadrature rule, described in 3.2.4, which uses only the linear weights. The numerical solution, in each of the examples, remains smooth over the corresponding time interval of interest. Therefore, it is not necessary to transform the linear quadrature weights to nonlinear ones. According to the analysis of the truncation error presented in [56, 44], retaining a single term in the partial sums for $\mathcal{D}_*$ should yield a first-order convergence rate, depending on the choice of α. Convergence results

CPU Type	Intel Xeon Gold 6148
C++ Compiler	ICC 2019.03
MPI Library	Intel MPI 2019.3.199
Optimization Flags	-O3 -xCORE-AVX512 -qopt-zmm-usage=high -qno-opt-prefetch
Thread Bindings	OMP_PROC_BIND=close, OMP_PLACES=threads
Team Size	Kokkos::AUTO()
Base Block Size	256^2
CFL	1.0
$\boldsymbol{\beta}$	1.0
$\boldsymbol{\epsilon}$	1×10^{-16}

Table 3.2: Architecture and code configuration for the numerical experiments conducted on the Intel 18 cluster at Michigan State University's Institute for Cyber-Enabled Research. As with the loop experiments in 3.4.3, we encourage the compiler to use AVX-512 instructions and avoid the use of prefetching. All available threads within the node (40 threads/node) were used in the experiments. Each node consists of two Intel Xeon Gold 6148 CPUs and at least 83 GB of memory. We wish to note that hyperthreading is not supported on this system. As mentioned in 3.4.3, when hyperthreading is not enabled, Kokkos::AUTO() defaults to a team size of 1. In cases where the base block size did not divide the problem evenly, this parameter was adjusted to ensure that blocks were nearly identical in size. The parameter β, which does not depend on Δt is used in the definition of α. For details on the range of admissible β values, we refer the reader to [56, 44], where this parameter was introduced. Lastly, recall that ϵ is the tolerance used in the NN constraints.

for each of the three test problems defined in sections 3.5.1, 3.5.1, and 3.5.1 are provided in 3.7.

Example 1: Linear Advection Equation

The first test problem considered in this work is the 2D linear advection equation

$$\partial_t u + \partial_x u + \partial_y u = 0, \quad (x, y) \in [0, 2\pi]^2,$$
$$u_0(x, y) = \frac{1}{4}(1 - \cos(x))(1 - \cos(y)),$$
s.t. two-way periodic BCs.

We evolve the numerical solution to the final time $T = 2\pi$. In the experiments, we used the same number of mesh points in both directions, with $\alpha_x = \alpha_y = \beta/\Delta t$, with $\beta = 1$. While this problem

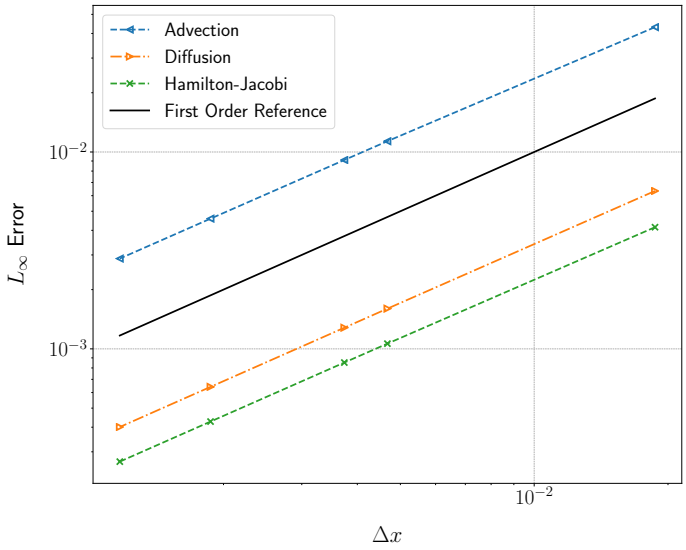


Figure 3.7: Convergence results for each of the 2-D example problems. Results were obtained using 9 MPI ranks with 40 threads/node. Also included is a first-order reference line (solid black). Our convergence results indicate first-order accuracy resulting from the low-order temporal discretization. The final reported L_∞ errors for each of the applications, on a grid containing 5277^2 total zones, are 2.874×10^{-3} (advection), 4.010×10^{-4} (diffusion), and 2.674×10^{-4} (H-J).

is rather simple and does not highlight many of the important features of our algorithm, it is nearly identical to the code for a nonlinear example. For initial experiments, a simple test problem is preferable because it gives more control over quantities which are typically dynamic, such as wave speeds. Moreover, the error can be easily computed from the exact solution

$$u(x, y, t) = u_0(x - t, y - t).$$

Example 2: Linear Diffusion Equation

The next test problem that we consider is the linear diffusion equation

$$\partial_t u = \partial_{xx} u + \partial_{yy} u, \quad (x, y) \in [0, 2\pi]^2,$$

$$u_0(x, y) = \sin(x) + \sin(y),$$

s.t. two-way periodic BCs.

The numerical solution is evolved from $(0, T]$, with $T = 1$, in order to prevent substantial decay. As with the previous example, we use an equal number of mesh points in both directions, so that

$\Delta x = \Delta y$. The fixed time stepping rule was used, with $\Delta t = \Delta x$. Compared with the previous example, we used the parameter definitions $\alpha_x = \alpha_y = 1/\sqrt{\Delta t}$, which corresponds to $\beta = 1$ in the definition for second derivative operators. The exact solution for this problem is given by

$$u(x, y, t) = e^{-t}\Big(\sin(x) + \sin(y)\Big).$$

Characteristically, this example is different from the advection equation in the previous example, which allows us to illustrate some key features of the method. Firstly, code developed for advection operators can be reused to build diffusion operators, an observation made in 3.2.1. More specifically, to construct the left and right-moving local integrals, we used the same linear WENO quadrature as with the advection equation in Example 1. However, we note that this particular example could, instead, use a more compact quadrature to eliminate the halo communication, which would remove a potential synchronization point. The second feature concerns the time-to-solution and is related to the unconditional stability of the method. Linear diffusion equations, when solved by an explicit method, are known to incur a harsh stability restriction on the time step, namely, $\Delta t \sim \Delta x^2$, making long-time simulations prohibitively expensive. The implicit aspect of this method drastically reduces the time-to-solution, as one can now select time steps which are, for example, proportional to the mesh spacing. This benefit is further emphasized by the overall speed of the method, which can be observed in sections 3.5.2 and 3.5.3.

Example 3: Nonlinear Hamilton-Jacobi Equation

The last test problem we consider is the nonlinear H-J equation

$$\partial_t u + \frac{1}{2}\left(1 + \partial_x u + \partial_y u\right)^2 = 0, \quad (x, y) \in [0, 2\pi]^2,$$
$$u_0(x, y) = 0,$$
s.t. two-way periodic BCs.

To prevent the characteristic curves from crossing, which would lead to jumps in the derivatives of the function u, the numerical solution is tracked over a short time, i.e., $T = 0.5$. We applied a

high-order linear WENO quadrature rule to approximate the left and right-moving local integrals and used the same parameter choices for α_x, α_y, and β, as with the advection equation in Example 1. However, since the wave speeds fluctuate based on the behavior of the solution u, we allow the time step to vary according to (3.44), which requires the use of the distributed adaptive time stepping rule outlined in 3.1.

Typically, an exact solution is not available for such problems. Therefore, to test the convergence of the method, we use a manufactured solution given by

$$u(x, y, t) = t\Big(\sin(x) + \sin(y)\Big),$$

with a corresponding source term included on the right-hand side of the equation. Methods employed to solve this class of problems are typically explicit, with a shock-capturing method being used to handle the appearance of "cusps" that would otherwise lead to jumps in the derivative of the solution. A brief summary of such methods is provided in our recent paper [45], where extensions of successive convolution were developed for curvilinear and non-uniform grids. The method follows the same structural format as an explicit method with the ability to take larger time steps as in an implicit method. However, the explicit-like structure of this method does not require iteration for nonlinear terms and allows for a more straightforward coupling with high-order shock-capturing methods. We wish to emphasize that despite the fact that this example is nonlinear, the only major mathematical difference with Example 1 is the evaluation of a different Hamiltonian function.

3.5.2 Weak Scaling Experiments

A useful performance property for examining the scalability of parallel algorithms describes how they behave when the compute resources are chosen proportionally to the size of the problem. Here, the amount of work per compute unit remains fixed, and the compute units are allowed to fluctuate. Weak scaling assumes ideal or best-case performance for the parallel components of algorithms and ignores the influence of bottlenecks imposed by the sequential components of a code. Therefore,

for N compute units, we shall expect a speedup of N. This motivates the following definitions for speedup and efficiency in the context of weak scaling:

$$S_N = \frac{NT_1}{T_N}, \quad E_N = \frac{S_N}{N} \equiv \frac{T_1}{T_N}.$$

Therefore, with weak scaling, ideal performance is achieved when the run times for a fixed work size (or, equivalently, the DOF/node/s) remain constant, as we vary the compute units. To scale the problem size, we take advantage of the periodicity for the test problems. The base problem on $[0, 2\pi] \times [0, 2\pi]$ can be replicated across nodes so that the total work per node remains constant. Provided in A.2 are plots of the weak scaling data — specifically the update metric 3.43 and the corresponding efficiency — obtained from the fastest of 10 trials of each configuration, using up to 49 nodes (1,960 cores).

These results generally indicate good performance, both in terms of the update frequency and efficiency, for a variety of problem sizes. Weak scalability appears to be excellent up to 16 nodes (640 cores), then begins to decline, most likely due to network effects. The performance behavior for advection and diffusion applications is quite similar, which is to be expected, since the parallel algorithms used to construct the base operators are nearly identical. With regard to the Hamilton-Jacobi application, we see that the performance is similar to the other applications at larger node counts. This seems to indicate that no major communication penalties are incurred by use of the adaptive time stepping method shown in 3.1, compared to fixed time stepping. Additionally, in the Hamilton-Jacobi application, we observe a sharp decline in the performance at 9 nodes in A.2. A closer investigation reveals that this is likely an artifact of the job scheduler for the system on which the experiments were conducted, as we were unable to secure a “contiguous" allocation of nodes. This has the unfortunate consequence of not being able to guarantee that data for a particular trial remain in close *physical* proximity. This could result in issues such as network contention and delays that exacerbate the cost of communication relative to the computation as discussed in 3.4. An non-contiguous placement of data is problematic for codes with inexpensive operations, such as the methods shown here, because the work may be insufficient to hide this increased cost of communication. For this reason, we chose to include plots containing the averaged weak

scaling data in A.2, which contains error bars calculated from the sample standard deviation. The noticeable size of the error bars in these plots generally indicates a large degree of variation in the timings collected from trials.

To more closely examine the importance of data proximity on the nodes, we repeated the weak scaling study, but with node counts for which a contiguous allocation count could be guaranteed. We have provided results for the fastest and averaged data in A.3. Data collected from the fastest trials indicates nearly perfect weak scaling, across all applications, up to 9 nodes, with a consistent update rate between $2 - 4 \times 10^8$ DOF/node/s. For convenience, these results were plotted with the same markers and formats so that results from the larger experiments in A.2 could be compared directly. A comparison of the fastest timings between the large and small runs supports our claim that data proximity is crucial to achieving the peak performance of the code. Furthermore, the error bars for the contiguous experiments displayed in A.3 show that the individual trials exhibit less overall variation in timings.

3.5.3 Strong Scaling Experiments

Another form of scalability considers a fixed problem size and examines the effect of varying the number of work units used to find the solution. In these experiments, we allow the work per compute unit to decrease, which helps identify regimes where sequential bottlenecks in algorithms become problematic, provided we are granted enough resources. Applications which are said to strong scale exhibit run times which are inversely proportional to the number of resources used. For example, when N compute units are applied to a problem, one expects the run to be N times faster than with a single compute unit. Additionally, if an algorithm's performance is memory bound, rather than compute bound, this will, at some point, become apparent in these experiments. Supplying additional compute units should not improve performance, if more time is spent fetching data, rather than performing useful computations. This motivates the following definitions for speedup and efficiency in the context of strong scaling:

$$S_N = \frac{T_1}{T_N}, \quad E_N = \frac{S_N}{N}.$$

Here, N is the number of nodes used, so that T_1 and T_N correspond to the time measured using a single node and N nodes, respectively. Results of our strong scaling experiments are provided in A.4. As with the weak scaling experiments, we have plotted the update metric 3.43 along with the strong scaling efficiency using both the fastest and averaged configuration data from a set of 10 trials. In contrast to weak scaling, strong scaling does not assume ideal speedup, so one could plot this information; however, the information can be ascertained from the efficiency data, so we refrain from plotting this data.

Results from these experiments show decent strong scalability for the N-N method. This method does not contain a substantial amount of work, so we do not expect good performance for smaller base problem sizes, as the work per node becomes insufficient to hide the cost of communication. On the other hand, larger base problem sizes, which introduce more work, are capable of saturating the resources, but will at some point become insufficient. This behavior is apparent in our efficiency plots. Increasing the problem size generally results in an improvement of the efficiency and speedup for the method. Part of these problems can be attributed to the use of a blocking pattern for loops structures discussed in 3.4.4. Depending on the size of the mesh, it may be the case that the block size and the team size set by the user result in idle threads. One possible improvement is to simply increase the team size so that there are fewer idle threads within an MPI task. Alternatively, one can adjust the number of threads per task, so that each task is responsible for fewer threads. While these approaches can be implemented with no changes to the code, they will likely not resolve this issue. Profiling seems to indicate that the source of the problem is the low arithmetic intensity of the reconstruction algorithms. In other words, the method is memory bound because the calculations required in the reconstructions are inexpensive relative to the cost of retrieving data from memory. As part of our future work, we plan to investigate such limitations through the use of detailed roofline models. We also plan to consider test problems in 3-D, which will introduce additional work.

3.5.4 Effect of CFL

In order to enforce a N-N dependency for our domain decomposition algorithm, we obtained several possible restrictions on Δt, depending on the problem and the choice of α. In the case of linear advection, we would, for example, require that

$$\Delta t \leq -\frac{\beta L_m}{c_{\max} \log(\epsilon)}.$$

with the largest possible time step permitting N-N dependencies being set by the equality. Admittedly, such a restriction is undesirable. As mentioned in 3.3.1, this assumption can be problematic if the problem admits fast waves ($c_{\max}$ is large) and/or if the block sizes are particularly small (L_m is small). In many applications, the former circumstance is quite common. However, our test problem contains fixed wave speeds so this is less of an issue. The latter condition is a concern for configurations which use many blocks, such as a large simulation on many nodes of a cluster. Another potential circumstance is related to the granularity of the blocks. For example, in these experiments, we use 1 MPI rank per compute node. However, it may be advantageous to consider different task configurations, e.g., using 1 (or more) rank(s) per NUMA region of a compute node.

A larger CFL parameter is generally preferable because it reduces the overall time-to-solution. Eventually, however, for a given CFL, there will be a crossover point, where the time step restriction causes the performance to drop due to the increasing number of *sequential* time steps. This experiment used a highly refined grid and varied the CFL number, using up to 9 nodes. Results from the CFL experiments are provided in 3.8. The data was obtained using an older version of our parallel algorithms, compiled with GCC 8.2.0-2.31.1, which does not use blocking. By plotting the behavior according to the number of nodes (ranks) used, we can fix the lengths of the blocks, hence L_m, and change the time step to identify the breakdown region. We observe a substantial decrease in performance for the 9 node configuration, specifically when the CFL number increases from 4 to 5. For more complex problems with dynamic wave behavior, this breakdown may be observed earlier. In response to this behavior, a user could simply increase or relax the tolerance, but the logarithm tends to suppress the impact of large relaxations. Another option,

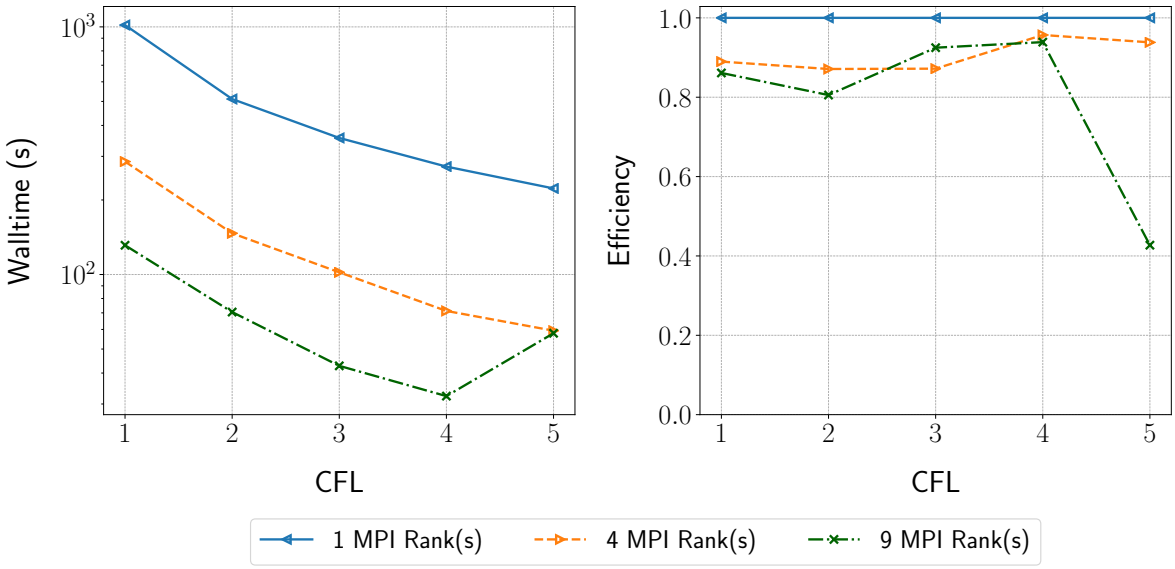


Figure 3.8: Results on the N-N method for the linear advection equation using a fixed mesh with 5377^2 total DOF and a variable CFL number. In each case, we used the fastest time-to-solution collected from repeating each configuration a total of 20 times. This particular data was collected using an older version of the code, compiled with GCC, which did not use the blocking approach. For larger block sizes, increasing the CFL has a noticeable improvement on the run time, but as the block sizes become smaller, the gains diminish. For example, if 9 MPI ranks are used, improvements are observed as long as CFL ≤ 4. However, when CFL = 5, the run times begin to increase, with a significant decrease in efficiency. As the blocks become smaller, Δt needs to be adjusted (decreased) so that the support of the non-local convolution data not extend beyond N-Ns.

which shall be considered in future work is to include more information from neighboring ranks by either eliminating this condition or, at the least, communicating enough information to achieve a prescribed tolerance.

3.6 Conclusion

In this paper, we presented hybrid parallel algorithms capable of addressing a wide class of both linear and nonlinear PDEs. To enable parallel simulations on distributed systems, we derived a set of conditions that use available wave speed (and/or diffusivity) information, along with the size of the sub-domains, to limit the communication through an adjustment of the time step. Although not considered here, these conditions, which are needed to ensure accuracy, rather than the stability of the method, can be removed at the cost of additional communication. Using these restrictions, boundary conditions are enforced across sub-domains in the decomposition. Results were obtained

for 2-D examples consisting of linear advection, linear diffusion, and a nonlinear H-J equation to highlight the versatility of the methods in addressing characteristically different PDEs.

As part of the implementation, we used constructs from the Kokkos performance portability library to parallelize the shared memory components of the algorithms. We extracted essential loop structures from the algorithms and analyzed a variety of parallel execution policies in an effort to develop an efficient application. These experiments considered several common optimization techniques, such as vectorization, cache-blocking, and placement of threads. From these experiments, we chose to use a blocked iteration pattern in which threads (or teams thereof) are mapped to blocks of an array with vector instructions being applied to 1-D line segments. These design choices offer a large degree of flexibility, which is an important consideration as we proceed with experimentation on other architectures, including GPUs, to leverage the full capabilities provided by Kokkos. By exploiting the stability properties of the representations, we also developed an adaptive time stepping method for distributed systems that uses “lagged" wave speed information to calculate the time step. While the methods presented here do not require adaptive time stepping for stability, it was included as an option because of its ability to prevent excessive numerical diffusion, as observed in previous work.

Convergence and scaling properties for the hybrid algorithms were established using at most 49 nodes (1960 cores), with a peak performance $> 10^8$ DOF/node/s. Larger weak scaling experiments, which used up to 49 nodes (1960 cores), initially performed reasonably well, with all applications later tending to 60% efficiency corresponding (roughly) to 2×10^8 DOF/node/s. While some performance loss is to be expected from network related complications, we found this to be much larger than what was observed in prior experiments. Later, it was discovered that the request for a contiguous allocation could not be accommodated so data locality in the experiments was compromised. By repeating the experiments on a smaller collection of nodes, which granted this request, we discovered that data locality plays a pivotal role in the overall performance of the method. We observe that a large base problem size is required to achieve good strong scaling. Furthermore, when threads are prescribed work at a coarse granularity (i.e., across blocks, rather

than entries within the blocks), one must ensure that the problem size is capable of saturating the resources to avoid idle threads. This approach introduces further complications for strong scaling, as the workload per node drops substantially while the block size remains fixed. Finite difference methods which, generally, do not generate a substantial amount of work, are quite similar to successive convolution. Therefore, we do not expect excellent strong scalability. Certain aspects of the algorithms can be tuned to improve the arithmetic intensity, which will improve the strong scaling behavior. At some point, the algorithms will be limited by the speed of memory transfers rather than computation. We also provided experimental results that demonstrate the limitations of the N-N condition in the context of strong scaling.

While we have presented several new ideas with this work, there is still much untapped potential with successive convolution methods. Firstly, optimizations on GPU architectures, which shall play an integral role in the upcoming exascale era, need to be explored and compared with CPUs. A roofline model should be developed for these algorithms to help identify key limitations and bottlenecks and formulate possible solutions. Although this work considered only first-order time discretizations, our future developments shall be concerned with evaluating a variety of high-order time discretization techniques in an effort to increase the efficiency of the method. Lastly, the parallel algorithms should be modified to enable the possibility of mesh adaptivity, which is a common feature offered by many state-of-the-art computing libraries.

CHAPTER 4

DEVELOPING A PARTICLE-IN-CELL METHOD

4.1 Introduction

This chapter concerns the development of a particle-in-cell (PIC) method for plasmas that is constructed using solvers for fields developed in the preceding chapters. We begin by introducing the concept of a macro-particle that is the foundation of all PIC methods in section 4.2. Next, in section 4.3, we discuss techniques used in our methods to enforce the involutions of the computational model. We then discuss the methods employed to advance the particle data, including standard integrators, which can be found in section 4.4, as well as a more recent integrators developed for problems with so-called non-separable Hamiltonians in section 4.5. In the sections on integrators, we propose modifications which allow for a coupling of these methods to the proposed field solvers. Numerical examples which demonstrate the performance of the proposed methods on several plasma test problems are presented in section 4.6. We provide a summary of our key findings in section 4.7.

4.2 Moving from Point-particles to Macro-particles

One of the essential features of PIC methods is that the simulation particles are not physical particles. Instead, they represent a sample of particles collected from an underlying distribution function. For this reason, they are often called super-particles or macro-particles. Moreover, the "size" of this sample is reflected in the weight associated with a given macro-particle w_{mp}, which can be calculated as

$$w_{mp} = \frac{N_{\text{real}}}{N_{\text{simulation}}}.$$

Here, we use N_{real} to denote the number of physical particles contained within a simulation domain and $N_{\text{simulation}}$ to be the number of simulation particles. The calculation of N_{real} is problem dependent, but can be expressed in terms of the average macroscopic number density $\bar{n}$ that

describes the plasma and a volume associated with either the domain or beam being considered. Once the weight for each particle is calculated it can be absorbed into properties of the particle species, e.g., the charge, so that $w_{mp}q_i$ is written as q_i.

In section 1.2.2, the charge density and current density were defined in equations (1.22) and (1.23) using a linear combination of Dirac delta distributions for a collection of N_p simulation particles. While PIC methods can certainly be developed to work with these point-particle representations, most PIC methods, including the ones developed in this work, represent particles using shape functions so that

$$\rho\left(\mathbf{x},t\right)=\sum_{p=1}^{N_p}q_pS\left(\mathbf{x}-\mathbf{x}_p(t)\right), \tag{4.1}$$

$$\mathbf{J}\left(\mathbf{x},t\right)=\sum_{p=1}^{N_p}q_p\mathbf{v}_pS\left(\mathbf{x}-\mathbf{x}_p(t)\right), \tag{4.2}$$

where the shape function S is now used to represent a simulation particle. The shape functions most often employed in PIC simulations are B-splines, which are compact (local), positive, and satisfy the partition of unity property. Furthermore, they can be easily extended to include additional dimensions using tensor products of univariant splines. While higher-order splines produce smoother mappings to the mesh and possess higher degrees of continuity, the extended support regions create complications for plasmas on bounded domains. The methods developed in this work employ linear splines to represent particle shapes. The linear spline function that represents the particle x_p on the mesh with spacing Δx is given by

$$S(x-x_p)=\begin{cases}1-\dfrac{|x-x_p|}{\Delta x}, & 0\le\dfrac{|x-x_p|}{\Delta x}\le 1,\\[2ex] 0, & \dfrac{|x-x_p|}{\Delta x}>1.\end{cases} \tag{4.3}$$

The shape function (4.3) generally serves two purposes: (1) It provides a way to map particle data onto the mesh (scatter operation) and (2) can be used to interpolate mesh based quantities to the particles during the time integration (gather operation). For consistency in a PIC method, it is important that these maps be identical.

In the next section, we address the issue of enforcing charge conservation with the proposed formulation. To this end, we discuss several approaches, including divergence cleaning methods, as well as a modification of the usual PIC charge mapping (4.1) that solves the continuity equation.

4.3 Methods for Controlling Divergence Errors

The formulation adopted in this work considers formulations of Maxwell's equations in the Lorenz gauge (1.8)-(1.10) as well as the Coulomb gauge (1.12)-(1.14). The benefit of adopting a gauge formulation is that the involution for the magnetic field

$$\nabla \cdot \mathbf{B} = 0,$$

will be trivially satisfied, since $\mathbf{B} = \nabla \times \mathbf{A}$; however, it is important to recognize that this formulation is over-determined only makes sense as long as the particular gauge condition (either (1.10) or (1.14)) is satisfied. The issue of enforcing the gauge condition is ultimately connected to charge conservation through the involution $\nabla \cdot \mathbf{E} = \rho/\epsilon_0$. When this condition is not satisfied or properly controlled in a numerical scheme, the solutions can become non-physical. In [19], the authors proposed techniques for controlling the growth of errors in Gauss' law, and demonstrated the usual behavior that results when these conditions are not properly enforced in PIC methods. Even if a Yee mesh [13] is used to represent the fields, this condition is not guaranteed to be satisfied in PIC codes due to errors introduced through the particle-to-mesh mappings [77]. In an effort to control this error, we examine two general classes of methods: (1) divergence cleaning through an auxiliary equation and (2) an analytical charge map for particles that is based on the continuity equation. The methods contained in the first class perform the cleaning through field solvers, which require a Poisson solve or a wave solve, the latter of which can be performed with the methods introduced in Chapter 2. The method in the second class enforces charge conservation through a path integral of the particle trajectories and is an analytical extension of the map presented in [25] for FEM-PIC. In contrast to [25], which computed the map numerically via quadrature, the map presented in this work is obtained by analytically integrating and differentiating the shape functions used to represent particles.

4.3.1 A Classic Elliptic Projection Method Based on Gauss' Law

In PIC methods based on the Boris method [4], which evolve particles using **E** and **B** fields, one can use a classic elliptic projection technique to reduce the errors in Gauss' law [2, 20, 77]. The idea is that one would like to enforce Gauss' law

$$\nabla \cdot \mathbf{E} = \frac{1}{\sigma_1}\rho.$$

During the simulation, a build up in violations of the continuity equation introduces certain errors that change the irrotational part of the electric field. If we let $\mathbf{E}^*$ denote the electric field computed with a numerical method, we can see that these violations take the form

$$\mathbf{E} = \mathbf{E}^* - \nabla\psi_E, \tag{4.4}$$

where ψ_E is some scalar function that should be determined. Taking the divergence of both sides and using Gauss' law, we obtain the elliptic equation

$$-\Delta\psi_E = \frac{1}{\sigma_1}\rho - \nabla \cdot \mathbf{E}^*. \tag{4.5}$$

The divergence term for the numerical electric field requires some form of a numerical derivative obtained using a difference approximation or a basis. If we assuming that the error is zero at the boundary, which is a realistic assumption, this equation can be solved with homogeneous Dirichlet boundary conditions. One then takes a discrete gradient, and corrects the numerical electric field using (4.4).

For the case in which fields are expressed in terms of potentials, recall that the electric field is calculated as

$$\mathbf{E} = -\nabla\psi - \frac{\partial \mathbf{A}}{\partial t},$$

which is valid for any gauge. After solving equation (4.5), we can correct the scalar potential and its gradient according to

$$\psi = \psi^* + \psi_E, \quad \nabla\psi = \nabla\psi^* + \nabla\psi_E,$$

where we have, again, used "*" to denote the initial approximation.

4.3.2 Elliptic Divergence Cleaning based on Potentials

Another elliptic divergence cleaning method was developed in the thesis [42] and also appeared in [41] for PIC simulations of the VM system in the Lorenz gauge. The idea is to construct an elliptic equation by combining Gauss's law with the electric field, which is expressed in terms of the potentials. In other words, we can substitute the definition

$$\mathbf{E} = -\nabla\psi - \frac{\partial \mathbf{A}}{\partial t},$$

into Gauss' law to arrive at the elliptic equation

$$-\Delta\psi = \frac{1}{\sigma_1}\rho + \frac{\partial\left(\nabla \cdot \mathbf{A}\right)}{\partial t}.$$

This equation was solved using a second-order centered finite-difference approximation of the Laplacian, and it was shown that a discrete form of Gauss' law will be satisfied as long as a staggered grid is used for the mesh data. This mesh staggering, which is problem dependent, is different from the more commonly used Yee method [13] and is largely motivated by the difference scheme used to approximate the Laplacian.

4.3.3 Enforcing the Lorenz Gauge through Lagrange Multipliers

In this section, we introduce a Lagrange multiplier to enable the enforcement of the Lorenz gauge condition (1.10) following [19], which proposed a divergence cleaning technique for the E-B form of Maxwell's equations using a hyperbolic model. The approach presented in this section considers the same system written in terms of the Lorenz gauge. To this end, we modify the Lorenz gauge condition by introducing a function ϕ which represents a residual:

$$\frac{\phi}{d^2} = \nabla \cdot \mathbf{A} + \frac{1}{\kappa^2}\frac{\partial \psi}{\partial t}. \tag{4.6}$$

As we will see, this function ϕ satisfies a wave equation whose purpose is to sweep away the residual in the Lorenz gauge. The non-physical parameter $d > 1$ is connected to the wave speed for ϕ and is selected to ensure that waves for ϕ propagate faster than the other waves of interest in

the problem. This choice is often problem dependent, but $d \approx 10$ is often used [19, 20, 21]. We consider a modification of the original system (1.41)-(1.43), which has the form

$$\begin{aligned}
\frac{1}{\kappa^2}\frac{\partial^2\psi}{\partial t^2} - \Delta\psi &= \frac{1}{\sigma_1}\rho, \\
\frac{1}{\kappa^2}\frac{\partial^2\mathbf{A}}{\partial t^2} - \Delta\mathbf{A} - \nabla\phi &= \sigma_2\mathbf{J}, \\
\nabla\cdot\mathbf{A} + \frac{1}{\kappa^2}\frac{\partial\psi}{\partial t} &= \frac{1}{d^2}\phi.
\end{aligned}$$

We have left the particle equations unspecified for generality. Observe that this modified system is equivalent to the original system when $\phi \equiv 0$. To develop the hyperbolic auxiliary equation, we first create a coupling between the wave equation for the scalar potential ψ and the modified gauge condition (4.6). This is accomplished by substituting $\frac{1}{\kappa^2}\partial_t\psi$ from the latter to the former to obtain

$$\frac{1}{d^2}\frac{\partial\phi}{\partial t} - \nabla\cdot\left(\frac{\partial\mathbf{A}}{\partial t}\right) - \Delta\psi = \frac{1}{\sigma_1}\rho,$$

We then take a time derivative of this equation to obtain

$$\frac{1}{d^2}\frac{\partial^2\phi}{\partial t^2} - \nabla\cdot\left(\frac{\partial^2\mathbf{A}}{\partial t^2}\right) - \Delta\left(\frac{\partial\psi}{\partial t}\right) = \frac{1}{\sigma_1}\frac{\partial\rho}{\partial t}.$$

Using the wave equation for $\mathbf{A}$ for the second time derivative in the above equation, we obtain, after rearranging the terms and with the aid of (4.6), a wave equation for ϕ, namely

$$\frac{1}{\kappa^2 d^2}\frac{\partial^2\phi}{\partial t^2} - \left(1 + \frac{1}{d^2}\right)\Delta\phi = \sigma_2\left(\frac{\partial\rho}{\partial t} + \nabla\cdot\mathbf{J}\right).$$

Hence, the system to be solved is

$$\begin{aligned}
\frac{1}{\kappa^2}\frac{\partial^2\psi}{\partial t^2} - \Delta\psi &= \frac{1}{\sigma_1}\rho, \\
\frac{1}{\kappa^2}\frac{\partial^2\mathbf{A}}{\partial t^2} - \Delta\mathbf{A} - \nabla\phi &= \sigma_2\mathbf{J}, \\
\frac{1}{\kappa^2 d^2}\frac{\partial^2\phi}{\partial t^2} - \left(1 + \frac{1}{d^2}\right)\Delta\phi &= \sigma_2\left(\frac{\partial\rho}{\partial t} + \nabla\cdot\mathbf{J}\right),
\end{aligned}$$

where the new wave equation for ϕ is used to enforce the gauge condition. Since the function ϕ represents the gauge error, which should be "swept away", it is prescribed outflow boundary

conditions. Initially, the data for the potentials should satisfy the gauge condition, so the initial data is $\phi(\mathbf{x}, 0) = 0$. Furthermore, notice that the source in this auxiliary equation contains

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{J},$$

which describes the evolution of charge in domain. If the charge in the problem is conserved, then

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{J} = 0,$$

and the gauge condition will be naturally satisfied. Otherwise, local violations of this constraint act as sources that generate waves for the residual function ϕ.

4.3.4 Enforcing the Coulomb Gauge

The Coulomb gauge (1.47) can be enforced using a projection technique that is nearly identical to the one discussed in section 4.3.1 that enforces Gauss' law. Any errors in the gauge condition manifest through irrotational components, which need to be removed. To this end, we apply a Helmholz decomposition to the numerical vector potential (indicated with "*")

$$\mathbf{A}^* = \mathbf{A}_{\text{rot}} + \mathbf{A}_{\text{irrot}}.$$

If the vector potential is purely rotational then it is the curl of another function, so naturally

$$\nabla \cdot \mathbf{A}_{\text{rot}} = 0.$$

The irrotational or curl-free part of the vector potential can be expressed as the gradient of a scalar function, so the decomposition for the vector potential is given by

$$\mathbf{A}^* = \mathbf{A}_{\text{rot}} - \nabla \eta, \tag{4.7}$$

where η is some scalar function. Taking the divergence of the equation (4.7), we see that

$$-\Delta \eta = \nabla \cdot \mathbf{A}^*.$$

After solving this equation for η and taking its derivative, we can obtain the rotational part of the vector potential by rearranging (4.7) so that

$$\mathbf{A}_{\text{rot}} = \mathbf{A}^* + \nabla\eta. \tag{4.8}$$

If we follow our formulation (1.45)-(1.47), the $\mathbf{A}^*$ will already be an approximation of the rotational part of the vector potential. Moreover, the rotational part of the wave equation (1.46) (i.e., (1.17)) can be evolved using the field solvers developed in chapter 2, which means that we can also compute these derivatives as well.

4.3.5 Analytical Maps for Enforcing Charge Conservation

A new approach to enforcing charge conservation was recently proposed in [25] in the context of finite-element particle-in-cell methods. The map for the charge density is based on the continuity equation, which is integrated in time:

$$\rho^{n+1} = \rho^n - \int_{t^n}^{t^{n+1}} \nabla \cdot \mathbf{J}\, dt. \tag{4.9}$$

Their map exchanges the time integral (4.9) for a spatial integral that effectively traces the motion of the particle on the mesh from t^n to t^{n+1}. In [25] and [26], this mapping was enforced in a weak sense through the finite element basis. Here, we shall extend this technique a bit further to devise a discrete analogue to (4.9), which can be computed analytically and enforces charge conservation to machine precision. We summarize the key identities presented in the paper [26], offering additional details which, we believe, make the method easier to understand.

We begin with the usual definitions for the charge density and current density, which were defined in equations (4.1) and (4.2) in terms of a shape function $S(\mathbf{x})$ as

$$\rho\left(\mathbf{x}, t\right) = \sum_{p=1}^{N_p} q_p S\left(\mathbf{x} - \mathbf{x}_p(t)\right),$$

$$\mathbf{J}\left(\mathbf{x}, t\right) = \sum_{p=1}^{N_p} q_p \mathbf{v}_p S\left(\mathbf{x} - \mathbf{x}_p(t)\right).$$

Again, N_p is the total number of macro-particles in the simulation, so that q_p now represents the charge of a macro-particle. Additionally, the position at time t for a given particle can be expressed in terms of its velocity using

$$\mathbf{x}_p(t) = \mathbf{x}_p(0) + \int_0^t \mathbf{v}_p(\tau)\, d\tau. \tag{4.10}$$

Various distributional identities are presented in [26]. In particular, they make repeated use of the Dirac delta distribution, which is defined in Fourier space as

$$\delta\left(\mathbf{x}-\mathbf{a}\right) = \frac{1}{(2\pi)^3}\int_{-\infty}^{\infty} e^{i\mathbf{k}\cdot(\mathbf{x}-\mathbf{a})}\, d^3\mathbf{k}. \tag{4.11}$$

Additionally, the delta distribution obeys the "sifting" properties

$$f\left(\mathbf{x}\right) = \int_{-\infty}^{\infty} f\left(\mathbf{y}\right)\delta\left(\mathbf{x}-\mathbf{y}\right)\, d^3\mathbf{y}. \tag{4.12}$$

and

$$\delta\left(\boldsymbol{\eta}-\boldsymbol{\xi}\right) = \int_{-\infty}^{\infty} \delta\left(\boldsymbol{\eta}-\mathbf{y}\right)\delta\left(\mathbf{y}-\boldsymbol{\xi}\right)\, d^3\mathbf{y}, \tag{4.13}$$

which shall be used in the discussion that follows.

