Technische Universität München
Fakultät für Informatik, Lehrstuhl für Wissenschaftliches Rechnen

High Order Adaptive Semi-Lagrangian/Volume-Integral Methods for Parallel Advection-Diffusion Simulations

ARASH BAKHTIARI

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des Akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Rüdiger Westermann Prüfer
der Dissertation:	1. Prof. Dr. Hans-Joachim Bungartz
	2. Prof. Dr. Miriam Mehl (Universität Stuttgart)
	3. Prof. George Biros, Ph.D. (The University of Texas at Austin, USA)

Die Dissertation wurde am 11.05.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 22.06.2017 angenommen.

Arash Bakhtiari: *High Order Adaptive Semi-Lagrangian/Volume-Integral Methods for Parallel Advection-Diffusion Simulations*

Dedicated to J. I.

ABSTRACT

In this dissertation we introduce a parallel, arbitrary-order accurate and unconditionally stable numerical method for solving the scalar *advection-diffusion* equation and the *incompressible Navier–Stokes* equations using dynamic Adaptive Mesh Refinement (AMR). Our work has several unique characteristics: (1) we combine the method of characteristics with the volume integral method by using explicit-implicit time integration methods which result in unconditionally stable schemes; (2) we support arbitrary-order accurate piecewise Chebyshev octree-based spatial discretization; (3) our advection-diffusion solver supports unsteady velocity fields; (4) we support shared- and distributed-memory architectures and vectorization for modern CPUs; and (5) we allow different parallel octrees for each quantity of interest and introduce a robust and novel partitioning scheme for hierarchical domain decompositions, which defines an upper-bound for the communication cost in distributed-memory Lagrangian schemes. In particular, in our approach, to alleviate the stability constraints proposed by the CFL condition, we treat the advective term of the PDE of interest with a second-order accurate, explicit, but unconditionally stable semi-Lagrangian scheme, which transforms the PDE of interest into a constant-coefficient elliptic problem. Then we solve the elliptic problem with a volume integral formulation at each time-step. The volume integrals can be computed with arbitrary precision in space with provable a priori error estimate with optimal complexity by deploying a Kernel Independent Fast Multipole Method (KIFMM). We study the convergence, single-node performance, strong scaling and weak scaling of our scheme for several challenging flows that cannot be resolved efficiently without using high-order accurate discretizations. We observe that using high order discretization results in significantly fewer unknowns and therefore faster time-to-solution. For example, we consider problems for which switching from low-order approximation to high-order approximation for a computation with a fixed target accuracy results in $15\times$ fewer unknowns and $40\times$ speedup. By using dynamic AMR, we further reduce the number of unknowns for a computation with a fixed target accuracy and thus gain orders of magnitude speedup compared to uniform discretization. Moreover, our novel distributed-memory octree partitioning scheme reduces the communication cost up to a factor of twenty. For our largest run, we solve a problem with 1.4 billion unknowns on a tree with maximum depth equal to 10 and using 14th-order elements on $16,384$ x86 cores on the "STAMPEDE" system at the Texas Advanced Computing Center (TACC). This is an effective resolution of nearly 100 billion unknowns with a uniform mesh. By using the technologies developed in this disserta-

tion both in terms of numerics and High Performance Computing (HPC), we are able to simulate challenging and compute-intensive realistic scenarios such as transport phenomena in porous medium with highly complex pore structure.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publication:

[1] A. Bakhtiari, D. Malhotra, A. Raoofy, M. Mehl, H.-J. Bungartz, and G. Biros, "A parallel arbitrary-order accurate amr algorithm for the scalar advection-diffusion equation," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, ser. SC '16, Salt Lake City, Utah: IEEE Press, 2016, 44:1–44:12, ISBN: 978-1-4673-8815-3. [Online]. Available: `http://dl.acm.org/citation.cfm?id=3014904.3014963`.

ACKNOWLEDGMENTS

First, and foremost, I would like to express my deepest gratitude to Prof. Hans-Joachim Bungartz for giving me the great possibility to work on this project. I have been extremely fortunate to have worked and researched in his research group. I am also extremely thankful to Prof. George Biros for the immense amount of help and advice that he has provided throughout my work on this thesis. His advice and expertise were indispensable throughout my research. I also greatly appreciate the support and advice I have received during my studies from Prof. Miriam Mehl.

I also would like to thank my colleagues at the chair of Scientific Computing at the Technical University of Munich (TUM), Institute for Advanced Study (IAS) and Parallel Algorithms for Data Analysis and Simulation Group (PADAS) at the university of Texas at Austin for creating a friendly and collaborate environment for research. In particular, I would like to express my gratitude to Dhairya Malhotra who generously shared his time with me through frequent discussions. I would also like to thank Razieh Rezaei, Benjamin Uekermann and Alfredo Para for proofreading of this dissertation.

My research would not have been possible without help from the patient staff and talented scientists at the IAS. I would like to also thank the Texas Advanced Computing Center (TACC), whose machines I have used during my research.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

pvfmm Parallel Volume Fast Multipole Method . . 7, 8, 11, 27, 28, 33, 34, 37, 40, 76, 84, 97, 99, 106, 119

AMR Adaptive Mesh Refinement v, 1, 2, 5, 6, 8, 11, 40, 59–62, 66, 71, 85, 99, 105, 110, 114, 118–120

BDF Backward Differentiation Formula 121

CFD Computational Fluid Dynamics 8, 14, 61, 73

CFL Courant–Friedrichs–Lewy . . v, 2, 3, 6, 13, 14, 24, 40, 50, 54, 58, 62, 106, 110, 111, 113, 114, 116, 118, 119

CM Complete-Merge 53, 54, 56–60, 107, 111, 113, 114, 116, 118

FMM Fast Multipole Method 5, 11, 27, 28, 30–34, 36, 37, 59, 77, 106

GPU Graphical Processing Unit 27

HPC High Performance Computing . . . . . vi, 1, 7, 27, 31, 39, 119

IAS Institute for Advanced Study ix

KIFMM Kernel Independent Fast Multipole Method . v, 5, 8, 24, 25, 27, 28, 31–34, 37, 77, 79, 119, 120

L2L Local-to-Local 30–32, 37

L2T Local-to-Target 30, 37

M2L Multipole-to-Local 30–32, 37

1 INTRODUCTION

Many problems in various scientific disciplines require understanding of the physics of fluid flows, which are modeled with Partial Differential Equations (PDEs) describing physical concepts such as conservation laws for mass, momentum and energy. Two well-known PDEs in this regard are the *advection-diffusion* equation and the *incompressible Navier–Stokes* equations. The advection-diffusion equation describes the transport of a physical quantity such as energy or substance in a flowing medium, while the incompressible Navier-Stokes equations are a set of coupled nonlinear PDEs which describe the flow of viscous fluids.

Our goal in this dissertation is to develop parallel numerical algorithms with unconditional stability and high order accuracy for the advection-diffusion and the incompressible Navier–Stokes equations with scalable Adaptive Mesh Refinement (AMR). This requires efficient and careful mathematical and algorithmic design and implementation.

1.1 MOTIVATION

Solving the advection-diffusion and the incompressible Navier–Stokes problems helps to understand transport phenomena in complex fluids [56], the dynamics of viscous flows such as blood flows [49], [75], porous media flows [63], [81], and multiphysics simulations [3], [16], [24], [51]. Numerical methods for such phenomena find applications in biomedical engineering, i. e. , controlling localized drug delivery from artificial vesicles or understanding oxygen transport in microcirculation to geophysical problems such as migration of contaminants in sub-surface flow or mantle convection [18]. Due to the multiple spatio-temporal scales in complex fluids and transport phenomena, fast solvers with efficient numerical methods and scalable AMR support are essential. Solving these PDEs presents various challenges both in terms of numerics and High Performance Computing (HPC). Below, we present a few of these hurdles:

1. Even the linear advection-diffusion PDE represents a challenge for AMR algorithms. Depending on the velocity field, initial conditions and the diffusion coefficient, the solution may develop sharp gradients that can be hard to resolve on uniform grids with low-order spatial approximations. For many cases, as the dynamics occur over a wide range of spatial and temporal scales, spatial high resolution is required in localized regions of the domain to resolve spatio-temporal

features of interest. This demands an adaptive non-uniform grid for accurate approximation of the solution, i. e. to avoid unphysical numerical dissipation in highly localized regions. Moreover, these regions can dynamically change as the simulation evolves over time. Consequently we need to adapt the mesh to the new field, which requires re-balancing of the computational and memory load of the new mesh.

2. If a conditionally stable scheme is deployed, depending on the spatial approximation order, stability considerations may impose severe constraints on the time-step sizes in regions that require high spatial resolution to be accurately resolved. More precisely, assuming q_{th}-order elements and the smallest element size to be h_{min}, then the time-step size δt should be $\mathcal{O}(h_{min} q^{-2})$, where for large q this can result in an excessive number of time-steps.

3. An efficient solver requires the solution of elliptic problems but although the underlying theory is well understood, scaling elliptic solvers for high-order discretization on non-uniform grids is not trivial since one needs appropriate smoothers [89].

In the following sections we provide a brief review of our methodology and compare our methods with the alternative approaches. Moreover, we explain how our methodology addresses the above stated challenges.

1.2 THE SEMI-LAGRANGIAN/VOLUME-INTEGRAL METHOD

In this dissertation we develop a parallel and highly optimized advection-diffusion and incompressible Navier–Stokes solver. We propose an AMR scheme that uses an explicit-implicit time-stepping scheme. In our approach we treat the advective term of the PDE of interest with an explicit, but unconditionally stable semi-Lagrangian scheme to alleviate the stability constraints imposed by the Courant–Friedrichs–Lewy (CFL) condition. By applying the semi-Lagrangian method we transform the PDE of interest to a constant-coefficient elliptic problem. Then we implicitly solve the elliptic problem with a volume integral formulation. That is, the solution of the elliptic PDE is given as a convolution of the right-hand side with the fundamental solution (also called the Green's function) of the elliptic PDE. For the advection-diffusion equation, the elliptic problem is a modified Poisson equation with right-hand side, while for the Navier–Stokes equations this becomes an unsteady Stokes equation, for which the fundamental solution is known and a similar formulation can be used.

In a nutshell our approach consists of two main building blocks: a semi-Lagrangian advection solver and a volume integral elliptic solver. In the fol-

lowing sections, we justify our methodological choices by comparing them with conventional alternative approaches.

1.2.1 *The Semi-Lagrangian Method*

The semi-Lagrangian method is a well established approach in the numerical approximation of advection dominated problems [29]. The basic idea is to discretize the Lagrangian derivative of the solution in time instead of the Eulerian derivative. The method involves backward time integration of the characteristics to find the departure points of fluid particles arriving at the Eulerian grid points. Once the departure points are computed, the values of the advected fields can be obtained by interpolation. The main strength of the semi-Lagrangian method is the unconditional stability and its ability to allow large CFL numbers with no loss in accuracy [28]. Moreover, by using the semi-Lagrangian method, instead of solving the differential equations, we solve the characteristics equation which is much simpler.

Applying the semi-Lagrangian method to the advective term and treating the non-advective terms with a semi-implicit method leads to a powerful unconditionally stable scheme. As a result of the work of André Robert in [77] and [78], the semi-Lagrangian semi-implicit methods have become very popular in Numerical Weather Prediction (NWP) models. A comprehensive review of semi-Lagrangian method for structured grids and its application in atmospheric models is provided in [85]. For unstructured grids, a complete references on semi-Lagrangian methods is given in [102]. The semi-Lagrangian method can be combined with any spatial discretization approach and is also easy to use in problems with nonuniform grids as we discuss in Chapter 4. The method has also proven to be a popular technique in simulation in many other areas of science and engineering [64]. The semi-implicit semi-Lagrangian method algorithms can be applied to the advection-diffusion or the Navier-Stokes equations as we show in Chapter 6 and Chapter 7.

In the semi-Lagrangian method both the trajectory integration and interpolation influence accuracy. The spatial interpolation scheme is critical since it is rather expensive and it can introduce too much numerical diffusion and the development of overshoots and undershoots. Hence, the semi-Lagrangian method depends strongly on the spatial discretization. There are two parts of the semi-Lagrangian method which require off-grid interpolation: (1) interpolating velocity values in trajectory calculations (2) interpolating the advecting field values at the departure points. It is much more important to interpolate the advecting field values accurately since the errors in this step will be carried through to the rest of the computation. Higher-order interpolation is therefore used in most semi-Lagrangian methods. For the one-dimensional linear advection equation, the dispersion and diffusion errors are analyzed in [37]. For a high-order conforming finite

element discretization a convergence proof can be found in [28]. It is also shown that the overall error of semi-Lagrangian methods is not monotonic with respect to the time-step size and in particular has the form

$$\mathcal{O}(\frac{\delta x^q}{\delta t} + \delta t^k),$$

where k refers to the order of backward time integration and q is the polynomial order of the spatial discretization scheme. Similar results has been obtained earlier in [65]. The arbitrary high-order spatial interpolation scheme used in this work is described in Section 3.3.1. Our basis functions do not form a conforming basis, they are essentially discontinuous Galerkin functions. However, we do not do any flux correction other than the pointwise semi-Lagrangian upwinding.

Typically the backward trajectory computation is performed by employing second-order implicit or explicit schemes (i. e. the mid-point rule). The fourth-order Runge-Kutta method was employed in [44], [61] but their results did not show any improvement over the second-order schemes. For unsteady velocity fields, in addition to spatial interpolation, also temporal interpolation is required. We discuss our approach and implementation of the temporal interpolation in Section 4.5.