The first identity, given as equation (10) in [26], is

$$\delta\left(\mathbf{x}-\mathbf{x}_p(t)\right) = \delta\left(\mathbf{x}-\mathbf{x}_p(0)\right) * \delta\left(\mathbf{x}-\int_0^t \mathbf{v}_p\left(\tau\right)\, d\tau\right), \tag{4.14}$$

where we use "$*$" to denote convolution. To see the equivalence, use the definition (4.10) to convert the particle position to velocity

$$\delta\left(\mathbf{x}-\mathbf{x}_p\right) = \delta\left(\mathbf{x}-\mathbf{x}_p(0)-\int_0^t \mathbf{v}_p(\tau)\, d\tau\right),$$

then appeal to identity (4.13), taking $\boldsymbol{\eta} = \int_0^t \mathbf{v}_i(\tau)\, d\tau$ and $\boldsymbol{\xi} = \mathbf{x}-\mathbf{x}_p(0)$ to obtain

$$\int_{-\infty}^{\infty} \delta\left(\mathbf{y}-\int_0^t \mathbf{v}_p(\tau)\, d\tau\right)\delta\left(\mathbf{x}-\mathbf{x}_p(0)-\mathbf{y}\right)\, d^3\mathbf{y} \equiv \delta\left(\mathbf{x}-\mathbf{x}_p(0)\right) * \delta\left(\mathbf{x}-\int_0^t \mathbf{v}_p\left(\tau\right)\, d\tau\right).$$

The next identity relates time and space derivatives of the delta distribution and is given by (see equation (11) in [26]) as

$$\partial_t \delta\left(\mathbf{x}-\mathbf{x}_p(t)\right) = -\nabla\cdot\left[\mathbf{v}_p(t)\delta\left(\mathbf{x}-\mathbf{x}_p(t)\right)\right], \tag{4.15}$$

where the divergence is taken over the spatial variable $\mathbf{x}$. This can be obtained from a direct calculation using the chain rule and the definition (4.11):

$$\begin{aligned}\partial_t \delta\left(\mathbf{x}-\mathbf{x}_p(t)\right) &= \partial_t\left(\frac{1}{(2\pi)^3}\int_{-\infty}^{\infty} e^{i\mathbf{k}\cdot\left(\mathbf{x}-\mathbf{x}_p(t)\right)}\, d^3\mathbf{k}\right),\\ &= -\int_{-\infty}^{\infty} i\mathbf{k}\cdot\mathbf{v}_p(t) e^{i\mathbf{k}\cdot\left(\mathbf{x}-\mathbf{x}_p(t)\right)}\, d^3\mathbf{k}\Bigg),\\ &= -\nabla\cdot\left[\mathbf{v}_p(t)\delta\left(\mathbf{x}-\mathbf{x}_p(t)\right)\right],\end{aligned}$$

The next identity (equation (12) in [26]) is the distributional equivalent of the continuity equation

$$\frac{\partial\rho}{\partial t}+\nabla\cdot\mathbf{J}=0.$$

This identity can be obtained by combining the definitions (4.1) and (4.2) with the identity (4.15). First, notice that

$$\partial_t\rho\left(\mathbf{x},t\right)=\sum_{p=1}^{N_p} q_p\partial_t S\left(\mathbf{x}-\mathbf{x}_p(t)\right).$$

Using the identity (4.15) along with the property (4.12), we can express the shape function as a convolution with a delta function, which yields

$$\begin{aligned}\partial_t\rho\left(\mathbf{x},t\right) &= \sum_{p=1}^{N_p} q_p\partial_t\left(S\left(\mathbf{x}\right)*\delta\left(\mathbf{x}-\mathbf{x}_p\right)\right),\\ &= \sum_{p=1}^{N_p} q_p S\left(\mathbf{x}\right)*\partial_t\delta\left(\mathbf{x}-\mathbf{x}_p\right),\\ &= \sum_{p=1}^{N_p} q_p S\left(\mathbf{x}\right)*\left(-\nabla\cdot\left[\mathbf{v}_p(t)\delta\left(\mathbf{x}-\mathbf{x}_p(t)\right)\right]\right).\end{aligned}$$

Next, observe that derivatives of convolution commute when the integrand and its derivatives decay rapidly away from zero. The shape function S is compactly supported so that these conditions will

be easily satisfied. This permits one to write

$$\begin{aligned}
\partial_t \rho\left(\mathbf{x}, t\right) &= \sum_{p=1}^{N_p} q_p S\left(\mathbf{x}\right) * \left(-\nabla \cdot \left[\mathbf{v}_p(t) \delta\left(\mathbf{x} - \mathbf{x}_p(t)\right)\right] \right), \\
&= -\nabla \cdot \left(\sum_{p=1}^{N_p} q_p \mathbf{v}_p(t) S\left(\mathbf{x}\right) * \delta\left(\mathbf{x} - \mathbf{x}_p(t)\right) \right), \\
&= -\nabla \cdot \left(\sum_{p=1}^{N_p} q_p \mathbf{v}_p(t) S\left(\mathbf{x} - \mathbf{x}_p(t)\right) \right), \\
&= -\nabla \cdot \mathbf{J},
\end{aligned}$$

which shows how the continuity equation can be obtained using distributions.

The last identity (equation (13) in [26]), which builds on these intermediate results, describes how charge density can be calculated on the mesh so that it satisfies the continuity equation. To start, they integrate the continuity equation from 0 to t, which yields

$$\rho\left(\mathbf{x}, t\right) = \rho\left(\mathbf{x}, 0\right) - \int_0^t \nabla \cdot \mathbf{J}\left(\mathbf{x}, \tau\right)\, d\tau.$$

Using the definition (4.2), this is equivalent to writing

$$\begin{aligned}
\rho\left(\mathbf{x}, t\right) &= \rho\left(\mathbf{x}, 0\right) - \sum_{p=1}^{N_p} q_p \int_0^t \nabla \cdot \left(\mathbf{v}_p(\tau) S\left(\mathbf{x} - \mathbf{x}_p(\tau)\right) \right) d\tau. \\
&= \rho\left(\mathbf{x}, 0\right) - \sum_{p=1}^{N_p} q_p \int_0^t \nabla \cdot \left(\mathbf{v}_p(\tau) S\left(\mathbf{x}\right) * \delta\left(\mathbf{x} - \mathbf{x}_p(\tau)\right) \right) d\tau, \\
&= \rho\left(\mathbf{x}, 0\right) - \sum_{p=1}^{N_p} q_p \int_0^t \nabla \cdot \left(\mathbf{v}_p(\tau) S\left(\mathbf{x}\right) * \frac{1}{\left(2\pi\right)^3} \int_{-\infty}^{\infty} e^{i\mathbf{k}\cdot\left(\mathbf{x} - \mathbf{x}_p(\tau)\right)}\, d^3\mathbf{k} \right) d\tau.
\end{aligned}$$

For convenience, we shall express the integrals in the last line in their component-wise form, which is given by the expression

$$\int_0^t \partial_j \left(v_p^{(j)}(\tau) S\left(\mathbf{x}\right) * \frac{1}{\left(2\pi\right)^3} \int_{-\infty}^{\infty} e^{i\mathbf{k}\cdot\left(\mathbf{x} - \mathbf{x}_p(\tau)\right)}\, d^3\mathbf{k} \right) d\tau.$$

Note that we adopt the usual summation convention in which repeated indices are summed and we have used $\partial_j = \frac{\partial}{\partial x_j}$ for brevity. It is (hopefully) clear from the notation that these spatial derivatives

act only on the mesh data rather than particles. Next, we use the definition (4.10) and rearrange the integrals to obtain

$$\partial_j \left(S(\mathbf{x}) * \frac{1}{(2\pi)^3} \int_{-\infty}^{\infty} \int_0^t v_p^{(j)}(\tau) e^{i\mathbf{k}\cdot\left(\mathbf{x}-\mathbf{x}_p(0)-\int_0^\tau \mathbf{v}_i(s)\,ds\right)}\, d\tau\, d^3\mathbf{k} \right).$$

This expression can be further simplified by using the fact that $\frac{d\mathbf{x}_p}{dt} = \mathbf{v}_p$, from which we deduce that

$$\partial_j \left(S(\mathbf{x}) * \frac{1}{(2\pi)^3} \int_{-\infty}^{\infty} \int_0^t v_p^{(j)}(\tau) e^{i\mathbf{k}\cdot\left(\mathbf{x}-\mathbf{x}_p(\tau)\right)}\, d\tau\, d^3\mathbf{k} \right). \tag{4.16}$$

The term above contains three separate contributions to the mesh corresponding to each index value j. We shall provide details for $j = 1$, and state the results for the remaining cases $j = 2, 3$. Focusing on the inner integral, we see that we need to evaluate the time integral

$$\int_0^t v_p^{(1)}(\tau) e^{i\mathbf{k}\cdot\left(\mathbf{x}-\mathbf{x}_p(\tau)\right)}\, d\tau.$$

Next, we use the definition (4.10) to see that $v_p^{(1)}\, d\tau$ is the total change in the position over a time frame $d\tau$, in other words, $v_p^{(1)}\, d\tau = dx_p^{(1)}$. Then the time integral can be converted into a path integral that connects $x_p^{(1)}(0)$ and $x_p^{(1)}(t)$:

$$\int_{x_p^{(1)}(0)}^{x_p^{(1)}(t)} e^{i\left(k^{(1)}\left(x^{(1)}-x_p^{(1)}\right)+k^{(2)}\left(x^{(2)}-x_p^{(2)}\right)+k^{(3)}\left(x^{(3)}-x_p^{(3)}\right)\right)}\, dx_p^{(1)}.$$

Here, the variables for the remaining particle coordinates are transformed to $x_p^{(2)}$ and $x_p^{(3)}$ and the differential element runs over the first component of the particle position vector, leaving these remaining variables unaffected. Inserting this result into (4.16) (with $j = 1$) and interchanging the integrals, we obtain

$$\partial_1 \left(S(\mathbf{x}) * \frac{1}{(2\pi)^3} \int_{x_p^{(1)}(0)}^{x_p^{(1)}(t)} \int_{-\infty}^{\infty} e^{i\mathbf{k}\cdot\left(\mathbf{x}-\mathbf{x}_p\right)}\, d^3\mathbf{k}\, dx_p^{(1)} \right),$$

which, by (4.11), is equivalent to writing

$$\partial_1 \left(\int_{x_p^{(1)}(0)}^{x_p^{(1)}(t)} S(\mathbf{x}) * \delta\left(\mathbf{x}-\mathbf{x}_p\right)\, dx_p^{(1)} \right) = \partial_1 \left(\int_{x_p^{(1)}(0)}^{x_p^{(1)}(t)} S\left(\mathbf{x}-\mathbf{x}_p\right)\, dx_p^{(1)} \right). \tag{4.17}$$

The remaining indices ($j = 2, 3$) in (4.16) yield

$$\partial_2\left(S\left(\mathbf{x}\right) * \frac{1}{\left(2\pi\right)^3}\int_{-\infty}^{\infty}\int_0^t v_p^{(2)}(\tau)e^{i\mathbf{k}\cdot\left(\mathbf{x}-\mathbf{x}_p(\tau)\right)}\,d\tau\,d^3\mathbf{k}\right) = \partial_2\left(\int_{x_p^{(2)}(0)}^{x_p^{(2)}(t)} S\left(\mathbf{x}-\mathbf{x}_p\right)\,dx_p^{(2)}\right), \quad (4.18)$$

$$\partial_3\left(S\left(\mathbf{x}\right) * \frac{1}{\left(2\pi\right)^3}\int_{-\infty}^{\infty}\int_0^t v_p^{(3)}(\tau)e^{i\mathbf{k}\cdot\left(\mathbf{x}-\mathbf{x}_p(\tau)\right)}\,d\tau\,d^3\mathbf{k}\right) = \partial_3\left(\int_{x_p^{(3)}(0)}^{x_p^{(3)}(t)} S\left(\mathbf{x}-\mathbf{x}_p\right)\,dx_p^{(3)}\right), \quad (4.19)$$

which can be more succinctly expressed in vector notation as

$$\nabla\cdot\left(S\left(\mathbf{x}\right) * \frac{1}{\left(2\pi\right)^3}\int_{-\infty}^{\infty}\int_0^t \mathbf{v}_p(\tau)e^{i\mathbf{k}\cdot\left(\mathbf{x}-\mathbf{x}_p(\tau)\right)}\,d\tau\,d^3\mathbf{k}\right) = \nabla\cdot\left(\int_{\mathbf{x}_p(0)}^{\mathbf{x}_p(t)} S\left(\mathbf{x}-\mathbf{x}_p\right)\,d\mathbf{x}_p\right), \quad (4.20)$$

with the vector integration bounds being interpreted in an entry-wise fashion. Once the shape function $S(\mathbf{x}-\cdot)$ is selected, one can analytically compute the derivatives of the shape functions in (4.20) via the equations (4.17) - (4.19).

Actually, it is not necessary to introduce delta distributions to obtain the mapping (4.20). In [26], a convolution between the shape functions and delta distributions was performed, so that derivatives could be transferred directly onto the delta distributions. These derivatives are to be interpreted in the sense of distributions. A far simpler approach can be obtained by starting from the continuity equation with the definition (4.2):

$$\rho\left(\mathbf{x},t\right) = \rho\left(\mathbf{x},0\right) - \sum_{p=1}^{N_p} q_p\nabla\cdot\Big(\int_0^t \mathbf{v}_p(\tau)S\left(\mathbf{x}-\mathbf{x}_p(\tau)\right)\,d\tau\Big).$$

Using the fact that $\frac{d\mathbf{x}_p}{dt} = \mathbf{v}_p$, we can immediately see that $d\mathbf{x}_p = \mathbf{v}_p\,d\tau$, hence

$$\nabla\cdot\Big(\int_0^t \mathbf{v}_p(\tau)S\left(\mathbf{x}-\mathbf{x}_p(\tau)\right)\,d\tau\Big) = \nabla\cdot\Big(\int_{\mathbf{x}_p(0)}^{\mathbf{x}_p(t)} S\left(\mathbf{x}-\mathbf{x}_p\right)\,d\mathbf{x}_p\Big).$$

The charge mapping involves derivatives of the shape function S according to equations (4.17) - (4.19). An evaluation of these terms can be illustrated by considering the evaluation of a single component, such as the first one, i.e.,

$$\frac{\partial}{\partial x^{(1)}}\left(\int_{x_p^{(1)}(0)}^{x_p^{(1)}(t)} S(\mathbf{x}-\mathbf{x}_p)\,dx_p^{(1)}\right).$$

Using the change of variables $\mathbf{z} = \mathbf{x} - \mathbf{x}_p$, this integral can be expressed as:

$$\frac{\partial}{\partial x^{(1)}}\left(\int_{x_p^{(1)}(0)}^{x_p^{(1)}(t)} S(\mathbf{x}-\mathbf{x}_p)\,dx_p^{(1)}\right) = -\frac{\partial}{\partial x^{(1)}}\left(\int_{x^{(1)}-x_p^{(1)}(0)}^{x^{(1)}-x_p^{(1)}(t)} S(z^{(1)},z^{(2)},z^{(3)})\,dz^{(1)}\right).$$

If we write the multivariant spline as a tensor product of univariant shape functions, then this is equivalent to writing

$$\frac{\partial}{\partial x^{(1)}}\left(\int_{x_p^{(1)}(0)}^{x_p^{(1)}(t)} S(\mathbf{x}-\mathbf{x}_p)\, dx_p^{(1)}\right) = -\frac{\partial}{\partial x^{(1)}}\left(\int_{x^{(1)}-x_p^{(1)}(0)}^{x^{(1)}-x_p^{(1)}(t)} S(z^{(1)})S(z^{(2)})S(z^{(3)})\, dz^{(1)}\right).$$

Next, we define the anti-derivative of the univariant spline as

$$I(x) := \int_0^x S(\xi)\, d\xi = \begin{cases} x + \frac{x^2}{2\Delta x}, & -\Delta x \le x \le 0, \\ x - \frac{x^2}{2\Delta x}, & 0 < x \le \Delta x, \\ 0, & |x| > \Delta x. \end{cases}$$

Let us temporarily ignore the functions of $z^{(2)}$ and $z^{(3)}$, as the integral is done only in $z^{(1)}$. At the end, we will reintroduce these shape functions with the time level of the coordinates in $z^{(2)}$ and $z^{(3)}$ being selected to match those of $z^{(1)}$. Using the anti-derivative, the integral reduces to

$$\begin{aligned}\int_{x^{(1)}-x_p^{(1)}(0)}^{x^{(1)}-x_p^{(1)}(t)} S(z^{(1)})\, dz^{(1)} &= \int_0^{x^{(1)}-x_p^{(1)}(t)} S(z^{(1)})\, dz^{(1)} - \int_0^{x^{(1)}-x_p^{(1)}(0)} S(z^{(1)})\, dz^{(1)}, \\ &= I\left(x^{(1)} - x_p^{(1)}(t)\right) - I\left(x^{(1)} - x_p^{(1)}(0)\right).\end{aligned}$$

Taking the derivative then with respect to $x^{(1)}$, we obtain

$$\begin{aligned}\frac{\partial}{\partial x^{(1)}}\int_{x^{(1)}-x_p^{(1)}(0)}^{x^{(1)}-x_p^{(1)}(t)} S(z^{(1)})\, dz^{(1)} &= \frac{\partial}{\partial x^{(1)}} I\left(x^{(1)} - x_p^{(1)}(t)\right) - \frac{\partial}{\partial x^{(1)}} I\left(x^{(1)} - x_p^{(1)}(0)\right), \\ &= S\left(x^{(1)} - x_p^{(1)}(t)\right) - S\left(x^{(1)} - x_p^{(1)}(0)\right).\end{aligned}$$

Re-introducing the functions of $x_p^{(2)}$ and $x_p^{(3)}$, and collecting the results, we get

$$-\frac{\partial}{\partial x^{(1)}}\int_{x^{(1)}-x_p^{(1)}(0)}^{x^{(1)}-x_p^{(1)}(t)} S(z^{(1)})S(z^{(2)})S(z^{(3)})\, dz^{(1)} = -S\left(\mathbf{x}-\mathbf{x}_p\right)\Big|_{\mathbf{x}_p=\mathbf{x}_p(0)}^{\mathbf{x}_p=\mathbf{x}_p(t)}. \tag{4.21}$$

Again, the time levels for $x_p^{(2)}$ and $x_p^{(3)}$, in (4.21), should be the same as the one used for $x_p^{(1)}$ for consistency. It is interesting to note that when this process is repeated for the remaining derivatives, the result is identical to (4.21). The final form of the map is given by

$$\rho\left(\mathbf{x},t\right) = \rho\left(\mathbf{x},0\right) - \sum_{p=1}^{N_p} q_p \nabla\cdot\Big(\int_{\mathbf{x}_p(0)}^{\mathbf{x}_p(t)} S\left(\mathbf{x}-\mathbf{x}_p\right)\, d\mathbf{x}_p\Big), \tag{4.22}$$

where the divergence of the integral can be evaluated through repeated use of (4.21). While the map defined by (4.21) and (4.22) is conservative in the sense that the total charge in the domain is preserved in time, it introduces an image charge on the mesh that corresponds to the starting position along a particle path. This is problematic for the case of the expanding beam problem considered in section 4.6, which injects an electron beam into a metal cavity. The image charge induces a large potential at the injection size that reverses the trajectory of the beam.

The appearance of an image charge can be illustrated with a simple example that considers a single test particle whose macro-particle charge $q = 1$. Its charge at any time level can be calculated with the mapping presented above, which, in 2-D, for a single particle, is given by

$$\rho^{n+1} = \rho^n + 2q\Big(S(\mathbf{x} - \mathbf{x}_p^{n+1}) - S(\mathbf{x} - \mathbf{x}_p^n)\Big).$$

In this example, we suppose that at time $t = 0$ the particle is in the center of the grid cell (i, j) and at time $t = \Delta t$, the particle has reached the cell boundary. We use a matrix convention so that the grid points in question align with entries of the matrix. The linear spline shape functions at these grid points are found to be

$$S(\mathbf{x} - \mathbf{x}_p^n) = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 \end{bmatrix}, \quad S(\mathbf{x} - \mathbf{x}_p^{n+1}) = \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \end{bmatrix}.$$

Plugging these directly into the charge mapping, we find that

$$\rho^{n+1} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 \end{bmatrix} + 2\begin{bmatrix} 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \end{bmatrix} - 2\begin{bmatrix} \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{4} & \frac{3}{4} & 0 \\ -\frac{1}{4} & \frac{3}{4} & 0 \end{bmatrix}.$$

From this, it is clear that the total charge is conserved in the sense that

$$\sum_{i,j} \rho_{ij}^{n+1} = \sum_{i,j} \rho_{ij}^n,$$

but the method introduces a non-physical image charge corresponding to the previous location of the particle. If we remove the factor of 2 that appears in the spline mapping, so that the charge map now reads

$$\rho^{n+1} = \rho^n + q\Big(S(\mathbf{x} - \mathbf{x}_p^{n+1}) - S(\mathbf{x} - \mathbf{x}_p^n)\Big).$$

Then we can repeat the process and find that

$$\rho^{n+1} = \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \end{bmatrix},$$

which removes the image charge with the opposing sign; however, this modification is equivalent to the usual spline mapping for charge. For this reason, we shall not consider this map in our numerical experiments.

4.4 Conventional Methods for Pushing Particles

In this section, we discuss standard methods used in the scientific computing community for pushing particles. These developments are important to the core task of this work because they not only provide a basis for comparison with other methods, but they also describe how other potential users may employ our field solvers in their own codes. Beginning with the well-known leapfrog method, we discuss how the proposed field solvers can be leveraged to develop solvers for electrostatic problems. We then address the electromagnetic case, which begins with a description of the Boris rotation algorithm.

4.4.1 Leapfrog Time Integration

Leapfrog time integration is a well-known technique in methods for evolving particles due to its simplicity, long-time accuracy, and symplectic nature. While a comprehensive treatment of this integrator can be found in any of the classic texts on particle methods, including Birdsall and Langdon [2] and Hockney and Eastwood [3], we provide a few details here so that we can discuss the coupling of the method to the collection of solvers introduced in chapter 2. To this end, we consider Newton's second law of motion for a single particle, which can be written in the first-order form

$$\dot{\mathbf{x}} = \mathbf{v}, \tag{4.23}$$

$$\dot{\mathbf{v}} = \frac{1}{m}\mathbf{F}(\mathbf{x}, t), \tag{4.24}$$

where m is the mass of the particle and $\mathbf{x}(t)$, and $\mathbf{v}(t)$ denote, respectively, the position and velocity of the particle at time t. Further, $\mathbf{F}$ is the force that is used to accelerate the particle, which depends *only* on the position data of the particle. The leapfrog method can be derived by integrating the position equation (4.23) from t^n to t^{n+1} and the velocity equation (4.24) from $t^{n-1/2}$ to $t^{n+1/2}$:

$$\mathbf{x}(t^{n+1}) = \mathbf{x}(t^n) + \int_{t^n}^{t^{n+1}} \mathbf{v}(\tau)\, d\tau,$$
$$\mathbf{v}(t^{n+1/2}) = \mathbf{v}(t^{n-1/2}) + \frac{1}{m}\int_{t^{n-1/2}}^{t^{n+1/2}} \mathbf{F}(\mathbf{x}(\tau), \tau)\, d\tau.$$

Then, the integrals are approximated with a second-order accurate midpoint rule to obtain the fully discrete update

$$\mathbf{v}(t^{n+1/2}) = \mathbf{v}(t^{n-1/2}) + \frac{\Delta t}{m}\mathbf{F}(\mathbf{x}(t^n), t^n), \tag{4.25}$$
$$\mathbf{x}(t^{n+1}) = \mathbf{x}(t^n) + \Delta t\mathbf{v}(t^{n+1/2}). \tag{4.26}$$

The $\frac{\Delta t}{2}$ offset in time between the position and velocity in this method gives the scheme its name. In practice, an initial offset in the velocity can be achieved by stepping the velocity backwards in time by $-\frac{\Delta t}{2}$ using an explicit Euler method, so that

$$\mathbf{v}(t^{-1/2}) = \mathbf{v}(t^0) - \frac{\Delta t}{2m}\mathbf{F}(\mathbf{x}(t^0), t^0).$$

Since the local truncation error for the Explicit Euler method is second-order in time, this step will not degrade the rate of convergence. Once the initialization is complete, a given time step consists of the following ingredients:

1. Compute the force $\mathbf{F}(\mathbf{x}(t^n), t^n)$ at time $t = t^n$.

2. Update the velocity by Δt using equation (4.25).

3. Update the position by Δt using equation (4.26).

For electrostatic problems, in the absence of external forces, the force term represents the motion caused by an electric field, i.e., $\mathbf{F}(\mathbf{x}(t^n), t^n) = q\mathbf{E}(\mathbf{x}(t^n), t^n) \equiv -q\nabla\psi(\mathbf{x}(t^n), t^n)$, where q is the charge of the particle, $\mathbf{E}$ is the electric field, and ψ is a scalar potential.

Depending on the context, the scalar potential may be provided as part of the problem or require its own solve. In the electrostatic applications considered in this thesis, the scalar potential ψ is the solution to either a Poisson equation

$$-\Delta\psi = \frac{1}{\sigma_1}\rho, \tag{4.27}$$

or the two-way wave equation

$$\frac{1}{\kappa^2}\partial_{tt}\psi - \Delta\psi = \frac{1}{\sigma_1}\rho, \tag{4.28}$$

with κ being the speed at which the wave propagates and ρ being the charge density deposited by particles on a mesh. In the case of the Poisson equation (4.27), the electric field can be computed using elliptic solvers and the leapfrog time stepping procedure is unchanged. When the potential is described by a wave equation (4.28), then some minor modifications are required depending on whether the source ρ is to be treated explicitly or implicitly by our methods. In the latter case, both ψ and its derivatives $\nabla\psi$ can be computed once the position update (4.26) is complete. Similarly, an advance for ψ with an explicit source (e.g., the time-centered or central schemes) would take place prior to the position update (4.26). Then, once the positions have been updated, the derivatives can be computed using the now implicit sources.

In the next section, we describe the Boris push, which is an extension of the leapfrog time integration scheme to include contributions from more general electromagnetic fields.

4.4.2 The Boris Push

The Boris push, introduced in a 1970 paper by Jay Boris [4], is an extension of the leapfrog method discussed in section 4.4.1 to address rotations supported by the more general Lorentz force

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

This technique is the standard method for pushing particles in electromagnetic fields and is widely adopted for its simplicity, speed, and long-time accuracy [2, 3]. While the method itself is not symplectic, its success has largely been attributed to its volume preserving feature [78]. Moreover,

despite its lack of symplecticity, the fluctuations in the energy introduced by the method are generally known to remain small, even for problems that require integration over large time intervals.

The setup for the Boris method begins with a discretization in time of the equations of motion for the particles that results in a leapfrog structure identical to (4.25) and (4.26). A key difference is that the velocity update equation for the particles involves the velocity itself, which is time averaged to obtain the implicit equation

$$\mathbf{v}^{n+1/2} = \mathbf{v}^{n-1/2} + \frac{q}{m}\left(\mathbf{E} + \frac{\left(\mathbf{v}^{n+1/2} + \mathbf{v}^{n-1/2}\right)}{2} \times \mathbf{B}\right).$$

While it is possible to rearrange this equation and analytically solve for $\mathbf{v}^{n+1/2}$, Boris realized that the electric and magnetic field contributions could be separated with a simple advance obtained through the use of rotations. Following the notation of [79], the Boris rotation method for velocity can be performed with the following steps:

1. Compute $\mathbf{v}^{-} = \mathbf{v}^{n-1/2} + \dfrac{q\Delta t}{2m}\mathbf{E}$
2. Compute $\mathbf{v}^{'} = \mathbf{v}^{-} + \mathbf{v}^{-} \times \mathbf{t}$, where $\mathbf{t} = \dfrac{q\Delta t}{2m}\mathbf{B}$
3. Compute $\mathbf{v}^{+} = \mathbf{v}^{-} + \mathbf{v}^{'} \times \mathbf{s}$, where $\mathbf{s} = \dfrac{2\mathbf{t}}{1+t^2}$
4. Lastly, the velocity update is given as $\mathbf{v}^{n+1/2} = \mathbf{v}^{+} + \dfrac{q\Delta t}{2m}\mathbf{E}$.

The position data can then be updated using the newly acquired velocity $\mathbf{v}^{n+1/2}$ following the usual leapfrog update (4.26).

We now describe an approach that couples our field solvers to the Boris method. The proposed approach treats the fields at integer time levels, with the particle data being stored in the usual leapfrog format. First, the initialization begins with $\left(\mathbf{x}^0, \mathbf{v}^0\right)$, along with the fields $\mathbf{E}^0$ and $\mathbf{B}^0$ (obtained from ψ^0 and $\mathbf{A}^0$). To create the velocity staggering in time, we use the initial fields to push the velocities back by $\Delta t/2$ using the Boris rotation method, which is second-order and results in $\mathbf{v}^{-1/2}$. Then, assuming we have the data $\left(\mathbf{E}^n, \mathbf{B}^n, \mathbf{x}^n, \mathbf{v}^{n-1/2}\right)$, the procedure for evolving the system from t^n to t^{n+1} consists of the following steps:

1. Apply the Boris rotation: $\left(\mathbf{E}^n, \mathbf{B}^n, \mathbf{v}^{n-1/2}\right) \mapsto \mathbf{v}^{n+1/2}$.

2. Advance the positions: $\left(\mathbf{x}^n, \mathbf{v}^{n+1/2}\right) \mapsto \mathbf{x}^{n+1}$.

3. Time average the velocities $\left(\mathbf{v}^{n-1/2}, \mathbf{v}^{n+1/2}\right) \mapsto \mathbf{v}^n$ and compute the current density $\mathbf{J}^n$.

4. Advance the $\mathbf{A}$ explicitly with the central scheme: $(\mathbf{A}^n, \mathbf{J}^n) \mapsto \mathbf{A}^{n+1}$.

5. Compute the charge density ρ^{n+1}.

6. Advance the ψ and its derivatives: $\left(\psi^n, \rho^{n+1}\right) \mapsto \left(\psi^{n+1}, \nabla\psi^{n+1}\right)$.

7. Compute the electric field $\mathbf{E}^{n+1} := -\nabla\psi^{n+1} - \partial_t \mathbf{A}^{n+1}$ using data from steps 4 and 6.

8. Iterate on the magnetic field data:

   a) Compute derivatives of $\mathbf{A}$ with the BDF update: $\left(\mathbf{A}^n, \mathbf{J}^{n+1,[k]}\right) \mapsto \nabla\mathbf{A}^{n+1,[k]}$.

   b) Approximate the magnetic field $\mathbf{B}^{n+1,[k]} := \nabla \times \mathbf{A}^{n+1,[k]}$ with $\nabla\mathbf{A}^{n+1,[k]}$.

   c) Advance the particle velocities: $\left(\mathbf{E}^{n+1}, \mathbf{B}^{n+1,[k]}, \mathbf{v}^{n+1/2}\right) \mapsto \mathbf{v}^{n+3/2}$.

   d) Time average the velocities $\left(\mathbf{v}^{n+1/2}, \mathbf{v}^{n+3/2}\right) \mapsto \mathbf{v}^{n+1}$ and compute $\mathbf{J}^{n+1,[k+1]}$.

   e) Repeat for some prescribed number of iterations or until convergence.

9. Prepare for the next time step: $\left(\mathbf{E}^{n+1}, \mathbf{B}^{n+1}, \mathbf{x}^{n+1}, \mathbf{v}^{n+1/2}\right) \mapsto \left(\mathbf{E}^n, \mathbf{B}^n, \mathbf{x}^n, \mathbf{v}^{n-1/2}\right)$.

The fields $(\psi, \mathbf{A})$, spatial derivatives $(\nabla\psi, \nabla\mathbf{A})$, and $\partial_t\mathbf{A}$, all live at the integer time levels in this approach. The reason is that the particle velocity evolution step over $[t^{n-1/2}, t^{n+1/2}]$ requires the field data at the mid-point t^n. The particle position data is at the integer levels. We generally use variables with square brackets as an upper index to indicate that it is an iteration variable. For instance, in step 8a, we use the source data at the wrong time level, which means that the BDF scheme will be using the incorrect source. By iterating on the current density $\mathbf{J}^{n+1}$, generally with a few iterates, we can obtain a good approximation of this source. To start the iteration in step 8,

we can apply Taylor expansion to the current density so that it is centered about time $t = t^n$. For second-order accuracy in time, we can start the iteration with

$$\mathbf{J}^{n+1,[0]} = \mathbf{J}^n + \left(\mathbf{J}^n - \mathbf{J}^{n-1}\right).$$

In the next section, we discuss a time integration method for particles that are evolved using non-separable Hamiltonians.

4.5 Time Integration with Non-separable Hamiltonians

In this section, we introduce the time integration methods for the particles in formulations, which evolve a certain non-separable Hamiltonian. An outline of the approach is provided in section 4.5.1. Once we have introduces the basic elements of the method, we propose several approaches for incorporating the field evolution into the scheme. We consider two perspectives. First, in section 4.5.1.1, we develop a naive implementation in which the particles “lead" the fields, i.e., the fields are modified through changes in the particles. The second approach we consider is presented in section 4.5.1.2 and is a variant of the first approach, allowing one to utilize methods which allow sources to be explicit.

4.5.1 The Molei Tao Integrator

For equations of motion of the form

$$\begin{aligned}
\dot{\mathbf{x}}_i &= \frac{1}{m_i}\left(\mathbf{P}_i - q_i\mathbf{A}\right) \equiv V\left(\mathbf{x}_i, \mathbf{P}_i\right), \\
\dot{\mathbf{P}}_i &= -q_i\nabla\psi + \frac{q_i}{m_i}(\nabla\mathbf{A})\cdot\left(\mathbf{P}_i - q_i\mathbf{A}\right) \equiv W\left(\mathbf{x}_i, \mathbf{P}_i\right),
\end{aligned}$$

the phase space variables are non-separable, which means traditional integrators, such as those in the previous section, can not be applied to the system. Recently, a paper by Tao [47] introduced methods, inspired by the work [80], to approximate non-separable Hamiltonians $H(\mathbf{x}, \mathbf{P})$ using an *augmented* form

$$\bar{H}\left(\mathbf{x}, \mathbf{P}, \mathbf{y}, \mathbf{Q}\right) := H_A + H_B + \omega H_C,$$

with

$$H_A := H(\mathbf{x}, \mathbf{Q}), \quad H_B := H(\mathbf{y}, \mathbf{P}), \quad H_C := \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + \frac{1}{2}||\mathbf{P} - \mathbf{Q}||^2.$$

By duplicating the phase space information, Tao was able to construct a family of explicit symplectic integration schemes of any even order degree of accuracy which do not require negative time steps. This latter point is significant for plasma problems modeled on bounded domains. In such problems, charged particles may be absorbed into the boundary within a single time step, and it is not clear how one should reverse this process.

To build these methods, Tao introduces the following set of flow maps that evolve the system forward in Δt-time:

$$\phi_{H_A}^{\Delta t} : \begin{bmatrix} \mathbf{x} \\ \mathbf{P} \\ \mathbf{y} \\ \mathbf{Q} \end{bmatrix} \mapsto \begin{bmatrix} \mathbf{x} \\ \mathbf{P} - \Delta t \partial_{\mathbf{x}} H(\mathbf{x}, \mathbf{Q}) \\ \mathbf{y} + \Delta t \partial_{\mathbf{Q}} H(\mathbf{x}, \mathbf{Q}) \\ \mathbf{Q} \end{bmatrix}, \quad \phi_{H_B}^{\Delta t} : \begin{bmatrix} \mathbf{x} \\ \mathbf{P} \\ \mathbf{y} \\ \mathbf{Q} \end{bmatrix} \mapsto \begin{bmatrix} \mathbf{x} + \Delta t \partial_{\mathbf{P}} H(\mathbf{y}, \mathbf{P}) \\ \mathbf{P} \\ \mathbf{y} \\ \mathbf{Q} - \Delta t \partial_{\mathbf{y}} H(\mathbf{y}, \mathbf{P}) \end{bmatrix},$$

$$\phi_{\omega H_C}^{\Delta t} : \begin{bmatrix} \mathbf{x} \\ \mathbf{P} \\ \mathbf{y} \\ \mathbf{Q} \end{bmatrix} \mapsto \frac{1}{2} \begin{bmatrix} \begin{pmatrix} \mathbf{x} + \mathbf{y} \\ \mathbf{P} + \mathbf{Q} \end{pmatrix} + R(\omega, \Delta t) \begin{pmatrix} \mathbf{x} - \mathbf{y} \\ \mathbf{P} - \mathbf{Q} \end{pmatrix} \\ \begin{pmatrix} \mathbf{x} + \mathbf{y} \\ \mathbf{P} + \mathbf{Q} \end{pmatrix} - R(\omega, \Delta t) \begin{pmatrix} \mathbf{x} - \mathbf{y} \\ \mathbf{P} - \mathbf{Q} \end{pmatrix} \end{bmatrix}.$$

Here, R is the block rotation matrix

$$R(\omega, \Delta t) = \begin{bmatrix} \cos(2\omega\Delta t) I & \sin(2\omega\Delta t) I \\ -\sin(2\omega\Delta t) I & \cos(2\omega\Delta t) I \end{bmatrix},$$

with I being the 2×2 or 3×3 identity matrix. Various integrators of any even order degree of accuracy can be obtained through a composition of these mappings. We refer the interested reader to the paper [47] for details. As an example, the paper provides the following second-order method:

$$\phi_2^{\Delta t} := \phi_{H_A}^{\Delta t/2} \circ \phi_{H_B}^{\Delta t/2} \circ \phi_{\omega H_C}^{\Delta t} \circ \phi_{H_B}^{\Delta t/2} \circ \phi_{H_A}^{\Delta t/2}.$$

In this composition, it is important to note that the coupling map $\phi_{\omega H_C}^{\Delta t}$ does not evolve the particles, but, instead, mixes the data in phase space.

A key element of this method is the use of a binding constant ω that synchronizes the two sets of phase space variables. Tao establishes an estimate on the accuracy of these methods in the context of long time simulations for integrable systems. For a method with order ℓ, time step size Δt, coupling parameter ω, and the simulation time T, he shows that the error is of the form

$$O\left(T\Delta t^{\ell}\omega\right),$$

as long as $T = O\left(\min\left(\Delta t^{-\ell}\omega^{-\ell}, \omega^{1/2}\right)\right)$. Based on this bound, he recommends that $\Delta t \ll \omega^{-1/\ell}$, and that it is both more accurate and efficient to use increase the order ℓ under a fixed ω.

The test problems shown in Tao's paper evolve particles in fixed fields which can be either static or non-static, but are known functions. In our applications, the electric and magnetic fields respond to changes in the plasma, which is represented using particles. Therefore, coupling these particle integration schemes to our methods for evolving fields is a non-trivial task, and we describe the necessary modifications in the sections that follow.