Clearly the semi-Lagrangian method is a powerful method however it has its limitations. For example simple semi-Lagrangian algorithms will be difficult to parallelize. We address the parallelization issue of the semi-Lagrangian method in Chapter 4 in detail. Mass or energy conservation are desirable properties that numerical methods that are used in physics simulations should possess. However, this is not guaranteed without further modification of the semi-Lagrangian method. Because the departure points do not necessarily coincide with the grid points, the semi-Lagrangian method requires an off-grid interpolation of the advective field values at the departure points, which can result in numerical dissipation and increased cost per time step compared with competing schemes. In Chapter 3 we introduce our arbitrary high-order spatial discretization to address the numerical dissipation issue. The interpolation cost and the single-node optimization techniques used in this thesis are discussed in Chapter 4.

1.2.2 *The Volume Integral Method*

In the second component of our methodology, which requires solution of elliptic problems, we focus on the volume integral formulation of the PDEs instead of the common PDE-based approaches such as finite difference, finite element or the spectral methods. That is, we obtain the solution of the elliptic PDE by using the volume integral formulation or in other words the convolution of the right-hand side of the PDE with the corresponding fundamental solution. For example solution of a linear constant-coefficient

PDE $\mathcal{L}(u)(x) = f(x)$, where $\mathcal{L}$ is the differential operator, is given as a direct volume integral

$$u(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y}, \quad \forall \mathbf{x} \in \Omega.$$

Here, G is the fundamental solution of the operator $\mathcal{L}$. In this section we provide a brief discussion of advantages and disadvantages of the volume integral formulation and the PDE-based approaches. The discussion in [54] also gives a comprehensive insight into the topic.

Achieving high order accuracy in the volume integral formulation is straightforward because it only depends on the accuracy of the approximation of the right-hand side. In contrast, for PDE-based approaches the high derivatives of the solution determines the accuracy of the scheme. Compared to PDE-based approaches, where computing the derivatives results in a loss in precision, in the integral form approach the accuracy does not tend to degrade as we compute the derivatives. The reason for this is the fact that in integral form, to compute the gradient of $u(\mathbf{x})$ we can simply compute the convolution with the gradient of the corresponding kernel, which is straightforward to calculate and directly leads to the gradient of u

$$\nabla u(\mathbf{x}) = \int_\Omega \nabla G(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y}.$$

However, one major drawback of the volume integral formulation is that the discretization of the integral equations results in dense linear systems. Therefore numerically computing the integral with a brute-force algorithm has a computational complexity of $O(N^2)$, assuming we look for the solution at N different points and the function $f(\mathbf{x})$ is given at N points. Acceleration methods such as the Fast Multipole Method (FMM) can reduce the complexity of the quadrature computation to $O(N)$ by separating the near- and far-field interactions and approximating the far interactions in a hierarchical manner with arbitrary precision in space with provable a priori error estimates. Therefore in our approach fast algorithms are essential to achieve high performance and parallel scalability. However, the classical FMM is kernel dependent, in the sense that using different kernels requires significant effort to modify the implementation and integrate the new kernel into the algorithm. Hence, we use a Kernel Independent Fast Multipole Method (KIFMM) introduced in [103]. Moreover, adaptivity to compute the quadrature can be achieved by using an adaptive quadrature rule which is relatively straightforward to implement by using a hierarchical domain decomposition.

1.3 CONTRIBUTIONS

In this dissertation we propose an AMR scheme with implicit-explicit time stepping. We use an explicit second-order semi-Lagrangian scheme for the

advective term, which is unconditionally stable, and an implicit volume integral solver for the elliptic PDE. For the the advection-diffusion and the incompressible Navier–Stokes equations, the elliptic PDEs are the unsteady diffusion and the Stokes operators with right-hand side, respectively.

Our contributions in this dissertation can be summarized as follows:

- We use an octree-based scheme with discontinuous q_{th}-order Chebyshev discretization at every octree node. Elliptic problems on this discretization are solved using a volume integral equation formulation. We use the same spatial discretization for the semi-Lagrangian advection solver, thus simplifying the coupling of the two components.

- To extend the scheme to parallel octrees with dynamic load-balancing support, we dynamically partition a Morton-ordered space-filling curve. We also exploit the locality properties of the Morton IDs to determine the location of the Lagrangian particles in a dynamic adaptive hierarchical domain.

- The velocity values are represented with tree-based data structure at discrete points in time. To extend the semi-Lagrangian scheme to unsteady velocity fields, we develop efficient multiple-tree evaluation algorithm for off-grid temporal interpolation/extrapolation of tree values.

- We allow for different distributed-memory partitioned trees for the velocity and concentration. Working with the two trees can create significant imbalances that can actually exceed memory resources due to load imbalance. We propose a novel partitioning scheme that addresses this problem and defines an upper-bound for communication cost for distributed-memory semi-Lagrangian schemes with minimal increase in computational cost.

- We use an AMR scheme with dynamic load-balancing. We show that our AMR scheme results in fewer unknowns by several orders of magnitude and thus faster time-to-solution for a fixed target accuracy.

- We resolve instabilities in the semi-Lagrangian solver by remapping the points to different grids and we optimize its FLOPS performance.

- We study the convergence of our schemes for steady and unsteady velocity fields. The strong and weak scaling of our solver is discussed in various challenging scenarios, where we test our algorithm with time-steps that are orders of magnitude larger than the CFL stability limit. Our largest runs were done with 1 billion unknowns reaching 10 levels of refinement with 14th order elements. This is an effective resolution of nearly 100 billion unknowns with a uniform mesh.

Our integral equation solver and discretization are based on the open-source Parallel Volume Fast Multipole Method (PVFMM) library [11], [62]. The semi-Lagrangian advection solver on octrees, novel partitioning algorithm for multiple octrees, support of unsteady velocity field and the HPC interpolation are all new technologies introduced in this thesis. Our implementation uses Intel intrinsics for vectorization, Open Multi-Processing (OpenMP) for shared memory parallelism, and the Message Passing Interface (MPI) for internode communication.

1.4 LITERATURE OVERVIEW

There is extensive literature on numerical approximation of the advection-diffusion and the incompressible Navier–Stokes equations. But for 3D methods with unconditional stability that use high-order discretization ($q > 2$), support dynamic non-uniform grids and scale on a large number of cores, the existing work is much more limited. For low-order discretization methods, comprehensive reviews and state-of-the-art can be found in [3], [15], [24], [67]. Low-order time adaptive semi-Lagrangian solvers are discussed in [41]. However, none of the codes realize high order. Most codes target fifth-order accurate schemes at most.

A third-order scheme is discussed in [36], but it uses regular grids, is only conditionally stable (the diffusion term is treated explicitly in time) and does not support distributed memory parallelism. A 11th-order accurate code was presented in [26] with excellent scalability, but it does not support adaptive mesh refinement. Perhaps the work closest to ours is the one in [17] in which high-order discontinuous Galerkin elements are discussed. A pure advection (no-diffusion) equation was solved using a 3rd-order discretization. An advection-diffusion problem was solved using lower-order discretization. A particle method for scalar advection-diffusion was described in [53]. However, it only supports regular Cartesian grids and no mesh refinement. Regarding the theoretical work on semi-Lagrangian methods, the time-step we use is described in [102]. Discontinuous Galerkin schemes are discussed in [76], and monotonicity preserving schemes are presented in [99]. Theoretical analysis for conforming finite elements was introduced in [28].

Using the semi-Lagrangian method for the advective term of the Navier–Stokes equations was first introduced in [6], [47], [71]. In [1], [44], the combination of the characteristics method with finite element method with a first-order time integration is studied. In [12] the method of the characteristics is combined with the integral equation formulation where the structured-grid finite elements are used. To solve the Dirichlet problem for the unsteady Stokes operator, they have used a double-layer boundary integral formulation. A 2D volume potential approximation with stream function formulation is studied in [39]. A velocity-vorticity formulation, where a La-

grangian approach is combined with an explicit diffusion step is discussed in [72]. In their work the volume integrals are only used to compute the vorticity.

1.5 OUTLINE OF THE DISSERTATION

This dissertation is organized into two major parts: (1) *Methodology and Algorithms*, where we discuss the mathematical foundation and numerical and algorithmic designs of the main building blocks of our methodology and (2) *Applications*, where we apply our methodology to three eminent PDEs in Computational Fluid Dynamics (CFD), namely, the diffusion equation, the advection-diffusion equation and the incompressible Navier–Stokes equations.

The content of this dissertation is structured as following:

- Chapter 2 gives an overview of the mathematical foundation of our methodology. In particular, we discuss the main idea of the semi-Lagrangian method. The stability and error analysis of the scheme is covered in this chapter. We also give a short overview of the potential theory and volume integral formulation of constant-coefficient elliptic PDEs such as the (modified) Laplace and the steady and unsteady Stokes operators.

- In Chapter 3 we discuss the PVFMM library, which is used in this work to compute the volume integrals. In this Chapter the machinery of the PVFMM library, in particular, the piecewise Chebyshev octree-based spatial discretization, KIFMM and the volume fast multipole method are discussed.

- Chapter 4 is devoted to the algorithmic details of the semi-Lagrangian method with a piecewise Chebyshev octree data structure. In this chapter we discuss shared- and distributed-memory algorithmic details of the semi-Lagrangian method. Our novel partitioning scheme, temporal interpolation/extrapolation for unsteady velocity fields and the AMR approach are all discussed in this chapter.

- In Chapter 5 we describe our volume integral formulation approach to solve the unsteady diffusion problem. Our goal in this chapter is to validate our volume integral method as an accurate and fast alternative to common PDE-based approaches.

- In Chapter 6 we develop a second-order temporal discretization for the advection-diffusion equation based on the Semi-Lagrangian/Volume-Integral approach. In this chapter we conduct various numerical experiments to study the convergence and also the single-node performance of our advection-diffusion solver.

- In Chapter 7 we extend the explicit-implicit Semi-Lagrangian/Volume-Integral method described in Chapter 6 to the incompressible Navier–Stokes equations. We validate our scheme by using well-know benchmark problems such as Taylor-Green vortex flow.
- In Chapter 8 we show parallel performance of our solver, including isogranular (or weak) scalability along with the strong scalability for various challenging flows. In this chapter we also compare scalability results of our novel partitioning scheme with alternative approaches.
- Chapter 9 concludes this dissertation with possible directions for future work.

Part I

METHODOLOGY AND ALGORITHMS

This part is devoted to giving an overview of mathematical foundation and algorithmic design decisions of the methodology used in this work.

Chapter 2 covers the mathematical aspects of the two main building blocks of our methodology, the semi-Lagrangian method and the volume integral method for elliptic PDEs.

In Chapter 3 we disucss the algorithmc details of the PVFMM library, which solves constant coefficient elliptic PDEs by computing the volume integrals with optimal complexity using accelerated methods such as FMM.

In Chapter 4 we propose parallel semi-Lagrangian algorithms for solving the scalar advection problem using dynamic Adaptive Mesh Refinement (AMR) with piecewise Chebyshev octree-based spatial discretization.

2

METHODOLOGY

This chapter gives a brief overview of the mathematical foundation of the methods used in this thesis. Our methodology consists of two main building blocks: the *semi-Lagrangian method* and the *volume integral method* for solving elliptic PDEs.

In a nutshell, our approach is as follows: First, by applying the semi-Lagrangian scheme to the convective term of the PDE of interest, e. g. the *Advection-Diffusion* or the *Navier-Stokes* equation, we transform the equations into an elliptic problem. Then we implicitly solve the elliptic problem with a volume integral formulation. The solution of the volume integral formulation is computed as a convolution of the right-hand side with the fundmental solution of the elliptic problem.

In this chapter we begin with a brief introduction to the Eulerian and Lagrangian schemes as alternative approaches to the semi-Lagrangian method. To justify our choice of the semi-Lagrangian method we first highlight the advantages and disadvantages of each scheme. Then to demonstrate the basic idea of the semi-Lagrangian method, we apply the method to the pure advection equation. We discuss convergence and stability properties of the semi-Lagrangian method. In particular, the main strengths of the scheme, namely the unconditional stability and its ability to allow large CFL numbers with no loss in accuracy are explained.

Our next step is to introduce the mathematical foundations of the elliptic problems and their volume integral formulation. We give a brief introduction to the *potential theory* and its applications in boundary value problems. As a reference for the next chapters we also give a summary of PDEs used in this thesis such as the (modified) Laplace and the steady and unstready Stokes operators. We discuss the volume integral formulations of the PDEs and derive the corresponding convolution kernels.

In this chapter we focus on the mathematical foundation of our methodology and postpone the numerical methods and the algorithmic details that we employ to solve the volume integrals and the semi-Lagrangian method to Chapter 3 and Chapter 4, respectively.

2.1 THE SEMI-LAGRANGIAN METHOD

There are two common ways to characterize a flow mathematically: *Eulerian* and *Lagrangian*. The *Eulerian* specification of a fluid flow describes the fluid properities from the perspective of an observer located in a fixed point in space as time passes. For example the flow velocity is represented by a

function $\boldsymbol{v}(\boldsymbol{x}, t)$ of position $\boldsymbol{x}$ in a fixed coordinate system and time t. On the other hand, in the *Lagrangian* specification of a fluid flow, the evolution of the flow will be observed from the perspective of a traveling individual parcel of fluid.

The Lagrangian and Eulerian specification of flow are related by the *material derivative*. Suppose we are given a flow field $\boldsymbol{v}$ and a generic field $\mathbf{F}(\boldsymbol{x}, t)$ in the Eulerian specification. $\mathbf{F}$ can be a scalar or a vector. The total rate of change of $\mathbf{F}$, which is also often called the Lagrangian rate of change, is given by

$$\frac{D\mathbf{F}}{Dt} = \frac{\partial \mathbf{F}}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} \mathbf{F}, \tag{1}$$

where $\boldsymbol{\nabla}$ denotes the gradient with respect to $\boldsymbol{x}$. The first term of the right-hand side is the local rate of change of $\mathbf{F}$ and the second term describes the rate of change due to convection. In a moving coordinate system the total derivatives become a partial time derivative and the convective term disappears due to change of the coordinates.

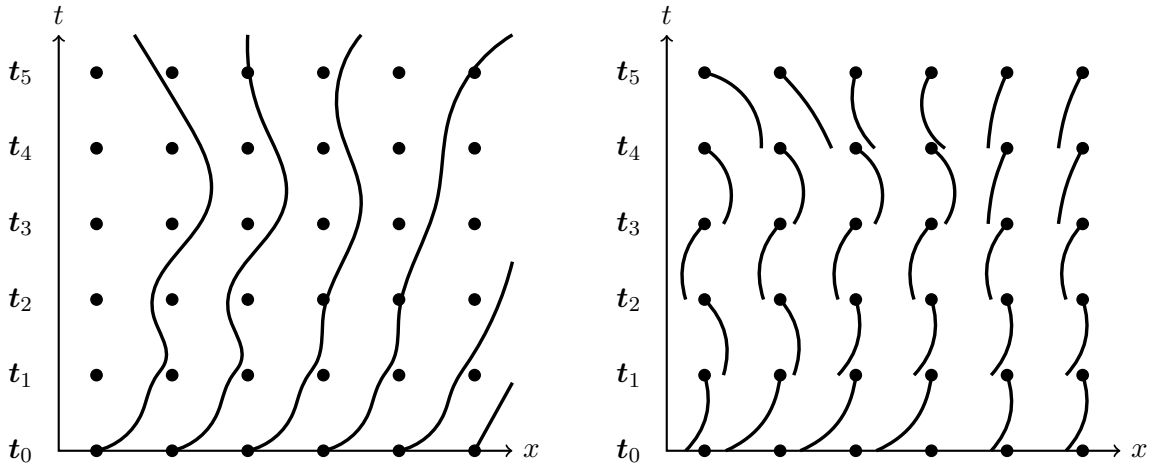
In general, due to numerical stability considerations, the CFL number for explicit methods with Eulerian schemes needs to be bounded by a constant:

$$\mathrm{CFL} = |\frac{\boldsymbol{v}\delta t}{\delta x}| < 1. \tag{2}$$

For high spatial resolution (very small mesh size δx), this severely restricts the time-step size δt for a given advecting wind. The severity of the constraint will depend upon the particular numerical scheme. However, in the Lagrangian schemes, because the advective time limit disappears, the maximum time-step size is controlled entirely by non-advective processes like viscosity and wave propagation. Therefore the Lagrangian schemes will be both stable and accurate for far larger CFL numbers than usual Eulerian limit. In the Lagrangian schemes one follows the evolution of the same set of particles for the entire time horizon. Consequently, in practical applications, the Lagrangian scheme results in nonuniform distribution of particles at later times. This leads to an inaccurate approximation of the computing quantity in domain areas where the particles are clustered. As a result, important spatial features of flow will not be well captured in the Lagrangian scheme.

Due to the problem of the chaotic distribution of particles in the Lagrangian schemes, this scheme has not become very popular in CFD simulations. To avoid the problem of clustered particles in the Lagrangian schemes, Wiin-Nielsen introduced the semi-Lagrangian method in the sixties [101]. The semi-Lagrangian method belongs to the general class of upwinding methods which exploits the advantages of both the Lagrangian and the Eulerian schemes by reseting the Lagrangian particles to the Eulerian grid

points at each time-step. By so doing, there is no mesh deformation as in the Lagrangian methods while the scheme is still able to use similarly large time-steps with no loss in accuracy. A schematic comparison of the Lagrangian and the semi-Lagrangian methods is depicted in Figure 1. As it is shown, in the semi-Lagrangian method, at each time-step, a new set of particles will be traced back to the initial *departure points*, while in the Lagrangian scheme the particles will be tracked for the entire time horizon.



(a) *Lagrangian Scheme* **(b)** *Semi-Lagrangian Scheme*

Figure 1: *A schematic comparison of the Lagrangian and the Semi-Lagrangian methods: (a) In the Lagrangian schemes one set of partciles will be initialized and tracked for the entire time horizon. (b) In the semi-Lagrangian schemes, at each time-step, a new set of particles will be traced back to their departure points.*

2.1.1 *One Dimensional Scalar Advection Equation*

To demonstrate the basic idea of the semi-Lagrangian method we consider the one dimensional scalar advection equation

$$\frac{Dc(x,t)}{Dt} = \frac{\partial c(x,t)}{\partial t} + \frac{dx}{dt}\frac{\partial c(x,t)}{\partial x} = 0, \tag{3}$$

where

$$\frac{dx}{dt} = \boldsymbol{v}\,(x,t)\,,$$

and $\boldsymbol{v}\,(x,t)$ is a given function. Equation (3) states that c is constant along a trajectory (characteristics) in space-time domain. In other words, the initial solution of c provides the exact solution for all time if the spatial arguments are replaced with the *characteristics coordinates* at the corresponding time. This provides the main idea of the semi-Lagrangian method: We imagine a particle that at time t_{k+1} resides at the Eulerian grid point x_E. We refer to

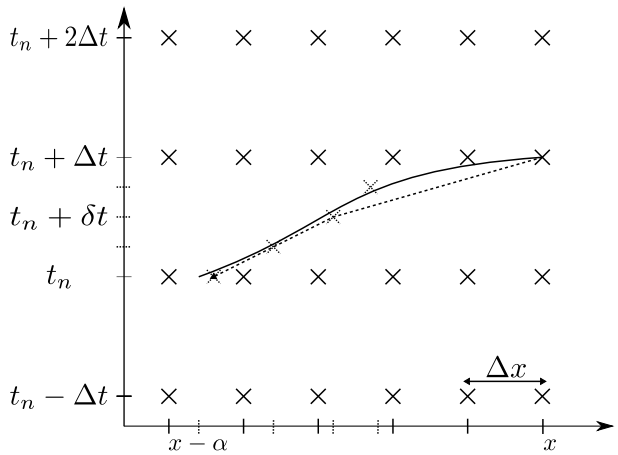
the position of the same particle at time t_k as its *departure point*, which in our notation is denoted by X_k. If the advection is the only process occurring, then the advected field at time t_{k+1} can be obtained from the the advected field values at time t_k at the departure points

$$c(x_E, t_{k+1}) = c(X_k, t_k). \tag{4}$$

The locations of the departure points can be obtained by solving the characteristics backward in time by using the given velocity field. The departure points normally do not coincide with the Eulerian grid points therefore obtaining the values of the advected field at the departure points requires spatial interpolation.

In a nutshell, the semi-Lagrangian scheme is made up of two main components: (1) backward in time integration of the characteristic equation to find the departure points of the particles arriving at the Eulerian grid points at time t_{k+1}, and (2) interpolation of the advected field values at time t_k at the departure points.

Figure 2: *Space-time mesh for trajectory computation for one dimensional semi-Lagrangian scheme. The solid curve is the actual trajectory. The dashed line represents the trajectory computed by using two steps explicit midpoint rule. Since the velocity values are available only at discrete points on space-time mesh, the computation of departure points requires temporal (in case of unsteady velocity fields) and spatial interpolation. Here,* Δt *represents the simulation time-step size while* δt *is the time-step size for computing the characteristics.*



Computing the trajectory requires the evaluation of velocity values at off-grid points on the space-time grid. However, in many practical applications, the velocity values are only available at discrete points of the space-time grid. Therefore depending on the integration method used to solve the characteristics, multiple temporal and spatial interpolation or extrapolation of velocity values are required. A schematic trajectory computation is given in Figure 2. In this figure we solve the characteristics backward in time by applying two steps explicit midpoint rule while using the Eulerian grid points as the initial condition. In Figure 2 the solid curve is the

actual trajectory and the dashed line is the computed trajectory. The points in space-time grid where the velocity values need to be evaluated are indicated by dashed crosses.

2.1.2 *Convergence and Stability Analysis*

Both the time integration of the characteristics and the interpolation of the advecting field values at the departure points influence the accuracy of the semi-Lagrangian method. In this section a summary of the accuracy and stability analysis of the semi-Lagrangian method for the pure advection equation is given. The work of J.S. Sawyer in [79] is the first study which showed that the semi-Lagrangian method can be used with longer time-step sizes without stability issues. J.R.Bates and A.McDonald proved the unconditional stability of the scheme for linear and quadratic interpolation on Cartesian grids in [5]. For the variable coefficient advection case with a finite element formulation, a general stability and convergence analysis is given in [28]. They have shown that the overall error of the semi-Lagrangian method is not monotonic with respect to the time-step size δt.

Theorem 2.1.1. *Assuming the error in the spatial interpolation is* $\mathcal{O}(\delta x^{p+1})$ *and the time discretization in the trajectory approximation method is* $\mathcal{O}(\delta t^{k})$*, (*$k$ *is the order of time integration method and* p *is the spatial interpolation order), then for a temporal resolution* $\delta t = T/N$ *in a time interval* $[0, T]$*, it can be shown that under some regularity assumptions the upper bound of the* L^{∞} *error grows proportionally to the number of time-steps used:*

$$\max_{n=1,N} \max_{i} |c(x_i, t^n) - c_i^n| \leqslant C[\mathcal{O}(\delta t^{k} + N\delta x^{p+1})]. \tag{5}$$

The rather strange consequence of this error bound is that increasing temporal resolution without increasing the spatial resolution might actually lead to an error increase by a factor that depends on the spatial resolution.

2.2 THE VOLUME INTEGRAL METHOD

The second component in our methodology is the *volume integral elliptic solver*. In this section we first give a brief introduction to the classification of PDEs to motivate our research into the elliptic PDEs and their volume integral formulations. We begin with the Laplace equation as the simplest non-trivial elliptic PDE. Then we derive a volume integral direct solution to the Poisson's equation by using the fundamental solution of the Laplace equation. We extend this approach to the modified Poisson's equation, which we employ in Chapter 5 and Chapter 6 to solve the advection-diffusion equation. In the next step the non-dimensional steady and unsteady Stokes operators and their fundamental solutions are given. In Chapter 7, we use

these equations to develop an explicit-implicit semi-Lagrangian method for the incompressible Navier–Stokes equations.

2.2.1 *Potential Theory*

In mathematics PDEs are classified as *elliptic*, *parabolic* and *hyperbolic*. To explain the classification of PDEs, for the sake of simplicity we only consider the following linear PDE of second order in two variables:

$$au_{xx} + 2bu_{xy} + cu_{yy} + du_x + eu_y + fu = g. \tag{6}$$

Here, the partial derivative of u with respect to dimension i is denoted by u_i. By replacing the u_x by α, u_y by β, u_{xx} by α^2, u_{yy} by β^2 and u_{xy} by $\alpha\beta$, we define the polynomial

$$P(\alpha, \beta) = a\alpha^2 + 2b\alpha\beta + c\beta^2 + d\alpha + e\beta + j. \tag{7}$$

The algebraic properties of the polynomial $P(\alpha, \beta)$ determine the mathematical nature of the solution of the Equation (6). Based on the value of the discriminat $b^2 - ac$ we classify the PDE as hyperbolic, parabolic, or elliptic. Equation (6) is considered as hyperbolic if the discriminant is positive. The wave equation $\partial^2 u/\partial^2 t = \Delta u$ is an eminent example of hyperbolic PDEs. If the discriminant is zero, the PDE is classified as parabolic. For the case that the discriminant is negative we categorize the PDE as elliptic. The diffusion equation $\partial u/\partial t = \Delta u$ and the Laplace equation $\Delta u = 0$ are well-known examples of parabolic and elliptic PDEs, respectively.

In this thesis most problems of interest are described by elliptic PDEs. Assuming $\mathcal{L}$ be a differential operator, based on the classical mathematical theories the solution to a linear constant-coefficient elliptic PDE

$$\mathcal{L}u(\mathbf{x}) = f(\mathbf{x}), \tag{8}$$

is given as a direct computation of the following volume integral over the domain Ω:

$$u(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y}. \tag{9}$$

Here, G is the Green's function, also known as the fundamental solution or the integral kernel of the operator $\mathcal{L}$ in free space. The fundamental solution is obtained by solving

$$\mathcal{L}G(\mathbf{x}, \mathbf{x_0}) = \delta(\mathbf{x} - \mathbf{x_0}), \tag{10}$$

where $\delta(\mathbf{x})$ is the Dirac delta function, $\mathbf{x_0}$ is the *source* point and $\mathbf{x}$ is the *observation* point. Based on the topology of the domain, Green's functions are in general classified into three categories: (1) The free-space Green's function

obtained by solving the Equation (10) for an infinite unbounded domain (2) The Green's function for a domain that is bounded by a solid surface and (3) the Green's function for a domain that is completely confined by solid surfaces. In the following sections we give a summary of the elliptic PDEs studied in this dissertation and their corresponding Green's functions.

2.2.2 *The Laplace Equation*

The Laplace equation

$$\Delta u(\mathbf{x}) = 0, \qquad \mathbf{x} \in \mathbb{R}^n, \tag{11}$$

and its inhomogeneous version, Poisson's equation

$$-\Delta u(\mathbf{x}) = f(\mathbf{x}), \tag{12}$$

are the simplest non trivial elliptic PDEs. In general, any other elliptic PDE can be considered as a generalization of these equations. Due to the fundamental role that the Laplace operator plays in the theory of PDEs, its solutions have a dedicated name and are called *harmonic functions*.

The Laplace equation occurs frequently in applied sciences. The diffusion process that has reached its equilibrium is an example of physics which leads to an elliptic problem. In Chapter 5 we describe our approach to solve the diffusion equation by applying the volume integral method. To develop the volume integral formulation for the Laplace equation we first require its fundamental solution. This is the function G satisfying

$$\Delta G(\mathbf{x}) = \delta(0), \quad \mathbf{x} \in \mathbb{R}^n,$$

where δ is the Dirac's delta function. Fourier analysis is one way to find the fundamental solution. However, here we use a more elementary way which exploits the symmetric nature of the Laplace equation. A comprehensive study of the Laplace operator and proofs for all theorems discussed in this and the following sections are given in [30].