4.5.1.1 Approach for Implicit Sources: Particles Lead the Fields

The wave equations for the potentials ψ and $\mathbf{A}$ are non-linear due to the source terms that couple with the particle data. We break this coupling through the use of duplicate fields, analogous to Tao's approach for the particles. To this end, we let $\left(\psi_{xq}^n, \nabla\psi_{xq}^n, \mathbf{A}_{xq}^n, \nabla\mathbf{A}_{xq}^n\right)$ and $\left(\psi_{yp}^n, \nabla\psi_{yp}^n, \mathbf{A}_{yp}^n, \nabla\mathbf{A}_{yp}^n\right)$ denote two pairs of field data at time level t^n. Moreover each pair of field data is associated with a set of particle data indicated by the subscripts. Assuming that we have the data $(\mathbf{x}^n, \mathbf{Q}^n)$, $(\mathbf{y}^n, \mathbf{P}^n)$, $\left(\psi_{xq}^n, \nabla\psi_{xq}^n, \mathbf{A}_{xq}^n, \nabla\mathbf{A}_{xq}^n\right)$, and $\left(\psi_{yp}^n, \nabla\psi_{yp}^n, \mathbf{A}_{yp}^n, \nabla\mathbf{A}_{yp}^n\right)$, the second order update $\phi_2^{\Delta t}$ can be modified to include updates to fields as follows:

1. Push particles: $(\mathbf{y}^n, \mathbf{P}^n) \mapsto \left(\mathbf{y}^{n+1/2}, \mathbf{P}^{n+1/2}\right)$ using $\left(\mathbf{x}^n, \mathbf{Q}^n, \nabla\psi_{xq}^n, \mathbf{A}_{xq}^n, \nabla\mathbf{A}_{xq}^n\right)$
2. Evolve the fields $\left(\psi_{yp}^n, \nabla\psi_{yp}^n, \mathbf{A}_{yp}^n, \nabla\mathbf{A}_{yp}^n\right) \mapsto \left(\psi_{yp}^{n+1/2}, \nabla\psi_{yp}^{n+1/2}, \mathbf{A}_{yp}^{n+1/2}, \nabla\mathbf{A}_{yp}^{n+1/2}\right)$ using the particle data $\left(\mathbf{y}^{n+1/2}, \mathbf{P}^{n+1/2}\right)$
3. Push particles: $(\mathbf{x}^n, \mathbf{Q}^n) \mapsto \left(\mathbf{x}^{n+1/2}, \mathbf{Q}^{n+1/2}\right)$ using $\left(\mathbf{y}^{n+1/2}, \mathbf{P}^{n+1/2}, \nabla\psi_{yp}^{n+1/2}, \mathbf{A}_{yp}^{n+1/2}, \nabla\mathbf{A}_{yp}^{n+1/2}\right)$

4. Coupling step: $\left(\mathbf{x}^{n+1/2}, \mathbf{P}^{n+1/2}, \mathbf{y}^{n+1/2}, \mathbf{Q}^{n+1/2}\right) \mapsto (\mathbf{x}^*, \mathbf{P}^*, \mathbf{y}^*, \mathbf{Q}^*)$

5. Recompute the field data $\left(\psi_{yp}^{n+1/2}, \nabla\psi_{yp}^{n+1/2}, \mathbf{A}_{yp}^{n+1/2}, \nabla\mathbf{A}_{yp}^{n+1/2}\right)$ using the particle data $(\mathbf{y}^*, \mathbf{P}^*)$

6. Evolve the fields $\left(\psi_{xq}^{n}, \nabla\psi_{xq}^{n}, \mathbf{A}_{xq}^{n}, \nabla\mathbf{A}_{xq}^{n}\right) \mapsto \left(\psi_{xq}^{n+1/2}, \nabla\psi_{xq}^{n+1/2}, \mathbf{A}_{xq}^{n+1/2}, \nabla\mathbf{A}_{xq}^{n+1/2}\right)$ using the particle data $(\mathbf{x}^*, \mathbf{Q}^*)$

7. Push particles: $(\mathbf{x}^*, \mathbf{Q}^*) \mapsto \left(\mathbf{x}^{n+1}, \mathbf{Q}^{n+1}\right)$ using $\left(\mathbf{y}^*, \mathbf{P}^*, \nabla\psi_{yp}^{n+1/2}, \mathbf{A}_{yp}^{n+1/2}, \nabla\mathbf{A}_{yp}^{n+1/2}\right)$

8. Evolve the fields $\left(\psi_{xq}^{n+1/2}, \nabla\psi_{xq}^{n+1/2}, \mathbf{A}_{xq}^{n+1/2}, \nabla\mathbf{A}_{xq}^{n+1/2}\right) \mapsto \left(\psi_{xq}^{n+1}, \nabla\psi_{xq}^{n+1}, \mathbf{A}_{xq}^{n+1}, \nabla\mathbf{A}_{xq}^{n+1}\right)$ using the particle data $\left(\mathbf{x}^{n+1}, \mathbf{Q}^{n+1}\right)$

9. Push particles: $(\mathbf{y}^*, \mathbf{P}^*) \mapsto \left(\mathbf{y}^{n+1}, \mathbf{P}^{n+1}\right)$ using $\left(\mathbf{x}^{n+1}, \mathbf{Q}^{n+1}, \nabla\psi_{xq}^{n+1}, \mathbf{A}_{xq}^{n+1}, \nabla\mathbf{A}_{xq}^{n+1}\right)$

10. Evolve the fields $\left(\psi_{yp}^{n+1/2}, \nabla\psi_{yp}^{n+1/2}, \mathbf{A}_{yp}^{n+1/2}, \nabla\mathbf{A}_{yp}^{n+1/2}\right) \mapsto \left(\psi_{yp}^{n+1}, \nabla\psi_{yp}^{n+1}, \mathbf{A}_{yp}^{n+1}, \nabla\mathbf{A}_{yp}^{n+1}\right)$ using the particle data $\left(\mathbf{y}^{n+1}, \mathbf{P}^{n+1}\right)$

There are several essential details embedded in the steps shown above which require further explanation. While it may not be apparent from the notation, the field updates involve additional time history, not just the most recent one. Quantities which are labeled with "$*$" live at time level $t^{n+1/2}$, but we use this notation to distinguish the data from $t^{n+1/2}$ pre/post-mixing when clarification is needed. The particle updates on pairs of coordinates may appear strange because of their implicit-like form, which merely reflects the coupling of phase space. As an example, in order to perform the particle push in step 3, we need to use the fields associated with the particle data $\left(\mathbf{y}^{n+1/2}, \mathbf{P}^{n+1/2}\right)$. This is reflected in step 2. Next, it would seem that we are forgetting to evolve fields between steps 3 and 4; however, the mixing step will modify the particle data regardless of the fields, so there is no need to perform this evolution. Instead, the fields are recomputed in step 5 because of the mixing that occurs in step 4.

Remark 4.5.1. This algorithm is quite costly both in terms of memory and computation. It requires duplicates of both particle and field information, which can be problematic when large numbers of particles and fine meshes are required. Computationally, a single time step of this second-order

scheme has a total of 10 steps in which fields and derivatives are updated. The total number of calls made to the wave solver methods depends entirely on the dimensionality of the problem, which will change the entries of the gradient vector, and the number of components retained for the vector potentials (if any).

4.5.1.2 Approaches for a Mixed Advance: Dealing with Explicit and Implicit Source Terms

This section presents a modification of the approach discussed in section 4.5.1.1 to develop a mixed approach where fields can be advanced using an explicit form of source terms with their corresponding derivatives using sources with an implicit form. Assuming that we have the data $(\mathbf{x}^n, \mathbf{Q}^n)$, $(\mathbf{y}^n, \mathbf{P}^n)$, $\left(\psi^n_{xq}, \nabla\psi^n_{xq}, \mathbf{A}^n_{xq}, \nabla\mathbf{A}^n_{xq}\right)$, and $\left(\psi^n_{yp}, \nabla\psi^n_{yp}, \mathbf{A}^n_{yp}, \nabla\mathbf{A}^n_{yp}\right)$, the second order update $\phi_2^{\Delta t}$ can be modified to include updates to fields as follows:

1. Evolve the fields $\left(\psi^n_{yp}, \mathbf{A}^n_{yp}\right) \mapsto \left(\psi^{n+1/2}_{yp}, \mathbf{A}^{n+1/2}_{yp}\right)$ using the particle data $(\mathbf{y}^n, \mathbf{P}^n)$
2. Push particles: $(\mathbf{y}^n, \mathbf{P}^n) \mapsto \left(\mathbf{y}^{n+1/2}, \mathbf{P}^{n+1/2}\right)$ using $\left(\mathbf{x}^n, \mathbf{Q}^n, \nabla\psi^n_{xq}, \mathbf{A}^n_{xq}, \nabla\mathbf{A}^n_{xq}\right)$
3. Compute the derivatives $\left(\nabla\psi^{n+1/2}_{yp}, \nabla\mathbf{A}^{n+1/2}_{yp}\right)$ using the particle data $\left(\mathbf{y}^{n+1/2}, \mathbf{P}^{n+1/2}\right)$
4. Evolve the fields $\left(\psi^n_{xq}, \mathbf{A}^n_{xq}\right) \mapsto \left(\psi^{n+1/2}_{xq}, \mathbf{A}^{n+1/2}_{xq}\right)$ using the particle data $(\mathbf{x}^n, \mathbf{Q}^n)$
5. Push particles: $(\mathbf{x}^n, \mathbf{Q}^n) \mapsto \left(\mathbf{x}^{n+1/2}, \mathbf{Q}^{n+1/2}\right)$ using $\left(\mathbf{y}^{n+1/2}, \mathbf{P}^{n+1/2}, \nabla\psi^{n+1/2}_{yp}, \mathbf{A}^{n+1/2}_{yp}, \nabla\mathbf{A}^{n+1/2}_{yp}\right)$
6. Coupling step: $\left(\mathbf{x}^{n+1/2}, \mathbf{P}^{n+1/2}, \mathbf{y}^{n+1/2}, \mathbf{Q}^{n+1/2}\right) \mapsto (\mathbf{x}^*, \mathbf{P}^*, \mathbf{y}^*, \mathbf{Q}^*)$
7. Recompute the derivatives $\left(\nabla\psi^{n+1/2}_{yp}, \nabla\mathbf{A}^{n+1/2}_{yp}\right)$ using the particle data $(\mathbf{y}^*, \mathbf{P}^*)$
8. Compute the derivatives $\left(\nabla\psi^{n+1/2}_{xq}, \nabla\mathbf{A}^{n+1/2}_{yp}\right)$ using the particle data $(\mathbf{x}^*, \mathbf{Q}^*)$
9. Evolve the fields $\left(\psi^{n+1/2}_{xq}, \mathbf{A}^{n+1/2}_{xq}\right) \mapsto \left(\psi^{n+1}_{xq}, \mathbf{A}^{n+1}_{xq}\right)$ using the particle data $(\mathbf{x}^*, \mathbf{Q}^*)$
10. Push particles: $(\mathbf{x}^*, \mathbf{Q}^*) \mapsto \left(\mathbf{x}^{n+1}, \mathbf{Q}^{n+1}\right)$ using $\left(\mathbf{y}^*, \mathbf{P}^*, \nabla\psi^{n+1/2}_{yp}, \mathbf{A}^{n+1/2}_{yp}, \nabla\mathbf{A}^{n+1/2}_{yp}\right)$
11. Compute the derivatives $\left(\nabla\psi^{n+1}_{xq}, \nabla\mathbf{A}^{n+1}_{xq}\right)$ using the particle data $\left(\mathbf{x}^{n+1}, \mathbf{Q}^{n+1}\right)$

12. Evolve the fields $\left(\psi_{yp}^{n+1/2}, \mathbf{A}_{yp}^{n+1/2}\right) \mapsto \left(\psi_{yp}^{n+1}, \mathbf{A}_{yp}^{n+1}\right)$ using the particle data $(\mathbf{y}^*, \mathbf{P}^*)$

13. Push particles: $(\mathbf{y}^*, \mathbf{P}^*) \mapsto \left(\mathbf{y}^{n+1}, \mathbf{P}^{n+1}\right)$ using $\left(\mathbf{x}^{n+1}, \mathbf{Q}^{n+1}, \nabla\psi_{xq}^{n+1}, \mathbf{A}_{xq}^{n+1}, \nabla\mathbf{A}_{xq}^{n+1}\right)$

14. Compute the derivatives $\left(\nabla\psi_{yp}^{n+1}, \nabla\mathbf{A}_{yp}^{n+1}\right)$ using the particle data $\left(\mathbf{y}^{n+1}, \mathbf{P}^{n+1}\right)$

Remark 4.5.2. Consideration of a mixed approach of this form, with regard to the field solvers developed in Chapter 2, is useful in preventing excessive dissipation. More specifically, the time-centered field solvers, which use an explicit source, are known to be *purely* dispersive [42].

Using the time-centered update for the fields, which requires an explicit source, in conjunction with duplicate field and particle data may create a mismatch in time levels between the fields and the particles. This motivates us to consider a variation on the first approach where particles lead the fields, but the field advances apply the time centered method where the source is made implicit by averaging. For example the charge density at time level t^n can be approximated to second-order accuracy in time by

$$\rho^n \approx \frac{1}{2}\left(\rho^{n+1} + \rho^{n-1}\right).$$

4.5.2 The Asymmetrical Euler Method

We recently became aware of an alternative particle push, suitable for non-separable Hamiltonian systems, which was proposed in [36]. For context, this paper considered mesh-free methods for solving the Darwin limit of the VM system using the Coulomb gauge. Their adoption of a generalized Hamiltonian model for particles was largely motivated by the numerical instabilities associated with time derivatives of the vector potential in this particular limit. The resulting model, which is essentially identical to the formulation (1.39)-(1.40) used in this work, trades additional coupling of phase space for numerical stability. They proposed a semi-implicit method, dubbed

the asymmetrical Euler method (AEM), which has the form

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^n \Delta t, \tag{4.29}$$

$$\mathbf{P}_i^{n+1} = \mathbf{P}_i^n + q_i \left(-\nabla \psi^{n+1} + \nabla \mathbf{A}^{n+1} \cdot \mathbf{v}_i^n \right) \Delta t, \tag{4.30}$$

$$\mathbf{v}_i^n \equiv \frac{1}{m_i} \left(\mathbf{P}_i^n - q_i \mathbf{A}^n \right). \tag{4.31}$$

This method, which is first-order in time, proceeds by, first, performing an explicit update of the particle positions using (4.29). Next, with the new positions, fields are updated, and finally, the generalized momentum update (4.30) is performed. While it may appear that iteration is required to compute gradients of the vector potential $\nabla \mathbf{A}^{n+1}$, since $\mathbf{A}^{n+1}$ requires $\mathbf{v}_i^{n+1}$ through the current, the authors use $\mathbf{v}_i^n$, which results in a fully-explicit update. It is recommended to avoid iteration, especially in the case that the vector potential is strong to prevent the formation of numerical instabilities. Further, this lagging of the velocity is consistent with a first-order method. At the end of the particle update, the velocity is modified according to (4.31) for use in the next time step.

In contrast to the method of Molei Tao, discussed in section 4.5.1, this method requires significantly less overall storage and fewer overall field solves. Despite the fact that this method is first order, experiments performed in [36] demonstrated good energy conservation and good accuracy, in addition to computational efficiency. Additionally, the lagging of the velocity in the generalized momentum update (4.30) means that this method avoids the pitfall of circular logic that occurs in the Molei Tao method when current needs to be mapped to the mesh. We apply this integrator to the expanding beam problems in section 4.6.

4.6 Numerical Examples

In this section we apply the proposed PIC methods to several well-known benchmark problems in the literature. First, we test the particle methods and our formulation in the case of a single particle moving through known fields. Then, we apply the methods to more interesting problems involving dynamic fields, which respond to the motion of the particles. We use the physical constants listed in Table 4.1 in the numerical experiments presented in this work.

Parameter	Value
Ion mass (m_i) [kg]	$9.108379025973462 \times 10^{-29}$
Electron mass (m_e) [kg]	$9.108379025973462 \times 10^{-31}$
Boltzmann constant (k_B) [kg m^2 s^{-2} K^{-1}]	$1.38064852 \times 10^{-23}$
Permittivity of free space (ϵ_0) [kg^{-1} m^{-3} s^4 A^2]	$8.854187817 \times 10^{-12}$
Permittivity of free space (μ_0) [kg m s^{-2} A^{-2}]	$1.25663706 \times 10^{-6}$
Speed of light (c) [m/s]	2.99792458×10^8

Table 4.1: Table of the physical constants (SI units) used in the numerical experiments.

4.6.1 Motion of a Single Charged Particle

We first compare the integrator proposed by Tao [47] with the Boris method [4], which are discussed in sections 4.4.2 and 4.5.1, respectively. This is a natural first step before applying the method to problems with dynamic fields that respond to particle motion. Here, we consider a simple model for the motion of a single charged particle that is given by

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{v}, \\ \dot{\mathbf{v}} &= \frac{q}{m} \left(\mathbf{E} + \mathbf{v} \times \mathbf{B} \right). \end{aligned}$$

We use electro- and magneto-static fields here and suppose that the magnetic field lies along the $\hat{\mathbf{z}}$ unit vector, so

$$\mathbf{B} = B_0 \hat{\mathbf{z}}, \quad \mathbf{E} = E^{(1)} \hat{\mathbf{x}} + E^{(y)} \hat{\mathbf{y}} + E^{(z)} \hat{\mathbf{z}},$$

where B_0 is a constant. Again, component-based definitions have been used for the fields $\mathbf{E} = \left(E^{(1)}, E^{(2)}, E^{(3)} \right)$ and $\mathbf{B} = \left(B^{(1)}, B^{(2)}, B^{(3)} \right)$. Consequently, we have that

$$\mathbf{v} \times \mathbf{B} = v^{(2)} B_0 \hat{\mathbf{x}} - v^{(1)} B_0 \hat{\mathbf{y}},$$

so the full equations of motion are

$$\begin{aligned} \frac{dx^{(1)}}{dt} &= v^{(1)}, \quad \frac{dv^{(1)}}{dt} = \frac{q}{m} \left(E^{(1)} + v^{(2)} B_0 \right), \\ \frac{dx^{(2)}}{dt} &= v^{(2)}, \quad \frac{dv^{(2)}}{dt} = \frac{q}{m} \left(E^{(2)} - v^{(1)} B_0 \right), \\ \frac{dx^{(3)}}{dt} &= v^{(3)}, \quad \frac{dv^{(3)}}{dt} = \frac{q}{m} E^{(3)}. \end{aligned}$$

We can then use the classical momentum $\mathbf{p} = m\mathbf{v}$ to obtain

$$\begin{aligned}
\frac{dx^{(1)}}{dt} &= \frac{1}{m}p^{(1)}, \quad \frac{dp^{(1)}}{dt} = q\left(E^{(1)} + \frac{1}{m}p^{(2)}B_0\right), \\
\frac{dx^{(2)}}{dt} &= \frac{1}{m}p^{(2)}, \quad \frac{dp^{(2)}}{dt} = q\left(E^{(2)} - \frac{1}{m}p^{(1)}B_0\right), \\
\frac{dx^{(3)}}{dt} &= \frac{1}{m}p^{(3)}, \quad \frac{dp^{(3)}}{dt} = qE^{(3)}.
\end{aligned}$$

Next, we show how to convert the electric and magnetic fields to potentials for use in Molei Tao's method.

Using the potentials ψ and $\mathbf{A} \equiv \left(A^{(1)}, A^{(2)}, A^{(3)}\right)$, one can compute the electric and magnetic fields via (1.11), which is equivalent to writing

$$\begin{aligned}
E^{(1)} &= -\partial_x\psi - \partial_t A^{(1)}, \quad B^{(1)} = \partial_y A^{(3)} - \partial_z A^{(2)}, \\
E^{(2)} &= -\partial_y\psi - \partial_t A^{(2)}, \quad B^{(2)} = -\partial_x A^{(3)} + \partial_z A^{(1)}, \\
E^{(3)} &= -\partial_z\psi - \partial_t A^{(3)}, \quad B^{(3)} = \partial_x A^{(2)} - \partial_y A^{(1)}.
\end{aligned}$$

The time-independence of the magnetic field for this problem implies that $\partial_t\mathbf{A} = 0$, so that

$$\mathbf{E} = -\nabla\psi \implies E^{(1)} = -\partial_x\psi, \quad E^{(2)} = -\partial_y\psi, \quad E^{(3)} = -\partial_z\psi.$$

Therefore, for this problem, we can use

$$\psi = -E^{(1)}x - E^{(2)}y - E^{(3)}z.$$

Moreover, since the magnetic field lives only in the z-direction, then this implies that the vector potential can be written as

$$\mathbf{B} = (0, 0, B_0) = (0, 0, \partial_x A^{(2)} - \partial_y A^{(1)}).$$

As the choice of functions for gauges are not unique, it suffices to pick

$$A^{(1)} \equiv 0, \quad A^{(2)} = B_0 x, \quad A^{(3)} \equiv 0.$$

In summary, the values and required derivatives for the potentials are given by

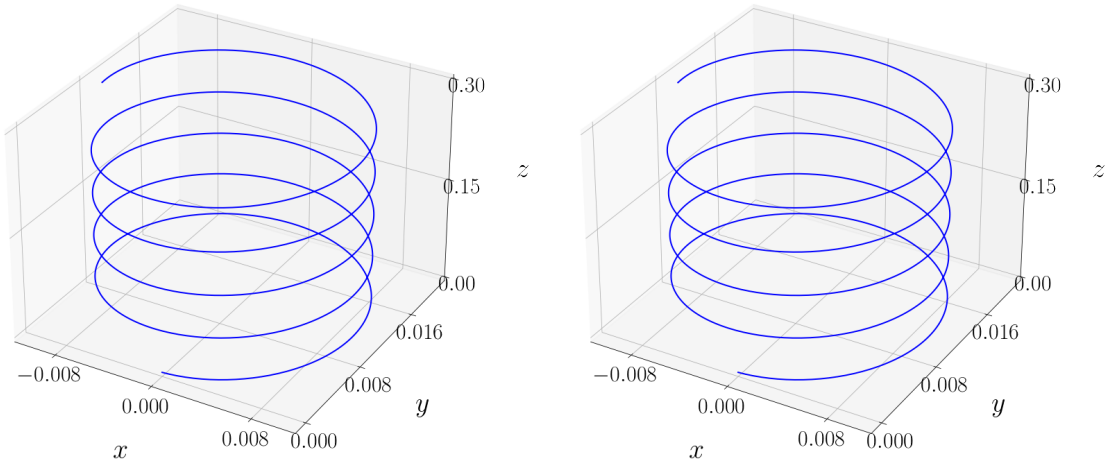
$$
\begin{aligned}
-\partial_x \psi = E^{(1)}, \quad -\partial_y \psi = E^{(2)}, \quad -\partial_z \psi = E^{(3)}, \\
A^{(1)} = 0, \quad A^{(2)} = B_0 x, \quad A^{(3)} = 0, \\
\partial_x A^{(1)} = 0, \quad \partial_y A^{(1)} = 0, \quad \partial_z A^{(1)} = 0, \\
\partial_x A^{(2)} = B_0, \quad \partial_y A^{(2)} = 0, \quad \partial_z A^{(2)} = 0, \\
\partial_x A^{(3)} = 0, \quad \partial_y A^{(3)} = 0, \quad \partial_z A^{(3)} = 0,
\end{aligned}
$$

which yield the simplified equations of motion

$$
\begin{aligned}
\frac{dx^{(1)}}{dt} &= \frac{1}{m} P^{(1)}, \\
\frac{dx^{(2)}}{dt} &= \frac{1}{m} \left(P^{(2)} - q B_0 x^{(1)} \right), \\
\frac{dx^{(3)}}{dt} &= \frac{1}{m} P^{(3)}, \\
\frac{dP^{(1)}}{dt} &= q E^{(1)} + \frac{q}{m} \left[B_0 \left(P^{(2)} - q B_0 x^{(1)} \right) \right], \\
\frac{dP^{(2)}}{dt} &= q E^{(2)}, \\
\frac{dP^{(3)}}{dt} &= q E^{(3)}.
\end{aligned}
$$

The setup for the test consists of a single particle with mass $m = 1.0$ and charge $q = -1.0$ whose initial position is at the origin of the domain i.e., $\mathbf{x}(0) = (0, 0, 0)$. Initially, the particles have momentum components only in the x and z directions to generate so called “cyclotron" motion. We choose the initial momenta to be $\mathbf{p}(0) = \mathbf{P}(0) = \left(1.0 \times 10^{-2}, 0, 1.0 \times 10^{-2}\right)$. The strength of the magnetic field in the z direction is selected to be $B_0 = 1.0$, and we ignore the contributions from the electric field, so that $\mathbf{E} = (0, 0, 0)$. Both methods are run to a final time of $T = 30.0$ and a total of 1000 time steps are used so that $\Delta t = 0.03$. Lastly, we select the coupling parameter $\omega = 100$ for the Molei Tao integrator. The particle’s position is tracked through time and plotted as a curve in 3-D. Figure 4.1 compares the particle trajectories obtained with both methods.

Next, we perform a refinement study of the two methods to examine the error properties using the same experimental parameters from the cyclotron test. Errors are measured using reference



(a) Boris method

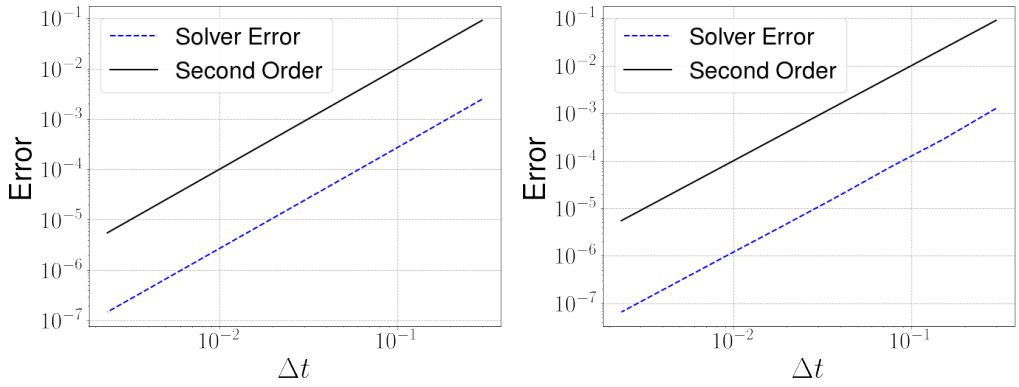
(b) Tao method ($\omega = 100$)

Figure 4.1: Trajectories for the single particle test, which are obtained using the Boris method 4.1a and the second-order integrator by Molei Tao (as presented in [47]) 4.1b. Both methods produce identical trajectories under identical experimental conditions. The particles rotate about the magnetic field which points in the z-direction.

solutions computed with 10^6 time steps, so that $\Delta t = 3.0 \times 10^{-5}$. Errors are measured using the ℓ_∞ norm. The test starts with using a total of 100 time steps and successively doubles the number of steps, using, at most, 1.28×10^4 steps. The results of the refinement study are shown in Figure 4.2. Both methods refine to second-order, with the Boris method producing a larger error in the solution compared to the Molei Tao integrator. While it may be possible to further reduce the size of the errors made by the Molei Tao method through better choices of the coupling parameter ω, the Boris method will likely remain more efficient in terms of error for a given amount of compute time. Despite the fact that we are not reporting timing results, the Boris method was notably faster than the Molei Tao integrator. This is not all that surprising given that the latter method requires more intermediate steps.

4.6.2 The Cold Two-Stream Instability

We consider the motion of “cold" streams of ions and electrons restricted to a one-dimensional periodic domain by a sufficiently strong (uniform) magnetic field in the two remaining directions.



(a) Boris method

(b) Tao method ($\omega = 100$)

Figure 4.2: Self-refinement for the single particle test using the Boris method 4.1a and the second-order integrator by Molei Tao (as presented in [47]) 4.1b. Second-order accuracy is achieved by both methods, but the ℓ_∞ errors for the Boris method are nearly a factor of 2 larger than those produced by the Molei Tao method. While we have not presented timing results, it is worth noting that the run times for the Boris method were considerably faster than those of the Molei Tao method due to the latter's additional "stages". The final error measurements taken from the refinement study are 1.4728×10^{-7} (Boris) and 6.5592×10^{-8} (Tao).

Ions are taken to be uniformly distributed in space, and sufficiently heavy compared to the electrons so that their motion can be ignored in the simulation. While the ions remain stationary, they act as a neutralizing background against the dynamic electrons. Mathematically, the electron velocities are represented as a sum of two Dirac delta distributions which are symmetric in velocity about the origin, i.e., streams move in opposite directions but have the same velocity magnitude. A slight perturbation in the electron velocities is then introduced to force a charge imbalance, as some particles move faster than others. This, in turn, generates an electric field that attempts to restore the neutrality of the system, causing the streams to interact, or "roll-up", creating regions of trapped particles.

In order to describe the models used in the simulation, let us denote the components of the position and momentum vectors for particle i as $\mathbf{x}_i \equiv \left(x_i^{(1)}, x_i^{(2)}, x_i^{(3)}\right)$ and $\mathbf{P}_i \equiv \left(P_i^{(1)}, P_i^{(2)}, P_i^{(3)}\right)$,

Models	Time Integration	Fields + Derivatives
$-\Delta\psi = \frac{1}{\sigma_1}\rho$	Leapfrog, Tao (with averaging)	FFT + FFT
$\frac{1}{\kappa^2}\partial_{tt}\psi - \Delta\psi = \frac{1}{\sigma_1}\rho$	Leapfrog, Tao (with averaging)	BDF-2 + BDF-2, Central-2 + BDF-2, Central-2 + BDF-4

Table 4.2: Summary of the algorithms explored for the two-stream instability example. Both time integration methods considered are second-order.

respectively, the equations for the motion of particle i assume the form

$$\frac{dx_i^{(1)}}{dt} = \frac{1}{r_i}P_i^{(1)},$$
$$\frac{dP_i^{(1)}}{dt} = -q_i\partial_x\psi.$$

Therefore, the motion in this plane requires knowledge of ψ, $\partial_x\psi$, which can be obtained by solving a two-way wave equation for the scalar potential:

$$\frac{1}{\kappa^2}\frac{\partial^2\psi}{\partial t^2} - \Delta\psi = \frac{1}{\sigma_1}\rho. \tag{4.32}$$

As this is an electrostatic problem, the gauge condition can be safely ignored. In the limit where $\kappa \gg 1$, the characteristic thermal velocities of the particles become well-separated from the speed of light. Rather than solve the two-way wave equation, one instead solves the Poisson equation

$$-\Delta\psi = \frac{1}{\sigma_1}\rho. \tag{4.33}$$

Using asymptotic analysis, it can be shown that the approximation error made by employing the Poisson model for the scalar potential is $O(1/\kappa)$ [42].

We benchmark the performance of several combinations of algorithms (see Table 4.2) for time stepping particles and evolving fields by comparing with well-known methods. This will help establish the baseline properties for the methods, and reduce the parameter space of viable methods. The setup for this test problem employs a spatial mesh defined on the interval $[-10\pi/3, 10\pi/3]$, which is discretized using 128 total grid points and supplied with periodic boundary conditions. The non-dimensional final time for the simulation is taken to be $T_f = 50.0$ with 4,000 time steps being used to evolve the system. The plasma is represented using a total of 20,000 macro-particles,

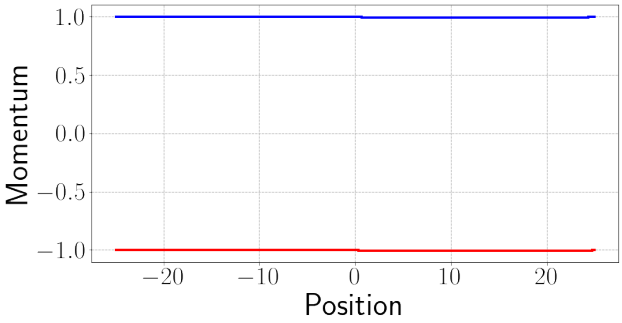


Figure 4.3: Initial configuration of electrons used in the two-stream experiments.

which are split equally between ions and electrons. As mentioned earlier, the positions of the ions and electrons are taken to be uniformly spaced along the grid. Ions remain stationary in the problem so we set their velocity to zero. The construction of the streams begins by first splitting the electrons into two equally sized groups, whose respective (non-dimensional) drift velocities are set to be ± 1. To generate an instability we add a perturbation to the electron velocities of the form

$$\epsilon \sin\left(\frac{2\pi k(x-a)}{L}\right).$$

Here, $\epsilon = 5\times 10^{-3}$ controls the perturbation strength, $k = 1$ is the wave number for the perturbation, x is the position of the electron, a is the left-most grid point, and L is the length of the domain. In a more physically realistic simulation, the perturbation would be induced by some external force, which would also result in a perturbation of the position data for the particles. Such a perturbation of the position data requires a self-consistent field solve to properly initialize the potentials. In our simulation, we assume that no spatial perturbation is present, so that the fields are identically zero at the initial time step. A plot of the electron streams at the initial condition is shown in Figure 4.3.

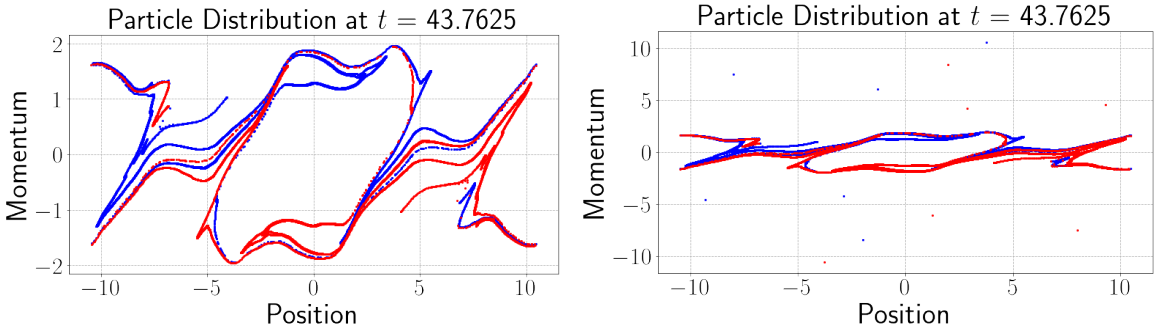
The plasma parameters used in the non-dimensionalization for this test problem, are displayed in Table 4.3. Note that under these scales, the normalized speed of light $\kappa = 50$. In some sense, this value is close to the relativistic regime, but far enough away to avoid the need for relativistic time integration methods. Additionally, we find that this configuration is sufficient to resolve the Debye length (≈ 6 cells/λ_D), angular plasma period (≈ 80 steps/ω_{pe}), and the particle CFL < 1, which are all necessary for maintaining stability in explicit PIC methods.

In order to get a sense of the behavior attributed to the particle integrator, we first considered the

Parameter	Value
Average number density ($\bar{n}$) [m^{-3}]	7.856060×10^{1}
Average temperature ($\bar{T}$) [K]	2.371698×10^{6}
Debye length (λ_D) [m]	1.199170×10^{4}
Inverse angular plasma frequency (ω_{pe}^{-1}) [s/rad]	2.000000×10^{-3}
Thermal velocity (v_{th}) [m/s]	5.995849×10^{6}

Table 4.3: Table of the plasma parameters used in the two-stream instability example.

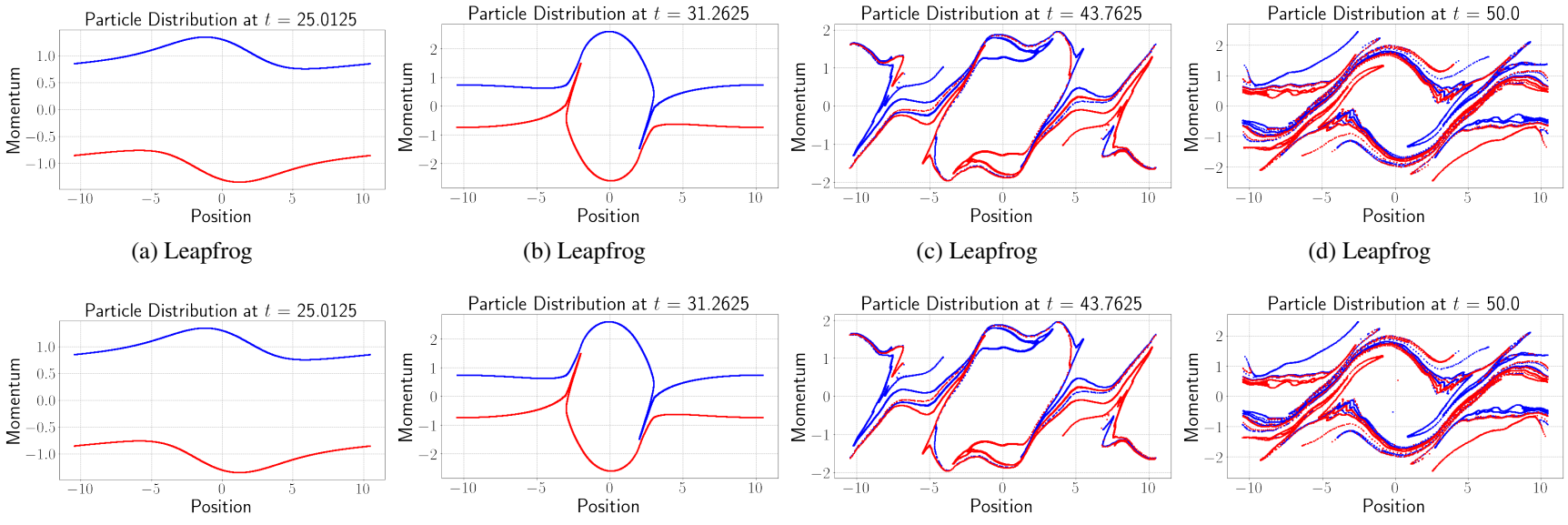
Poisson model (4.33) for the scalar potential. Since the combination of leapfrog time integration with an FFT field solver is such a commonly used approach to this problem, it allowed us to identify key differences attributed solely to the choice of time integration method used for particles. We found that a direct application of the Molei Tao integrator to this problem, which includes the corresponding field solves, gave nonphysical results. As the streams began to develop interesting structures, the particles appeared to "jump" off their smooth trajectories manifesting as some form of noise. This can be attributed to the duplicate field and particle data required by Molei Tao's method, which are not guaranteed to remain close. Eventually this leads to differences in the potentials that are used to move the particles. Adjustments to the coupling parameter in Molei Tao's method were unsuccessful at controlling this behavior, and, in fact, exacerbated the phenomenon. Having to deal with a parameter in a method, which can lead to exceptionally nonphysical results is a highly undesirable feature of a method. In an attempt to fix this problem, we chose to adjust the particle data at the end of the time step by replacing the values with the averages of $\left(\mathbf{x}^{n+1}, \mathbf{Q}^{n+1}\right)$ and $\left(\mathbf{y}^{n+1}, \mathbf{P}^{n+1}\right)$. In Figure 4.4, we compare the Molei Tao method and demonstrate the behavior of the method with and without averaging. This approach while most likely not symplectic, seems to be reasonably effective at controlling this difference and the number of particles that leave the stream lines. Therefore, note that this modification shall be used in all subsequent experiments that use the Molei Tao integrator. Having (somewhat) resolved the issue posed by the Molei Tao method, we then compared this integrator against the Leapfrog method using the Poisson model (4.33) for the scalar potential. We present snapshots of the electron streams obtained with both methods in Figure 4.5. We observe nearly identical behaviors from both methods, with the exception of later times, at which point, several particles have moved off their trajectories.



(a) With averaging

(b) Without averaging

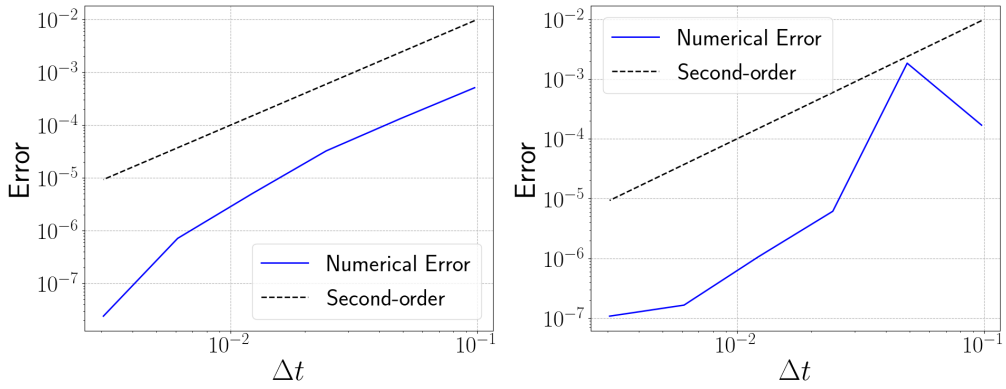
Figure 4.4: A comparison of the Molei Tao particle integrator with and without averaging for the two-stream example with the Poisson model. Over time, the pairs of phase space data, including the associated fields, can grow apart leading to vastly different potentials that kick particles off their smooth trajectories. Averaging appears to be fairly effective at controlling this behavior.



(a) Leapfrog (b) Leapfrog (c) Leapfrog (d) Leapfrog

(e) Tao with averaging (ω = 500) (f) Tao with averaging (ω = 500) (g) Tao with averaging (ω = 500) (h) Tao with averaging (ω = 500)

Figure 4.5: We present plots of the electrons in phase space obtained using the Poisson model for the two-stream example. Results obtained using leapfrog time integration are shown in the top row, while the bottom row uses the second-order integrator based on Molei Tao and applies averaging. We selected ω = 500 as the value of the coupling parameter in the Molei Tao integrator. The FFT is used to compute the scalar potentials in both methods. At later times, despite improvements from "averaging" the particle data, the Tao method causes particles to move off the stream lines. This phenomena is a numerical artifact that is not present in the leapfrog method.



(a) Leapfrog

(b) Tao with averaging ($\omega = 500$)

Figure 4.6: Time refinement of a tracer particle's position for the two-stream instability using the Poisson model for the potential with leapfrog (a) and the Molei Tao integrator with averaging (b). We selected $\omega = 500$ as the value of the coupling parameter in the Molei Tao integrator. Both methods converge to second-order accuracy with leapfrog generally displaying a larger absolute error than the Tao method. The exception to this is the smallest Δt used in the leapfrog experiments.

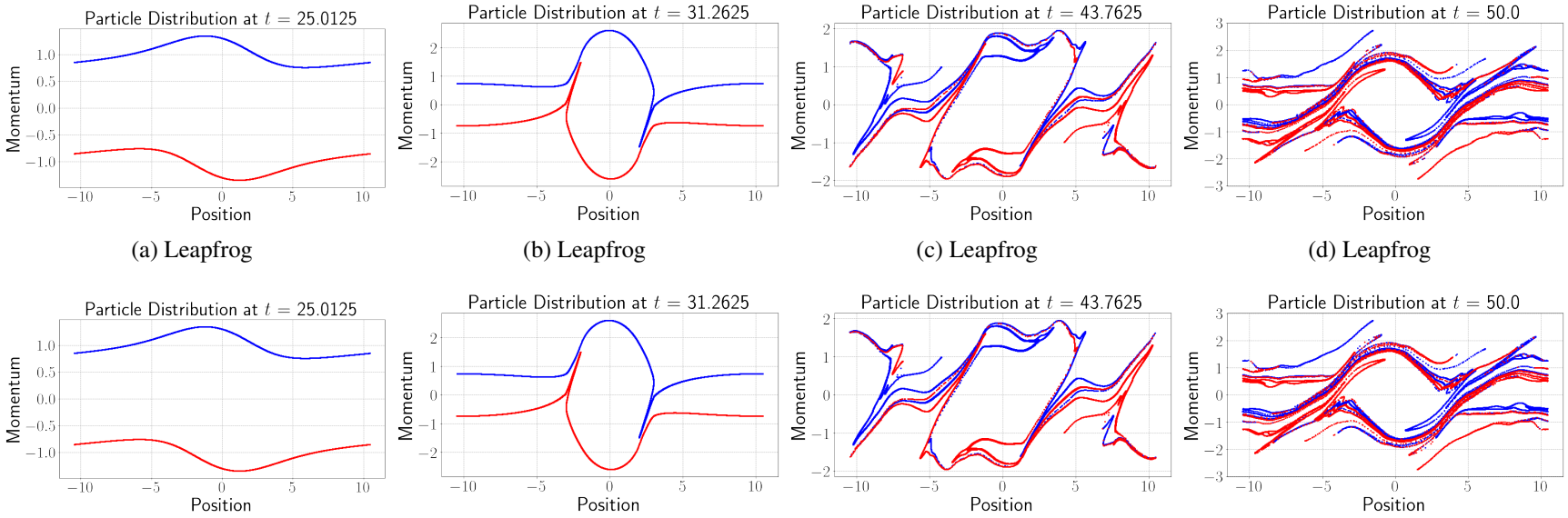
A time refinement experiment for the two-stream example with the Poisson model was also performed using the same numerical parameters outlined in Table 4.3. Errors were measured by following the trajectory of a single tracer particle, which is arbitrarily selected from the middle of the particle array, for several values of Δt. The location of this tracer particle, in each case, is stored at the non-dimensional final time $T_f = 25$, which we note is prior to the occurrence of the "roll-up" in the streams. The reference solution used to measure the error was computed using 16,384 time steps. We ran the code starting with 256 time steps and successively doubled this until reaching 8192 steps. Errors were measured using the ℓ_∞ norm, which in one dimension, becomes the absolute value. We plot the errors against the time step size in Figure 4.6. The plots show second-order time accuracy in both methods, with cleaner refinement behavior observed in the leapfrog integrator despite a larger overall error. Additionally, the Molei Tao integrator displays a noticeable increase in the error when larger time steps are taken. It is worth noting that the initial values of Δt (e.g., where the jump in the error occurs) are in violation of the particle CFL number, which should be < 1 for explicit particle methods; however, this jump in the error is not observed in the leapfrog method, which is known for its long-time accuracy. Moreover, it is difficult to identify the exact time at which the breakdown in the Molei Tao method occurs. The accuracy guarantees

are given by Tao in an asymptotic form that involves the duration of the simulation, the order of the method, and the value of the coupling parameter. Additionally, these estimates were obtained only in the case where of particles moving through fields known at all positions in space and time, so these estimates may no longer be valid for the dynamic fields considered in this work.

This same experiment was repeated using the two-way wave model (4.32) in the place of the Poisson model (4.33) for the scalar potential. For problems which are strongly electrostatic (i.e., $\kappa \gg 1$), the wave model should produce results which are similar to those of the Poisson model shown in Figure 4.5. This setting allows us to benchmark the performance of the wave solvers and methods for derivatives discussed in chapter 2. We include the second-order time-centered and BDF field solvers, as well as derivative methods based on BDF-2 and BDF-4 discretizations in this experiment. Recall that in section 2.5.1, we mentioned concerns of stability for field solvers based on BDF-3 and BDF-4 discretizations. Here, we are using the higher-order methods only to compute derivatives, which are not evolved in time, so stability less of a concern. Moreover, since this problem is one-dimensional in space, we avoid the splitting error, so the derivatives will be more accurate. We considered three pairings of these methods: (1) BDF-2 with BDF-2, (2) central-2 with BDF-2, and (3) central-2 with BDF-4. Results obtained with each of these pairings for the field solvers are shown in Figures 4.7, 4.8, and 4.9, respectively. A notable difference with the Poisson model is that particles stayed attached to their smooth trajectories. This can be attributed to the use of a wave model, which, due to the finite speed of propagation, responds more slowly to changes in the charge density. Among the results for the wave models, we observe excellent agreement between the leapfrog and Molei Tao integrators. In these experiments, the run parameters we selected gave a CFL ≈ 3.79, which is not large enough to see noticeable improvements gained by moving to higher-order methods. Nevertheless, the overall consistency among the results is quite encouraging. We note that time averaging is used on the charge density in solvers which combine Molei Tao with the central-2 scheme for the fields (see section 4.5.1.2 for details). Without this averaging, the streams interact at an accelerated rate, which is nonphysical.

A time refinement of the proposed methods was also performed following the same procedure

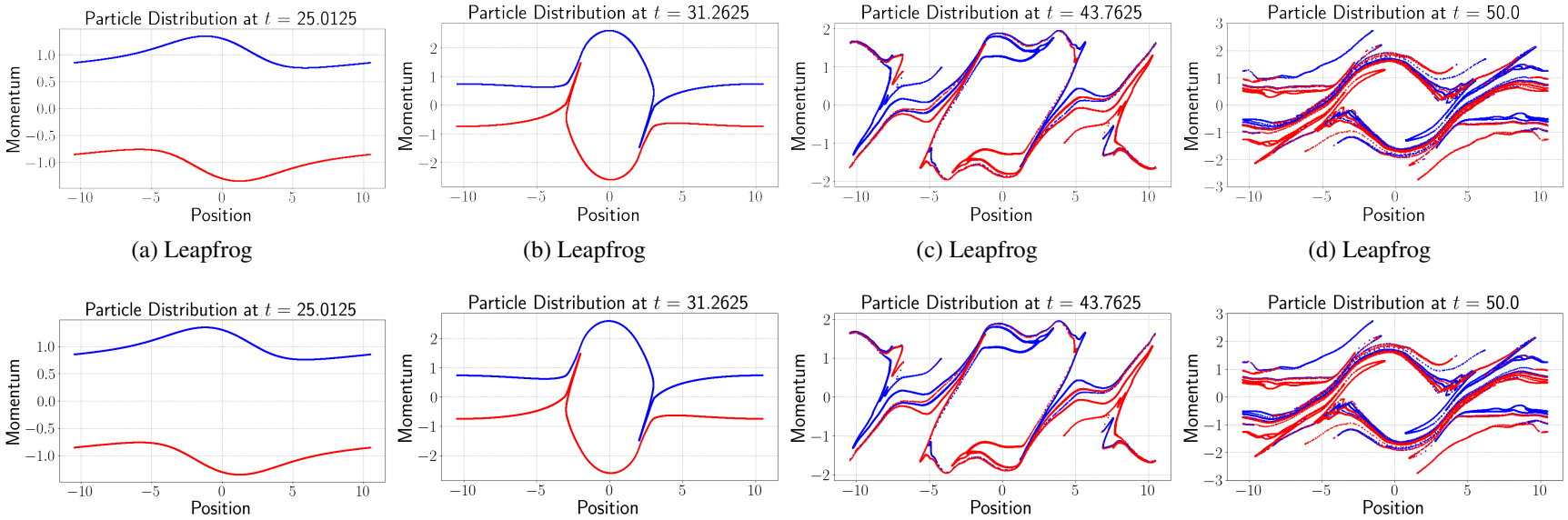
used for the Poisson model based on a tracer particle. The results of the refinement study are presented in Figure 4.10. We (generally) observe second-order temporal accuracy with each of the methods. When leapfrog is used to move the particles, we observe fairly clean second-order accuracy. The Molei Tao method, in contrast, shows some irregularities in the refinement pattern, which in some cases appears to diverge when large time steps are used. This is likely due to the time step simply being too large for an explicit method, despite satisfying the CFL condition for the particles. Additionally, it is worth noting that the error in the methods with Molei Tao display a smaller error. In terms of efficiency, however, the increased number of field solves required by the Molei Tao method may not be offset by this improvement in the error.



(a) Leapfrog (b) Leapfrog (c) Leapfrog (d) Leapfrog

(e) Tao with averaging ($\omega = 500$) (f) Tao with averaging ($\omega = 500$) (g) Tao with averaging ($\omega = 500$) (h) Tao with averaging ($\omega = 500$)

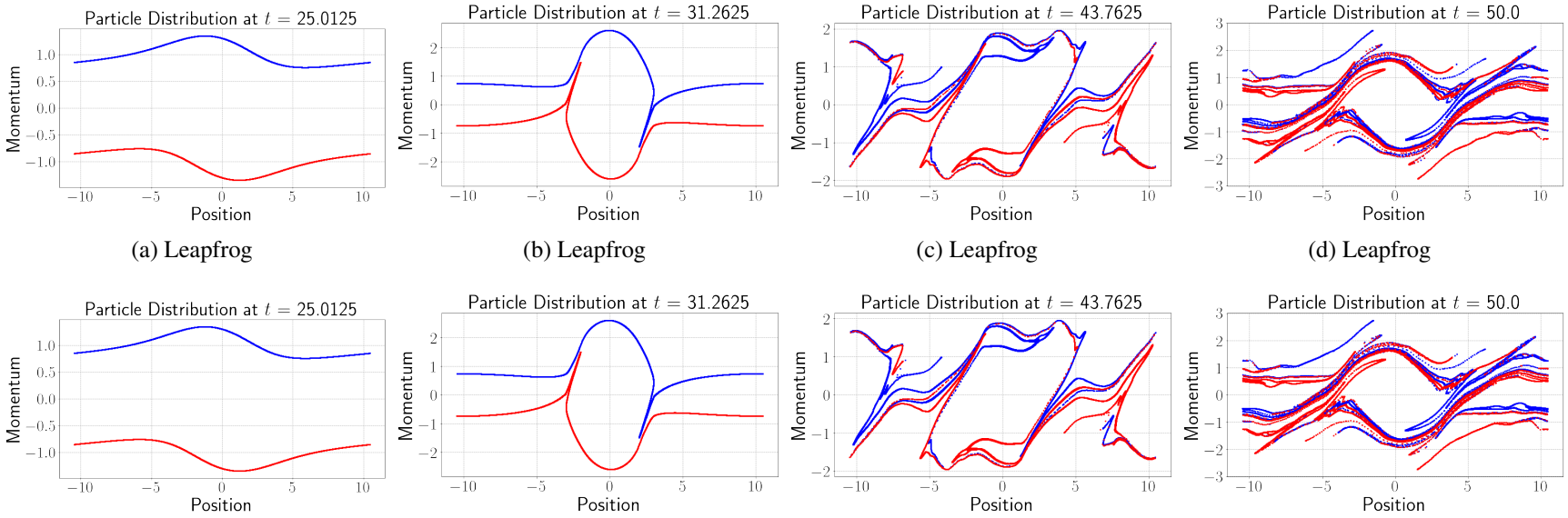
Figure 4.7: We present plots of the electrons in phase space obtained using the wave model for the two-stream example. Results obtained using leapfrog time integration are shown in the top row, while the bottom row uses the second-order integrator based on Molei Tao and applies averaging. We selected $\omega = 500$ as the value of the coupling parameter in the Molei Tao integrator. The second-order (diffusive) BDF scheme (BDF-2) is used to compute the scalar potentials and their derivatives in for both methods. Unlike the results obtained with the Poisson model, which used the FFT as the field solver (shown in Figure 4.5), the particles at the later times in the Molei Tao method seem to stay attached to their trajectories.



(a) Leapfrog (b) Leapfrog (c) Leapfrog (d) Leapfrog

(e) Tao with averaging (ω = 500) (f) Tao with averaging (ω = 500) (g) Tao with averaging (ω = 500) (h) Tao with averaging (ω = 500)

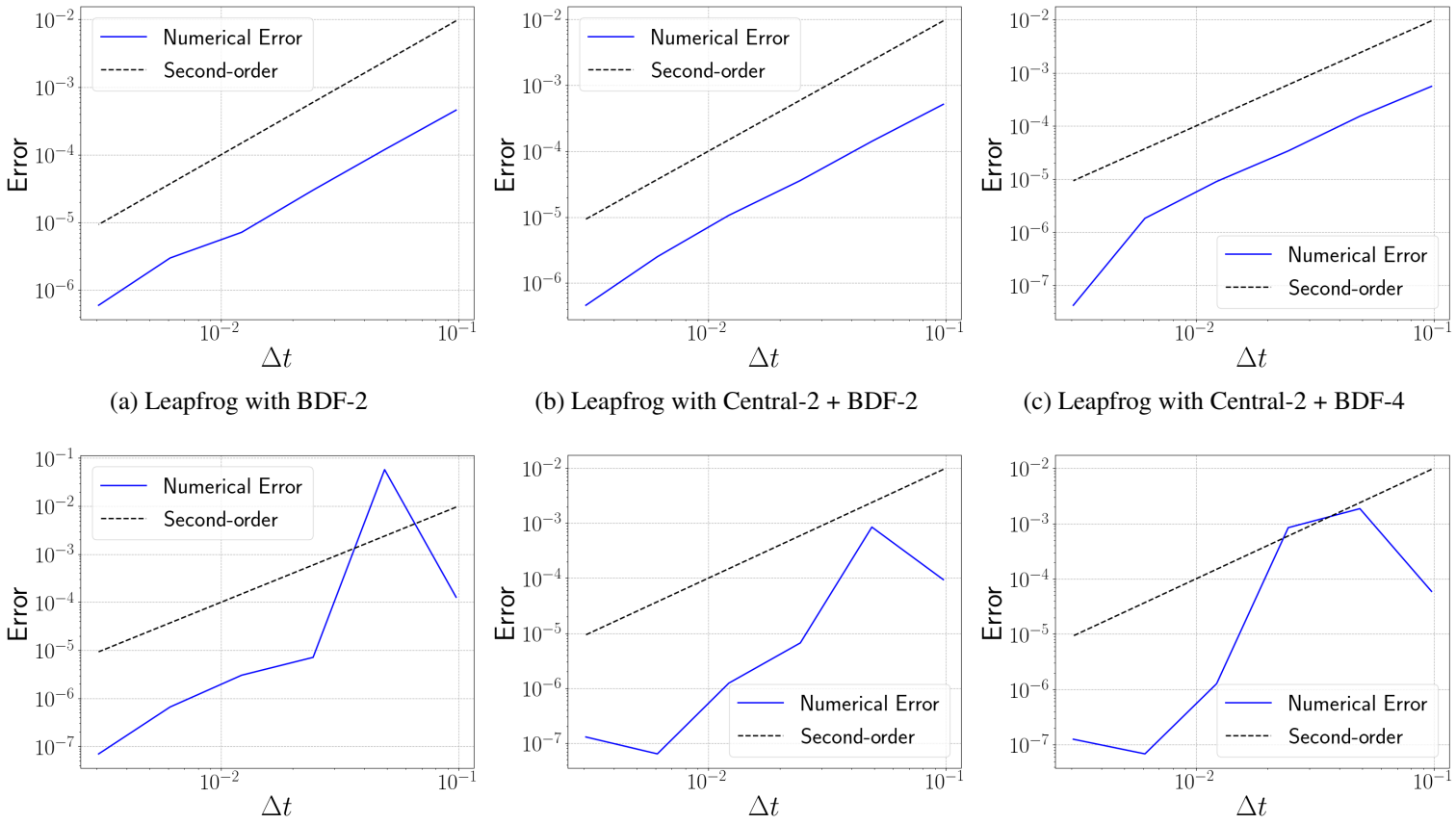
Figure 4.8: We present plots of the electrons in phase space obtained using the wave model for the two-stream example. Results obtained using leapfrog time integration are shown in the top row, while the bottom row uses the second-order integrator based on Molei Tao and applies averaging. We selected ω = 500 as the value of the coupling parameter in the Molei Tao integrator. The scalar potentials are evolved using the second-order central scheme (central-2), while the derivatives are computed at each step with the second-order BDF scheme (BDF-2). In the bottom row, which uses the Molei Tao method, we obtain results that are similar to the BDF-2 method (see 4.7) in the sense that particles do not seem to jump off of their trajectories.



(a) Leapfrog (b) Leapfrog (c) Leapfrog (d) Leapfrog

(e) Tao with averaging (ω = 500) (f) Tao with averaging (ω = 500) (g) Tao with averaging (ω = 500) (h) Tao with averaging (ω = 500)

Figure 4.9: We present plots of the electrons in phase space obtained using the wave model for the two-stream example. Results obtained using leapfrog time integration are shown in the top row, while the bottom row uses the second-order integrator based on Molei Tao and applies averaging. We selected ω = 500 as the value of the coupling parameter in the Molei Tao integrator. The scalar potentials are evolved using the second-order central scheme (central-2), while the derivatives are computed at each step with the fourth-order BDF scheme (BDF-4). As with the other wave solver methods, the particles in the Molei Tao experiments seem to stay attached to their smooth trajectories, even at the later times.



(a) Leapfrog with BDF-2 (b) Leapfrog with Central-2 + BDF-2 (c) Leapfrog with Central-2 + BDF-4

(d) Tao with BDF-2 (e) Tao with Central-2 + BDF-2 (f) Tao with Central-2 + BDF-4

Figure 4.10: Time refinement of a tracer particle's position for the two-stream instability. For the particle push, we consider both leapfrog and the Molei Tao method with averaging, in combination with different methods for fields and their derivatives. We selected $\omega = 500$ as the value of the coupling parameter in all of the Molei Tao integrator experiments. Each of the methods converge to second-order accuracy with the error in the Tao method being smaller than leapfrog.

Parameter	Value
Average number density ($\bar{n}$) [m^{-3}]	1.129708×10^{14}
Average temperature ($\bar{T}$) [K]	2.371698×10^{6}
Debye length (λ_D) [m]	1.000000×10^{-2}
Inverse angular plasma frequency (ω_{pe}^{-1}) [s/rad]	1.667820×10^{-9}
Thermal velocity (v_{th}) [m/s]	5.995849×10^{6}

Table 4.4: Table of the plasma parameters used in the numerical heating example.

4.6.3 Numerical Heating Study

We now perform a numerical heating study using the same combination of models and algorithms shown in Table 4.2 for the two-stream problem. The primary purpose of this test is to characterize the effect of resolving the Debye length λ_D in a steady-state problem to the Vlasov equation. Moreover, it allows us to benchmark the degree of the heating phenomenon observed under different selections of models, particle integrators, and field solvers. These numerical properties turn out to be connected to the symplecticity of the method. Explicit PIC methods are not symplectic because the fields and the particles are not self-consistent with one another. A consequence of this is that the grid should be sufficiently fine so that a given particle can "see" the correct potential, which is otherwise screened by particles of opposite charge. In other words, with explicit PIC methods, one needs to resolve the charge separation inside of the plasma, which is determined by the Debye length. A general rule of thumb for explicit PIC simulations is that the grid spacing Δx should be chosen to satisfy $4\Delta x < \lambda_D$. The phenomenon of heating occurs in simulations which do not adequately resolve this scale. In such cases, the system will try to increase the temperature of the plasma until it becomes adequately resolved on the given mesh. Fully-implicit methods, which break this restriction [9], allow for a substantially coarser mesh to be used for a given calculation and will be the subject of future work.

The setup for this problem is slightly different from the two-stream example discussed earlier. Here, we provide, as input a Debye length λ_D as well as a normalized speed of light κ, which can be used to calculate the average number density $\bar{n}$ and macroscopic temperature $\bar{T}$ for the plasma. The remaining parameters related to the plasma can be derived from these values and are shown in

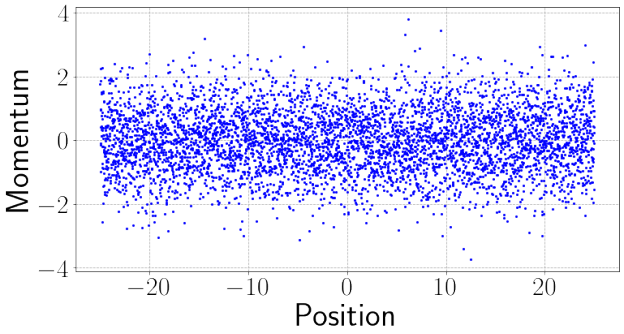


Figure 4.11: Initial electron data in phase space used for the numerical heating tests.

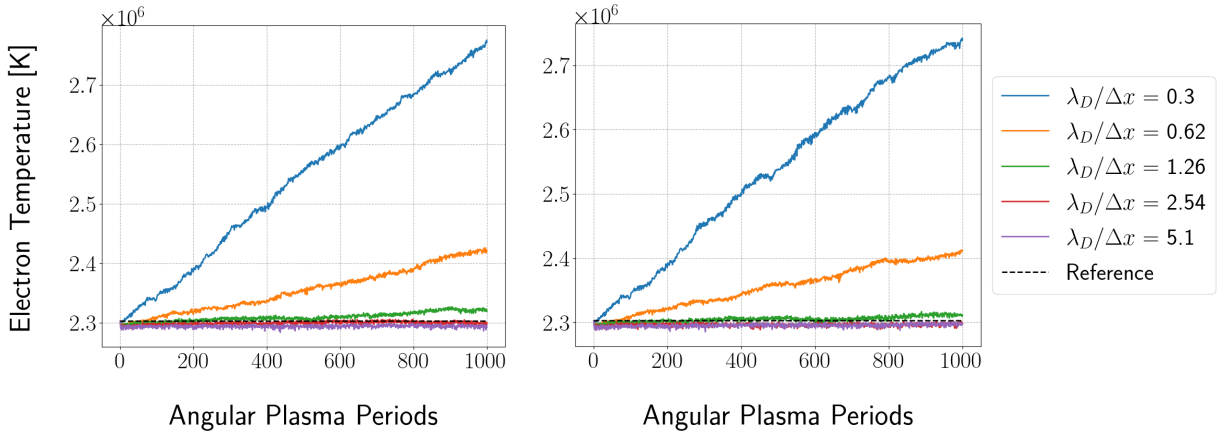
Table 4.4. The non-dimensional grid used in this problem is taken to be periodic on the interval $[-25, 25]$. This grid is refined by successively doubling the number of mesh points from 16 to 256. In each case, the simulation uses 5×10^5 time steps to the non-dimensional final time $T_f = 1 \times 10^3$.

A total of 5×10^3 particles are used for each species in the simulation, which consist of ions and electrons. As before, we assume that ions will remain stationary since they are heavier than the electrons. Electrons are given uniform positions in space and their velocities are initialized by sampling from the standard normal distribution. There is no drift velocity present in this problem. This non-dimensional standard normal distribution corresponds to a Maxwellian distribution that has mean zero and a temperature $\bar{T}$. To ensure consistency across the runs, we seed the random number generator. A plot of the electrons in phase space at the initial condition is displayed in Figure 4.11.

In order to monitor heating during the simulations, we track the time history of the variance for the electron velocities, which is connected to the temperature of a Maxwellian distribution. Note that the variance data at $N + 1$ time levels $\{\text{var}(v^n)\}_{n=0}^N$ can be converted to a temperature history using

$$\left\{\bar{T}^n\right\}_{n=0}^N = \frac{m_e}{k_B} \left\{\text{var}(v^n)\right\}_{n=0}^N .$$

The results of our heating study can be found in Figures 4.12 and 4.13, which represent the Poisson and wave models for the potential, respectively. In the case of the Poisson model, identical heating properties are observed when particles are integrated with either leapfrog or the averaged version of the Molei Tao integrator. The degree of heating becomes noticeably larger as the

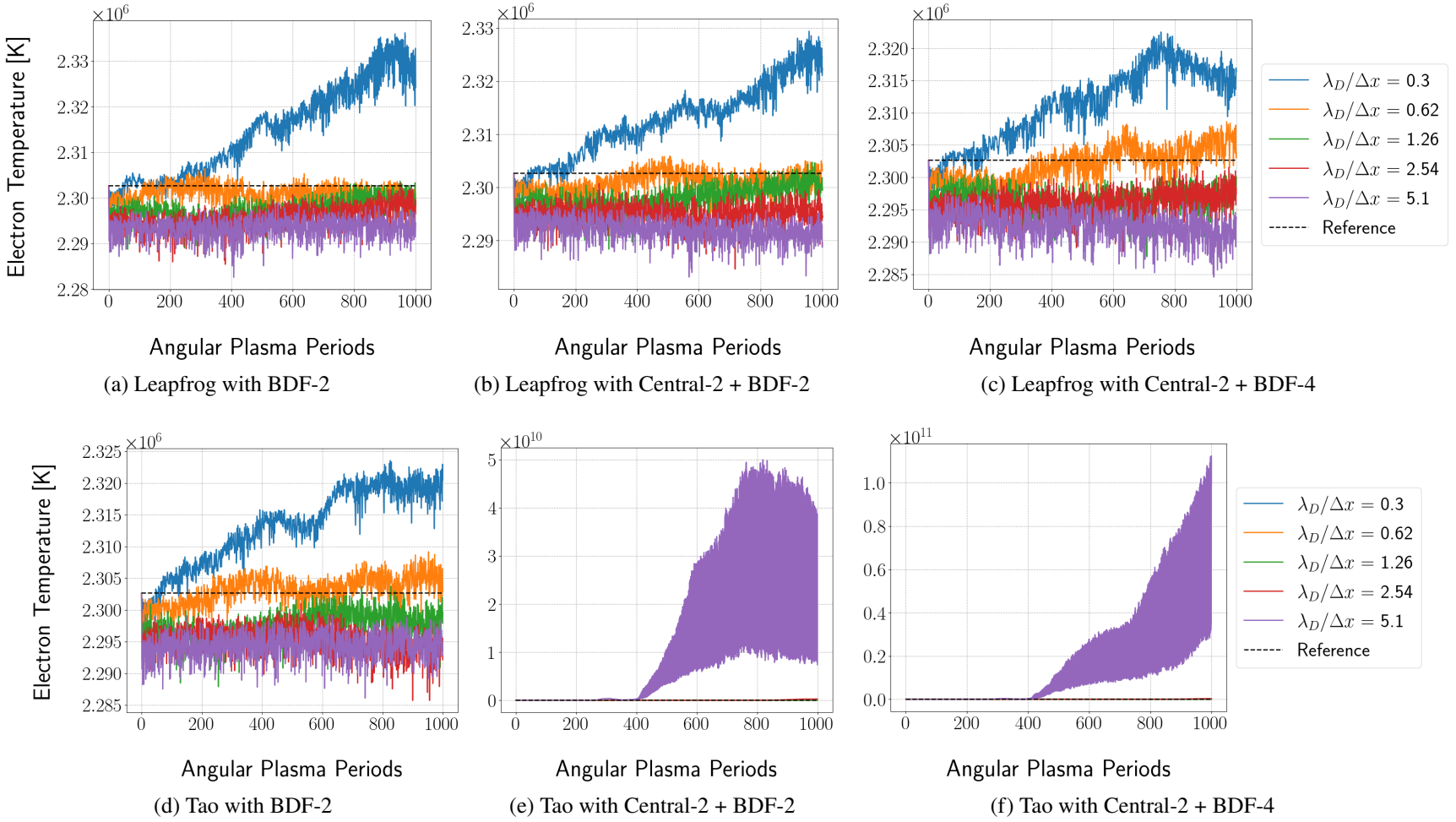


(a) Leapfrog with FFT

(b) Tao with FFT

Figure 4.12: We present results from the numerical heating tests based on the Poisson model. Plots show the average electron temperature as a function of the number of angular plasma periods using leapfrog (left) and the second-order integrator by Molei Tao with averaging (right). Fields and their derivatives are obtained using the FFT.

grid is coarsened. Similar behaviors are observed in the wave model for the potential with some caveats. Firstly, we observe a less substantial degree of heating in cases where the Debye length is underresolved due to the finite speed of propagation in the wave model. Additionally, we see that two of the configurations, specifically the ones which combine Molei Tao with the time-centered method, become unstable in time; however, the same approaches with leapfrog behave as expected. A likely source of the problem is the time averaging applied to the source terms used in the Molei Tao method, which is not applied in other methods that use either the Molei Tao integrator or the time-centered scheme for the scalar potential.



(a) Leapfrog with BDF-2 (b) Leapfrog with Central-2 + BDF-2 (c) Leapfrog with Central-2 + BDF-4

(d) Tao with BDF-2 (e) Tao with Central-2 + BDF-2 (f) Tao with Central-2 + BDF-4

Figure 4.13: We display results from the numerical heating tests that use the wave model for the potentials. Plots show the average electron temperature as a function of the number of angular plasma periods using leapfrog (top) and the second-order integrator by Molei Tao with averaging (bottom). We selected $\omega = 500$ as the value of the coupling parameter in the Molei Tao integrator. The scalar potentials and derivatives are computed with the scheme label provided in the individual captions.

4.6.4 The Bennett Equilibrium Pinch

To benchmark the performance of our method on electromagnetic problems, we first consider the Bennett Equilibrium pinch, named after its discoverer W. H. Bennett, who first analyzed the problem [81]. In this paper, Bennett constructed a certain steady-state solution for the ideal MHD equations in cylindrical coordinates. Electrons are modeled as a fluid that drifts along the z-direction, creating currents that generate magnetic fields with x and y components that confine or "squeeze" the plasma towards the axis of the cylinder. The fluid velocity along the axis of the cylinder is carefully chosen to create a proper equilibrium balance between the plasma and the confining magnetic field that surrounds it.

Our particle simulation of the Bennett pinch closely follows the description provided in chapter 13 of Bittencourt [82]. We consider the motion of electrons inside a cross-section of a cylindrical column of plasma whose radius is R_b and suppose that the axis of the beam is centered at the origin of a bounding box $\Omega = [-2R_b, 2R_b] \times [-2R_b, 2R_b]$. Let $\mathbf{J} = \left(J^{(1)}, J^{(2)}, J^{(3)}\right)$ denote the components of the current density. In the x-y plane, the particles are sampled from a Maxwellian distribution, so we ignore the contributions from $J^{(1)}$ and $J^{(2)}$ because they will have a net-zero current. Since the particles drift along the axis of the beam, we retain the third component of the current density $J^{(3)}$. Consequently, we ignore the wave equations for $A^{(1)}$ and $A^{(2)}$, choosing to retain only $A^{(3)}$. Ions are spatially distributed by sampling the same distribution as the electrons and are assumed to be stationary within the cross-section, since their mass is taken to be much larger than the electrons. If we denote the components of the position and momentum vectors for particle i as $\mathbf{x}_i \equiv \left(x_i^{(1)}, x_i^{(2)}, x_i^{(3)}\right)$ and $\mathbf{P}_i \equiv \left(P_i^{(1)}, P_i^{(2)}, P_i^{(3)}\right)$, respectively, the equations for the

motion of each particle assume the form

$$\begin{aligned}
\frac{dx_i^{(1)}}{dt} &= \frac{1}{r_i} P_i^{(1)}, \\
\frac{dx_i^{(2)}}{dt} &= \frac{1}{r_i} P_i^{(2)}, \\
\frac{dP_i^{(1)}}{dt} &= -q_i \partial_x \psi + \frac{q_i}{r_i} \left(\partial_x A^{(3)} \right) \left(P_i^{(3)} - q_i A^{(3)} \right), \\
\frac{dP_i^{(2)}}{dt} &= -q_i \partial_y \psi + \frac{q_i}{r_i} \left(\partial_y A^{(3)} \right) \left(P_i^{(3)} - q_i A^{(3)} \right).
\end{aligned}$$

Note that while we retain the last component of the momentum, we do not include an equation for $P_i^{(3)}$, since each term involves z-derivatives which are ignored. In a more realistic simulation, we would retain the full 3D-3P system to monitor changes in $P_i^{(3)}$. That said, this test is still useful because it serves as a benchmark to assess the quality of particle confinement inside the beam.

The particle equations of motion for this problem can also be formulated in terms of **E** and **B** and solved with the Boris method discussed in section 4.4.2. Given the potentials ψ and **A**, we can calculate the **E** and **B** fields using (1.11):

$$\mathbf{E} = -\nabla\psi - \partial_t \mathbf{A}, \quad \mathbf{B} = \nabla \times \mathbf{A}.$$

These can be used to evolve particles through the non-dimensional model

$$\begin{aligned}
\frac{dx_i^{(1)}}{dt} &= v_i^{(1)}, \\
\frac{dx_i^{(2)}}{dt} &= v_i^{(2)}, \\
\frac{dv_i^{(1)}}{dt} &= \frac{q_i}{r_i} \left(E^{(1)} - v_i^{(3)} B^{(2)} \right), \\
\frac{dv_i^{(2)}}{dt} &= \frac{q_i}{r_i} \left(E^{(2)} + v_i^{(3)} B^{(1)} \right), \\
\frac{dv_i^{(3)}}{dt} &= \frac{q_i}{r_i} \left(\left[v_i^{(1)} B^{(2)} - v_i^{(2)} B^{(1)} \right] \right),
\end{aligned}$$

where the last equation for the velocity is neglected so that this form is consistent with the generalized momentum formulation.

Parameter	Value
Beam radius (R_b) [m]	1.0×10^{-6}
Average number density ($\bar{n}$) [m^{-3}]	4.391989×10^{19}
Average temperature ($\bar{T}$) [K]	5.929245×10^{1}
Debye length (λ_D) [m]	8.019042×10^{-8}
Inverse angular plasma frequency (ω_{pe}^{-1}) [s/rad]	2.674864×10^{-12}
Thermal velocity (v_{th}) [m/s]	2.997925×10^{4}
Electron drift velocity (v_{drift}) [m/s]	2.997925×10^{7}
Fraction of particles contained in the beam (α) [non-dimensional]	0.99

Table 4.5: Table of the parameters used in the setup for the Bennett pinch problem.

Particles that leave the domain due to inadequate confinement by the fields can be prescribed new positions by sampling the initial distribution, which essentially re-injects them into the beam. The velocity and momenta are left unchanged in an effort to keep the total current density constant. Next, we provide the experimental parameters for the simulations along with details regarding the initialization procedure for the problem.

During the initialization and time evolution phases of the simulation, we use an analytical solution for the toroidal magnetic field to verify the correctness of the numerical fields, which maintain the steady-state. To derive the analytical solution for the toroidal magnetic field $B^{(\theta)}$, we solve the differential equation (equation 13.3.5 in [82])

$$\frac{d}{dr}\left(rB^{(\theta)}\right) = \mu_0 q_e v_e^{(3)} r n(r),$$

where μ_0 is the permeability of free space, q_e is the electron charge, $v_e^{(3)}$ is the z-component of the electron velocity, and $n(r)$ is the *Bennett distribution* for the electrons (equation 13.3.8 in [82]) given as

$$n(r) = \frac{n_0}{\left(1 + n_0 b r^2\right)^2}. \tag{4.34}$$

The on-axis number density n_0 is calculated according to equation 13.3.14 of [82]

$$n_0 = \frac{1}{bR_b^2}\left(\frac{\alpha}{1-\alpha}\right),$$

where $\alpha \in [0, 1)$ is a parameter the fraction of particles in the cross-section contained within the

beam. We use the constant b, whose form is given in equation 13.3.9 of [82], namely

$$b = \frac{\mu_0 \left(q_e v_e^{(3)}\right)^2}{8 k_B \bar{T}}.$$

Note that the above expression is equivalent to assuming that the ions are cold so that $T_i = 0$ and $T_e = \bar{T}$. To solve this differential equation, we integrate both sides from 0 to r:

$$\int_0^r \frac{d}{dr'}\left(r' B^{(\theta)}(r')\right) dr' = \int_0^r \mu_0 q_e v_e^{(3)} \frac{n_0 r'}{\left(1 + n_0 b r'^2\right)^2} dr'.$$

The left side simplifies to $rB^{(\theta)}(r)$ and the right side can be evaluated using the substitution $\tau = 1 + n_0 b r'^2$ which yields

$$\begin{aligned} rB^{(\theta)}(r) &= \frac{\mu_0 q_e v_e^{(3)}}{2b} \int_1^{1+n_0 b r^2} \tau^{-2}\, d\tau, \\ &= \frac{\mu_0 q_e v_e^{(3)}}{2b}\left(1 - \frac{1}{1 + n_0 b r^2}\right), \\ &= \frac{\mu_0 q_e v_e^{(3)}}{2} \frac{n_0 r^2}{1 + n_0 b r^2}. \end{aligned}$$

Hence, the analytical solution for the toroidal magnetic field is

$$B^{(\theta)}(r) = \frac{\mu_0 q_e v_e^{(3)}}{2} \frac{n_0 r}{1 + n_0 b r^2}. \tag{4.35}$$

Numerically, we solve the problem in a Cartesian coordinate system, instead of a cylindrical coordinate system. Therefore, we will have to convert $B^{(1)} \equiv B^{(x)}$ and $B^{(2)} \equiv B^{(y)}$ to $B^{(\theta)}$. The required transformation that converts between these coordinate systems is

$$\begin{bmatrix} B^{(r)} \\ B^{(\theta)} \\ B^{(z)} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B^{(x)} \\ B^{(y)} \\ B^{(z)} \end{bmatrix},$$

Note that this transformation can be further simplified with $\cos(\theta) = \frac{x}{r}$ and $\sin(\theta) = \frac{y}{r}$, with $r > 0$, so that we obtain for the θ-component

$$B^{(\theta)}(r) = -\frac{y}{r} B^{(x)} + \frac{x}{r} B^{(y)}, \quad r = \sqrt{x^2 + y^2} > 0.$$

For the case of $r = 0$, we can appeal to the analytical solution (4.35), so that $B^{(\theta)}(0) = 0$.