Since the Laplace operator commutes with rotations, we assume its solutions should be *radial*. That is, if u on $\mathbb{R}^n$ is harmonic then $u(\mathbf{x}) = v(\|\mathbf{x}\|) = v(r)$, where $\|.\|$ is the Euclidean norm $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2}$.

Proposition. *If* $u(\mathbf{x}) = v(r)$ *where* $\mathbf{x} \in \mathbb{R}^n$ *and* $r = \|\mathbf{x}\|$ *then,*

$$\Delta u = \frac{n-1}{r} v'(r) + v''(r).$$

Proof. For $r > 0$, the first and second partial derivatives of r with respect to dimension x_i are

$$\frac{\partial}{\partial x_i} r = \frac{x_i}{r},$$

and

$$\frac{\partial^2}{\partial x_i^2} r = \frac{1}{r} - \frac{x_i^2}{r^3}.$$

Applying the chain rule and using the partial derivatives of r yields the following equations for the second derivate of the function u with respect to x_i

$$\frac{\partial}{\partial x_i} u = \nu'(r)\frac{x_i}{r},$$

$$\frac{\partial^2}{\partial x_i^2} u = \nu'(r)\left(\frac{1}{r} - \frac{x_i^2}{r^3}\right) + \nu''(r)\frac{x_i^2}{r^2},$$

where $\nu'(r) = \partial\nu/\partial r$ and $\nu''(r) = \partial^2\nu/\partial^2 r$. Then the Laplacian of u as a function of the derivatives of $\nu(r)$ can be written as:

$$\Delta u = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial^2 x_i} = \frac{n-1}{r}\nu'(r) + \nu''(r).$$

□

Corollary. *If $u(\mathbf{x}) = \nu(r)$ is a radial function on $\mathbb{R}^n$ then u satisfies the Laplace equation on $\mathbb{R}^n/\{0\}$ if and only if:*

$$\nu(r) = \begin{cases} c_1 \ln r + c_2 & \text{if } n = 2 \\ \frac{c_1}{(2-n)r^{n-2}} + c_2 & \text{if } n \geqslant 3, \end{cases}$$

where c_1 and c_2 are constants.

Proof. In order to ν be a radial solution of the Laplace equation, it should satisfies

$$\frac{n-1}{r}\nu'(r) + \nu''(r) = 0.$$

Therefore what remains to be done is to solve this equation for $\nu(r)$:

$$\begin{aligned} \nu'' &= \frac{1-n}{r}\nu' \\ \Rightarrow \frac{\nu''}{\nu'} &= \frac{1-n}{r} \\ \Rightarrow \ln \nu' &= (1-n)\ln r + C \\ \Rightarrow \nu'(r) &= \frac{C}{r^{n-1}}, \end{aligned}$$

which implies

$$\nu(r) = \begin{cases} c_1 \ln r + c_2 & \text{if } n = 2 \\ \frac{c_1}{(2-n)r^{n-2}} + c_2 & \text{if } n \geqslant 3. \end{cases}$$

□

This is a solution of Laplace's equation in $\mathbb{R}^n/\{0\}$ for any constants c_1 and c_2. Note that the solution at $r = 0$ is undefined. However, we can choose constants c_1 and c_2 such that $-\Delta u = \delta(0)$. We define the function $G(\mathbf{x})$ as follows:

$$G(\mathbf{x}) = \begin{cases} -\frac{1}{2\pi} \ln \|\mathbf{x}\| & \text{if } n = 2 \\ \frac{1}{n(n-2)\alpha(n)} \frac{1}{\|\mathbf{x}\|^{n-2}} & \text{if } n \geqslant 3, \end{cases} \tag{13}$$

where $\alpha(n)$ is the volume of the unit ball in $\mathbb{R}^n$. Notice that the function $G(\mathbf{x})$ satisfies the Laplace's equation on $\mathbb{R}^n/\{0\}$.

Theorem 2.2.1 (Fundamental Solution of the Laplace Equation)**.** *The function* $G(\mathbf{x})$ *defined in Equation* (13) *satisfies* $-\Delta G(\mathbf{x}) = \delta(0)$*. That is, for all* g

$$-\int_{\mathbb{R}^n} G(\mathbf{x}) \Delta g(\mathbf{x}) d\mathbf{x} = g(0),$$

and therefore G *is the* ***fundamental solution*** *of the Laplace equation.*

To motivate a solution of the Poisson's equation based on the fundamental solution of the Laplace Equation, we define

$$v(x) = \int_{\mathbb{R}^n} G(x-y) f(y) dy,$$

then we compute the Laplacian of $v(x)$

$$\begin{aligned} -\Delta_x v &= -\int_{\mathbb{R}^n} \Delta_x G(x-y) f(y) dy \\ &= -\int_{\mathbb{R}^n} \delta_x f(y) dy = f(x). \end{aligned}$$

This implies that $v(x)$ satisfies the Poisson's equation. Next, we represent the solution of the PDE by means of potentials.

Theorem 2.2.2 (Integral Representation Formula)**.** *Assume that* $u \in C^2(\Omega)$ *where* Ω *is a bounded domain. Let* G *be the fundamental solution of the Laplace operator in* $\mathbb{R}^n$*. Then the following representation formula for* $u(\mathbf{x})$ *is valid for every* $\mathbf{x} \in \Omega$*:*

$$u(\mathbf{x}) = \underbrace{\int_\Omega G(\mathbf{x}-\mathbf{y}) \Delta_y u(\mathbf{y}) d\mathbf{y}}_{\textit{volume(Newton) potential}} - \underbrace{\int_{\partial\Omega} G(\mathbf{x}-\sigma) \nabla_{\hat{N}(\sigma)} u(\sigma) d\sigma}_{\textit{single layer potential}} - \underbrace{\int_{\partial\Omega} \nabla_{\hat{N}(\sigma)} G(\mathbf{x}-\sigma) u(\sigma) d\sigma}_{\textit{double layer potential}}. \tag{14}$$

where $\nabla_{\hat{N}(\sigma)}$ *denotes the outward normal derivate taken with respect to the domain boundary* $\partial\Omega$*.*

Theorem 2.2.2 states that the function u can be represented in Ω as the sum of the three integrals. The three integrals in this theorem are referred to as the *volume potential*, the *single-layer potential* and the *double-layer potential*. These potentials have an important rule in representation of functions and solutions of the principal boundary value problems in potential theory.

Now, let f be a function of compact support in $\mathbb{R}^n$ which means $f \equiv 0$ outside of a ball B_R of sufficiently large radius R. Suppose the function u is the solution of the whole-space Poisson's equation. We assume that u has the property that the integrals over $\partial\Omega$ of $\nabla_{\hat{N}(\sigma)} Gu$ and $G\nabla_{\hat{N}(\sigma)} u$ tend to 0 as $R \to \infty$. Now by applying the *represenation formula* (Theorem 2.2.2) and letting $R \to \infty$, we obtain for any $\mathbf{x} \in \mathbb{R}^n$

$$u(\mathbf{x}) = \int_{\mathbb{R}^n} G(x-y)\Delta u(y)dy = \int_{\mathbb{R}^n} G(x-y)f(y)dy.$$

Theorem 2.2.3 (Solution of the Possion's Equation)**.** *Assume f has compact support and $f \in C^2(\mathbb{R}^n)$. Let*

$$u(\mathbf{x}) \equiv \int_{\mathbb{R}^n} G(x-y)f(y)dy,$$

where G is the fundamental solution of the Laplace equation. Then u is the solution of the Poisson's equation $-\Delta u = f$ and $u \in C^2(\mathbb{R}^n)$.

2.2.3 *The Modified Laplace Equation*

The modified Laplace equation and its inhomogeneous version, the modified Poisson equation arise in many areas of science and engineering. The equation of the modified Poisson's equation reads

$$\alpha u(\mathbf{x}) - \Delta u(\mathbf{x}) = f(\mathbf{x}), \tag{15}$$

where α is a constant and f is an arbitrary function of position. When α is zero, the equation transforms to the Poisson's equation. One way to solve the modified Poisson equation is using the the method of Green's function as we used for the Poisson's equation. Here, without proof, we state the fundamental solution of the modified Laplace equation

$$G(\mathbf{x}, \mathbf{y}) = \frac{e^{-\lambda r}}{4\pi r}, \tag{16}$$

where $\lambda = \sqrt{\alpha}$, $\mathbf{x}$ is the location of the evaluation point, $\mathbf{y}$ is the location of the singularity, $\mathbf{r} = \mathbf{x} - \mathbf{y}$ and $r = \|\mathbf{r}\|$. Then, similar to the Poisson's problem, the solution to the modified Poission's equation is given by the following direct volume integral:

$$u(\mathbf{x}) = \int_\Omega \frac{e^{-\lambda r}}{4\pi r} f(y)dy.$$

In Chapter 5 and Chapter 6, we use the direct volume integral solution of the Poission's and modified Poission's PDEs to solve the advection-diffusion equation.

2.2.4 *The Stationary Stokes Equation*

The equations that govern the motion of an incompressible Newtonian fluid are the continuity equation for the velocity field

$$\nabla \cdot \mathbf{u} = 0, \tag{17}$$

and Newton's second law for a small parcel of fluid, which is described by the incompressible Navier-Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \boldsymbol{\nabla} \mathbf{u} - \mu \Delta \mathbf{u} + \nabla p = b. \tag{18}$$

Here, μ is the viscosity of the fluid, p the pressure and b is a body force, which for simplicity we assume to be constant.

Fluid flows where the viscous forces dominate the advective inertial forces are called *Stokes* flow which are also known as creeping flow. The equations governing this kind of flow are called the Stokes equations which in fact are a linearization of the Navier-Stokes equations and are obtained by leaving out the time dependence and the advective terms:

$$-\mu \Delta \mathbf{u} + \nabla p = 0, \quad \text{and} \quad \nabla \cdot \mathbf{u} = 0. \tag{19}$$

In this section we focus on the mathematical properties of the Stokes equations and postpone the study of the physical properties of the Stokes and the incompressible Navier–Stokes equations to Chapter 7. The common methods to solve the linear PDEs can be applied to the Stokes equations. However, in this thesis we solve the Equation (19) by computing the direct volume integral formulation, which requires the fundamental solution of the PDE. The fundamental solution of the Stokes flow is obtained by solving the continuity equation $\nabla \cdot \mathbf{u}$ and the singularity forced Stokes equation

$$-\mu \Delta \mathbf{u} + \nabla p = \delta(\mathbf{x} - \mathbf{x}_0), \tag{20}$$

where $\mathbf{x}_0$ is an arbitrary point. The solution to Equation (20) in three dimensions is given by:

$$G(\mathbf{x}, \mathbf{y}) = \frac{1}{8\pi\mu} \left(\frac{1}{r} I + \frac{\mathbf{r} \otimes \mathbf{r}}{r^3} \right), \tag{21}$$

where I is the identity operator and $\otimes$ denotes the tensor product. This kernel is known as *Stokeslet*. The derivation of the Stokeslet and a complete discussion of this material is given in [73].

2.2.5 *The Unsteady Stokes Equation*

When the inertial convective term $\mathbf{u}\cdot\boldsymbol{\nabla}\mathbf{u}$ in the incompressible Navier–Stokes equations is small compared to other terms and thus can be neglected, the equations governing the flow are described by the *unsteady Stokes* equations, which in non-dimensional form are given by:

$$\alpha\mathbf{u} - \mu\Delta\mathbf{u} + \nabla p = 0, \quad \text{and} \quad \nabla\cdot\mathbf{u} = 0. \tag{22}$$

By solving the singularity forced unsteady Stokes equation $\alpha\mathbf{u} - \mu\Delta\mathbf{u} + \nabla p = \delta(\mathbf{x}-\mathbf{x}_0)$, we obtain the *unsteady Stokeslet*

$$G(\mathbf{x},\mathbf{y}) = \frac{1}{8\pi\mu}\left(\frac{A(R)}{r}I + \frac{B(R)}{r^3}(\mathbf{r}\otimes\mathbf{r})\right). \tag{23}$$

We define $\lambda = \sqrt{\frac{\alpha}{\mu}}$ and $R = \lambda r$. Then $A(R)$ and $B(R)$ are defined as

$$A(R) = 2e^{-R}\left(1 + \frac{1}{R} + \frac{1}{R^2}\right) - \frac{2}{R^2}, \tag{24}$$

$$B(R) = -2e^{-R}\left(1 + \frac{3}{R} + \frac{3}{R^2}\right) + \frac{6}{R^2}. \tag{25}$$

In Chapter 7 we employ the unsteady Stokeslet to solve the Navier-Stokes equations.

2.3 SUMMARY

In this chapter we began with a brief introduction to the semi-Lagrangian method, highlighting the main strength of the scheme, namely the unconditional stablity and and its ability to allow large CFL numbers compared to the Eulerian schemes. We discussed that combing the semi-Lagrangian method with semi-implicit methods is an elegant mean of designing unconditionally stable schemes that have proven to be quite valuable in practice. More precisely, in our approach we treat the advective term with the Semi-Lagrangian method and transform the advection-diffusion or the incompressible Navier-Stokes equations to an elliptic problem, which we implicitly solve with a volume integral formulation. We recalled the mathemtical foundation of the elliptic problems and the potential theory. In particular, we studied the (modified) Laplace and the steady and unsteady Stokes operators and their volume integral formulations. By using hierarchical domain decomposition and fast algorithms such as the Kernel Independent Fast Multipole Method, the volume integral formulation is an attractive alternative to common PDE-based approaches. In the next chapter, we discuss our numerical and algorithmic approach for the volume integral elliptic solver.