The setup of our PIC simulation requires an average or macroscopic number density $\bar{n}$. This can be obtained from a variation of equation 13.3.11 in [82] to obtain

$$\bar{n} = \frac{2\pi}{16R_b^2} \int_0^{R_b} n(r) r \, dr = \frac{n_0 \pi R_b^2}{16R_b^2 \left(1 + n_0 b R_b^2\right)}, \tag{4.36}$$

where we have used the definition (4.34). A summary of the plasma parameters used in the simulation of the Bennett pinch is presented in Table 4.5.

Next, we describe the initialization procedure for the fields ψ and $\mathbf{A}$ that recovers the steady-state solution with enough accuracy to achieve particle confinement. This problem is defined over free-space, so we prescribe outflow boundary conditions along the boundary of the box that encloses the beam. To initialize the steady-state with the wave solvers, we first set $\psi = 0$ and $\mathbf{A} = 0$, and then proceed by stepping the corresponding wave equations to steady-state, holding the sources ρ and $\mathbf{J}$ fixed. Although it is more expensive than direct evaluation of the free-space integral solution, this technique allows for a self-consistent initialization of the data used by the outflow procedure for this problem. Further, since the data is stepped to steady-state, this approach is general enough that it can also be used to initialize both the Poisson and wave models. In Figure 4.14, we show plots of the steady-state toroidal magnetic field obtained this initialization procedure using a 128×128 mesh. The errors in the initial condition are primarily concentrated along the edge of the simulation domain and near the origin. Along the domain boundary, the errors here are primarily due to the explicit outflow procedure. Errors occurring near the origin are due to the steep gradients created by the beam, which can be corrected by using a finer mesh.

As a first test, we compare the Molei Tao integrator with the Boris method for the true steady-state problem, which is described by the elliptic system

$$\begin{aligned} -\Delta\psi &= \frac{1}{\sigma_1}\rho, \\ -\Delta A^{(3)} &= \sigma_2 J^{(3)}, \end{aligned}$$

which is defined over free-space. An approach based on Green's functions is quite natural, as the

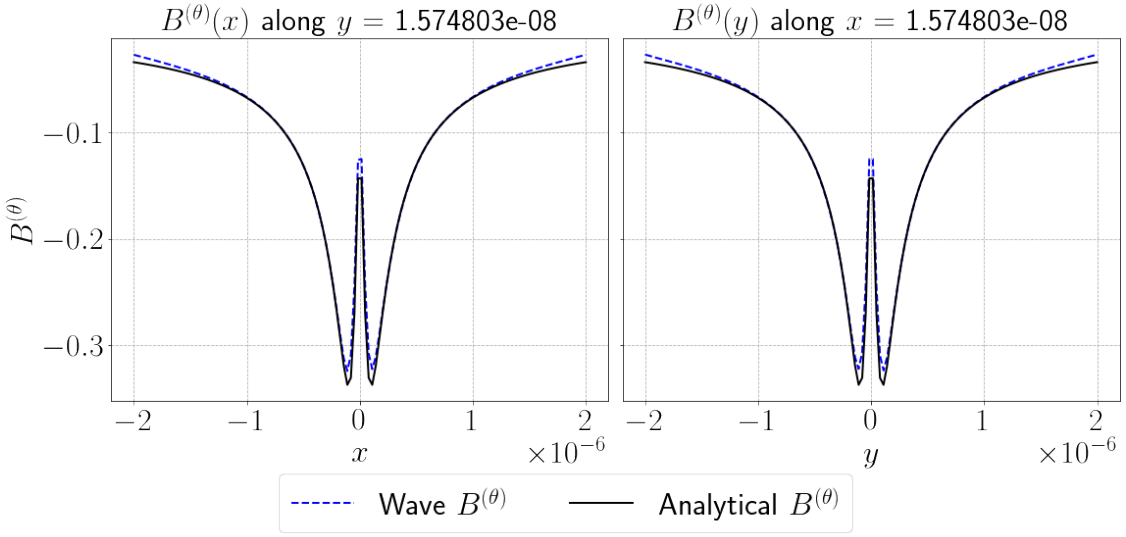


Figure 4.14: Initialization of the steady-state toroidal magnetic field in the Bennett problem computed with the BDF-2 wave solver after 1000 steps against a fixed current density. The derivatives of the vector potential $A^{(3)}$ are also obtained with the BDF-2 method.

free-space solution of this decoupled elliptic system requires the evaluation of the volume integrals

$$\psi(\mathbf{x}) = -\frac{1}{2\pi\sigma_1}\int_\Omega \ln\left(||\mathbf{x}-\mathbf{x}'||\right)\rho(\mathbf{x}')\,d\mathbf{x}',$$
$$A^{(3)}(\mathbf{x}) = -\frac{\sigma_2}{2\pi}\int_\Omega \ln\left(||\mathbf{x}-\mathbf{x}'||\right)J^{(3)}(\mathbf{x}')\,d\mathbf{x}',$$

with $||\cdot||$ being the usual Euclidean distance. The evaluation of these integrals can be performed efficiently with a fast summation method such as a tree-code [83] or fast multipole method [48], which is beyond the scope of the present work. Instead, we solve this elliptic system using second-order finite-differences with a sparse direct solver that applies Dirichlet boundary conditions. Derivatives of the fields are computed with second-order finite-differences. The Dirichlet data used for the elliptic solves is supplied by evolving the wave equations for the potentials with the BDF-2 method using outflow boundary conditions. While there are many possible approaches to this problem, it is important that this data be updated at each time step so that the boundary data is consistent with the source. We applied both the Boris and Tao solvers for 50 thermal crossings using a total of 1×10^6 steps, which gives a CFL ≈ 15.90. For each species, i.e., ions and electrons, we used 102,400 particles. In Figure B.1, we show the state of the beam and corresponding fields after 50 thermal crossings obtained with the Boris method. We observe satisfactory preservation

of the steady-state fields for the problem. We note that the CFL for this test is quite large for the second-order wave solver; however, we only use the data from the wave solver along the boundary of the simulation domain. In these regions the fields are mostly flat, so this is an acceptable approximation, even with a diffusive solver. We also observe good agreement with the analytical solution around the axis of the beam, specifically in capturing the wells in the field. Results obtained with the Molei Tao integrator, using $\omega = 500$, are shown in Figure B.2. The fields, which are shown before the final time, demonstrate a loss of the steady-state attributed to an excess of particles escaping from the beam. This causes the current density to spread outwards, resulting in changes in the potentials used to calculate the fields. We have not determined the source of this issue in the Tao method; however, given the other issues encountered with the approach, we ultimately decided that this integrator was no longer a viable option for experimentation and chose not to pursue it further.

Allowing for time variation in the fields requires that we now solve a system of two wave equations for the potentials

$$\frac{1}{\kappa^2}\frac{\partial^2 \psi}{\partial t^2} - \Delta\psi = \frac{1}{\sigma_1}\rho,$$
$$\frac{1}{\kappa^2}\frac{\partial^2 A^{(3)}}{\partial t^2} - \Delta A^{(3)} = \sigma_2 J^{(3)},$$

subject to outflow boundary conditions. In the Boris method, all derivatives for ψ and $A^{(3)}$, as well as ψ itself, are computed with the BDF-2 scheme; however, the vector potential $A^{(3)}$ is evolved explicitly using the central-2 method. To obtain the current density $\mathbf{J}^{n+1}$, which is required for the derivatives obtained with BDF-2, we use the iterative approach presented in section 4.4.2, which uses a Taylor expansion to create an initial guess for the current density. A total of 5 iterates are used in each time step. We ran the solver for 35 thermal crossings using a total of 3.5×10^6 steps, which gives a CFL ≈ 3.18. For each of the particle species, i.e., ions and electrons, we used 102,400 particles. In Figure B.3, we show the state of the beam and corresponding fields after 35 thermal crossings obtained with the Boris method. There is some slight dissipation in the regions surrounding the axis of the beam, which is caused by the BDF method. As mentioned earlier, the

use of a finer mesh would improve the quality of the solution in these regions. Along the boundary, we can see a discrepancy with the analytical solution due to inaccuracies in the treatment of outflow boundary conditions by the wave solver. Future work will seek corrections to this behavior so that the fields and their derivatives will be more accurate along the boundary. Despite these slight inaccuracies, we observe satisfactory preservation of the steady-state fields for the problem.

4.6.5 The Expanding Beam Problem

We now apply the proposed methods to the expanding beam test problem [24]. This example is well-known for its sensitivity to issues concerning charge conservation, which makes it well-suited for evaluating methods used to enforce gauge conditions and involutions. While this particular example is normally solved in cylindrical coordinates, our simulation is performed using a two-dimensional rectangular box that retains the fields $E^{(1)}$ and $E^{(2)}$, as well as $B^{(3)}$. An injection zone, which is placed on one of the faces of the box, injects a steady beam of particles into the domain. The beam expands as particles move along the box due to the electric field and eventually settles into a steady-state. Particles are absorbed or “collected" once they reach the edge of the domain and are removed from the simulation.

Based on the results of the previous test, we decided to abandon the Molei Tao integrator for this example; however, we recently became aware of another particle integrator that can be used to evolve the generalized momentum formulation used in this work. In [36], a semi-implicit Euler discretization was used in a mesh-free simulation of the VM system, in the Darwin limit. To avoid time derivatives of the potentials, the particle equations were cast in terms of a generalized momentum. This method, while only first-order accurate, is simple to implement, efficient, and based on the results in [36], surprisingly accurate. Moreover, they point out that this approach can likely be generalized to obtain high-order extensions. Note, an overview of this integrator was presented in section 4.5.2. The generalized momentum formulation for this problem evolves the

particle equations

$$
\begin{aligned}
\frac{dx_i^{(1)}}{dt} &= \frac{1}{r_i}\left(P_i^{(1)} - q_i A^{(1)}\right), \\
\frac{dx_i^{(2)}}{dt} &= \frac{1}{r_i}\left(P_i^{(2)} - q_i A^{(2)}\right), \\
\frac{dP_i^{(1)}}{dt} &= -q_i \partial_x \psi + \frac{q_i}{r_i}\left[\left(\partial_x A^{(1)}\right)\left(P_i^{(1)} - q_i A^{(1)}\right) + \left(\partial_x A^{(2)}\right)\left(P_i^{(2)} - q_i A^{(2)}\right)\right], \\
\frac{dP_i^{(2)}}{dt} &= -q_i \partial_y \psi + \frac{q_i}{r_i}\left[\left(\partial_y A^{(1)}\right)\left(P_i^{(1)} - q_i A^{(1)}\right) + \left(\partial_y A^{(2)}\right)\left(P_i^{(2)} - q_i A^{(2)}\right)\right].
\end{aligned}
$$

The formulation, shown above, is written in terms of scalar and vector potentials, which are obtained by solving Maxwell's equations in the Coulomb gauge. As shown in section 1.2.2.2, the complete system in the Coulomb gauge (1.45)-(1.47) can be simplified to obtain an equivalent system in which the vector potential $\mathbf{A}$ is purely rotational. The system in this form is given by

$$-\Delta\psi = \frac{1}{\sigma_1}\rho, \tag{4.37}$$

$$\frac{1}{\kappa^2}\frac{\partial^2 A^{(1)}}{\partial t^2} - \Delta A^{(1)} = \sigma_2 J_{\text{rot}}^{(1)}, \tag{4.38}$$

$$\frac{1}{\kappa^2}\frac{\partial^2 A^{(2)}}{\partial t^2} - \Delta A^{(2)} = \sigma_2 J_{\text{rot}}^{(2)}. \tag{4.39}$$

Note that we have abused notation here, since the vector potentials $A^{(1)}$ and $A^{(2)}$, as written above, are actually the rotational components of $\mathbf{A}$. Along the boundary of the domain, the electric and magnetic fields are prescribed perfectly electrically conducting (PEC) boundary conditions, which, in two spatial dimensions, is equivalent to enforcing homogeneous Dirichlet boundary conditions on the potentials ψ, $A^{(1)}$, and $A^{(2)}$. The rotational part of the current density is obtained by solving the elliptic equation

$$-\Delta\eta = \partial_x J^{(1)} + \partial_y J^{(2)}, \tag{4.40}$$

which is then used to adjust the current density according to

$$J_{\text{rot}}^{(1)} = J^{(1)} + \partial_x \eta, \tag{4.41}$$

$$J_{\text{rot}}^{(2)} = J^{(2)} + \partial_y \eta. \tag{4.42}$$

Since the problem is PEC, there can be no currents or charges on the boundary. This suggests we enforce homogeneous Neumann boundary conditions for equation (4.40). Despite the projection (4.40)-(4.42) used for the current density, irrotational components of the vector potential can be introduced through the discretization of the wave equations (4.38) and (4.39). After solving the wave equations, we extract the rotational components of the vector potential by first solving the elliptic equation

$$-\Delta\xi = \partial_x J^{(1)} + \partial_y J^{(2)}. \tag{4.43}$$

In the second step, the gradient of ξ is used to remove the irrotational parts of the potential via

$$A^{(1)}_{\text{rot}} = A^{(1)} + \partial_x \xi, \tag{4.44}$$

$$A^{(2)}_{\text{rot}} = A^{(2)} + \partial_y \xi. \tag{4.45}$$

As in the projection step for the current, we solve the elliptic equation (4.43) using homogeneous Neumann boundary conditions. The sequence of corrections described by (4.43)-(4.45) are identical to the elliptic divergence cleaning method discussed in section 4.3.4 to enforce the gauge condition.

As with the Bennett problem, we shall compare results obtained using the generalized momentum formulation with the formulation that employs the Boris method, in which the equations of motion for the particles are expressed in terms of **E** and **B** and take the form

$$\begin{aligned}
\frac{dx_i^{(1)}}{dt} &= v_i^{(1)}, \\
\frac{dx_i^{(2)}}{dt} &= v_i^{(2)}, \\
\frac{dv_i^{(1)}}{dt} &= \frac{q_i}{r_i}\left(E^{(1)} + v_i^{(2)} B^{(3)}\right), \\
\frac{dv_i^{(2)}}{dt} &= \frac{q_i}{r_i}\left(E^{(2)} - v_i^{(1)} B^{(3)}\right).
\end{aligned}$$

A key difference with the generalized momentum formulation is that the fields in the Boris method use the Lorenz gauge, rather than the Coulomb gauge. Therefore, the Boris approach requires

fields, which are obtained by solving the system

$$\begin{aligned}
\frac{1}{\kappa^2}\frac{\partial^2\psi}{\partial t^2} - \Delta\psi &= \frac{1}{\sigma_1}\rho, \\
\frac{1}{\kappa^2}\frac{\partial^2 A^{(1)}}{\partial t^2} - \Delta A^{(1)} &= \sigma_2 J^{(1)} \\
\frac{1}{\kappa^2}\frac{\partial^2 A^{(2)}}{\partial t^2} - \Delta A^{(2)} &= \sigma_2 J^{(2)}.
\end{aligned}$$

Using equation (1.11), the potentials ψ, $A^{(1)}$, and $A^{(2)}$, can be used to obtain $\mathbf{E}$ and $\mathbf{B}$ for the particle updates. At the present time, we do not have a working method for enforcing the Lorenz gauge condition, so a cleaning method is not used this approach. The formulation involving the Boris push could certainly be modified to work with the Coulomb gauge rather than the Lorenz gauge condition, and will be explored in later work.

To setup the simulation, we first create a box specified by the region $[0, 1] \times [-\frac{1}{2}, \frac{1}{2}]$, which has been normalized by some length scale L. We shall further assume that the beam consists only of electrons, which are prescribed some injection velocity $v_{\text{injection}}$ and travel along the x-axis of the box. An estimate of the crossing time for a particle can be obtained using the injection velocity and the length of the domain, which sets the time scale T for the simulation. The duration of the simulation is given in terms of particle crossings, which are then used to set the time step Δt. At each time step, particles are initialized in an injection region specified by the interval $[-L_{\text{ghost}}, 0) \times [-R_b, R_b]$, where R_b is the radius of the beam, and the width of the injection zone L_{ghost} is chosen such that

$$L_{\text{ghost}} = v_{\text{injection}}\Delta t.$$

This ensures that all particles initialized in the injection zone will be in the domain after one time step. Particle positions in the injection region are set according to samples taken from a uniform distribution, and the number of particles injected for a given time step is set by the injection rate. In each time step, the injection procedure is applied before the particle position update, so that, at the end of the time step, the injection zone is empty. To prevent the introduction of an impulse response in the fields due to the initial injection of particles, we apply a linear ramp function to

Parameter	Value
Beam radius (R_b) [m]	8.0×10^{-3}
Average number density ($\bar{n}$) [m^{-3}]	7.8025×10^{14}
Largest box dimension (L) [m]	1.0×10^{-1}
Electron injection velocity ($v_{\text{injection}}$) [m/s]	5.0×10^{7}
Electron crossing time (T) [s]	2.0×10^{-9}
Electron macro-particle weight (w_{mp}) [non-dimensional]	1.67×10^{-5}
Injection rate (per Δt) [1/s]	10

Table 4.6: Table of the parameters used in the setup for the expanding beam problem.

the macro-particle weights whose duration is one particle crossing. A summary of the parameters used to setup the problem are presented in Table 4.6.

At some point, the particles will reach the boundary and should be removed from the simulation. A list is a natural choice for managing the injection and deletion of particles, since particles can be easily added or removed. The types of problems we are considering require many particles, so having to constantly resize a list can become quite expensive. Moreover, implementations of lists are not cache-friendly, so we lose opportunities for vectorization. Instead, we use arrays whose lengths are determined by (over)estimating the total number of particles in the domain, at any given time. We estimate the number of particles by first calculating the number of time steps required for a particle to cross the box, which is then multiplied by the injection rate. Finally we double this result for additional safety, since the beam will spread to some degree. We store a running total of the number of particles in the domain at any given time, which identifies the entries of the array to update, with entries beyond this value being considered “deleted." Consequently, at any given time step, we need to sort the particle arrays so that particles outside of the domain are placed after this counter variable. This sort step is performed by first creating a Boolean array which indicates if particle is outside the domain. A sorting method is then applied to the Boolean array, which is quite fast, as the data is mostly sorted. Then, the sorted Boolean array is used to remap the entries of the particle arrays.

We first test the formulation which combines the Boris method with the fields in the Lorenz gauge (1.43). Initially, we applied the time-centered method to update the components of the vector

potentials; however, the lack of dissipation in the time-centered approach led to certain noise in the fields due to dispersion error, which was further amplified by the application of the time derivative. Instead, we used the BDF-2 method, which is dissipative, to perform these calculations. The effect on the time derivatives, which is shown in Figure B.4, is quite apparent. We ran the simulation with this configuration to a final time of 1000 particle crossings. A mesh of 128×128 grid points was used in the calculation, which gave a CFL ≈ 0.761. The results for this experiment are shown in Figure B.5. The beam, itself, is surprisingly stable and does not display significant issues associated with violating the gauge condition. Along the edge of the beam, we observe some small oscillations that will eventually grow over time, causing the beam to break apart. This is also reflected in the growth of the error in the Lorenz gauge toward the end of the run. In Figure B.6, we show the smooth potentials and their partial derivatives, which are constructed using the proposed BDF-2 wave solver. As mentioned earlier, we have not had success with enforcing the Lorenz gauge in this particular work, but it is something we plan to revisit in the future.

Next, we test the Coulomb gauge formulation, which applies the AEM for time integration [36] (discussed in section 4.5.2). In this experiment, we ran the code for 3000 particle crossings without a cleaning technique to enforce the gauge condition. We used the same 128×128 mesh for the fields as in the Boris approach. The Poisson solves are performed using second-order finite-differences with a sparse linear solver. Derivatives of the data obtained from the Poisson solves are computed with second-order finite-differences. After 3000 particle crossings, the structure of the beam is largely destroyed due to violations in the gauge condition. The results presented in Figure B.7 show the beam at an earlier time, corresponding to 2000 crossings, at which point the striations and clumping in the beam are quite apparent. In Figure B.8, we show the beam after 3000 crossings, which uses the same approach but applies the cleaning procedure described by equations (4.43)-(4.45). The impact of the cleaning is quite remarkable, as the integrity of the beam is no longer compromised. The cleaning approach displays some violations in the gauge condition at the boundaries of the domain, which are expected because particles enter or leave the domain at these points. On the interior the fluctuations are in the sixth decimal position, which can likely be

improved through the use of a more accurate Poisson solver along with additional particles. The smooth potentials and their derivatives, which were obtained with this formulation are presented in Figure B.9. The derivatives used to evaluate the gauge condition show some jumps along the boundary where particles enter and leave, which we plan to investigate in greater detail. Comparing Figures B.6 and B.9, it is interesting to see the structural differences in the potentials (and their derivatives) obtained with different formulations.

As mentioned earlier, we have not yet constructed a functioning cleaning method for the Lorenz gauge formulation. In spite of this, we find the Lorenz gauge formulation to be quite appealing because it avoids the use of elliptic solvers. For this reason, we combined the AEM for the particles with a first-order BDF field solver. No cleaning method is used for the fields. We ran the simulation out to 3000 particle crossings on the same 128×128 mesh for the fields. The beam, which, remains surprising intact without a cleaning method, is shown in Figure B.10a. The time trace of the Lorenz gauge error, which is displayed in Figure B.10b, shows some oscillations that appear to be bounded. The fields from the same experiment, at the final step, are presented in Figure B.11. The fields are quite similar to those obtained with the second-order BDF scheme combined with the Boris method presented in Figure B.6. Despite the low order accuracy, it can be shown that a first-order time discretization of the fields is consistent with a discrete form of the Lorenz gauge (see B.1 for details). Furthermore, the low-order time accuracy of the fields does not introduce significant dissipation. While the goal of our work is to build higher-order field solvers for plasma applications, this result is interesting due to its practicality. More specifically, it demonstrates that it is possible to obtain a reasonable solution in an inexpensive manner.

The last point we wish to mention concerns the metrics used to assess charge conservation. One of the advantages of working with a gauge formulation is that the metrics for charge conservation are embedded in the gauge condition, which can be calculated using the derivatives of the potentials. Compare this with measurements based on Gauss' law $\nabla \cdot \mathbf{E} = \rho/\sigma_1$, which utilizes particle data that may be under-resolved by the mesh. If either ρ or the divergence term is not smooth, this could give the impression that the method is ineffective at conserving charge. As an example, in the last

experiment, which enforced the Coulomb gauge through an elliptic method, we found that the error in the gauge condition was quite small, especially away from the boundary. For the same problem, the point-wise error in Gauss' law, which is shown in Figure B.12b, indicates that the method is not conserving charge, even in regions away from the boundaries. On the other hand, if we compute the residual

$$\int_{\Omega} \left(\nabla \cdot \mathbf{E}(\mathbf{x}) - \frac{1}{\sigma_1} \rho(\mathbf{x}) \right) dV_{\mathbf{x}} \approx \sum_{i,j} \left(\nabla \cdot \mathbf{E}(x_i, y_j) - \frac{1}{\sigma_1} \rho(x_i, y_j) \right) \Delta x_i \Delta y_j, \tag{4.46}$$

then we can say whether or not the method conserves charge in a bulk sense. In the above definition, the divergence is interpreted in a discrete sense and the sum runs over the mesh points so that $\Delta x_i \Delta y_j$ is the volume of the grid-cell (i, j). In Figure B.12a, we plot the bulk error as a function of time, which shows far smaller violations and is symmetric about zero. The large violations in Gauss' law for the case of cleaning (shown in Figure B.12b) are likely the result of the treatment used for the divergence term. If we compare this with the same formulation that skips the cleaning step (shown in Figure B.13), then we can see that the point-wise violations occur on a much greater scale. This discrepancy was similarly in [19], where they showed the ℓ_2 error in Gauss' law to be $O(1)$, even if cleaning methods were applied. It is also important to note that if components of the electric field display non-smooth features such as cusps or steep gradients, then the finite-difference derivatives over modest stencils will show nonphysical oscillations, which will directly impact the results of the point-wise error in Gauss' law. One approach we have considered to deal with this is the application of WENO derivatives, which we plan to explore in future work.

4.6.6 A Narrow Beam Problem and the Effect of Particle Count

In this last test, we slightly modify our setup from the previous example in section 4.6.5 to construct a beam with a lower density, which will be used to conduct a certain type of refinement study. This modification primarily concerns the prescription of particle weights. In Table 4.6 for the previous problem, we provided a value for the number density $\bar{n}$ in addition to a particle weight w_{mp}. The particle weighting from the previous example was obtained by essentially hard-coding an estimate

for the number of simulation particles. While it is not incorrect to do this, the currents in the beam may become too large as the injection rate increases, causing particles to move back to the injection zone. To fix this problem, we compute the particle weight w_{mp} using an estimate for the number of particles in the beam that accounts for the injection rate. Since we discussed how to estimate this number in the previous section, we omit these details for brevity. Therefore, the particle weight no longer needs to be prescribed in the setup, and it naturally adjusts according to the injection rate specified by the user. This modification ultimately allows us to examine the effect of the particle count on the solution by fixing the number density and varying the particle injection rate. Aside from this modification, all other details, e.g., the models, injection procedure, etc. are identical to those provided in section 4.6.5, so we shall not describe them further.

We test the effect of an increased particle count on the numerical solution using the solver that combines the AEM for particles with the BDF-2 field solver in the Coulomb gauge, along with elliptic projections to enforce the gauge condition. We use the same 128×128 mesh as in the previous problem and an injection rate of 400 particles per timestep. The remaining parameters are specified in Table 4.7. We run the problem for a total of 5 particle crossings, which is sufficient for the refinement purposes of this problem, using the same CFL ≈ 0.761 as the previous problem. We plot the narrow beam and the gauge error in Figure B.14. The increased smoothness in the charge density due to the increase in the number of particles is apparent. Moreover, the error in the gauge condition seems to be quite small away from the boundary, where particles are injected and removed. The corresponding fields and derivatives, which are used to move the particles are presented in Figure B.15. The fields appear to be smooth. We also plot the “bulk" error in Gauss' law associated as a function of time and show the point-wise error as a surface in Figure B.16. The bulk error shows a jump at the time that corresponds to the first crossing, at which point some particles begin to leave the domain. Shortly after this jump, the error settles. As before, the point-wise violations in Gauss' law seem to indicate a loss of charge conservation. For convenience, we show the derivatives of the electric field, which are used to measure the error in Gauss' law in Figure B.17. We note that the derivative in the x shows some oscillations in the interior of the

Parameter	Value
Beam radius (R_b) [m]	8.0×10^{-3}
Average number density ($\bar{n}$) [m^{-3}]	1.552258×10^{14}
Largest box dimension (L) [m]	1.0×10^{-1}
Electron injection velocity ($v_{\text{injection}}$) [m/s]	5.0×10^{7}
Electron crossing time (T) [s]	2.0×10^{-9}

Table 4.7: Table of the parameters used in the setup for the narrow beam problem.

beam, and the y derivative contains a mix of sharp and uniform features.

Lastly, we show the effect of increasing the particle count on the gauge condition by considering injection rates of 100, 200, and 400 particles per time step. These results are presented in Figure B.18. While the error at the boundaries remains largely unchanged in the runs, there is a noticeable improvement in the error on the interior. Specifically, we see the more jagged features on the interior become smoother and smaller in size due to the increased particle count.

4.7 Conclusion

In this work, we developed a PIC method by coupling dimensionally-split integral equation solvers for the fields with standard and non-standard time integration methods for particles. After introducing the concepts of a general PIC method, we presented several approaches for enforcing gauges and charge conservation. We then introduced methods for integrating the particle equations of motion that are necessitated by the formulations considered in this work. The discussion was primarily focused on two methods designed for problems with non-separable Hamiltonians. While the time integration methods employed in this work are not new, the novel contribution of our work is that we demonstrated how existing methods for particles can effectively leverage the proposed field solvers in simulations of plasmas. This includes the construction of spatial derivatives, which can be obtained directly from the field solvers. To this end, we applied the proposed methods to several application problems involving beams. Results were compared with standard methods based on leapfrog time integration, and in nearly all examples, the proposed methods recovered similar behavior. The results not only validate the generalized momentum formulation, but also demonstrate the versatility and flexibility of the proposed field solvers in simulating plasma phenomena.

CHAPTER 5

CONCLUSION AND FUTURE DIRECTIONS

In this thesis, we have presented a collection of algorithms for evolving fields in plasmas with specific applications to the Vlasov-Maxwell system. Maxwell's equations are reformulated in term of the Lorenz gauge, as well as the Coulomb gauge to obtain systems involving wave equations. These wave equations are solved using the methods proposed in this work, and are combined with a particle-in-cell method [2, 3] to simulate plasmas. This particle description of the Vlasov equation couples directly to the fields, which are solved using a mesh. We considered two formulations of the equations for the particles. First, a standard approach was presented, which is based on the Newton-Lorentz equations, while the other used a generalized Hamiltonian to write the particle equations in terms of the potentials (and their derivatives) used in the gauge formulation. The advantage offered by the generalized Hamiltonian framework is that it eliminates the need to compute time derivatives, which reduce the time accuracy of the fields and can lead to instabilities in certain limits [35, 36].

In the first part of this thesis, we developed and extended methods for scalar wave equations, which can be used to update the potentials in these formulations. Our developments are based on a class of algorithms known as the MOL^T, which combines a dimensional splitting technique with a one-dimensional integral equation method. The resulting methods are unconditionally stable, can address geometry, and are $O(N)$, where N is the number of mesh points. Our work contributed methods to construct derivatives of the potentials for this class of dimensionally-split methods. These derivatives, which are used to evolve particles, were constructed directly from the data used in the one-dimensional integral solution. Consequently, the methods naturally inherit the speed, stability, and geometric flexibility offered by the base solver. Moreover, we established, through refinement experiments, that these derivatives converge at the same rate in both space and time, as the base method. We also presented a more systematic treatment of outflow boundary conditions for the second-order (in time) methods, including refinement studies, which were not presented in

earlier work. While the outflow procedure used in this work is convergent, more work should be done to reduce magnitude of the error and improve the rate of convergence.

The core algorithms used in the MOL^T and the related class of successive convolution methods were also explored in the context of high-performance computing environments. We developed a novel domain decomposition approach, which ultimately allowed the method to be used on distributed memory computing platforms. Shared memory algorithms were developed using the Kokkos performance portability library, which allows a user to write a single version of a code that can be executed on various computing devices with the architecture-dependent details being managed by the library. We optimized predominant loop structures in the code and settled on a blocking pattern that prescribed parallelism at multiple levels. Moreover, the proposed iteration pattern is flexible enough to work with shared memory features available on GPU systems. While the results indicated a high sensitivity to data locality, which is a feature of memory bound algorithms, the methods were shown to be quite fast. Scaling experiments demonstrated that the proposed algorithms could sustain an update rate in excess of 2.5×10^6 grid points per second, per physical core. On a shared memory system with 40 cores (per node), this translates to an update rate of 1×10^8 grid points per second (per node). This was true even for the largest experiment conducted in that same study, which used nearly 35 billion grid points.

We also presented particle-in-cell methods for the Vlasov-Maxwell system, which leveraged the methods for fields and derivatives developed in this work. We showed how to combine the proposed methods with standard and non-standard time integration methods for the particles and applied these methods to a variety of plasma test problems. The focus on beam problems is primarily motivated by the preference of particle methods over mesh-based discretizations, which are overly diffusive near the edge of the beam. Our results are generally encouraging and demonstrate the capabilities of the proposed field solvers in simulating plasma phenomena. Additionally, our results serve to validate the generalized Hamiltonian formulation, which will be the foundation of our future work.

The results presented in this thesis suggest several interesting directions for future research. In terms of field solvers, the methods used for outflow boundary conditions should be reconsidered.

This is especially true in the case of the more intricate methods based on successive convolution, which, with the current methods, requires a fairly substantial amount of storage for the time history along the boundary. The success of the BDF methods suggests that higher-order approaches should be considered and analyzed in a rigorous fashion. In light of the stability issues associated with elongated time stencils, it may be worthwhile to construct higher-order methods using extrapolation or other correction techniques, which can simultaneously address the splitting error associated with multi-dimensional problems.

Extensions of the parallel algorithms presented in this thesis should also be considered. First, the current approach should be evaluated on GPUs. A generalization of the decomposition, which extends beyond nearest-neighbors and eliminates the artificial CFL-like condition, should also be evaluated. This would provide a clear path for addressing problems involving non-uniform mesh patches that are frequently encountered in algorithms that support adaptivity. Future work should also evaluate other fast summation algorithms which are more aligned with the vectorization capabilities supported by new hardware.

The research directions above also have implications for the development of new solvers for the Vlasov-Maxwell system. In the near-term future, we plan on revisiting the formulation involving the Lorenz gauge, which avoids the use elliptic solvers. This suggests a pairing with the hyperbolic divergence cleaning method discussed in this thesis, which requires effective approaches for enforcing outflow boundary conditions. On the other hand, we believe the Coulomb gauge formulation has great potential despite the three elliptic solves required to properly enforce the gauge condition. In order to support problems with geometry, the Poisson equations should be discretized as integral equations rather than finite-differences. This suggestion is motivated by the access to analytical derivatives offered by integral equation methods, which are aligned with the techniques used in this work. Furthermore, there have been several interesting developments involving fast summation methods using a technique known as barycentric Lagrange interpolation [84]. These approaches which are kernel-independent, have also been explored on GPUs [85, 86] and show great promise in addressing challenges posed by new computational hardware.

APPENDICES

APPENDIX A

APPENDIX FOR CHAPTER 3

A.1 Example for Linear Advection

Suppose we wish to solve the 1D linear advection equation:

$$\partial_t u + c\partial_x u = 0, \quad (x,t) \in (a,b) \times \mathbb{R}^+, \tag{A.1}$$

where $c > 0$ is the wave speed and leave the boundary conditions unspecified. The procedure for $c < 0$ is analogous. Discretizing (A.1) in time with backwards Euler yields a semi-discrete equation of the form

$$\frac{u^{n+1}(x) - u^n(x)}{\Delta t} + c\partial_x u^{n+1}(x) = 0.$$

If we rearrange this, we obtain a linear equation of the form

$$\mathcal{L}[u^{n+1}; \alpha](x) = u^n(x), \tag{A.2}$$

where we have used

$$\alpha := \frac{1}{c\Delta t}, \quad \mathcal{L} := \mathcal{I} + \frac{1}{\alpha}\partial_x.$$

By reversing the order in which the discretization is performed, we have created a sequence of BVPs at discrete time levels. If we had discretized equation (A.1) using the MOL formalism, then $\mathcal{L}$ would be an algebraic operator. To solve equation (A.2) for u^{n+1}, we analytically invert the operator $\mathcal{L}$. Notice that this equation is actually an ODE, which is linear, so the problem can be solved using methods developed for ODEs. If we apply the integrating factor method to the problem, we obtain

$$\partial_x \left[e^{\alpha x} u^{n+1}(x) \right] = \alpha e^{\alpha x} u^n(x).$$

To integrate this equation, we use the fact that characteristics move to the right, so integration is performed from a to x. After rearranging the result, we arrive at the update equation

$$\begin{aligned} u^{n+1}(x) &= e^{-\alpha(x-a)}u^{n+1}(a) + \alpha\int_a^x e^{-\alpha(x-s)}u^n(s)\,ds, \\ &\equiv e^{-\alpha(x-a)}A^{n+1} + \alpha\int_a^x e^{-\alpha(x-s)}u^n(s)\,ds, \\ &\equiv \mathcal{L}^{-1}[u^n;\alpha](x). \end{aligned}$$

This update displays the origins of the implicit behavior of the method. While convolutions are performed on data from the previous time step, the boundary terms are taken at time level $n+1$.

Now that we have obtained the update equation, we need to apply the boundary conditions. Clearly, if the problem specifies a Dirchlet boundary condition at $x = a$, then $A^{n+1} = u^{n+1}(a)$. We can compute a variety of boundary conditions using the update equation

$$u^{n+1}(x) = e^{-\alpha(x-a)}A^{n+1} + \alpha\int_a^x e^{-\alpha(x-s)}u^n(s)\,ds,$$

where

$$I[u^n;\alpha](x) = \alpha\int_a^x e^{-\alpha(x-s)}u^n(s)\,ds.$$

For example, with periodic boundary conditions, we would need to satisfy

$$u^{n+1}(a) = u^{n+1}(b), \tag{A.3}$$

$$\partial_x u^{n+1}(a) = \partial_x u^{n+1}(b). \tag{A.4}$$

Applying condition (A.3), we find that

$$A^{n+1} = e^{-\alpha(b-a)}A^{n+1} + \alpha\int_a^b e^{-\alpha(b-s)}u^n(s)\,ds.$$

Solving this equation for A^{n+1} shows that

$$A^{n+1} = \frac{I[u^n;\alpha](b)}{1-\mu},$$

with $\mu = e^{-\alpha(b-a)}$. Alternatively, we could have started with (A.4), which would give an identical solution. While this particular procedure is only applicable to linear problems, this exercise motivates some of the choices made to define operators in the method.

```
// Distribute tiles of the array to teams of threads dynamically
Kokkos::parallel_for("team loop over tiles", team_policy(total_tiles,
    Kokkos::AUTO()),
    KOKKOS_LAMBDA(team_type &team_member)
{
    // Determine the flattened tile index via the team rank
    // and compute the unflattened indices of the tile T_{i,j}
    const int tile_idx = team_member.league_rank();
    const int tj = tile_idx % num_tiles_x;
    const int ti = tile_idx / num_tiles_x;

    // Retrieve tile sizes & offsets and
    // obtain subviews of the relevant grid data on tile T_{i,j}
    // ...

    // Use a team's thread range over the lines
    Kokkos::parallel_for(Kokkos::TeamThreadRange<>(team_member, Ny_tile),
    [&](const int iy)
    {
        // Slice to extract a subview of my line's data and
        // call line methods which use vector loops
        // ...
    }
});
```

Scheme A.1: An example of coarse-grained parallel nested loop structure.

```
// Distribute the threads to lines
Kokkos::parallel_for("Fast sweeps along x", range_policy(0, Ny),
    KOKKOS_LAMBDA(const int iy)
{
    // Slice to obtain the local integrals to which we apply
    // the convolution kernel to the entire line
    // ...
});
```

Scheme A.2: Kokkos kernel for the fast-convolution algorithm.

A.2 Kokkos Kernels

This section provides listings, which outline the general format of the Kokkos kernels used in this work. Specifically, we provide structures for the tiled/blocked algorithms (Scheme A.1) in addition to the kernel that executes the fast summation method along a line (Scheme A.2).