We explain our approach to hierarchical domain decomposition and how we achieve arbitrary-order accuracy in space. We also briefly explain the KIFMM algorithm.

3

THE VOLUME INTEGRAL ELLIPTIC SOLVER

Many problems require to compute potentials for discrete or continuous source distributions. Applications vary from Millennium simulations of galaxy formation in astrophysics [43], [60], [84], [100] to acoustic scattering [82], [95] or fluid flows [33], [59].

Computing the potential for a discrete source distribution requires summation over points. For the case of a continous source distribution, summation transforms to integration over the source density. The computational complexity of both cases is $\mathcal{O}(N^2)$. However, by using fast algorithms such as FMM the computational complexity can be reduced to $\mathcal{O}(N)$.

As we discussed in Chapter 2 our approach in this thesis involves solving constant-coefficient elliptic PDEs by computing volume potentials with continuous source distributions. To solve the elliptic PDEs we use the PVFMM library [11], [62], which is a highly optimized, high-order and adaptive volume integral elliptic solver, which exploits the FMM technology to compute the volume integrals. In this chapter we recall the main components of the PVFMM library and its underlying numerical and algroithmic designs.

The volume FMM for two dimensions was first introduced in [27]. The work in [54] extended the method to three dimensions. The numerical approach used in PVFMM is based on the method introduced in [54]. The PVFMM library includes new algorithmic as well as HPC optimizations like shared memory parallelism using OpenMP and distributed memory parallelism using MPI. Its algorithms are cache optimized and support vectorized kernel functions. In addition, accelerators like Intel Xeon Phi coprocessors and Graphical Processing Units (GPUs) are supported.

For the spatial discretization the PVFMM library uses a piecewise Chebyshev octree data structure for both source distribution and the computed potentials. We use the same spatial discretization in Chapter 4 for our Semi-Lagrangian advection solver. We explain the piecewise Chebyshev octree data structures in Section 3.3.1 in detail.

By using the KIFMM the PVFMM library is capable of computing the volume integrals for various kernels of constant coefficient elliptic PDEs like (modified)Laplace, (un)steady Stokes and Helmholtz for free-space and periodic boundary conditions. Depending on the kernel, additional boundary conditions such as Dirichlet or Neumann are also possible to implement. In [63], it is shown that by using iterative linear solvers, the PVFMM library can also be used to solve variable coefficient elliptic PDEs.

In this chapter we first provide an introduction to the particle N-Body problems and briefly describe the main ideas behind FMM. Understanding

basics of FMM is useful for understanding KIFMM, which we briefly explain in the next step. Finally we explain how the PVFMM library adapts the KIFMM to evaluate the potential due to a continuous source distribution.

The brief introduction given in this chapter provides the context for our volume integral approach. However, here we focus only on the key components of the PVFMM library related to the subject of this thesis and avoid numerical, algorithmic and performance optimization details. For a more comprehensive explanation of the PVFMM library, we refer the interested reader to the relevant publications [11], [62].

3.1 FAST MULTIPOLE METHOD

Assume we have N source and target points. We would like to compute the potential u_i due to source points y_j at each target point located at x_i. This problem is called N-*body problem* and can arise in many physical phenomena. The total potential u_i at each target point can be computed by the sum over the potential contributed by each source point

$$u_i = \sum_{j=1}^{N} G(x_i, y_j) f_j, \quad \forall i = 1, \cdots, N. \tag{26}$$

Here y_j and f_j are the coordinates and the source density of the source points. We refer to G as the kernel function which specifies the physics of the problem.

The direct computation of the sum has complexity of $\mathcal{O}(N^2)$. In order to solve large scale N-body problems, it is essential to develop more efficient algorithms. A number of algorithms have been proposed to address this issue. However, the FMM has been the most successful one. The FMM computes the *approximation* of this sum with $\mathcal{O}(N)$ complexity with a guaranteed user-specified accuracy. The FMM was first introduced by Greengard and Rokhlin [38], [40] and was developed to speed up the calculation of N-body problem for the long-ranged potentials. The FMM is considered to be one of the top ten algorithms of the 20th century [22].

In FMM, the near and far interactions are separated. That is, the summation in Equation (26) will be split into two parts

$$u_i = \sum_{y_j \in \mathcal{N}} G(x_i, y_j) f_j + \sum_{y_j \in \mathcal{F}} G(x_i, y_j) f_j. \tag{27}$$

Here, $\mathcal{N}$ and $\mathcal{F}$ denote domains areas considered as near and far, respectively. The near interactions are computed exactly by direct summation. However, the contribution from far interactions can be approximated. In order to indicate which computational domain areas are considered as near or far, the FMM algorithm exploits advantages of hierarchical domain decompositions such as quadtrees in two dimensions or octrees in three dimensions.

In Figure 3 we illustrate a quadtree data structure. In this figure the tree nodes for far and near field interactions for a particular target point x are indicated with blue and green color, respectively. The tree nodes further away from the target point (tree nodes indicated by blue color) are also called *well-separated* tree nodes. The potential due to well-separated tree nodes for a given target node are evaluated hierarchically. That is, the far interactions are broken into parts that are evaluated at different tree levels.

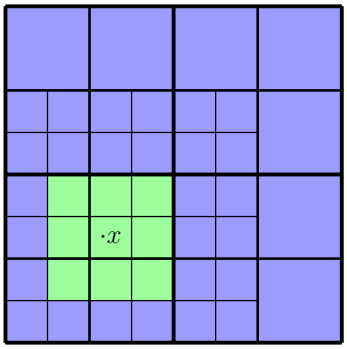
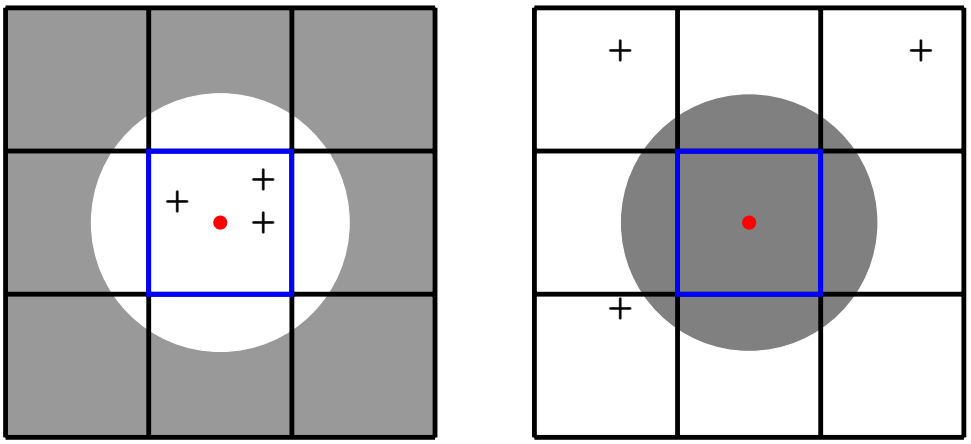


Figure 3: *Far (blue) and near (green) field interaction areas. The image is from [11]*

In the following we discuss how to approximate the potentials due to source points in one tree node and efficiently translate the approximated values among different tree levels. To efficiently represent the potentials, two types of series will be associated with each tree node:

- **Multipole Expansions** approximate the potentials due to source points within a particular tree node at target points in well-separated areas.
- **Local Expansions** approximate the potentials due to source points located in well-separated areas at the points inside a particluar tree node.



(a) *Multipole expansion: approximating the potentials due to source points within a tree node at far areas.*

(b) *Local expansion: approximating the potentials due to source points located in well-separated areas at the points inside a tree node.*

Figure 4: *Schematic illustration of multipole and local expansion.*

In Figure 4 we schematically illustrate the multipole and local expansions. In this figure the source points are indicated by black crosses. The areas where the potentials needs to be approximated are highlighted with gray

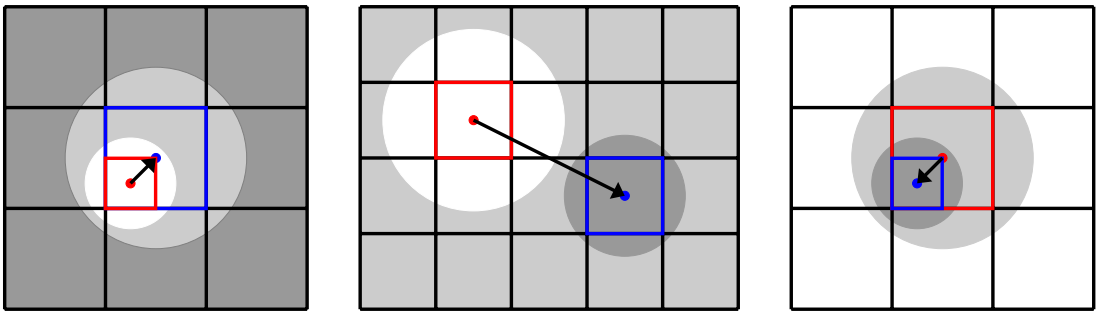
color. Figure 4a illustrates the multipole expansion, where the potential due to the source points located inside a tree node will be approximated at the target points located in well-separated areas. The local expansion is depicted in Figure 4b, where the potential due the source points located in far areas are approximated inside the tree node.

In order to develop a fast algorithm with $\mathcal{O}(N)$ complexity we require to accumulate expansions from several tree nodes and combine them so that they may be evaluated at the target points only once. More precisely, we require first a means of combining the multipole expansions from child nodes into multipole expansion of the parent node. This will be used to compute the coarse-scale multipole expansions from the fine-scales expansions. Second we need to convert several multipole expansions into a single local expansion in the target node. This is a translation between source nodes and target nodes at the same tree level. Finally we require a translation of the local expansion of the target node to the local expansion within each of the target node's children at the next tree level. Therefore the following translation operators are defined:

- **Source-to-Multiple (S2M) Translation:** computes the multipole expansion coefficients due to source points in each leaf box.
- **Multipole-to-Multiple (M2M) Translation:** a linear operator, which obtains the multipole coefficients of coarser level tree nodes from multipole coefficients of the children nodes by mapping the coefficients with respect to children and parent nodes' centers.
- **Multipole-to-Local (M2L) Translation:** a linear operator, which translates the multipole expansion of well separated nodes into local expansion of a target node.
- **Local-to-Local (L2L) Translation:** a linear operator, which obtains the contributions of far fields to the local expansion of a target node by shifting the local expansion of its parent's node to the center of the target node.
- **Local-to-Target (L2T) Translation:** computes the far field contributions of potentials for a leaf node at its target points by evaluating its local expansion.

In Figure 5, M2M, M2L and L2L translations are depicted.

The total field at a target point in a leaf node will be computed as the sum of the field due to the source points in the nodes of near fields and the contributions from sources in the far field. The later is approximated by evaluating the local expansion of the node at the target point. Hence, the main task in FMM is to hierarchically construct the local expansion of each tree node. Thus the FMM algorithm can be summarized into two steps:



(a) *M2M Translation* **(b)** *M2L Translation* **(c)** *L2L Translation*

Figure 5: *Schematic illustration of M2M, M2L and L2L translations.*

- **The Upward Pass:** In a post-order traversal of the tree, we compute the multipole expansions for all leaf nodes. Then for all the non-leaf nodes we compute the M2M translations.
- **The Downward Pass:** For all nodes we compute the M2L translations from well-separated source nodes. Then in a pre-order traversal of the tree we compute the L2L translations. In the next step we evaluate the local expansions at the target points for all leaf nodes. Finally we compute the direct interactions from leaf nodes that are not well-separated and add them to the target potentials.

The expansion and translation operators introduced in the classical FMM algorithm depend on the underlying PDE. Therefore developing an optimized FMM software for various kernels can be quite cumbersome and time consuming. In the next section we introduce the KIFMM, which addresses this issue.

3.2 KERNEL INDEPENDENT FAST MULTIPOLE METHOD

In this section we discuss the Kernel Independent Fast Multipole Method algorithm. This method has the same structure as the classical FMM but does not require implementation of expansions and translation operators for each underlying kernel. Instead, it requires only a black-box kernel evaluation. The original KIFMM was introduced in [103]. The parallel implementation and HPC optimizations of the method were developed in [57], [104]. In a nutshell, the crucial element of this approach is to use the concept of the *equivalent density* representations instead of the analytic expansions and translations. That is, to represent the potential due to sources inside a tree node, we use an equivalent density on a surface enclosing the tree node (*equivalent surface*) that has the same contribution to potentials in far field as the original sources inside the surface. To find this equivalent density, we match the potential due to original sources with the potential of the equivalent densities at another surface in the far field (*check surface*). To

compute the equivalent densities, by using the integral equation formulation, the local exterior and interior problems on the check surface will be solved. For instance, in KIFMM instead of the construction of the multipole expansion in the upward pass in classical FMM, we solve the local exterior inverse problems to compute the equivalent densities. More precisely, we define a set of source points $(r_k^{eq,\mathcal{B}}, q_k^{eq,\mathcal{B}})$ on the equivalent surface around the tree node $\mathcal{B}$. This is depicted in Figure 6a. The points are arranged as a $m \times m$ grid on each face of the surface, where m, the multipole order, specifies the accuracy of the expansion. To compute the equivalent density q_k^{eq}, we first compute the potential $u_i^{ch,\mathcal{B}}$ at points on the check surface $r^{ch,\mathcal{B}}$ due to original sources inside the equivalent surface

$$u_i^{ch,\mathcal{B}} = \sum_{y_j \in \mathcal{B}} G(r_i^{ch,\mathcal{B}}, y_j) q_j, \quad \forall i, \tag{28}$$

where y_j is position of a source inside the equivalent surface of $\mathcal{B}$. Then we solve the following linear system to obtain the equivalent density $q_k^{eq,\mathcal{B}}$

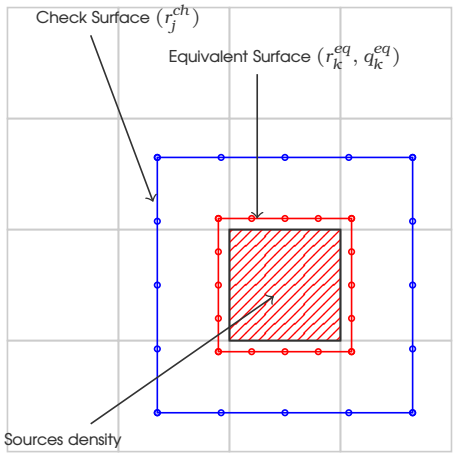
$$u_i^{ch,\mathcal{B}} = \sum_k G(r_i^{ch,\mathcal{B}}, r_k^{eq,\mathcal{B}}) q_k^{eq,\mathcal{B}}, \quad \forall i. \tag{29}$$

The potential generated by the equivalent source densities $q_k^{eq,\mathcal{B}}$ can accurately approximate the potential due to source points inside the tree node $\mathcal{B}$ at target points in well-separated nodes.