A.3 WENO Quadrature

We provide the various expressions for the coefficients and smoothness indicators used in the reconstruction process for $J_R^{(r)}$. Defining $\nu \equiv \alpha\Delta x$, the coefficients for the fixed stencils are given in [44] as follows:

$$
\begin{aligned}
c_{-3}^{(0)} &= \frac{6 - 6\nu + 2\nu^2 - (6 - \nu^2)e^{-\nu}}{6\nu^3}, \\
c_{-2}^{(0)} &= -\frac{6 - 8\nu + 3\nu^2 - (6 - 2\nu - 2\nu^2)e^{-\nu}}{2\nu^3}, \\
c_{-1}^{(0)} &= \frac{6 - 10\nu + 6\nu^2 - (6 - 4\nu - \nu^2 + 2\nu^2)e^{-\nu}}{2\nu^3}, \\
c_{0}^{(0)} &= -\frac{6 - 12\nu + 11\nu^2 - 6\nu^3 - (6 - 6\nu + 2\nu^2)e^{-\nu}}{6\nu^3},
\end{aligned}
$$

$$
\begin{aligned}
c_{-2}^{(1)} &= \frac{6 - \nu^2 - (6 + 6\nu + 2\nu^2)e^{-\nu}}{6\nu^3}, \\
c_{-1}^{(1)} &= -\frac{6 - 2\nu - 2\nu^2 - (6 + 4\nu - \nu^2 - 2\nu^3)e^{-\nu}}{2\nu^3}, \\
c_{0}^{(1)} &= \frac{6 - 4\nu - \nu^2 + 2\nu^3 - (6 + 2\nu - 2\nu^2)e^{-\nu}}{2\nu^3}, \\
c_{1}^{(1)} &= -\frac{6 - 6\nu + 2\nu^2 - (6 - \nu^2)e^{-\nu}}{6\nu^3},
\end{aligned}
$$

$$
\begin{aligned}
c_{-1}^{(2)} &= \frac{6 + 6\nu + 2\nu^2 - (6 + 12\nu + 11\nu^2 + 6\nu^3)e^{-\nu}}{6\nu^3}, \\
c_{0}^{(2)} &= -\frac{6 + 4\nu - \nu^2 - 2\nu^3 - (6 + 10\nu + 6\nu^2)e^{-\nu}}{2\nu^3}, \\
c_{1}^{(2)} &= \frac{6 + 2\nu - 2\nu^2 - (6 + 8\nu + 3\nu^2)e^{-\nu}}{2\nu^3}, \\
c_{2}^{(2)} &= -\frac{6 - \nu^2 - (6 + 6\nu + 2\nu^2)e^{-\nu}}{6\nu^3}.
\end{aligned}
$$

The corresponding linear weights are

$$
\begin{aligned}
d_0 &= \frac{6 - \nu^2 - (6 + 6\nu + 2\nu^2)e^{-\nu}}{3\nu(2 - \nu - (2 + \nu)e^{-\nu})}, \\
d_2 &= \frac{60 - 60\nu + 15\nu^2 + 5\nu^3 - 3\nu^4 - (60 - 15\nu^2 + 2\nu^4)e^{-\nu}}{10\nu^2(6 - \nu^2 - (6 + 6\nu + 2\nu^2)e^{-\nu})},
\end{aligned}
$$

$$d_1 = 1 - d_0 - d_2.$$

The expressions for the smoothness indicators are given in [56] as

$$\begin{aligned}
\beta_0 &= \frac{13}{12}\left(-v_{i-3} + 3v_{i-2} - 3v_{i-1} + v_i\right)^2 + \frac{1}{4}\left(v_{i-3} - 5v_{i-2} + 7v_{i-1} - 3v_i\right)^2, \\
\beta_1 &= \frac{13}{12}\left(-v_{i-2} + 3v_{i-1} - 3v_i + v_{i+1}\right)^2 + \frac{1}{4}\left(v_{i-2} - v_{i-1} - v_i + v_{i+1}\right)^2, \\
\beta_2 &= \frac{13}{12}\left(-v_{i-1} + 3v_i - 3v_{i+1} + v_{i+2}\right)^2 + \frac{1}{4}\left(-3v_{i-1} + 7v_i - 5v_{i+1} + v_{i+2}\right)^2.
\end{aligned}$$

To obtain the analogous expressions for $J_L^{(r)}$, we exploit the "mirror-symmetry" property of WENO reconstructions. That is, one can keep the left side of each of the expressions, then reverse the order of the expressions on the right. Expressions for calculating one particular smoothness indicator, if interested, can be found in [44].

A.4 Some Larger Figures from Experiments

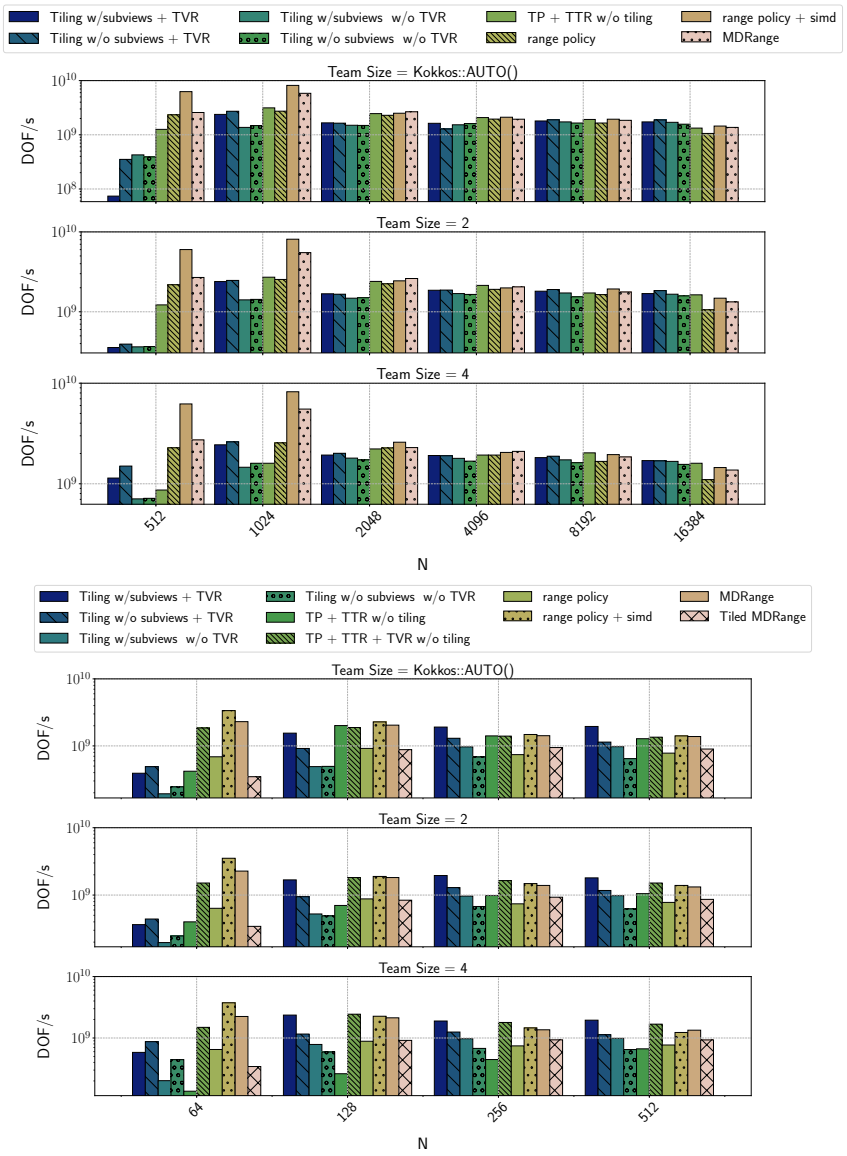


Figure A.1: Plots comparing the performance of different parallel execution policies for the pattern in Scheme 3.2 using test cases in 2-D (top) and 3-D (bottom). Tests were conducted on a single node that consists of 40 cores using the code configuration outlined in 3.1. Each group consists of three plots, whose difference is the value selected for the team size. We note that hyperthreading is not enabled on our systems, so Kokkos::AUTO() defaults to a team size of 1. Tile experiments used a block size of 256^2, in 2-D problems, and 32^3 in 3-D. A tiled MDRange was not implemented in the 2-D cases because the block size was larger than some of the problems. The results generally agree with those presented in 3.5. For smaller problem sizes, using the non-portable range_policy with OpenMP simd directives is clearly superior over the policies. However, when enough work is available, we see that blocked policies with subviews and vectorization generally become the fastest. In both cases, MDRange seems to have fairly good performance. Tiling, when used with MDRange, in the 3-D cases, seems to be slower than plain MDRange. Again, we see that the use of blocking provides a more consistent update rate if enough work is available.

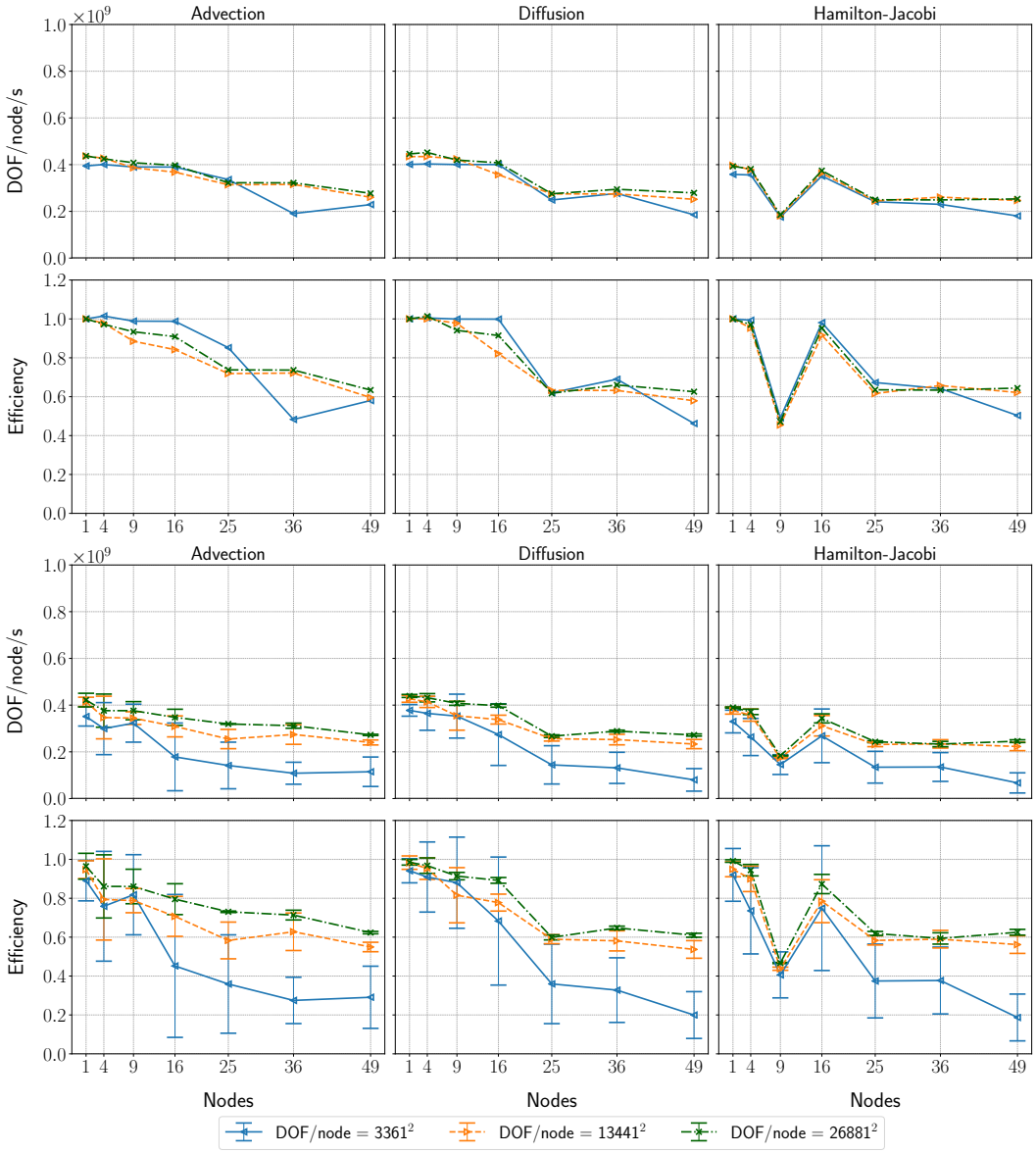


Figure A.2: Weak scaling results, for each of the applications, using up to 49 nodes (1960 cores). For each of the applications, we have provided the update rate and weak scaling efficiency computed via the fastest time/step (top) and average time/step (bottom). Results for advection and diffusion applications is quite similar, despite the use of different operators. The results for the H-J application seem to indicate that no major performance penalties are incurred by use of the adaptive time stepping method. Scalability appears to be excellent, up to 16 nodes (640 cores), then begins to decline. While some loss in performance, due to network effects, is to be expected, this loss appears to be larger than was previously observed. The nodes used in the runs were not contiguous, which hints at a possible sensitivity to data locality.

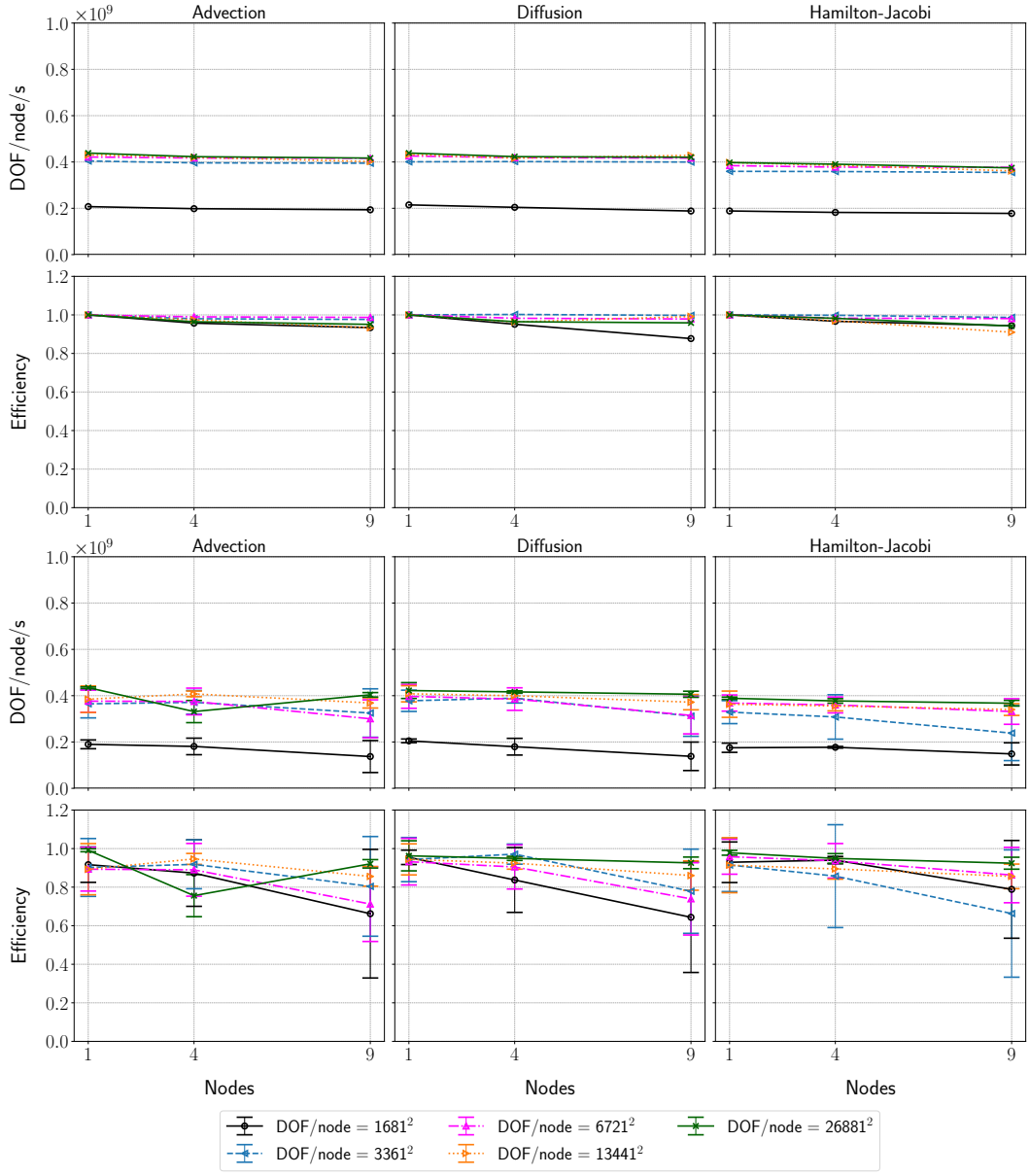


Figure A.3: Weak scaling results obtained with contiguous allocations of up to 9 nodes (360 cores) for each of the applications. For comparison, the same information is displayed as in A.2. Data from the fastest trials indicates nearly perfect weak scaling, across all applications, up to 9 nodes, with a consistent update rate between $2-4\times10^8$ DOF/node/s. A comparison of the fastest timings between the large and small runs supports our claim that data proximity is crucial to achieving the peak performance of the code. Note that size the error bars are generally smaller than those in A.2. This indicates that the timing data collected from individual trials exhibits less overall variation.

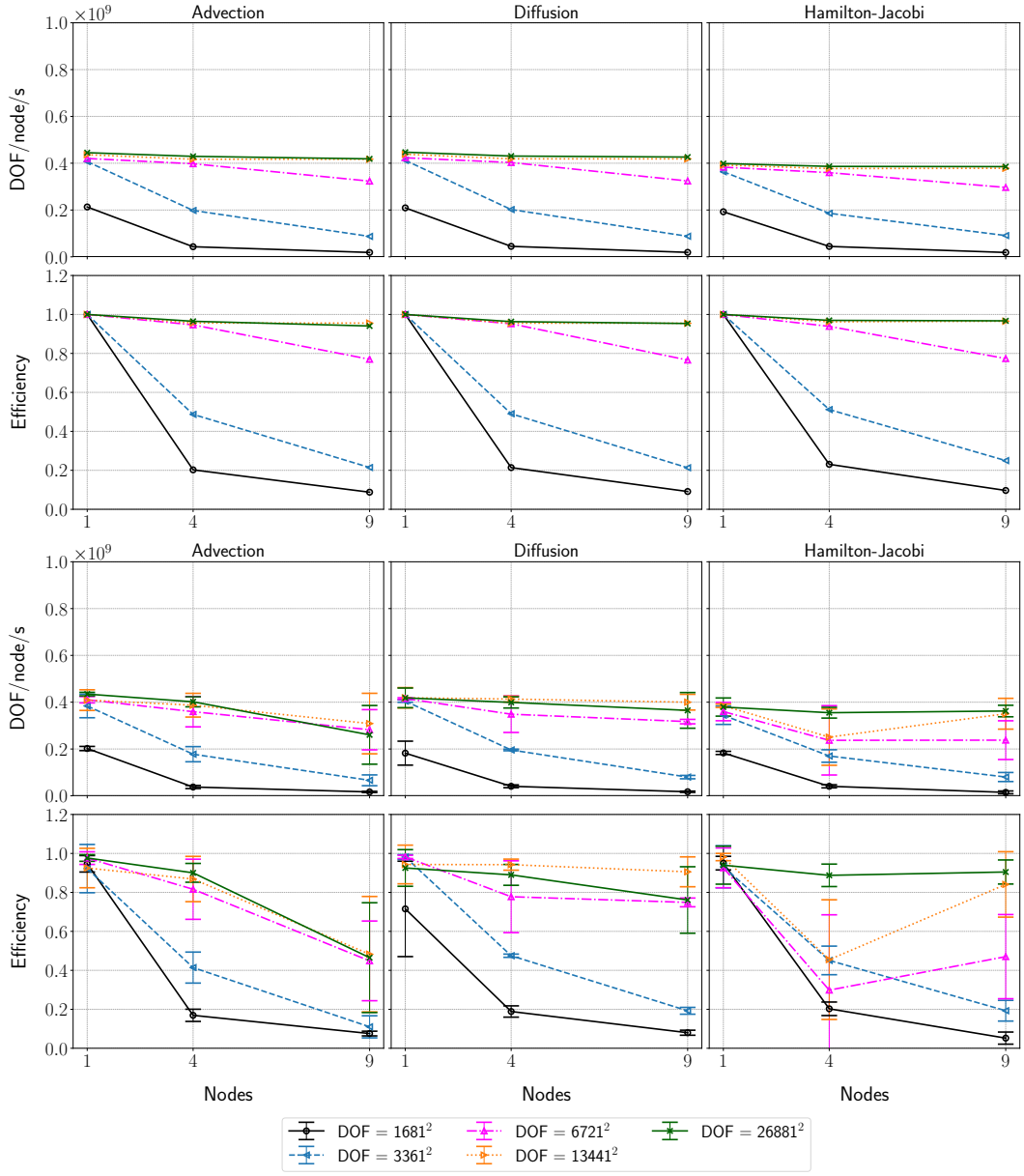


Figure A.4: Strong scaling results for each of the applications obtained on contiguous allocations of up to 9 nodes (360 cores). Displayed among each of the applications are the update rate and strong scaling efficiency computed from the fastest time/step (top) and average time/step (bottom). This method does not contain a substantial amount of work, so we do not expect good performance for smaller base problem sizes, as the work per node becomes insufficient to hide the cost of communication. Larger base problem sizes, which introduce more work, are capable of saturating the resources, but will at some point become insufficient. Moreover, threads become idle when the work per node fails to introduce enough blocks.

APPENDIX B

APPENDIX FOR CHAPTER 4

B.1 Semi-discrete Time Consistency of the Lorenz Formulation with BDF-1

Here we show that the Lorenz gauge formulation of Maxwell's equations (1.8)-(1.10) satisfies a certain time consistency relation when a first-order BDF time discretization is applied. By time consistent, we mean that the semi-discrete system for the potentials induces both a gauge condition and continuity equation at the semi-discrete level. In the treatment of the semi-discrete equations, we shall ignore effects of dimensional splittings.

Following the procedure used in section 2.2.1, we can derive the semi-discrete equations for the scalar and vector potentials using first-order backwards differences for each of the time derivatives. Proceeding, one obtains the following semi-discrete equations for the Lorenz gauge formulation:

$$\mathcal{L}\psi^{n+1} = 2\psi^n - \psi^{n-1} + \frac{1}{\alpha^2\epsilon_0}\rho^{n+1}, \tag{B.1}$$

$$\mathcal{L}\mathbf{A}^{n+1} = 2\mathbf{A}^n - \mathbf{A}^{n-1} + \frac{\mu_0}{\alpha^2}\mathbf{J}^{n+1}, \tag{B.2}$$

$$\frac{\psi^{n+1} - \psi^n}{c^2\Delta t} + \nabla \cdot \mathbf{A}^{n+1} = 0, \tag{B.3}$$

where we have used the usual operator notation

$$\mathcal{L} := \mathcal{I} - \frac{1}{\alpha^2}\Delta, \quad \alpha := \frac{1}{c\Delta t}. \tag{B.4}$$

We can verify that this semi-discrete system is time consistent in the sense of the semi-discrete Lorenz gauge (B.3) through a direct calculation.

First, note that the linear operator $\mathcal{L}$ can be inverted in the equation for the scalar potential to obtain the update

$$\psi^{n+1} = \mathcal{L}^{-1}\left(2\psi^n - \psi^{n-1} + \frac{1}{\alpha^2\epsilon_0}\rho^{n+1}\right). \tag{B.5}$$

If we evaluate this equation at time level n, we obtain

$$\psi^n = \mathcal{L}^{-1}\left(2\psi^{n-1} - \psi^{n-2} + \frac{1}{\alpha^2\epsilon_0}\rho^n\right). \tag{B.6}$$

Next, we take the divergence of $\mathbf{A}$ in equation (B.2) and find that

$$\mathcal{L}\left(\nabla\cdot\mathbf{A}^{n+1}\right) = 2\nabla\cdot\mathbf{A}^{n} - \nabla\cdot\mathbf{A}^{n-1} + \frac{\mu_0}{\alpha^2}\nabla\cdot\mathbf{J}^{n+1}.$$

Formally inverting the operator $\mathcal{L}$, we obtain the relation

$$\nabla\cdot\mathbf{A}^{n+1} = \mathcal{L}^{-1}\left(2\nabla\cdot\mathbf{A}^{n} - \nabla\cdot\mathbf{A}^{n-1} + \frac{\mu_0}{\alpha^2}\nabla\cdot\mathbf{J}^{n+1}\right). \tag{B.7}$$

We now use equations (B.5), (B.7), and (B.6) to evaluate the semi-discrete Lorenz gauge (B.3). Using the linearity of the operator $\mathcal{L}$, we obtain

$$\frac{\psi^{n+1}-\psi^{n}}{c^2\Delta t} + \nabla\cdot\mathbf{A}^{n+1} = \mathcal{L}^{-1}\left[\frac{2\psi^{n}-3\psi^{n-1}+\psi^{n-2}}{c^2\Delta t} + 2\nabla\cdot\mathbf{A}^{n} - \nabla\cdot\mathbf{A}^{n-1} + \frac{\mu_0}{\alpha^2}\left(\frac{\rho^{n+1}-\rho^{n}}{\Delta t} + \nabla\cdot\mathbf{J}^{n+1}\right)\right].$$

Note that we have used the relation $c^2 = (\mu_0\epsilon_0)^{-1}$. From these calculations, we can see that the corresponding semi-discrete continuity equation appears as a residual for the gauge condition (B.3). The remaining terms in the operand for the inverse can be also be expressed directly in terms of this semi-discrete gauge, since

$$\frac{2\psi^{n}-3\psi^{n-1}+\psi^{n-2}}{c^2\Delta t} + 2\nabla\cdot\mathbf{A}^{n} - \nabla\cdot\mathbf{A}^{n-1} = 2\left(\frac{\psi^{n}-\psi^{n-1}}{c^2\Delta t} + \nabla\cdot\mathbf{A}^{n}\right) - \left(\frac{\psi^{n-1}-\psi^{n-2}}{c^2\Delta t} + \nabla\cdot\mathbf{A}^{n-1}\right).$$

This gives rise to an inductive argument for the time consistency. The initial data for the problem satisfies both the semi-discrete gauge condition and the continuity equation. If the discrete gauge condition

$$\frac{\psi^{n+1}-\psi^{n}}{c^2\Delta t} + \nabla\cdot\mathbf{A}^{n+1} = 0,$$

holds for any time level n, then it follows that the analogous semi-discrete continuity equation

$$\frac{\rho^{n+1}-\rho^{n}}{\Delta t} + \nabla\cdot\mathbf{J}^{n+1} = 0,$$

holds as well. We briefly sketch the idea for both directions.

The forward direction can be easily seen by assuming that the semi-discrete gauge condition holds up to time level $n+1$, which is equivalent to writing

$$0 = \mathcal{L}^{-1}\left[\frac{\mu_0}{\alpha^2}\left(\frac{\rho^{n+1}-\rho^{n}}{\Delta t} + \nabla\cdot\mathbf{J}^{n+1}\right)\right].$$

The result follows by applying the operator $\mathcal{L}$ to both sides.

A similar argument can be used for the converse. The discrete gauge condition is assumed to be satisfied by the initial condition and all relevant earlier times, i.e.,

$$\frac{\psi^{n+1} - \psi^n}{c^2 \Delta t} + \nabla \cdot \mathbf{A}^{n+1} = 0, \quad n = -2, -1.$$

Now, we assume that the continuity equation

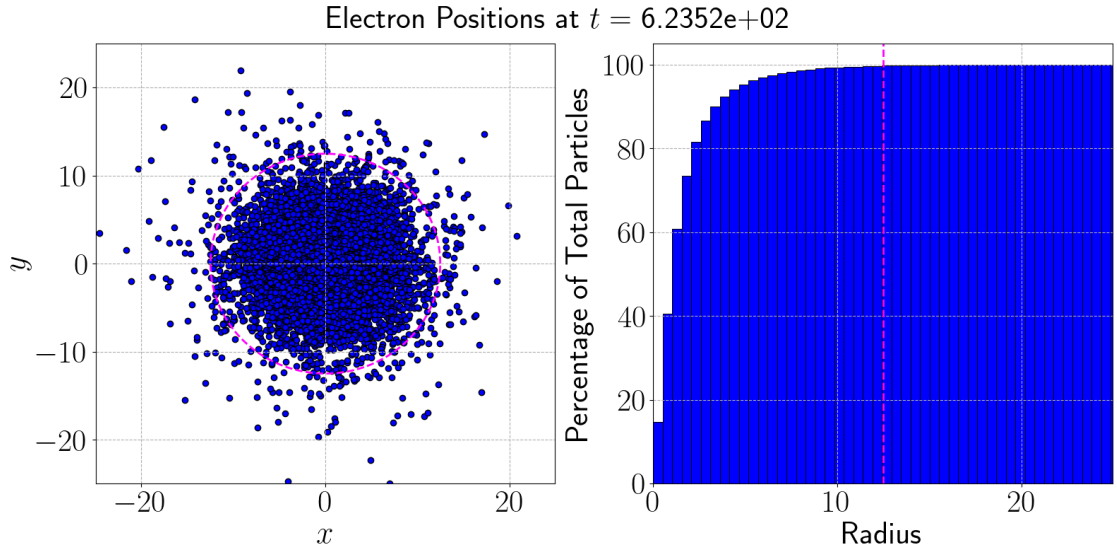
$$\frac{\rho^{n+1} - \rho^n}{\Delta t} + \nabla \cdot \mathbf{J}^{n+1} = 0,$$

is true for any time level n. Then, the gauge condition at $n = 0$ also satisfied because

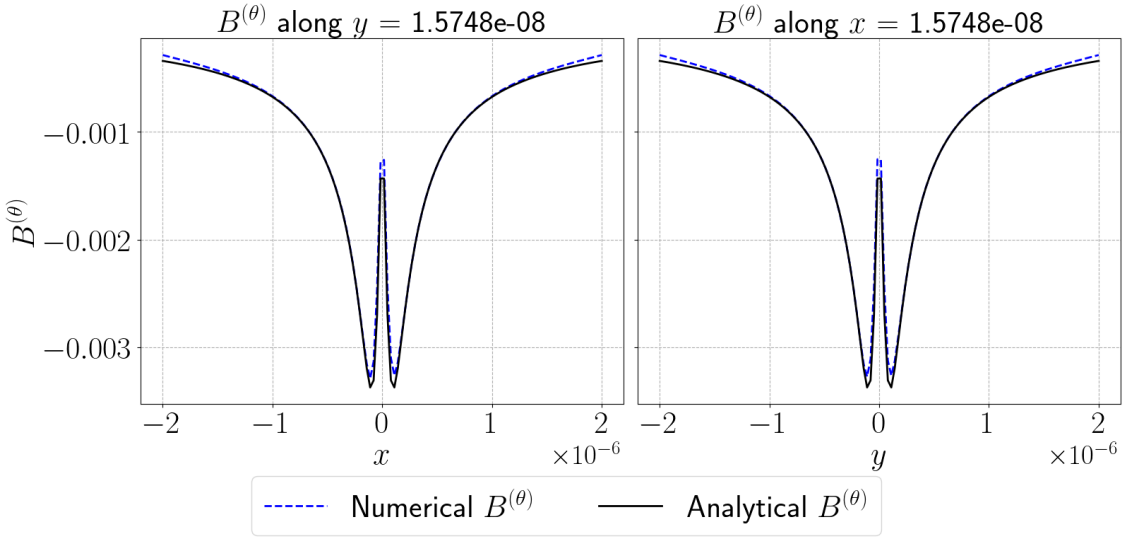
$$\frac{\psi^1 - \psi^0}{c^2 \Delta t} + \nabla \cdot \mathbf{A}^1 = \mathcal{L}^{-1}\,[0] \equiv 0.$$

This argument can be iterated n more times to obtain the result.

B.2 Some Larger Figures from Experiments

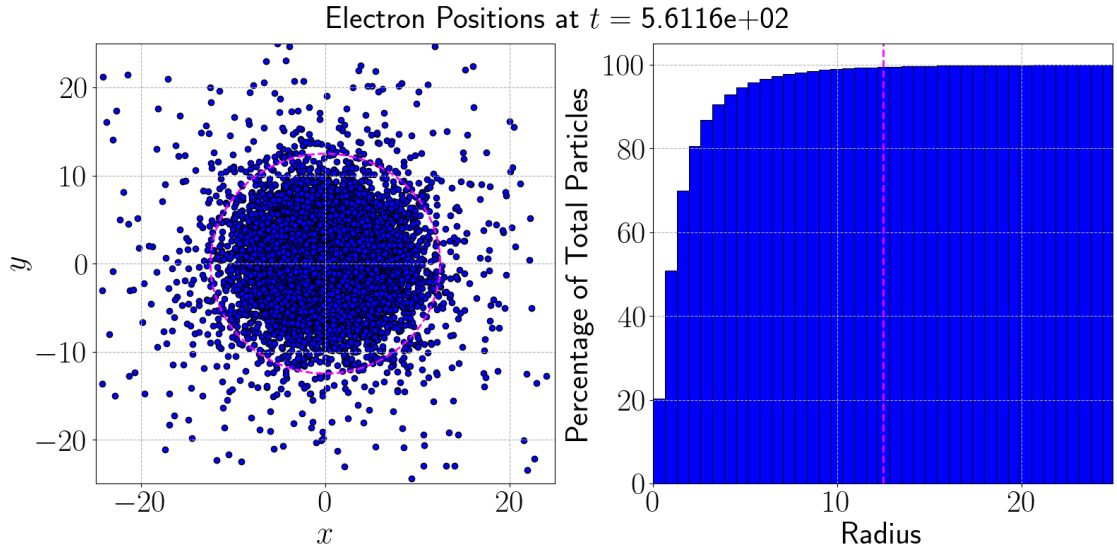


(a) The beam of electrons (left) and the corresponding distribution in terms of their radii (right)

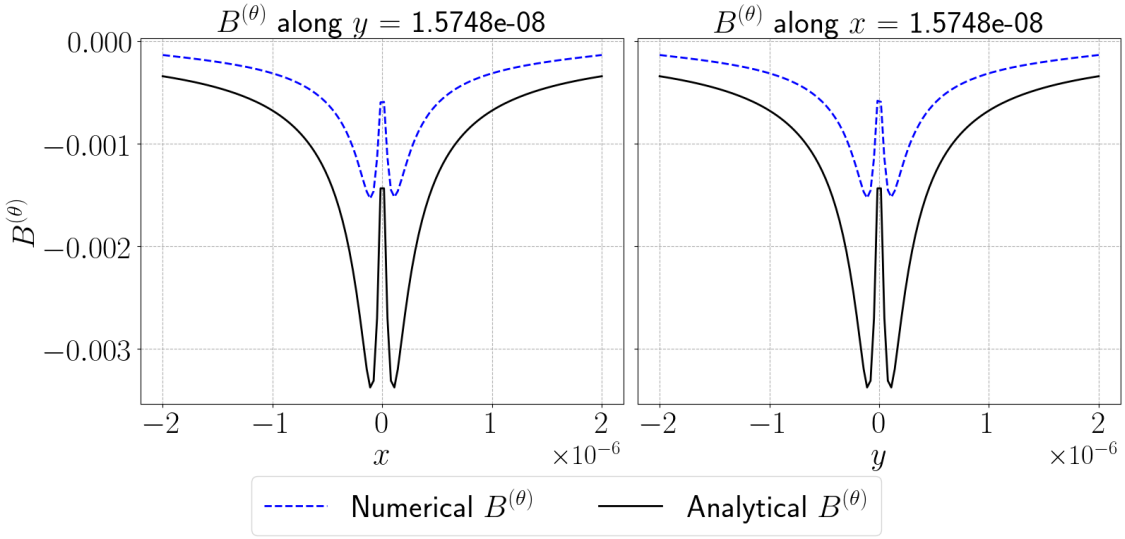


(b) Slices in x (left) and y (right) of the toroidal magnetic field $B^{(\theta)}(r)$

Figure B.1: The state of the Bennett problem after 50 thermal crossing times using the Boris method with the steady-state Poisson model for the fields. The top figure shows the electrons in the non-dimensional grid and plots the radius of the beam as a reference. We also include a cumulative histogram of the electrons based on their radii, which uses a total of 50 bins. The plots on the bottom are cross-sections of the steady-state magnetic field $B^{(\theta)}$, which are plotted against the analytical field. We see good agreement in the magnetic field with its analytical solution, which is enough to confine most of the particles within the beam.

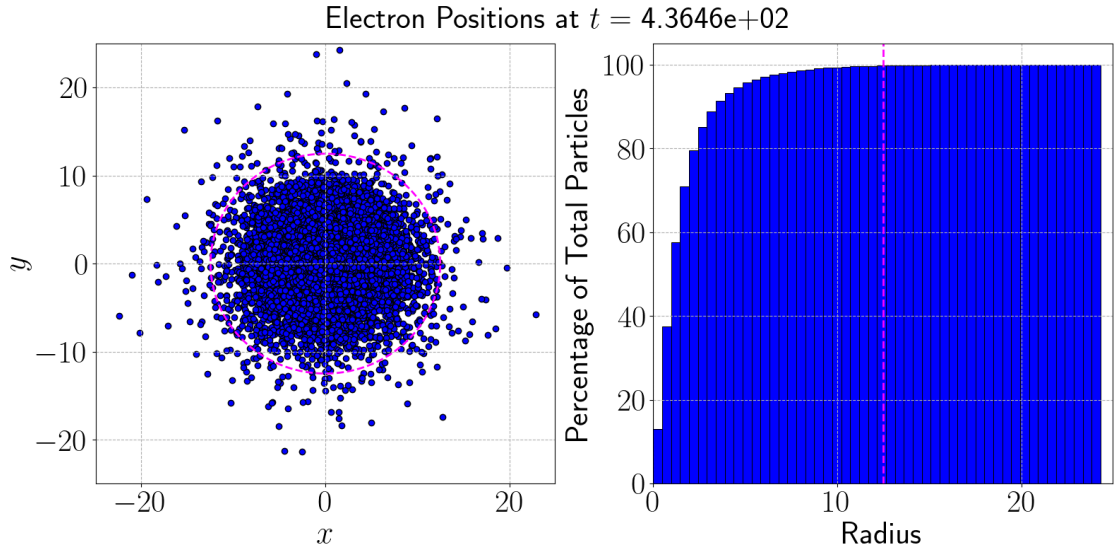


(a) The beam of electrons (left) and the corresponding distribution in terms of their radii (right)

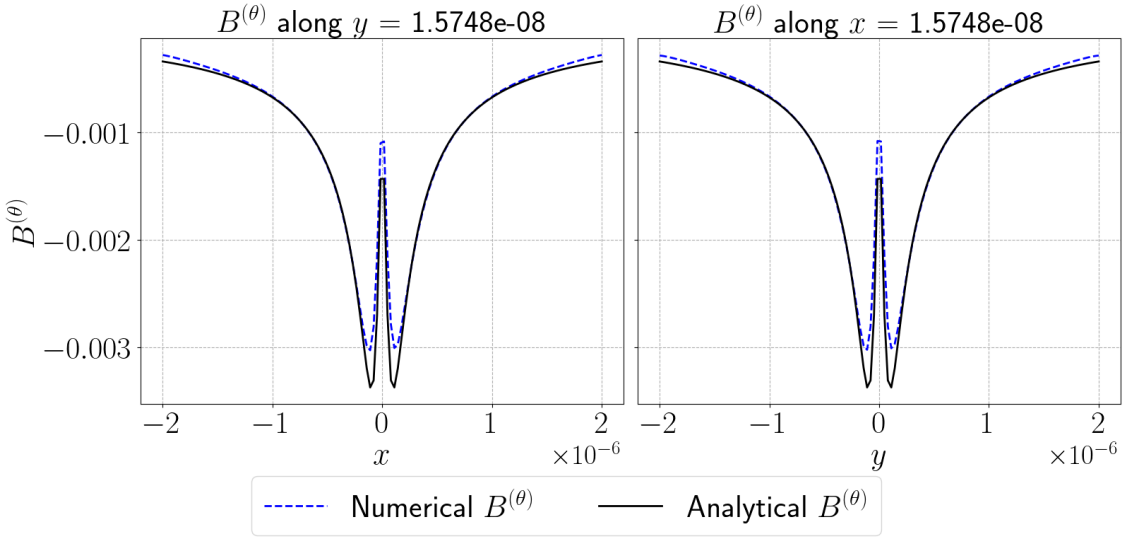


(b) Slices in x (left) and y (right) of the toroidal magnetic field $B^{(\theta)}(r)$

Figure B.2: The state of the Bennett problem after 45 thermal crossing times obtained with the Molei Tao method (ω = 500) using the steady-state Poisson model for the fields. The top figure shows the electrons in the non-dimensional grid and plots the radius of the beam as a reference. We also include a cumulative histogram of the electrons based on their radii, which uses a total of 50 bins. The plots on the bottom are slices of the steady-state magnetic field $B^{(\theta)}$, which is plotted against the analytical field. We observe a significant drift in the numerical field away from its steady-state that results in a loss of confinement of the particles to the beam.

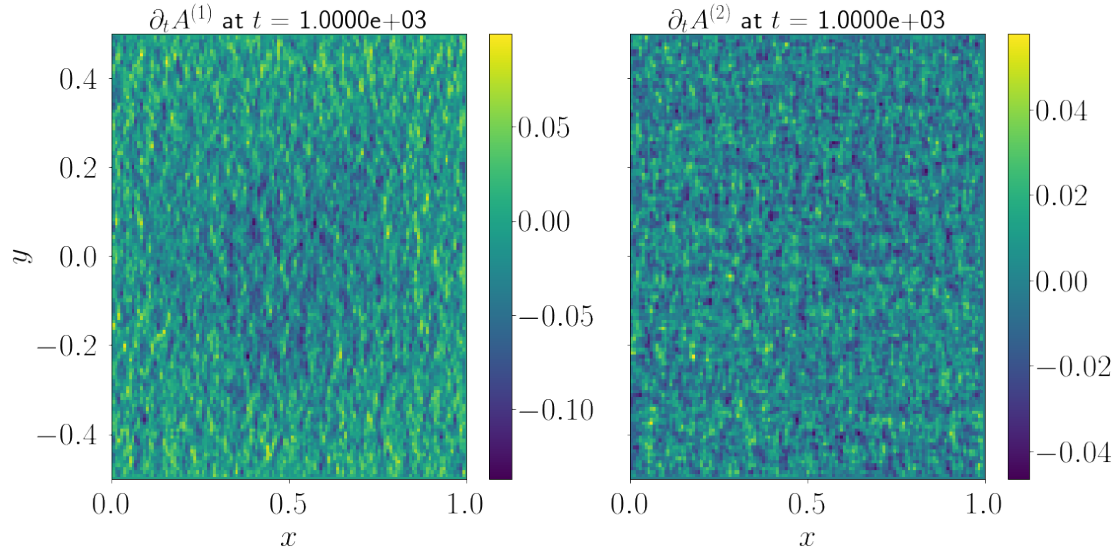


(a) The beam of electrons (left) and the corresponding distribution in terms of their radii (right)

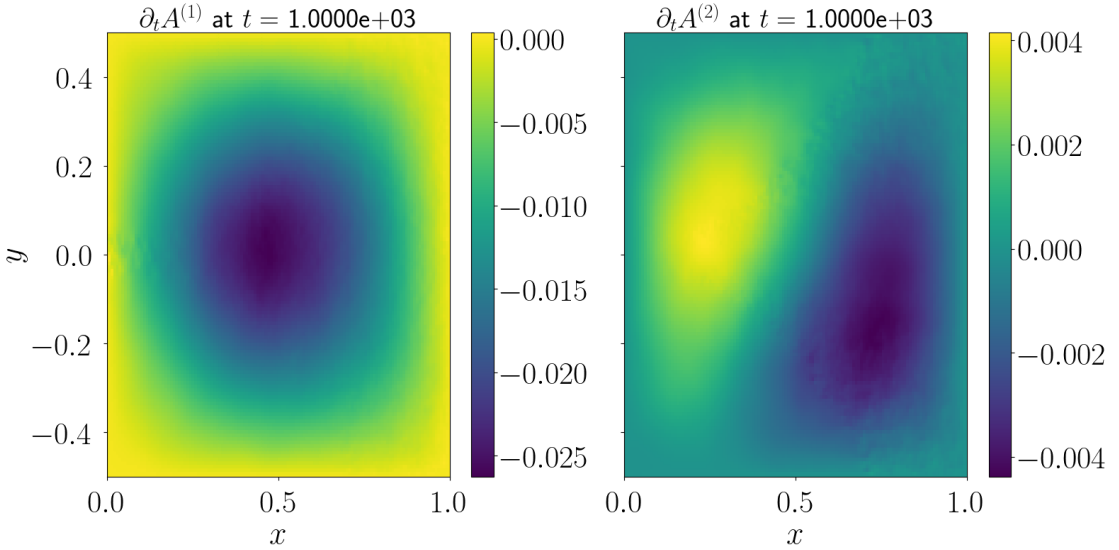


(b) Slices in x (left) and y (right) of the toroidal magnetic field $B^{(\theta)}(r)$

Figure B.3: The state of the Bennett problem after 35 thermal crossing times using the Boris method with the wave model for the fields. The top figure shows the electrons in the non-dimensional grid and plots the radius of the beam as a reference. We also include a cumulative histogram of the electrons based on their radii. Again, the beam radius is indicated as a reference. A total of 50 bins are used in the plot. The plots on the bottom are slices of the steady-state magnetic field $B^{(\theta)}$, which is plotted against the analytical field. We see good agreement in the magnetic field with its analytical solution, which is enough to confine most of the particles within the beam.

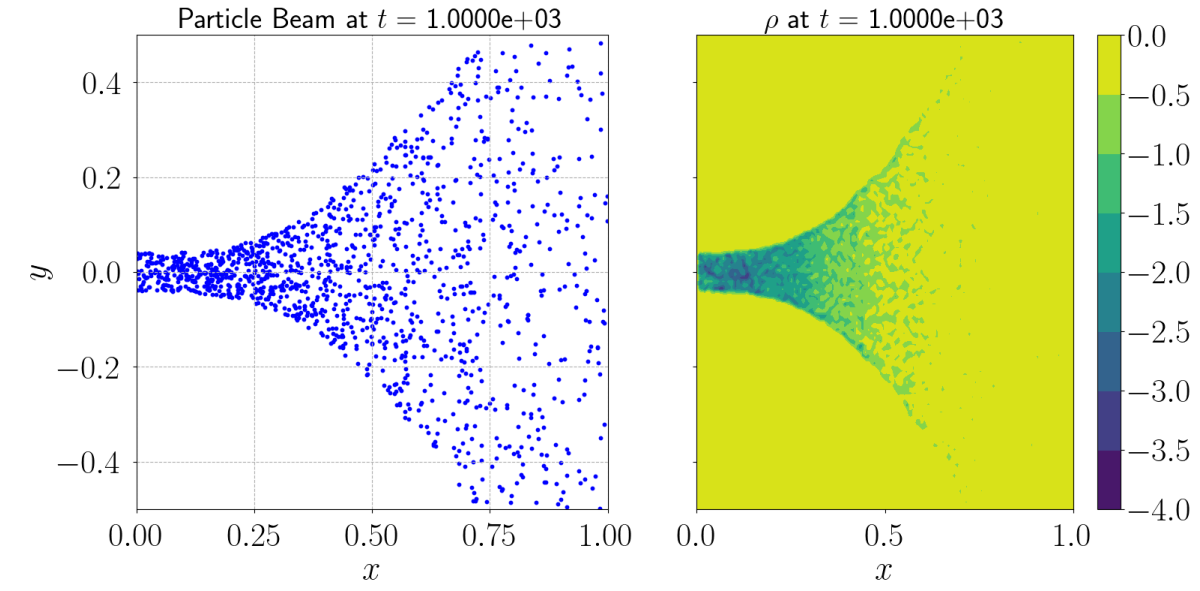


(a) Time-centered method

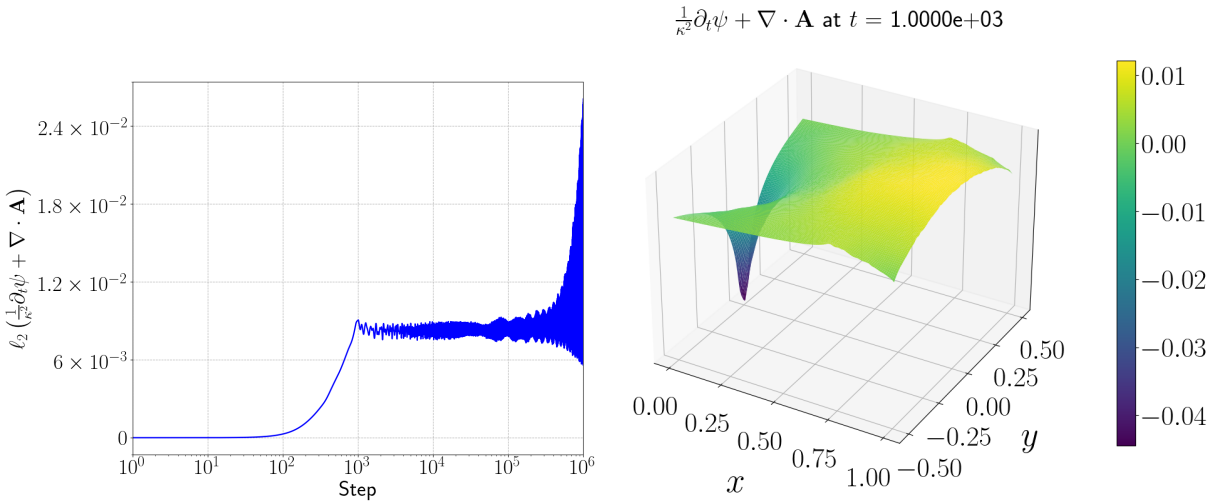


(b) BDF method

Figure B.4: A comparison of the time derivatives of the vector potentials after 1000 particle crossings for the expanding beam problem. This particular data was obtained using the Lorenz gauge formulation for the fields with the Boris method for particles. In the top row, the vector potentials are updated with the time-centered approach, which is purely dispersive and generates noisy time derivatives. The bottom row performs the same experiment, but uses the BDF method, which is purely dissipative. The differences in the quality of the results are quite apparent. This was discussed in [42], but results were not shown to illustrate the severity of the effects due to dispersion.

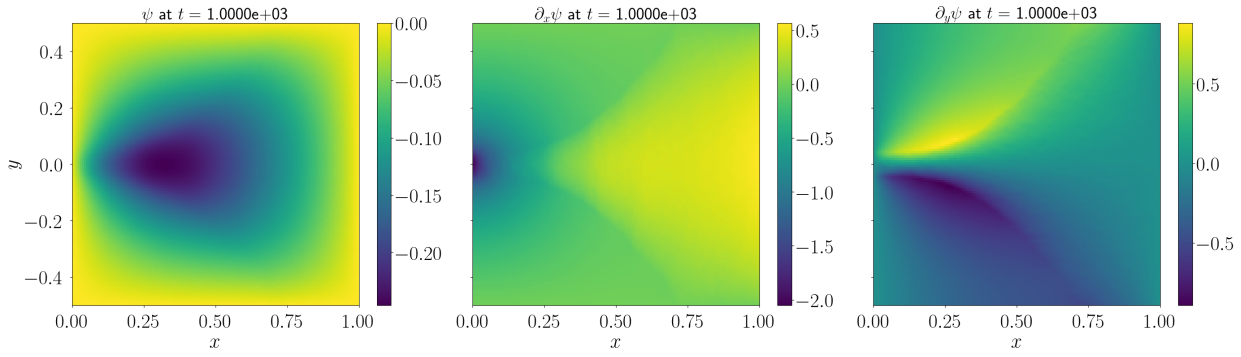


(a)

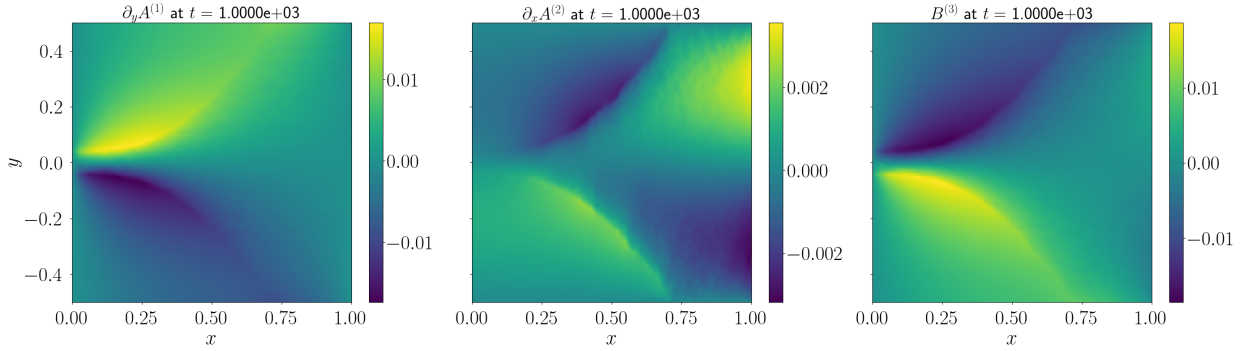


(b) (c)

Figure B.5: We plot the expanding beam after 1000 particle crossings obtained with the Lorenz gauge formulation that combines the Boris method with the BDF-2 field solver. In Figure B.5a, we plot the beam and the corresponding charge density. We observe some oscillations along the top edge of the beam, which also appear in the charge density. In Figure B.5b, we observe an increase in the size of violations of the Lorenz gauge condition, which indicates that the method will eventually fail. We plot the Lorenz gauge error as a surface in Figure B.5c using data from the final step. The most significant violations occur near the injection region and along the boundary where particles are removed.



(a)



(b)

Figure B.6: Here we show the potentials (and their derivatives) for the expanding beam problem after 1000 particle crossings. This data was obtained using the Lorenz gauge formulation which combines the Boris method with the BDF-2 wave solver. The first row plots the scalar potential ψ and its partial derivatives. Similarly, in the second row, we plot the derivatives of the vector potentials $A^{(1)}$ and $A^{(2)}$, which are used to construct the magnetic field $B^{(3)}$ (shown in the right-most plot). Note that the time derivative data for the vector potentials were plotted in Figure B.4b, so we exclude them here.

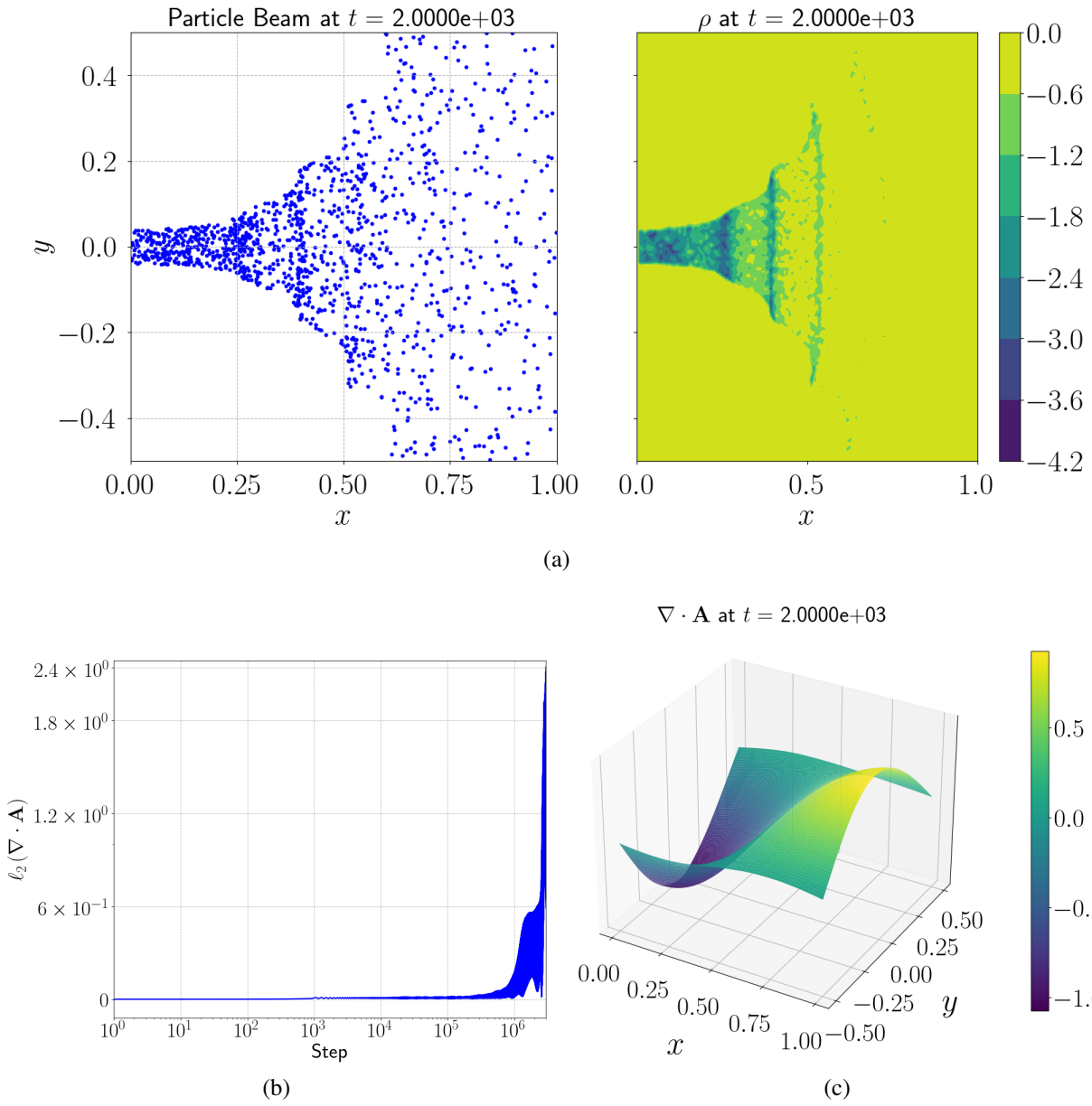


Figure B.7: We show the expanding beam after 2000 particle crossings obtained with the Coulomb gauge formulation, which uses the AEM for time stepping without a cleaning method. In Figure B.7a, we plot the beam and the corresponding charge density, which show visible striations and oscillations along the edge of the beam due to violations in the gauge condition. The growth in the errors associated with the gauge condition is reflected in Figure B.7b, which exhibits unbounded growth. The surface plot of the gauge condition at 2000 crossings shows large errors, especially near the injection region and along the boundary where particles are removed.

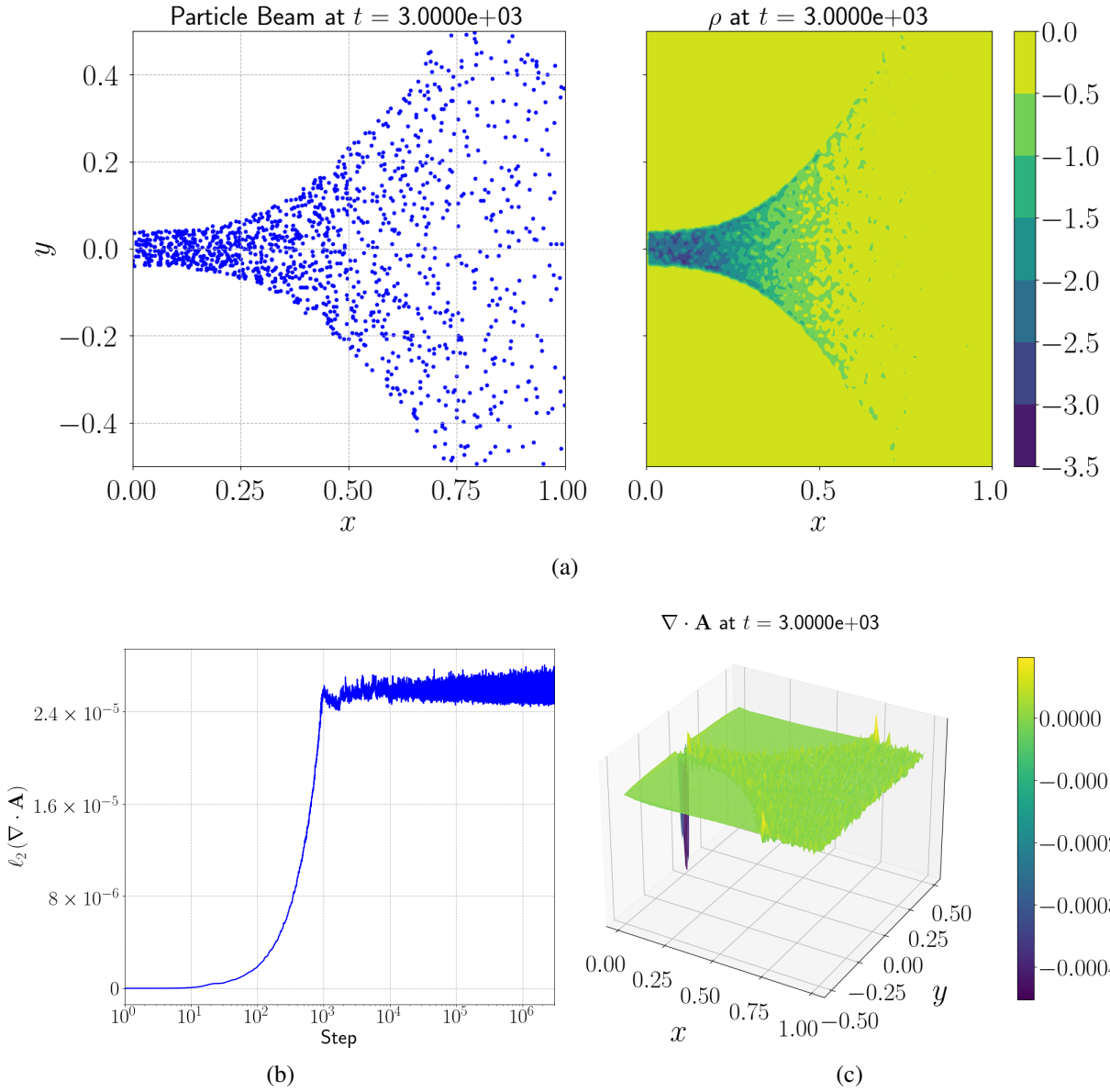
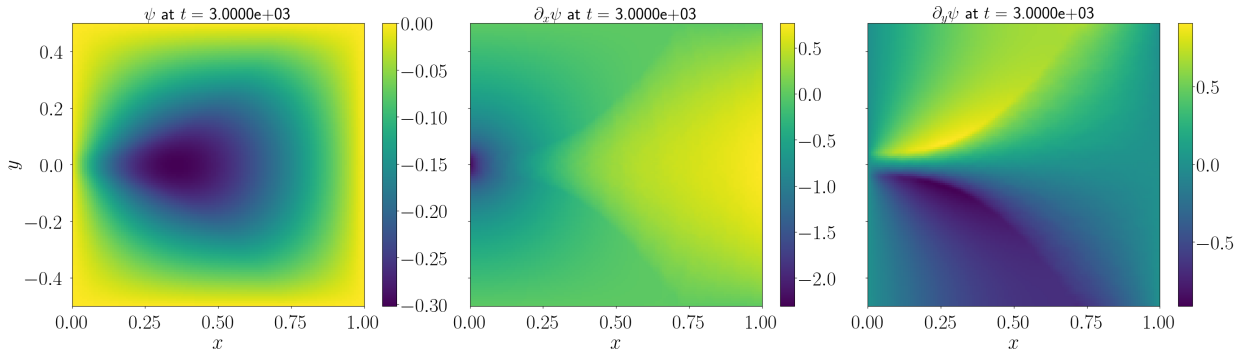
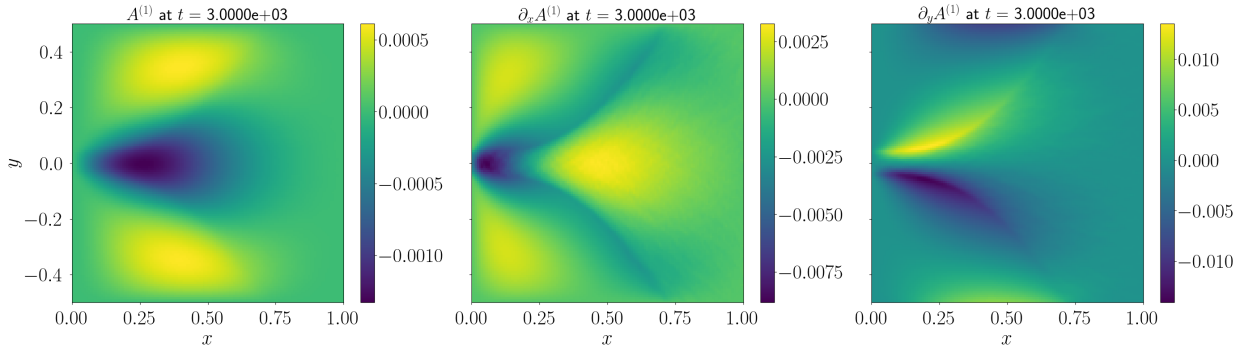


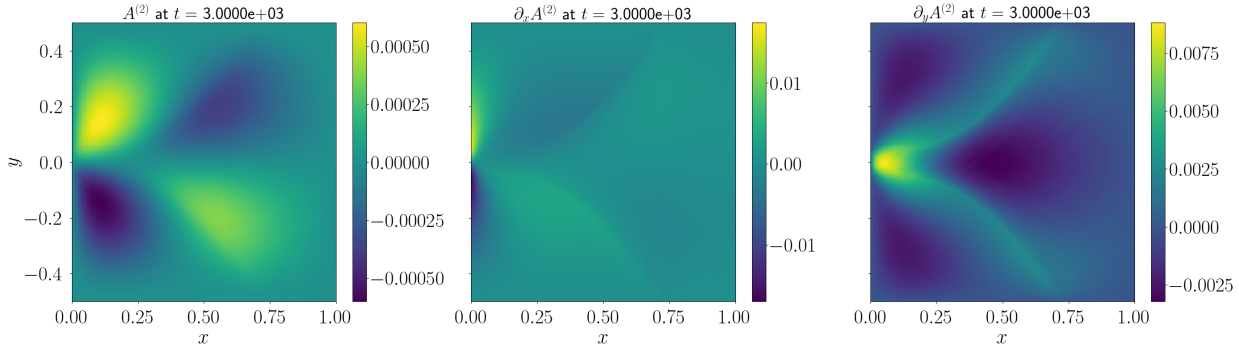
Figure B.8: We show the expanding beam after 3000 particle crossings obtained with the Coulomb gauge formulation that uses the AEM for time stepping with elliptic divergence cleaning. In Figure B.8a, we plot the beam and the corresponding charge density. The elliptic divergence cleaning seems effective at controlling the errors in the gauge condition, compared to the results shown in Figure B.7, which do not apply the cleaning method. The fluctuations of the gauge error away from the boundaries is now in the sixth decimal position, which is a notable improvement over the result shown in Figure B.7c.



(a)

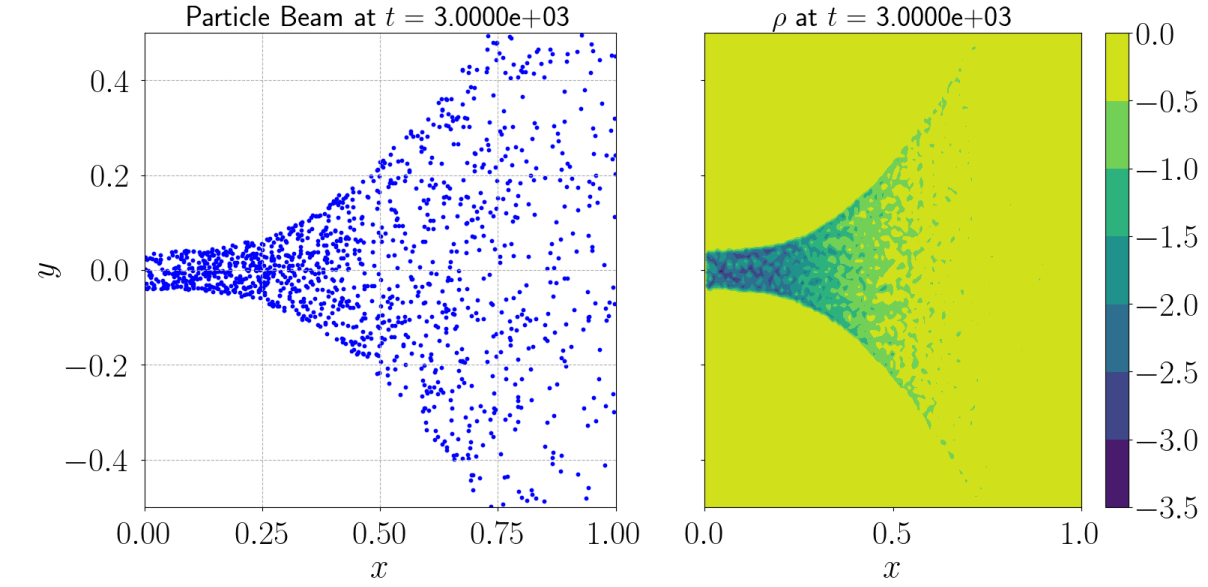


(b)

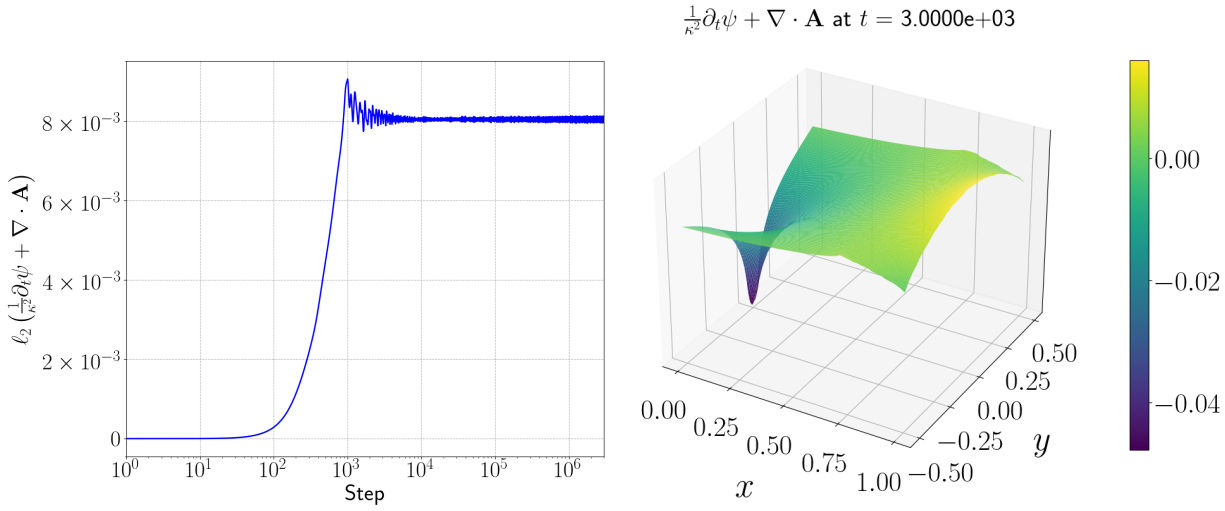


(c)

Figure B.9: Here we show the potentials (and their derivatives) for the expanding beam problem after 3000 particle crossings. This data was obtained using the Coulomb gauge formulation which combines the AEM for time integration with the BDF-2 wave solver. Elliptic divergence cleaning was applied to the vector potential. In each row, we plot a field quantity and is corresponding derivatives. The top row shows the scalar potential ψ and its derivative, which are computed with a finite-differences. The middle and last row show the vector potential components $A^{(1)}$ and $A^{(2)}$, respectively, along with their derivatives, which are computed with the BDF method.

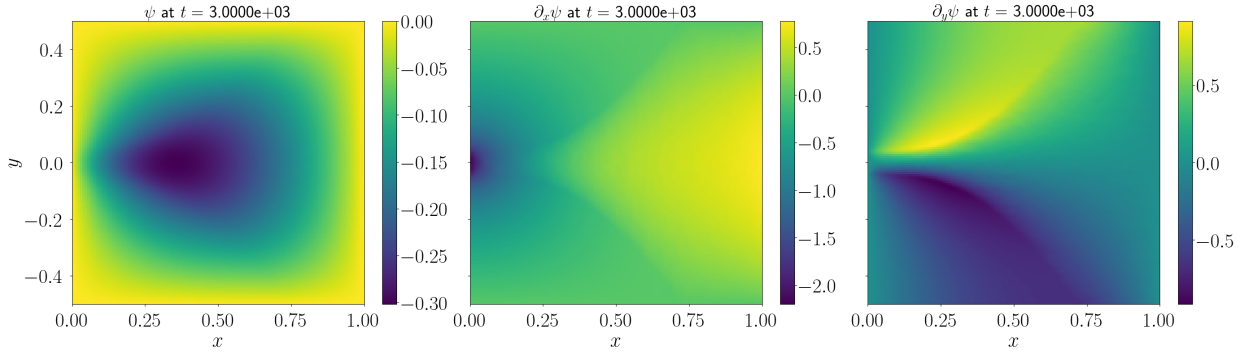


(a)

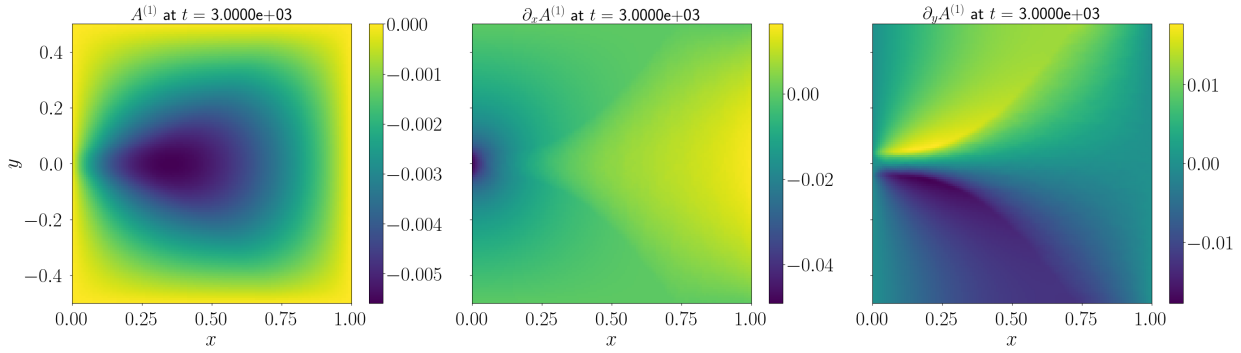


(b) (c)

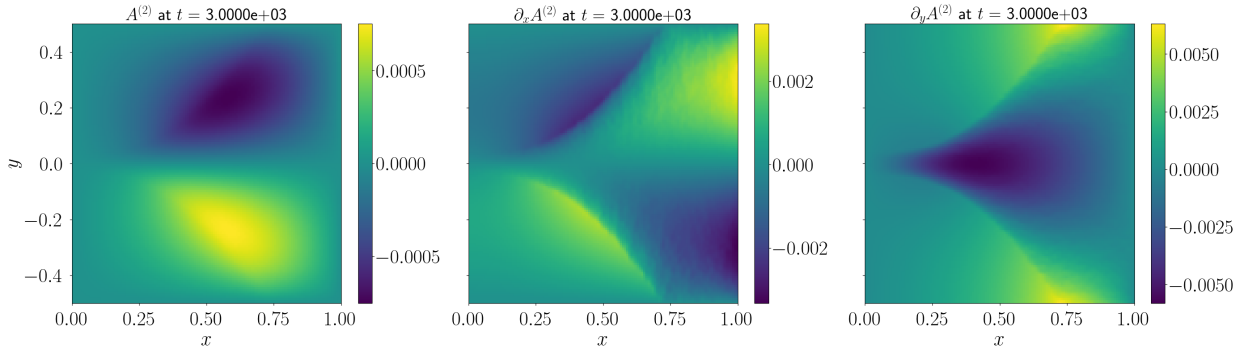
Figure B.10: We show the expanding beam after 3000 particle crossings obtained with the Lorenz gauge formulation that uses the AEM for time stepping along with a first-order BDF solver. No divergence cleaning is applied. In Figure B.10a, we plot the beam and the corresponding charge density. The beam surprisingly remains intact after many particle crossings without the use of a cleaning method. The fluctuations of the gauge error over time are quite small. We do not observe the growth in the gauge error shown earlier in Figure B.5b for the Boris method.



(a)

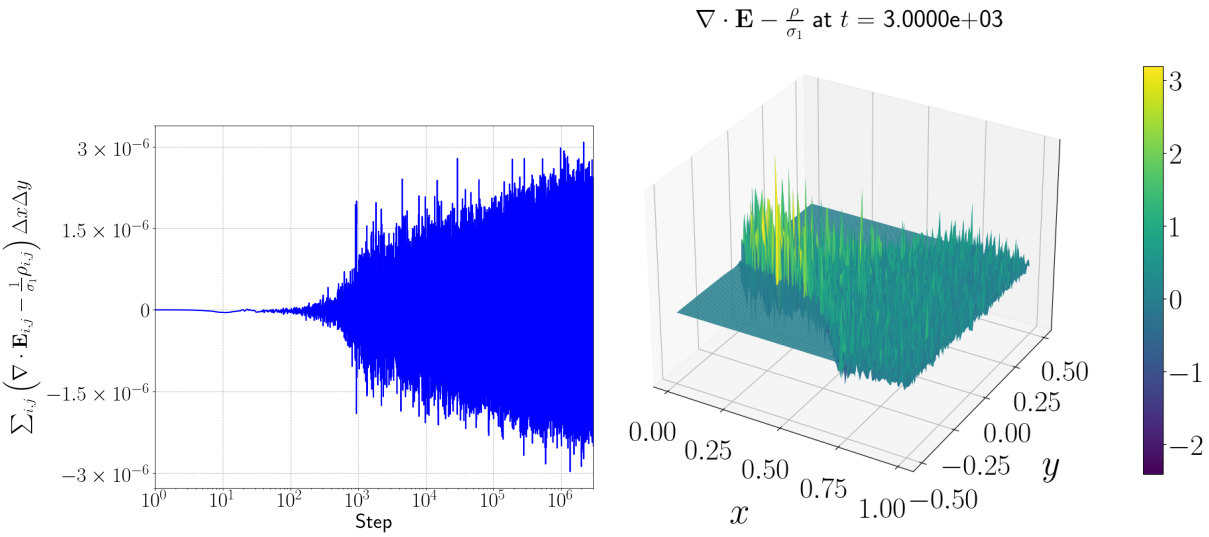


(b)



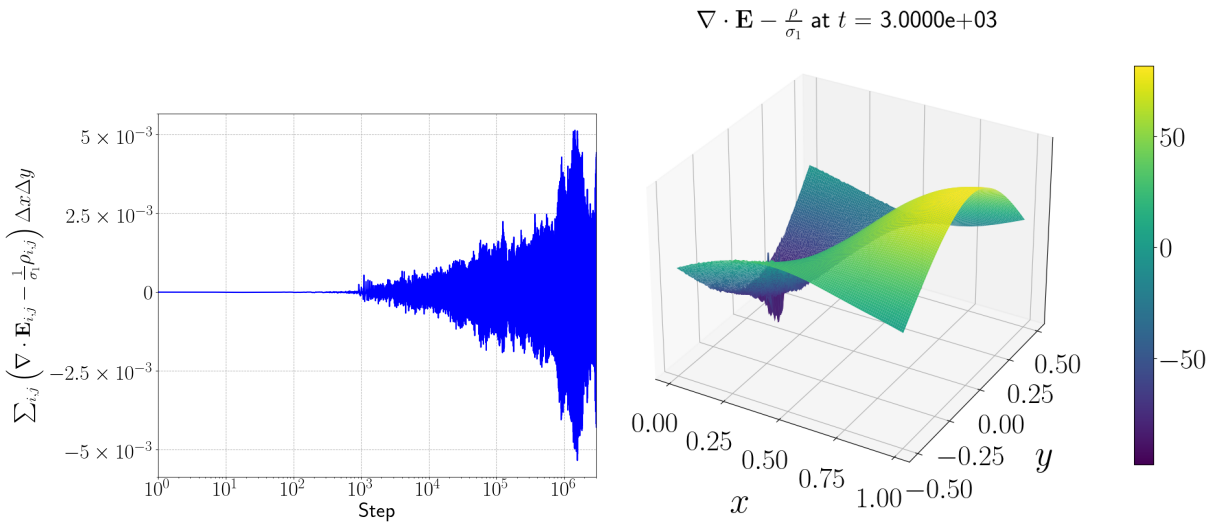
(c)

Figure B.11: Here we show the potentials (and their derivatives) for the expanding beam problem after 3000 particle crossings. This data was obtained using the Lorenz gauge formulation which combines the AEM for time integration with the BDF-1 wave solver. A divergence cleaning method is not used in this example. In each row, we plot a field quantity and is corresponding derivatives. The top row shows the scalar potential ψ and its derivative, while the middle and last row shows the vector potential components $A^{(1)}$ and $A^{(2)}$, respectively, along with their derivatives.