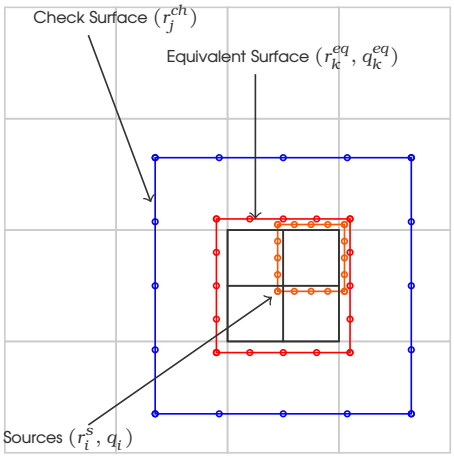
In Figure 6b, we illustrate the M2M translation in the context of KIFMM. To compute the equivalent density for a non-leaf node $\mathcal{B}$, we evaluate the potentials due to equivalent densities of its children on the check surface of non-leaf node. Then we compute the equivalent density of the non-leaf node by solving the linear system as explained before.

In Figures 6c and 6d we depict the M2L expansion and the L2L translation operator for the KIFMM approach. Here, we use a similar procedure as the multipole expansion and M2M translation to compute the equivalent densities. However, notice the relative location of the equivalent and check surfaces for local expansion and L2L translation.

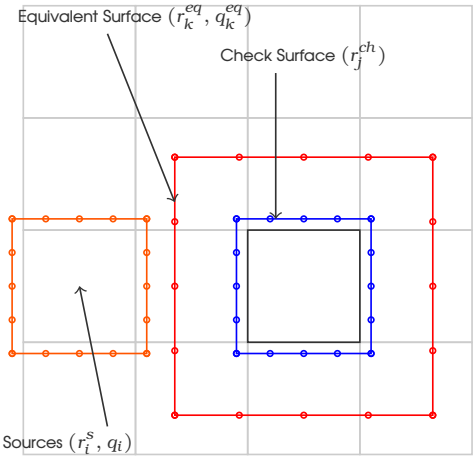
Once M2M, M2L and L2L translation operators in the context of KIFMM are available we can use the same algorithm as the classical FMM to solve the N-body problems. Since computing the equivalent densities requires only the evaluation of the kernel function G at some points on the check surfaces (instead of multipole and local expansion of the underlying kernel), this scheme has the advantage that it is relatively simple to extend to more general kernels as long as the kernel is associated with a non-oscillatory second-order elliptic PDEs such as the Laplacian, the modified Laplacian, the steady or unsteady Stokes operators.



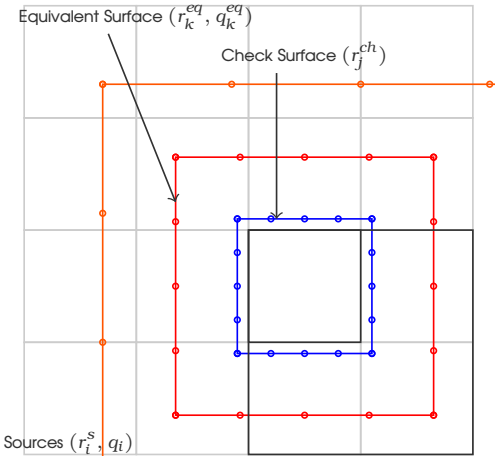
(a) *Multipole expansion*



(b) *Multipole to Multipole translation*



(c) *Multipole to Local translation*



(d) *Local to Local translation*

Figure 6: *Schematic illustration of multipole and local expansions as well as M2M, M2L and L2L translations in the context of KIFMM. The images are from [11].*

3.3 VOLUME FAST MULTIPOLE METHOD

In this section we describe the basic idea behind the volume FMM and how to adapt the KIFMM to compute the potentials with continuous source distribution where the summation in particle FMM turns to an integral. The volume FMM computes the convolution of a given density function f with a kernel function G

$$u(x) = \int_\Omega G(x,y)\, f(y)\, dy, \tag{30}$$

where G is considered to be the fundamental solution for an elliptic PDE (see Section 2.2.1). To represent the input density function f and the potential u, the PVFMM library uses octree based spatial discretization method

which we describe in Section 3.3.1. Thus the integral in Equation (30) is computed as the sum of integrals over each leaf octant $\mathcal{B}$ in octree $\mathcal{T}$:

$$u(x) = \sum_{\mathcal{B}\in\mathcal{T}} \int_{y\in\mathcal{B}} G(x,y)\, f(y) dy. \tag{31}$$

Similar to particle FMM we split each integral into near and far field interactions (well-separated from the target evaluation point x)

$$u(x) = \sum_{\mathcal{B}\in\mathcal{T}} \left(\int_{\mathcal{N}(x)} G(x,y)\, f(y) dy + \int_{\mathcal{F}(x)} G(x,y)\, f(y) dy \right). \tag{32}$$

The far interactions are approximated by using multipole and local expansions as in KIFMM and can be computed efficiently using standard Gaussian quadrature. The near interactions are evaluated exactly however due to the $1/\|x-y\|$ factor in kernel functions for the target points inside or close to the boundary of an octant this leads to singular or near-singular integrals.

In this section we first introduce the piecewise Chebyshev octree based data structure used in PVFMM library. Then we discuss the approach used in PVFMM library to efficiently compute the singular integrals in the context of this spatial disctretization. The details of the volume FMM and in particular the error and complexity analysis of the PVFMM software can be found in [11], [54].

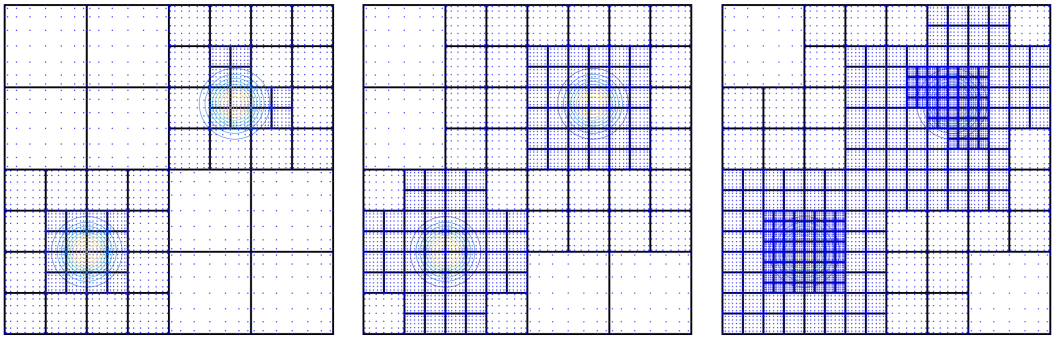
3.3.1 *Chebyshev Octree-based Spatial Discretization*

As we explained in previous section the PVFMM library discretizes the computational domain using an octree data structure. Hierarchical data structures such as octrees have a variety of important applications in areas such as adaptive finite element methods, adaptive mesh refinement methods, and many-body algorithms. Design and use of large scale distributed tree data structures that scale to thousands of processors is still a major challenge and is an area of active research. The PVFMM library uses distributed-memory parallelized *Chebyshev octree data structure*, which we explain in the following. The distributed parallelization of this domain discretization is discussed in Section 4.2.1.

PIECEWISE CHEBYSHEV OCTREE DATA STRUCTURE In our approach, the computational domain Ω is discretized by an octree $\mathcal{T}$. We use piecewise polynomial representations for discretizing the field values on this domain. More precisely, we approximate the field values by constructing Chebyshev polynomials of degree q in each leaf octant $\mathcal{B} \in \mathcal{T}$

$$\hat{u}(x,y,z) = \sum_{i+j+k\leqslant q} \alpha^{\mathcal{B}}_{ijk} T_i(x) T_j(y) T_k(z), \tag{33}$$

Figure 7: *Quadtree Chebyshev approximation of two Gaussian functions positioned at two corners of the domain. The Chebyshev grid in each octant is demonstrated via blue dots. Recall that each octant has the same grid resolution. The approximation accuracy is controlled by tree error tolerance ϵ_{tree}. Adaptive refinement of tree by increasing the spatial accuracy of the approximation is shown in multiple snapshots.*



where $T_k(x)$ is the Chebyshev polynomial of degree k in x. To construct the Chebyshev coefficients we evaluate the field function on $(q+1)^3$ Chebyshev grid points in leaf octant $\mathcal{B}$. We use these values to construct the Chebyshev coefficients $\alpha^{\mathcal{B}}_{ijk}$ $(0 \leqslant i, j, q \leqslant q)$. In Equation (33) for computational efficiecy we use only $(q+1)(q+2)(q+3)/6$ coefficients to approximate the field values. In the current implementation q is fixed for each octant, thus hp-adaptivity is not supported. To achieve spatial adaptivity we estimate the truncation error per octant by computing the absolute sum of the highest order coefficients

$$\epsilon^{\mathcal{B}} = \sum_{i+j+k=q} |\alpha^{\mathcal{B}}_{ijk}|. \tag{34}$$

The leaf nodes with truncation errors larger than a prescribed threshold ϵ_{tree} are refined recursively until the required accuracy is achieved. In case of a multidimensional field, we treat each component independently with a similar approach.

In Figure 7 we depict an adaptive construction of Chebyshev quadtrees presenting two Gaussian functions positioned at two corners of the domain. The Chebyshev grid in each octant is demonstrated via blue dots. Recall that each octant has the same grid resolution. The approximation accuracy is controlled by a predefined tree error tolerance ϵ_{tree}. As it is shown in this figure, by decreasing ϵ_{tree}, more refinement is required to resolve the field values with the desired accuracy. In the next section, we show how to compute the singular or near singular integrals discussed in Equation (32) on top of this spatial discretization.

3.3.2 *Singular Quadratures*

We would like to compute the potential due to a single leaf octant $\mathcal{B}$ where the density function in $\mathcal{B}$ is given as a Chebyshev polynomial approximation as described in the previous section. If the target point is located sufficiently far away, we can use normal Gaussian quadrature. However, for target points inside or close to the boundaries of the octant this requires solving singular or near singular integrals, which can be very expensive. For a the density function f approximated by a Chebyshev polynomials $\hat{f}(y) = \sum_{i,j,k} \alpha^{\mathcal{B}}_{i,j,k} T_{i,j,k}(y)$, the potential at a target point x due to the source density in octant $\mathcal{B}$ can be computed as follows

$$\begin{aligned} u(x) &= \int_{y\in\mathcal{B}} G(x,y)\hat{f}(y) \\ &= \int_{y\in\mathcal{B}} G(x,y) \left[\sum_{i,j,k} \alpha^{\mathcal{B}}_{i,j,k} T_{i,j,k}(y)\right] \\ &= \sum_{i,j,k} \alpha^{\mathcal{B}}_{i,j,k} \underbrace{\left[\int_{y\in\mathcal{B}} G(x,y) T_{i,j,k}(y)\right]}_{I_{i,j,k}}. \end{aligned} \tag{35}$$

Then the potential due to a leaf octant $\mathcal{B}$ at any point can be computed as a summation

$$u(x) = \sum_{i,j,k} \alpha^{\mathcal{B}}_{i,j,k} I_{i,j,k}, \tag{36}$$

where $I_{i,j,k}$ is the integral over the Chebyshev basis functions for each evaluation point x (the integral term in Equation (35)). These integrals can be precomputed using the Duffy transformation method [25] and a tensor-product Gauss quadrature rule. In this way, the S2M translations and near interactions between leaf nodes can be represented as matrix-vector products. For example, in the case of S2M translations, the potential quadrature at each point on the check surface will be precomputed. Then the check potential can be computed as

$$u_{check} = M_{s2ch} \times \alpha, \tag{37}$$

where $\alpha = [\alpha_1, \cdots, \alpha_n]^T$ are the Chebyshev coefficients representing the source density of the leaf octant. Similar to before, we compute the equivalent density by solving the linear system. A similar approach can be used for near interactions between leaf nodes.

In volume FMM in contrast to classical particle FMM, the source density is given as a polynomial approximation. Therefore all transformations in classical FMM which involve source terms need to be modified. In addition, in volume FMM the target points are Chebyshev node points in each

octant such that after computing the potentials at these targets points we can compute the final results as a polynomial approximation by using a linear transformation. Since the positions of the target points (Chebyshev nodes in each octant) are fixed, the translation operators involving the target points such as L2T can be precomputed. M2M, L2L, M2L remain the same as the particle FMM.

2:1 BALANCE CONSTRAINT In S2M translations the target octant and the source octant can be at the same level or can be at arbitrary coarser or finer level relative to each other. The quadratures in Equation (36) need to precomputed for all possible combinations of these cases. For trees with high depth precomputing the quadrature for every possible combination leads to enormous memory consumption and becomes quickly infeasible. Therefore we set a maximum allowed depth for the tree. In addition, the *2:1 balance constraint* needs to be enforced. That is, the adjacent leaf nodes can be at most one level finer or coarser relative to each other. By applying these techniques the PVFMM library is capable of limiting the memory consumption.

3.4 SUMMARY

This chapter gives a brief description of the main numerical and algorithmic ideas behind the PVFMM library, which is a FMM-based volume integral elliptic solver. We began with an introduction to the N-body problem and the FMM algorithm. We showed that the FMM can solve the N-body problems with optimal complexity by separating the near and far interactions and approximating the far interactions in a hierarchical manner. In the next step we introduced the KIFMM, which uses the concept of *equivalent density* representations instead of multipole and local expansions and therefore compared to classical FMM is much simpler to extend to more general non-oscillatory second-order elliptic kernels. Finally we described the volume FMM approach where we compute the potentials due to continuous source distributions instead of discrete particles. In this case the summation in particle FMM transforms to an integral, which we compute by using the KIFMM. We also described the piecewise Chebyshev octree spatial discretization depolyed in the PVFMM library. In the next chapter we develop our semi-Lagrangian advection solver on top of this spatial discretization.

4

THE SEMI-LAGRANGIAN ADVECTION SOLVER

In this chapter we propose parallel, scalable and spatially-adaptive semi-Lagrangian algorithms for solving the scalar advection problem on top of a Chebyshev octree data structure. More precisely, we solve the following advection equation for the *concentration* $c(\mathbf{x}, t)$:

$$\frac{\partial c(\mathbf{x}, t)}{\partial t} + \boldsymbol{v}(\mathbf{x}, t) \cdot \boldsymbol{\nabla} c(\mathbf{x}, t) = 0, \ \mathbf{x} \in \Omega, \tag{38}$$

with initial condition $c(\mathbf{x}, 0) = c_0(\mathbf{x})$ and with either free-space or periodic boundary conditions at the boundary $\partial\Omega$ of a unit cube. Here $\boldsymbol{v}$ is a given time-dependent velocity field. The spatial discretization for both the concentration and velocity fields is based on the arbitrary-order piecewise Chebyshev octree data structure described in Section 3.3.1. This problem appears in porous media flows [81], transport phenomena in complex fluids [56], and multiphysics simulations [3], [24].

The advection equation presents several hurdles, both with respect to its discretization and its solution on HPC systems. First, depending on the velocity field and the initial condition, the solution $c(\mathbf{x}, t)$ can develop sharp gradients that are hard to resolve. Second, the spatio-temporal scales in $\boldsymbol{v}$ need not to be consistent with the scales of the initial condition c_0. Third, special discretization schemes are necessary for the advection scheme. If a conditionally stable scheme is used with q_{th}-order elements and the smallest element size is h_{min}, then the time step δt should be $\mathcal{O}(h_{min} q^{-2})$. For large q this can result in an excessive number of time steps. The effective solution of these problem is the subject of this chapter.

As we explained our methodology in Chapter 2, we apply the unconditionally stable semi-Lagrangian method to solve the Equation (38). In Section 4.1, we begin with a brief introduction to the shared-memory semi-Lagrangian scheme, highlighting our piecewise Chebyshev octree discretization and using the Morton IDs to determine the location of the Lagrangian particles in a dynamic adaptive hierarchical domain decomposition. Evaluating the field values at the location of the Lagrangian particles is a fundamental component of the semi-Lagrangian algorithm. Hence, as the next step, we discuss our optimization techniques, such as optimization for cache usage as well as vectorization for modern CPUs by using Intel intrinsics, to compute the Chebyshev polynomial interpolations at an arbitrary set of Lagrangian points.

In Section 4.2, we extend our scheme to distributed-memory parallelization by using MPI for internode communication. First, we describe the ap-

proach used in the PVFMM library to construct distributed-memory Chebyshev octrees, where Morton space-filling curves are used to partition the tree octants among MPI processes. We then discuss the the algorithm for the semi-Lagrangian scheme on top of parallel octrees. The next step will be to efficiently evaluate an arbitrary set of Lagrangian points at a distributed-memory Chebyshev octree.

The tree evaluation at the positions of a set of Lagrangian particles requires each process to find the leaf in the octree in which the particle is located. If the given particle is not locarted on the local process, the remote process that owns the element must be determined. This can be done efficiently given the linear order of the octree [96].

In our implementation, in order to resolve each field with a desired accuracy we allow for different adaptive trees for the velocity and concentration. However, working with two trees that are constructed and partitioned completely independent of each other can create significant imbalances that can actually exceed memory resources due to load imbalance. In addition, the communication cost of the semi-Lagrangian will be independent of CFL number and depend solely on the partitioning of the concentration and velocity trees. We propose a novel partitioning scheme that addresses these issues by defining an upper-bound for communication cost in distributed-memory semi-Lagrangian schemes with minimal increase in computational cost. We elaborate on this in Section 4.3.

In Section 4.4 of this chapter, we discuss our dynamic AMR algorithm. By using AMR, we show that the number of unknowns for a computation with a fixed target accuracy can be reduced by orders of magnitude compared to the same computation using uniform grid for the same target accuracy.

Since the velocity values are given only at discrete points in time, computing the trajectories of the Lagrangian particles in unsteady velocity fields requires interpolation of velocity values in time. In Section 4.5 we address this issue.

Finally, in Section 4.6, we study the convergence of our scheme in different scenarios for steady and unsteady velocity fields and test our algorithm with time-steps that are orders of magnitude larger than the CFL stability limit.

4.1 SHARED-MEMORY SEMI-LAGRANGIAN SOLVER

In this section we describe our algorithms for implementing the semi-Lagrangian method on top of the Chebyshev octree data structure. Recall that the semi-Lagrangian method consists of two main components (see Section 2.1):

1. Backward trajectory computation to obtain the departure points of the Lagrangian particles

2. Interpolation of concentration values at the departure points

The semi-Lagrangian time-stepping scheme requires the evaluation of the concentration and velocity fields at a large number of arbitrary points in the domain. So, given the octree-based piecewise polynomial representation of a function, as discussed in 3.3.1, we need to efficiently evaluate it at a large number of arbitrary points $\{x_1, \cdots, x_n\}$. To assign an evaluation point to a leaf node we use the *Morton encoding* technology which is explained in the next section. The algorithm to evaluate arbitrary target points at a Chebyshev octree is given in Section 4.1.2. Once these building blocks are available, we discuss the algorithmic details of our semi-Lagrangian advection solver with a Chebyshev octree spatial discretization in Section 4.1.4.

4.1.1 *Morton Encoding*

Morton encoding is a mapping of multidimensional data to a one dimensional array while preserving the locality of the data points. In this thesis, we use the Morton encoding as a locational code to identify the Lagrangian particles as well as the octants of a tree. By using Morton ordering, we can represent a tree as a linear array of leaf octants.

To construct the Morton encoding for a tree octant, given three coordinates of the octant's anchor (x, y, z), and the maximum permissible depth of the tree D_{max}, we first interleave the D_{max} bits length binary representation of the three coordinates. Then, we append the binary representation of octant's level to this sequence of bits [9], [21], [97]. To compute the Morton Identification (Morton ID) of a point instead of an octant, we proceed with the same procedure, however, we use the coordinates of the point as the (x, y, z) and D_{max} as its tree depth. In the followlling, we list a few interesting properties of the Morton encodings that we exploit in our algorithms in this thesis [88]:

PROPERTY 1 Sorted list of the Morton IDs of all leaves of an octree in an ascending order is eqivalent to Preorder traversal of the leaves of the octree. By connecting the centers of the leaves, we can construct the Morton space-filling curve (see Figure 9).

PROPERTY 2 Given three octants $\mathcal{B}_a < \mathcal{B}_b < \mathcal{B}_c$ such that $\mathcal{B}_c$ is not one of the descendants of the octant $\mathcal{B}_b$ then for each descendant of the octant $\mathcal{B}_b$, the inequality $\mathcal{B}_a < \mathcal{B}_d < \mathcal{B}_c$. is valid.

PROPERTY 3 The Morton ID of any descendents of a node is is bigger than the node itself.

PROPERTY 4 The Morton id of any node and of its first child are consecutive.

By using the Morton IDs and exploiting their properties, an octree can be represented as a linear array of leaf-octants sorted by their Morton IDs. We call the sorted array of leaf-octants the *linear representation* of the octree.

In our semi-Lagrangian advection scheme, we need to evaluate the Chebyshev tree at points along the characteristics. For each evaluation point, we have to determine the leaf-octant containing that point. An efficient method for doing this is to sort all the evaluation points by their Morton ID using a sorting algorithm. It then becomes trivial to build correspondences between the array of evaluation points and the array of leaf-octants when both arrays are sorted by Morton ordering. This becomes clear in the next section.

4.1.2 *Single-Node Octree Evaluation*

In order to evaluate arbitrary target points at a piecewise Chebyshev octree we first need to assign the evaluation points to leaf octants. Therefore we compute the Morton ID m_i for each evaluation point x_i and then sort the points by their Morton ID to $\{m_{k_1}, \cdots, m_{k_n}\}$. This requires $\mathcal{O}(n \log n)$ work. Then for each leaf octant $\mathcal{B}$ we determine the Morton IDs $M_{\mathcal{B}}$ for $\mathcal{B}$ and $M_{\mathcal{B}'}$ for the next leaf octant in the tree $\mathcal{B}'$. In the sorted array of point Morton IDs, we determine the index range $I_{\mathcal{B}}$ such that $M_{\mathcal{B}} \leqslant m_{k_i} < M_{\mathcal{B}'}$ for each $i \in I_{\mathcal{B}}$. This requires just two binary searches in the sorted array of point Morton IDs for each $\mathcal{B}$. Now, we evaluate the Chebyshev approximation at each point $(x, y, z) \in \{x_{k_i} : i \in I_{\mathcal{B}}\}$,

$$c_{k_i} = \sum_{i \leqslant q} T_i(x) \sum_{i+j \leqslant q} T_j(y) \sum_{i+j+k \leqslant q} T_k(z) \alpha^{\mathcal{B}}_{ijk} \tag{39}$$

After evaluating the Chebyshev approximation at all evaluation points for each leaf octant, we rearrange the values $\{c_{k_1}, \cdots, c_{k_n}\}$ according to the original ordering of the points $\{c_1, \cdots, c_n\}$. For shared memory systems, we use an OpenMP merge-sort algorithm for sorting the Morton IDs. The procedure for evaluating a Chebyshev tree at arbitrary target points is given in Algorithm 1. In the next section, we discuss how to optimize this evaluation process.

4.1.3 *Chebyshev Interpolation Optimization*

Computing the sum in Equation (39) requires $\mathcal{O}(q^3)$ floating point operations. Even for high order approximations ($q = 14$), the coefficients $\alpha^{\mathcal{B}}_{ijk}$ easily fit in the L1 CPU cache. Therefore for all $n_{\mathcal{B}}$ evaluation points, the coefficients $\alpha^{\mathcal{B}}_{ijk}$ must be read from the main memory only once. The $3(q+1)$ Chebyshev polynomial values $T_i(x), T_j(y), T_k(z)$ are also available in the L1 cache. Therefore, the ratio of floating-point operations to the number of memory accesses to main memory (arithmetic intensity) is high and when

Algorithm 1 Evaluate Sequential Chebyshev Octree at Arbitrary Target Points

Input:
$\mathcal{T}$: Input tree
$\mathcal{X}$: List of target points
Output:
$\mathcal{V}$: Evaluated tree values at target points

```
1: procedure EvaluateSequentialTree(T, X)
2:     M_X ← MortonID(X)
3:     N ← LeafNodes(T)
4:     for B ∈ N do
5:         M_B ← MortonID(B)
6:         X_B ← BinarySearch(M_X, M_B)
7:         V ← V ∪ ChebyshevPoly(B, X_B)
8:     end for
9: end procedure
```

carefully implemented achieves high flop-rates. We have also vectorized the above computation for double precision using AVX vector intrinsics to maximize the intranode performance. For double-precision AVX vectorization, we vectorize to evaluate for 4 points together. In addition, we have parallelized the loop over the evaluation points using OpenMP. With these optimizations, we are able to achieve about 150GFLOPS on a single node of Stampede achieving 43% of peak double-precision floating-point performance.

4.1.4 *Semi-Lagrangian on a Chebyshev Octree*

Our approach to implementing the semi-Lagrangian with Chebyshev octree discretization is illustrated in Figure 8. We assume that the velocity and concentration fields at time t_k are given. The process of applying the semi-Lagrangian method on top of a Chebyshev octree can thus be summarized as follows: we first initialize a new tree $\mathcal{T}_{k+1}$ with the same refinement as the concentration tree $\mathcal{T}_k^c$ at time t_k. Then we select a set of interpolation points at each leaf-octant in $\mathcal{T}_{k+1}$. To obtain the departure points $\mathbf{X}_k = \mathbf{x}(t_k)$ at time t_k, we start from the interpolation points $\mathbf{x}_{grid}$ at time step t_{k+1} and use the explicit second order midpoint rule (Runge-Kutta) with backward velocity:

$$\hat{\mathbf{x}} = \mathbf{x}_{grid} - \frac{\delta t}{2}\boldsymbol{\nu}(\mathbf{x}_{grid}, t_{k+1}), \tag{40}$$

$$\mathbf{X}_k = \mathbf{x}_{grid} - \delta t\boldsymbol{\nu}(\hat{\mathbf{x}}, t_k + \delta t/2). \tag{41}$$

Computing the trajectory with the midpoint scheme requires evaluating the velocity values at times t_{k+1} and $t_k + \delta t/2$ for all the Lagrangian points. For steady velocity fields where the velocity does not change over time, it requires two tree evaluations at $\mathbf{x}_{grid}$ and $\hat{\mathbf{x}}$. However, for unsteady velocity fields, since the velocity values are given only at discrete time points, determining the velocity values at time $t_k + \frac{\delta t}{2}$ requires temporal interpolation which is discussed in Section 4.5. Once the departure points are determined, to obtain the concentration at time t_{k+1}, we need to construct the interpolants of the form $c(\mathbf{X}_k, t_k)$ by evaluating the concentration solution at time t_k at the semi-Lagrangian point $\mathbf{X}_k$. From these values at the interpolation points, we compute the coefficients in the Chebyshev approximation, by solving a least-squares problem for each leaf octant in $\mathcal{T}_{k+1}$. To do this, we have to construct a piecewise polynomial approximation of the form discussed in Section 3.3.1. For a given set of n interpolation points (x_i, y_i, z_i) and $i \in \{1, \cdots, n\}$, we construct the matrix $M_{ij} = T_j(x_i, y_i, z_i)$. Here, $T_j(x, y, z)$ are the Chebyshev polynomials of the form $T_{j_1}(x)T_{j_2}(y)T_{j_3}(z)$ such that $j_1 + j_2 + j_3 \leqslant q$. We precompute the pseudoinverse M^{-1} for the matrix M. Then, from the set of n values $c_i^{\mathcal{B}}$ at the interpolation points for a tree octant $\mathcal{B}$, the coefficients for its Chebyshev approximation are given by the matrix-vector product $\alpha^{\mathcal{B}} = M^{-1}c^{\mathcal{B}}$.