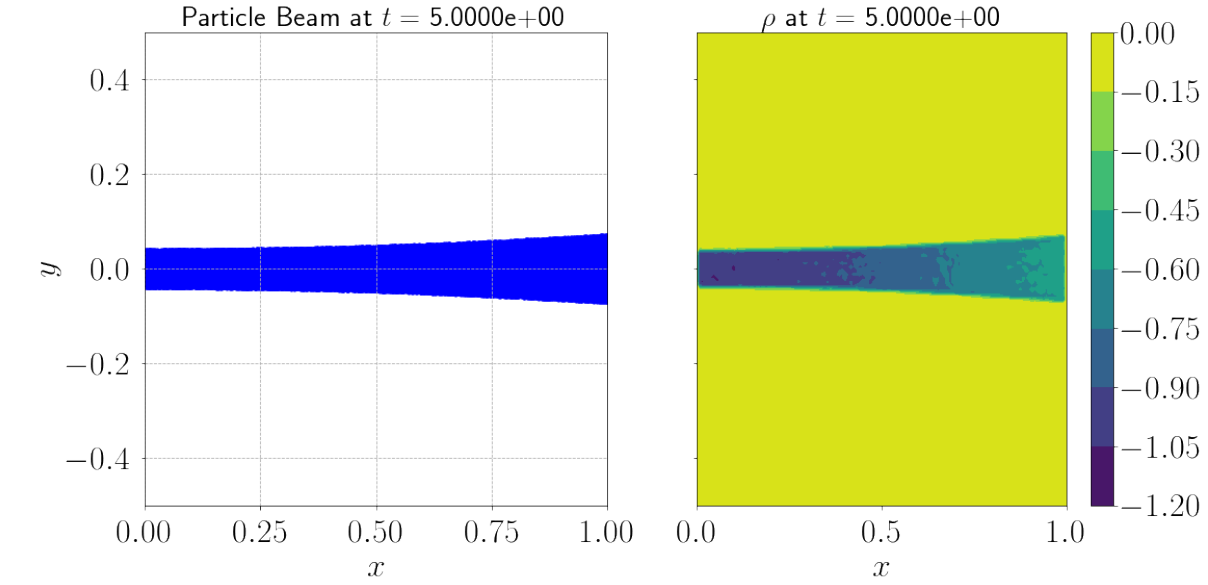
(a) (b)

Figure B.12: Error in Gauss' law for the Coulomb gauge formulation of the expanding beam problem which applies the AEM for time integration and uses elliptic divergence cleaning. On the left, we show the time evolution of an "averaged" residual in Gauss' law. The plot on the right is a surface of the error in Gauss' law taken after 3000 particle crossings. Even though cleaning is used to control violations in the gauge condition, whose corresponding surface was shown in Figure B.8c, the metric based on point-wise violations in Gauss' law seems to indicates a significant loss of conservation. On the other hand, the plot on the left implies that Gauss' law is satisfied in an integral sense.

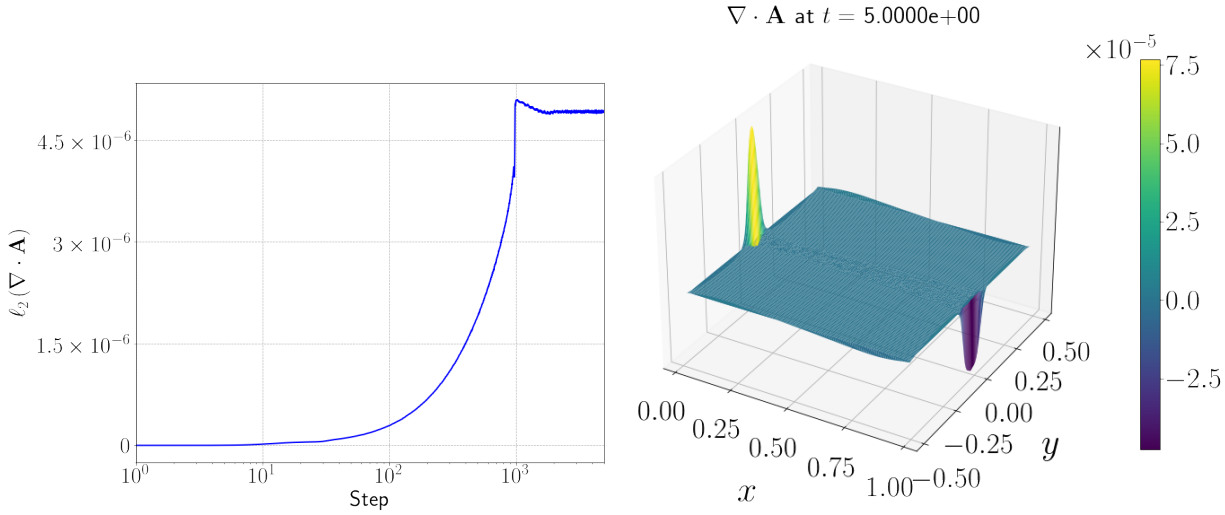


(a) (b)

Figure B.13: Error in Gauss' law for the Coulomb gauge formulation of the expanding beam problem which applies the AEM for time integration. Elliptic divergence cleaning is not used here. On the left, we show the time evolution of the "averaged" residual in Gauss' law. The plot on the right is a surface of the error in Gauss' law taken after 3000 particle crossings. The point-wise violations in Gauss' law are much larger than we observed in Figure B.12b. Similarly, the time evolution of the average defect in Gauss' law is roughly three orders of magnitude larger than B.12a.

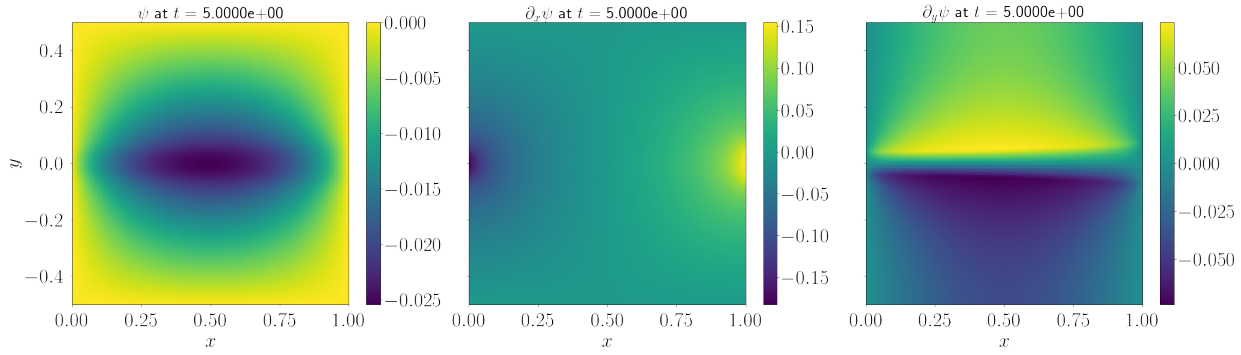


(a)

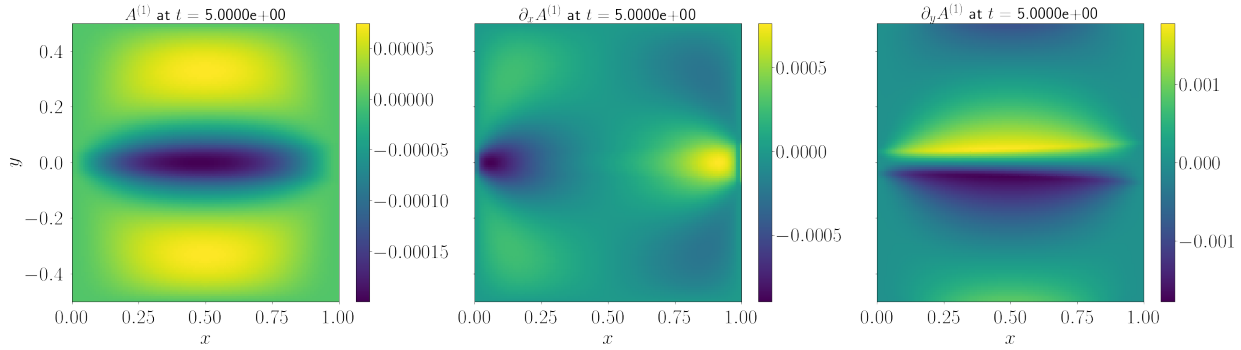


(b) (c)

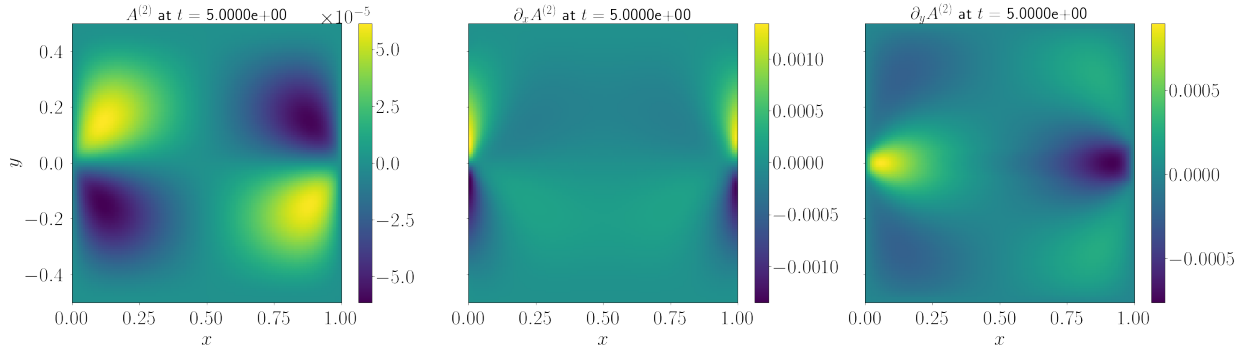
Figure B.14: We show the narrow beam after 5 particle crossings obtained with the Coulomb gauge formulation that uses the AEM for time stepping with elliptic divergence cleaning. We injected 400 particle per time step. In Figure B.14a, we plot the beam and the corresponding charge density. The plot of the particles appears more solid due to the increased injection rate. The density itself is quite smooth due to the use of additional particles. As before, we see there are violations in the gauge condition along the boundaries due to the injection and removal of particles there. Additionally, the gauge error appears to be quite small away from the boundaries due to the increased smoothness offered by the use of additional particles.



(a)

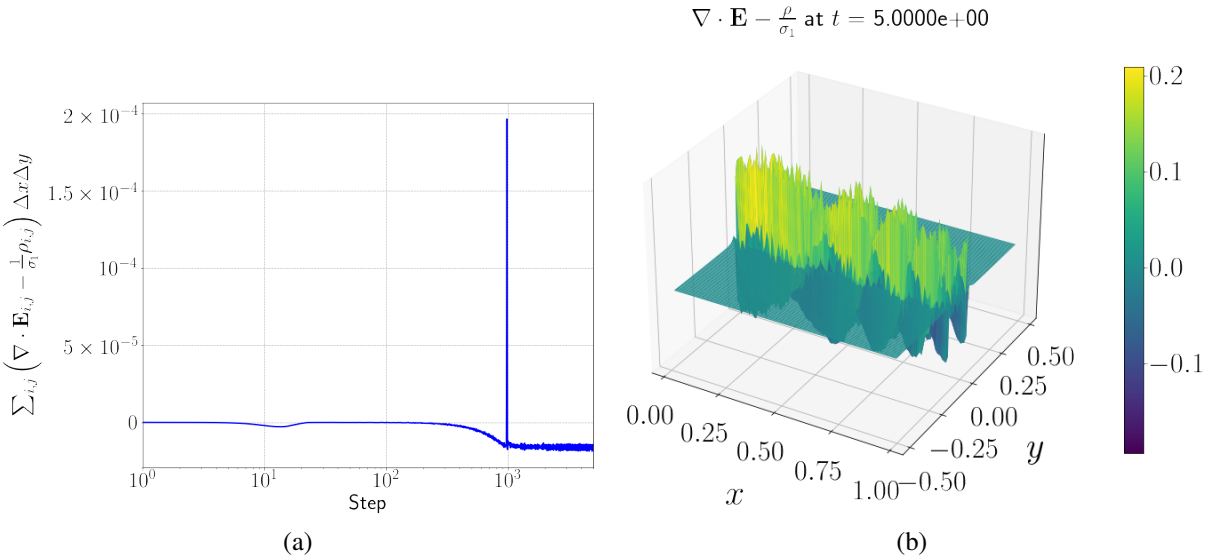


(b)



(c)

Figure B.15: Here we show the potentials, as well as their derivatives, for the narrow beam problem after 5 particle crossings using an injection rate of 400 particles per step. We used the Coulomb gauge formulation which combines the AEM for time integration with the BDF-2 wave solver for the fields. Elliptic divergence cleaning was applied to the vector potential. In each row, we plot a field quantity and is corresponding spatial derivatives. The top row shows the scalar potential ψ and its derivative, which are computed with a finite-differences. The middle and last row show the vector potential components $A^{(1)}$ and $A^{(2)}$, respectively, along with their derivatives, which are computed with the BDF method. The structure of the fields and their derivative are quite smooth here.



(a) (b)

Figure B.16: Error in Gauss' law for the narrow beam problem that uses an injection rate of 400 particles per step. On the left, we show the time evolution of an "averaged" residual in Gauss' law. There is a jump in the "bulk" error for Gauss' law at step 1000, since this coincides with the beam's first crossing, before stabilizing. The plot on the right is a surface of the error in Gauss' law taken after 5 particle crossings. Even though cleaning is used to control violations in the gauge condition, whose corresponding surface was shown in Figure B.14c, the metric based on point-wise violations in Gauss' law seems to indicates a loss of charge conservation similar to the previous example.

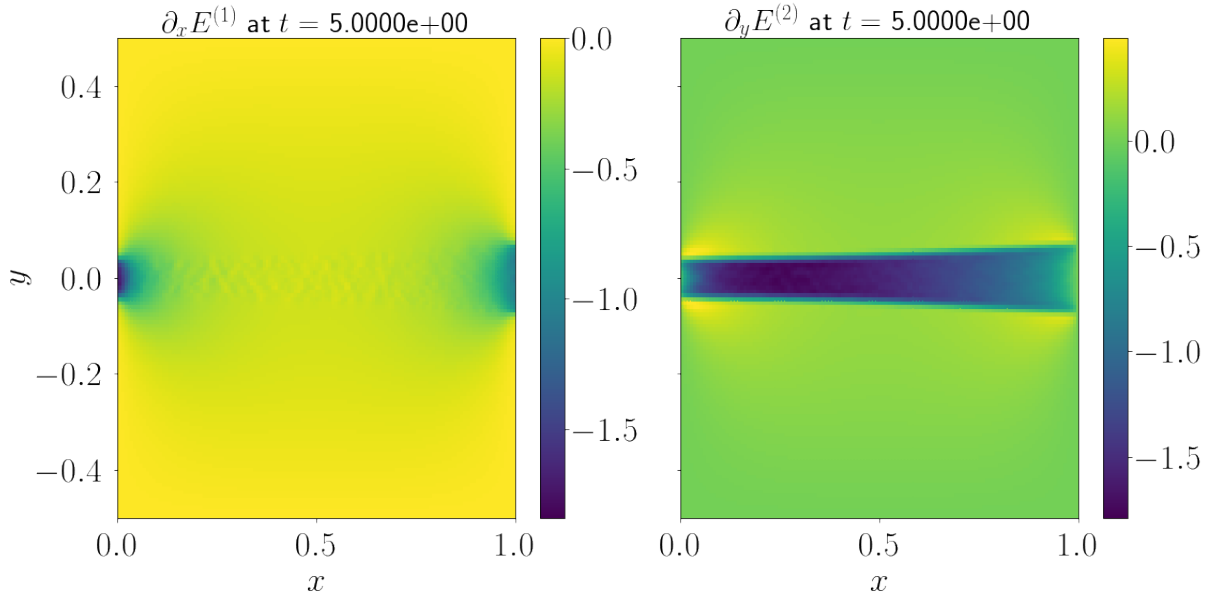


Figure B.17: We show the derivatives used to calculate the divergence of the electric field for the narrow beam problem at 5 particle crossings. We used an injection rate of 400 particles. Derivatives are computed with second-order finite-differences. We note the appearance of small oscillations in the x derivative, which is shown on the left. The plot to the right, which corresponds to the y-derivative is largely uniform on the interior of the beam, but is sharp along the edge of the beam.

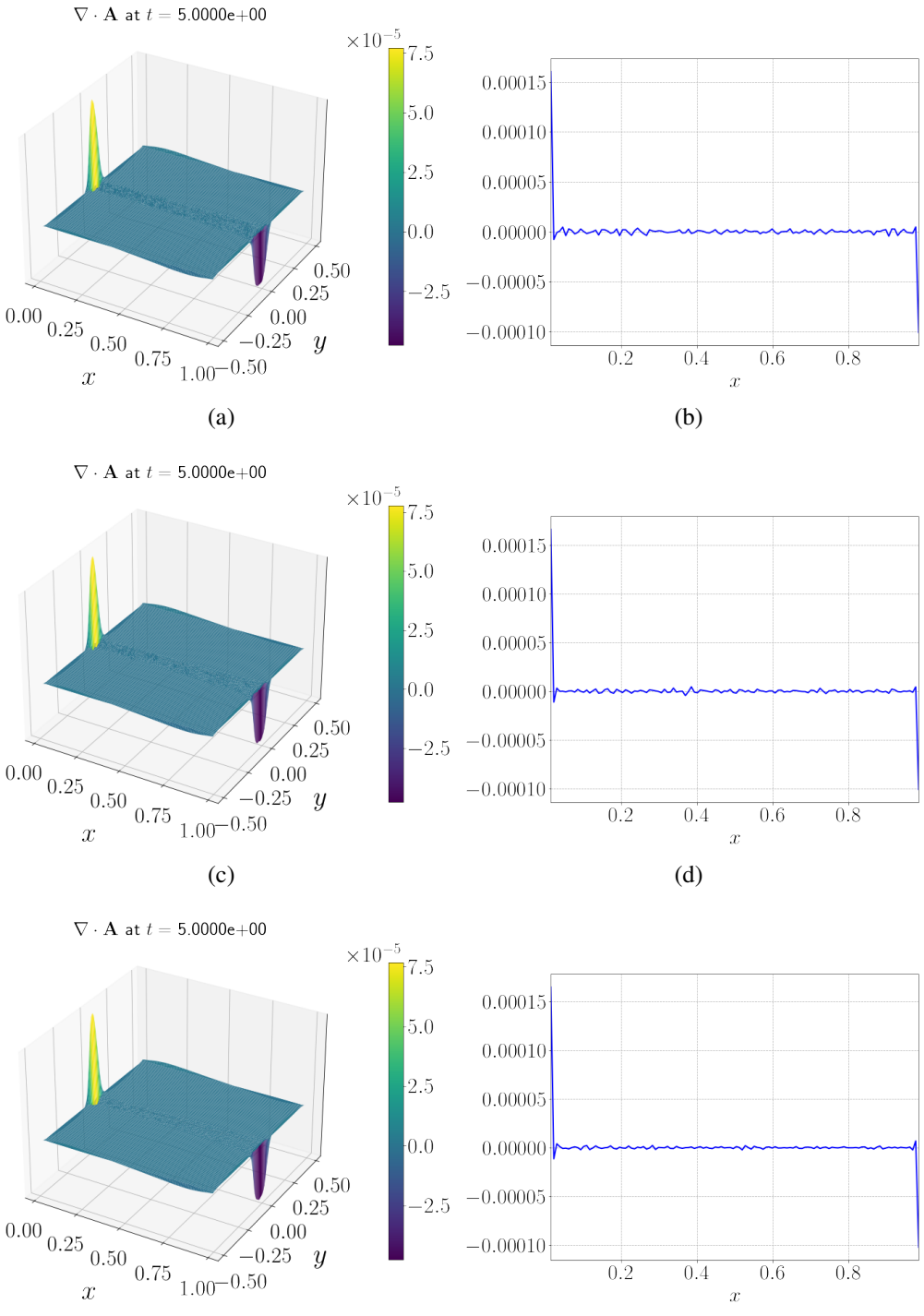


Figure B.18: We show the effect of the particle injection rate on the gauge error for the narrow beam problem at 5 particle crossings. In each row, we plot the error in the Coulomb gauge as a surface (left column) and as a slice in x along the middle of the beam (right) column. The rows correspond to injection rates of 100, 200, and 400 particles per time step, respectively, from top to bottom. We can see that the increase in particle count reduces the gauge error on the interior of the domain due to the smoothing effect on the particle data.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] J. P. Verboncoeur, "Particle simulation of plasmas: Review and advances," *Plasma Physics and Controlled Fusion*, vol. 47, A231–A260, 5A 2005.

[2] C. K. Birdsall and A. B. Langdon, *Plasma Physics via Computer Simulation*. McGraw-Hill Book Company, 1985.

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

[4] J. Boris, "Relativistic plasma simulation-optimization of a hybrid code," in *Proceedings of the Fourth Conference on Numerical Simulations of Plasmas*, 1970, pp. 3–67.

[5] A. Langdon, B. Cohen, and A. Friedman, "Direct implicit large time-step particle simulation of plasmas," *Journal of Computational Physics*, vol. 51, pp. 107–138, 1 1981.

[6] J. Brackbill and D. Forslund, "An implicit method for electromagnetic plasma simulation in two dimensions," *Journal of Computational Physics*, vol. 46, pp. 271–308, 2 1982.

[7] R. Mason, "An electromagnetic field algorithm for 2D implicit plasma simulation," *Journal of Computational Physics*, vol. 71, pp. 429–473, 2 1987.

[8] B. Cohen, A. Langdon, D. Hewett, and R. Procassini, "An implicit method for electromagnetic plasma simulation in two dimensions," *Journal of Computational Physics*, vol. 81, pp. 151–168, 1 1989.

[9] G. Chen, L. Chacón, and D. Barnes, "An energy- and charge-conserving, implicit, electrostatic particle-in-cell algorithm," *Journal of Computational Physics*, vol. 230, pp. 7018–7036, 18 2011.

[10] D. Knoll and D. Keyes, "Jacobian-free Newton–Krylov methods: A survey of approaches and applications," *Journal of Computational Physics*, vol. 193, pp. 357–397, 2 2004.

[11] L. Chacón, G. Chen, and D. Barnes, "A charge-and energy-conserving implicit, electrostatic particle-in-cell algorithm on mapped computational meshes," *Journal of Computational Physics*, vol. 233, pp. 1–9, 2012.

[12] G. Chen, L. Chacón, L. Yin, B. Albright, J. Stark, and R. Bird, "A semi-implicit, energy- and charge-conserving particle-in-cell algorithm for the relativistic Vlasov-Maxwell equations," *Journal of Computational Physics*, vol. 407, p. 109 228, 2020.

[13] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's

equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, vol. 14, pp. 302–307, 3 1966.

[14] A. Taflove and S. C. Hagness, *Computational electrodynamics: the finite-difference time-domain method*, Third. Artech House Publishers, 2005.

[15] A. D. Greenwood, K. L. Cartwright, J. W. Luginsland, and E. A. Baca, "On the elimination of numerical Cerenkov radiation in PIC simulations," *Journal of Computational Physics*, vol. 201, pp. 665–684, 2 2004.

[16] J. P. Verboncoeur, "Aliasing of electromagnetic fields in stair step boundaries," *Computer physics communications*, vol. 164, pp. 344–352, 1 2004.

[17] B. Engquist, J. Häggblad, and O. Runborg, "On energy preserving consistent boundary conditions for the Yee scheme in 2D," *BIT Numerical Mathematics*, vol. 52, pp. 615–637, 3 2012.

[18] E. Sonnendrücker, J. Ambrosiano, and S. Brandon, "A finite element formulation of the Darwin PIC model for use on unstructured grids," *Journal of Computational Physics*, vol. 121, pp. 281–297, 2 1995.

[19] C.-D. Munz, P. Omnes, R. Schneider, E. Sonnendrücker, and U. Voss, "Divergence correction techniques for Maxwell solvers based on a hyperbolic model," *Journal of Computational Physics*, vol. 161, no. 2, pp. 484–511, 2000.

[20] G. Jacobs and J. Hesthaven, "High-order nodal discontinuous Galerkin particle-in-cell method on unstructured grids," *Journal of Computational Physics*, vol. 214, pp. 96–121, 1 2006.

[21] ——, "Implicit–explicit time integration of a high-order particle-in-cell method with hyperbolic divergence cleaning," *Journal of Computational Physics*, vol. 180, pp. 1760–1767, 10 2009.

[22] M. Pinto, S. Jund, S. Salmon, and E. Sonnendrücker, "Charge-conserving FEM–PIC schemes on general grids," *Comptes Rendus Mécanique*, vol. 342, pp. 570–582, 10-11 2014.

[23] M. Pinto, K. Kormann, and E. Sonnendrücker, *Variational framework for structure-preserving electromagnetic particle-in-cell methods*, 2021. DOI: 10.48550/ARXIV.2101.09247. [Online]. Available: https://arxiv.org/abs/2101.09247.

[24] S. O'Connor, Z. D. Crawford, J. P. Verboncoeur, J. Luginsland, and B. Shanker, "A set of benchmark tests for validation of 3-D particle in cell methods," *IEEE Transactions on Plasma Science*, vol. 49, pp. 1724–1731, 5 Apr. 2021.

[25] S. O'Connor, Z. D. Crawford, O. H. Ramachandran, J. Luginsland, and B. Shanker, "Time

integrator agnostic charge conserving finite element PIC," *Physics of Plasmas*, vol. 28, p. 092 111, 9 Sep. 2021.

[26] Z. D. Crawford, S. O'Connor, J. Luginsland, and B. Shanker, "Rubrics for charge conserving current mapping in finite element electromagnetic particle in cell methods," *IEEE Transactions on Plasma Science*, vol. 49, pp. 3719–3732, 11 Nov. 2021.

[27] Y. Saad and M. H. Schultzn, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, pp. 856–869, 3 1986.

[28] D. Appelö, L. Zhang, T. Hagstrom, and F. Li, "An energy-based discontinuous Galerkin method with tame CFL numbers for the wave equation," 2021. DOI: 10.48550/ARXIV.2110.07099. [Online]. Available: https://arxiv.org/abs/2110.07099.

[29] O. Beznosov and D. Appelö, "Hermite-discontinuous Galerkin overset grid methods for the scalar wave equation," *Communications on Applied Mathematics and Computation*, vol. 3, pp. 391–418, 3 2021.

[30] F. Zheng, Z. Chen, and J. Zhang, "A finite-difference time-domain method without the Courant stability conditions," *IEEE Microwave and Guided Wave Letters*, vol. 9, pp. 497–523, 11 1999.

[31] ——, "Toward the development of a three-dimensional unconditionally stable finite-difference time-domain method," *IEEE Transactions on Microwave Theory and Techniques*, vol. 48, pp. 1550–1558, 9 2000.

[32] J. Lee and B. Fornberg, "Some unconditionally stable time stepping methods for the 3D Maxwell's equations," *Journal of Computational and Applied Mathematics*, vol. 166, pp. 497–523, 2 2004.

[33] M. F. Causley, A. J. Christlieb, Y. Güçlü, and E. Wolf, "Method of lines transpose: A fast implicit wave propagator," 2013. DOI: 10.48550/ARXIV.1306.6902. [Online]. Available: https://arxiv.org/abs/1306.6902.

[34] M. Causley, A. Christlieb, and E. Wolf, "Method of lines transpose: An efficient unconditionally stable solver for wave propagation," *Journal of Scientific Computing*, vol. 70, pp. 896–921, 2 2017.

[35] M. Maüsek and P. Gibbon, "Mesh-free magnetoinductive plasma model," *IEEE Transactions on Plasma Science*, vol. 38, pp. 2377–2382, 9 2010.

[36] L. Siddi, G. Lapenta, and P. Gibbon, "Mesh-free Hamiltonian implementation of two dimensional Darwin model," *Physics of Plasmas*, vol. 24, pp. 1–11, 8 2017.

[37] Y. Cheng, A. J. Christlieb, W. Guo, and B. Ong, "An asymptotic preserving Maxwell solver resulting in the Darwin limit of electrodynamics," *Journal of Scientific Computing*, vol. 71, no. 3, pp. 959–993, 2017.

[38] M. F. Causley and A. J. Christlieb, "Higher order A-stable schemes for the wave equation using a successive convolution approach," *SIAM Journal on Numerical Analysis*, vol. 52, no. 1, pp. 220–235, 2014.

[39] A. J. Christlieb, P. T. Guthrey, W. A. Sands, and M. Thavappiragasm, "Parallel algorithms for successive convolution," *Journal of Scientific Computing*, vol. 86, pp. 1–44, 1 2021.

[40] M. Thavappiragasm, A. Christlieb, J. Luginsland, and P. Guthrey, "A fast local embedded boundary method suitable for high power electromagnetic sources," *AIP Advances*, vol. 10, p. 115 318, 11 2020.

[41] E. Wolf, M. Causley, A. Christlieb, and M. Bettencourt, "A particle-in-cell method for the simulation of plasmas based on an unconditionally stable field solver," *Journal of Computational Physics*, vol. 326, pp. 342–372, 2016.

[42] E. Wolf, "A particle-in-cell method for the simulation of plasmas based on an unconditionally stable wave equation solver," Ph.D. dissertation, Michigan State University, 2015.

[43] M. Causley, H. Cho, and A. Christlieb, "Method of lines transpose: Energy gradient flows using direct operator inversion for phase-field models," *SIAM Journal on Scientific Computing*, vol. 39, no. 5, B968–B992, 2017.

[44] A. Christlieb, W. Guo, Y. Jiang, and H. Yang, "Kernel based high order explicit unconditionally-stable scheme for nonlinear degenerate advection-diffusion equations," *Journal of Scientific Computing*, vol. 82:52, pp. 1–29, 3 2020.

[45] A. Christlieb, W. Sands, and H. Yang, "A kernel-based explicit unconditionally stable scheme for Hamilton-Jacobi equations on nonuniform meshes," *Journal of Computational Physics*, vol. 415, pp. 1–25, 2020, Art. No. 109543.

[46] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 3202–3216, 12 2014.

[47] M. Tao, "Explicit symplectic approximation of nonseparable Hamiltonians: Algorithm and long time performance," *Phys. Rev. E*, vol. 94, p. 043 303, 4 Oct. 2016. DOI: 10.1103/PhysRevE.94.043303.

[48] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, 1987.

[49] M. C. A. Kropinski and B. D. Quaife, "Fast integral equation methods for Rothe's method applied to the isotropic heat equation," *Computers and Mathematics with Applications*, vol. 61, pp. 2436–2446, 9 2011.

[50] M. Causley, A. Christlieb, B. Ong, and L. Van Groningen, "Method of lines transpose: An implicit solution to the wave equation," *Mathematics of Computation*, vol. 83, no. 290, pp. 2763–2786, 2014.

[51] G.-S. Jiang and C.-W. Shu, "Efficient implementation of weighted ENO schemes," *Journal of Computational Physics*, vol. 126, no. 1, pp. 202–228, 1996.

[52] A. Christlieb, W. Guo, and Y. Jiang, "A WENO-based method of lines transpose approach for vlasov simulations," *Journal of Computational Physics*, vol. 327, pp. 337–367, 2016.

[53] H. Kreiss, N. Petersson, and J. Yström, "Difference approximations of the Neumann problem for the second order wave equation," *SIAM Journal of Numerical Analysis*, vol. 42, pp. 1292–1323, 3 2004.

[54] M. Thavappiragasam, "A-stable implicit rapid scheme and software solution for electromagnetic wave propagation," Ph.D. dissertation, Michigan State University, 2019.

[55] M. F. Causley, H. Cho, A. J. Christlieb, and D. C. Seal, "Method of lines transpose: High order L-stable $O(N)$ schemes for parabolic equations using successive convolution," *SIAM Journal on Numerical Analysis*, vol. 54, no. 3, pp. 1635–1652, 2016.

[56] A. Christlieb, W. Guo, and Y. Jiang, "A kernel-based high order "explicit" unconditionally stable scheme for time dependent Hamilton–Jacobi equations," *Journal of Computational Physics*, vol. 379, pp. 214–236, 2019.

[57] S. Gottlieb, C.-W. Shu, and E. Tadmor, "Strong stability-preserving high-order time discretization methods," *SIAM review*, vol. 43, no. 1, pp. 89–112, 2001.

[58] A. Salazar, M. Raydan, and A. Campo, "Theoretical analysis of the exponential transversal method of lines for the diffusion equation," *Numerical Methods for Partial Differential Equations*, vol. 16, no. 1, pp. 30–41, 2000.

[59] M. Schemann and F. A. Bornemann, "An adaptive Rothe method for the wave equation," *Computing and Visualization in Science*, vol. 1, no. 3, pp. 137–144, 1998.

[60] G. Biros, L. Ying, and D. Zorin, *An embedded boundary integral solver for the unsteady incompressible Navier-Stokes equations (preprint)*, 2002.

[61] ——, "A fast solver for the Stokes equations with distributed forces in complex geometries," *Journal of Computational Physics*, vol. 193, pp. 317–348, 1 2004.

[62] S.-H. Chiu, M. N. J. Moore, and B. Quaife, "Viscous transport in eroding porous media," *Journal of Fluid Mechanics*, vol. 893, A3, 2020. DOI: 10.1017/jfm.2020.228.

[63] B. D. Quaife and M. N. J. Moore, "A boundary-integral framework to simulate viscous erosion of a porous medium," *Journal of Computational Physics*, vol. 375, pp. 1–21, 2018.

[64] H. Wang, T. Lei, J. Li, J. Huang, and Z. Yao, "A parallel fast multipole accelerated integral equation scheme for 3D Stokes equations," *International journal for numerical methods in engineering*, vol. 70, pp. 812–839, 7 2007.

[65] L. Ying, G. Biros, and D. Zorin, "A high-order 3D boundary integral equation solver for elliptic PDEs in smooth domains," *Journal of Computational Physics*, vol. 219, pp. 247–275, 1 2006.

[66] O. P. Bruno and M. Lyon, "High-order unconditionally stable FC-AD solvers for general smooth domains I. Basic elements," *Journal of Computational Physics*, vol. 229, pp. 2009–2033, 6 2010.

[67] ——, "High-order unconditionally stable FC-AD solvers for general smooth domains II. Elliptic, parabolic and hyperbolic PDEs. Theoretical considerations," *Journal of Computational Physics*, vol. 229, pp. 3358–3381, 9 2010.

[68] J. Douglas Jr., "On the numerical integration of $\partial_{xx}U + \partial_{yy}U = \partial_t U$ by implicit methods," *Journal of the Society for Industrial and Applied Mathematics*, no. 3, pp. 42–65, 1955.

[69] ——, "Alternating direction methods for three space variables," *Numerische Mathematik*, no. 3, pp. 41–63, 1 1962.

[70] D. W. Peaceman and H. H. Rachford Jr., "The numerical solution of parabolic and elliptic differential equations," *Journal of the Society for Industrial and Applied Mathematics*, no. 3, pp. 28–41, 1 1955.

[71] N. Albin and O. P. Bruno, "A spectral FC solver for the compressible Navier-Stokes equations in general domains I: Explicit time-stepping," *Journal of Computational Physics*, vol. 230, pp. 6248–6270, 16 2011.

[72] T. G. Anderson, O. P. Bruno, and M. Lyon, "High-order, dispersionless "fast-hybrid" wave equation solver part I: $O(1)$ sampling cost via incident-field windowing and recentering," *SIAM Journal on Scientific Computing*, vol. 42, pp. 1348–1379, 2 2020.

[73] C.-W. Shu, "High order weighted essentially nonoscillatory schemes for convection dominated problems," *SIAM review*, vol. 51, no. 1, pp. 82–126, 2009.

[74] R. D. Hornung and J. A. Keasler, "The RAJA portability layer: Overview and status," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, United States, Tech.

Rep., Sep. 2014. DOI: 10.2172/1169830.

[75] P. Grete, F. W. Glines, and B. W. O'Shea, "K-Athena: A performance portable structured grid finite volume magnetohydrodynamics code," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, pp. 85–97, 1 2020.

[76] C. J. White, J. M. Stone, and C. F. Gammie, "An extension of the Athena++ code framework for GRMHD based on advanced Riemann solvers and staggered-mesh constrained transport," *The Astrophysical Journal Supplement*, vol. 225, 2 2016.

[77] P. Mardahl and J. Verboncoeur, "Charge conservation in electromagnetic PIC codes; spectral comparison of Boris/DADI and Langdon-Marder methods," *Computer Physics Communications*, vol. 106, pp. 219–229, 1997.

[78] H. Qin, S. Zhang, J. Xiao, J. Liu, Y. Sun, and W. Tang, "Why is Boris algorithm so good?" *Physics of Plasmas*, vol. 20, 8 Aug. 2013. DOI: 10.1063/1.4818428.

[79] L. Brieda. "Particle push in magnetic field (boris method)." (2011), [Online]. Available: https://www.particleincell.com/2011/vxb-rotation/ (visited on 04/15/2022).

[80] P. Pihajoki, "Explicit methods in extended phase space for inseparable Hamiltonian problems," *Celestial Mechanics and Dynamical Astronomy*, vol. 121, pp. 211–231, 2015. DOI: 10.1007/s10569-014-9597-9.

[81] W. H. Bennett, "Magnetically self-focussing streams," *Phys. Rev.*, vol. 45, pp. 890–897, 12 1934. DOI: 10.1103/PhysRev.45.890.

[82] J. A. Bittencourt, *Fundamental of Plasma Physics*, Third. Springer-Verlag, 2010.

[83] J. Barnes and P. Hut, "A hierarchical $O(N \log N)$ force-calculation algorithm," *Nature*, vol. 324, pp. 446–449, 1986.

[84] L. Wang, R. Krasny, and S. Tlupova, "A kernel-independent treecode based on barycentric Lagrange interpolation," *Communications in Computational Physics*, vol. 28, no. 4, pp. 1415–1436, 2020. DOI: https://doi.org/10.4208/cicp.OA-2019-0177. [Online]. Available: http://global-sci.org/intro/article_detail/cicp/18106.html.

[85] N. Vaughn, L. Wilson, and R. Krasny, "A GPU-accelerated barycentric Lagrange treecode," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 701–710. DOI: 10.1109/IPDPSW50202.2020.00125.

[86] L. Wilson, N. Vaughn, and R. Krasny, "A GPU-accelerated fast multipole method based on barycentric Lagrange interpolation and dual tree traversal," *Computer Physics Communications*, vol. 265, 2021. DOI: https://doi.org/10.1016/j.cpc.2021.108017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465521001296.