Typically, we want to choose the interpolation points in such a way that the matrix M is well-conditioned. If we consider a tree octant defined by the box $[-1, 1]^3$ and choose the Chebyshev points (x_i, y_j, z_j) for $i, j, k \in \{1, \cdots, n\}$ with $x_i, y_i, z_i = \cos((2i-1)\pi/(2q))$ as interpolation nodes, then the columns of the matrix M are orthogonal and the matrix is well-conditioned. However, because these node points are strictly in the interior of the box, for sufficiently small time-step size or low velocity, it results in an unstable advection scheme because no information is obtained from the octree octants in the upstream direction. Therefore, we scale the Chebyshev interpolation node coordinates by $1/\cos(\pi/(2q))$.

Notice that we are using $(q+1)^3$ node points to compute approximately $(q+2)^3/6$ coefficients for the Chebyshev approximation. However, we observed that using fewer interpolation points results in a larger condition number for the matrix M, which leads to a numerically unstable scheme for long time-horizon simulations.

4.2 DISTRIBUTED-MEMORY SEMI-LAGRANGIAN SOLVER

To target large problems with complex geometries and high spatial resolution, due to restricted available memory at each compute node, we need to extend our solver to support large parallel systems with multiple compute nodes. For such problems, the tree needs to be partitioned across compute nodes. For distributed-memory parallelism we use the Morton ID of leaf octants for tree construction, partitioning of octants across MPI processes

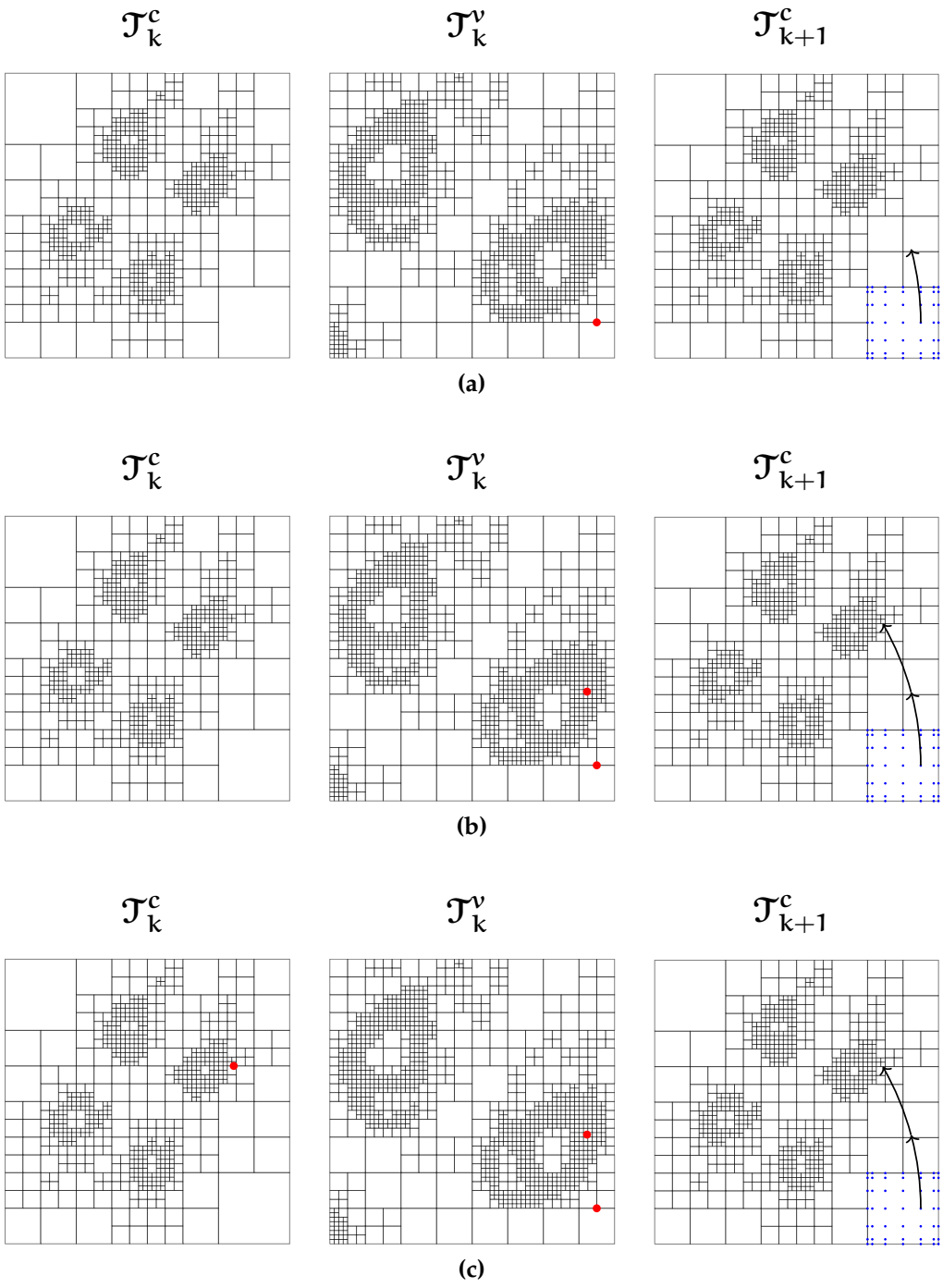


Figure 8: *Semi-Lagrangian with a piecewise Chebyshev octree spatial disctretization: we assume that the velocity and concentraiton fields at time t_k are given. We first initialize a new tree $\mathcal{T}_{k+1}$ with the same refinement as the concentration tree $\mathcal{T}_k^c$ at time t_k. In this figure, the interpolation points (Chebyshev grid) for a leaf octant $\mathcal{B}$ are indicated by blue dots. We compute the backward characteristics for these points using a second-order Runge-Kutta scheme (requires two evaluations of the velocity tree) and evaluatation of the concentration values at departure points. The evaluation points are highlighted with red dots. (a) (First row) First evaluation of velocity tree along the characteristics. (b) (Second row) The second evaluation of velocity tree along the characteristics. (c) (Third row) Evaluation of concentration tree at departure points. From the values at the interpolation points, we compute the coefficients in the Chebyshev approximation, by solving a least-squares problem for the leaf octant $\mathcal{B}$.*

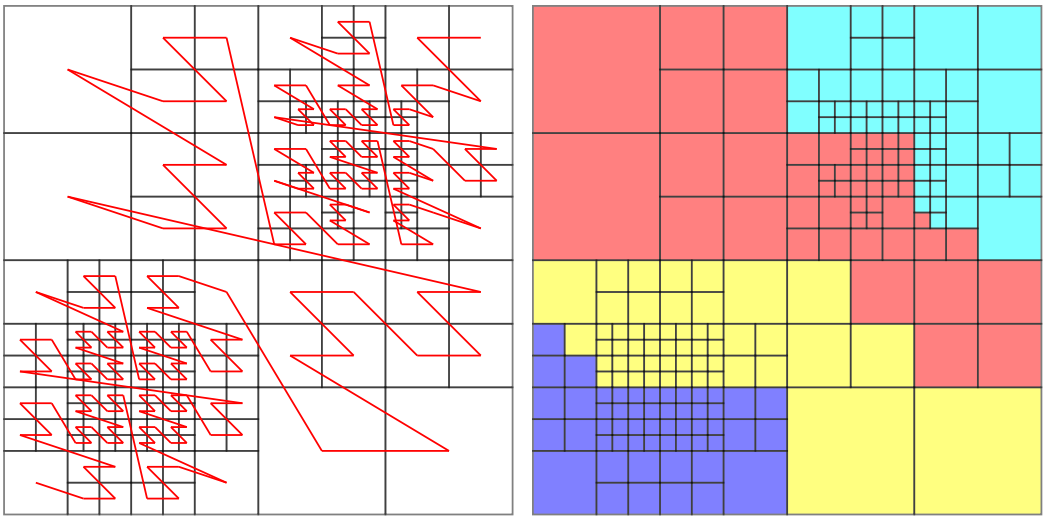
and load-balancing [100]. In the Section 4.2.1, we briefly discuss these algorithms.

The algorithm to evaluate an arbitrary set of Lagrangian particles at a distributed-memory octree and the communication steps for the evaluation process is comprehensively discussed in Section 4.2.2. Finally, in Section 4.2.3, we extend our sequential semi-Lagrangian algorithm explained in Section 4.1.4 to a distributed-memory parallel implementation.

4.2.1 *Distributed-Memory Chebyshev Octrees*

To construct a parallel adaptive Chebyshev octree, first we start with a set of uniformly positioned initial seed points. We compute the Morton ID of the seed points. Then we parallel sort the points by using the Morton ID as the sort key and partition the points uniformly among the processes. Each process constructs a linear octree by using its local points. The parameter maximum number of points per octant is used as the refinement criterion. After this step, we might need to remove the duplicate octants at the boundaries of the partitions. This gives us a uniform octree where the leaf octants are sorted by their Morton IDs. In the next step, each process performs the adaptive Chebyshev refinement procedure based on the given initial values as explained in Section 3.3.1. Depending on the field values being represented by the tree, we may encounter significant load imbalance during the adaptive refinement. Therefore we repartition the leaf octants whenever the load imbalance exceeds a threshold. To partition the leaf octants of a distributed memory octree among MPI processes, we rely on space-filling curves wich provide a unique linear ordering of all the leaf octants of the octree. In our approach we use a Morton space-filling curve which is one of many other space-filling curves [21]. The *Hilbert curve* is considered as an alternative approach with a better order-preserving behavior. However, the calculations for the Hilbert curve are significantly more compute intensive. By comparision the Morton space-filling curve is simpler to implement. In addition, in algorithms employed in this thesis, we exploit the considerable advantages of the ordering properties of Morton IDs to locate the Lagrangian particles in a dynamic adaptive hierarchical domain decomposition (see Sections 4.1.1 and 4.2.2).

The Morton space-filling curve is constructed as a distributed-memory sorted array of the Morton IDs of all the leaf octants. Recall that the computational cost of the semi-Lagrangian method is proportional to the number of octants in the concentration tree. Hence, to enforce the load balance we repartition the sorted octants among processes such that each partition has the same number of octants while we maintain the Morton ordering of the octants. This load balancing procedure requires only point-to-point communication. An example of a Morton space-filling curve is shown in Figure 9a.



(a) (b)

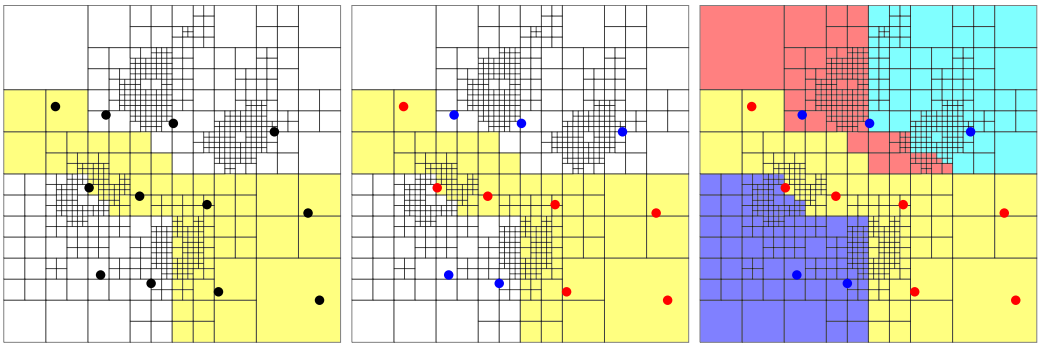
Figure 9: *Distributed memory tree partitioning by using Morton space-filling curves: (a) To partition the leaf octants of a distributed memory octree among MPI processes, we use the Morton space-filling curves. The Morton space-filling curve is constructed by creating and distributed memory sorting of an array of the Morton IDs of all the leaf octants. (b) Then we equally partition the sorted octants among processes. The octants are color-coded based on the MPI process they belong to.*

Figure 9b depicts the octants that are distributed over four MPI processes. The octants are color-coded according to the MPI process they belong to.

4.2.2 *Distributed-Memory Octree Evaluation*

The most expensive task in our semi-Lagrangian solver is evaluating the piecewise Chebyshev polynomial discretizations at a set of arbitrary Lagrangian points along the characteristics. In Section 4.1.2 we discussed our approach to address this problem for the shared-memory systems. In this section we extend our octree evaluation algorithm to distributed-memory systems.

The procedure for evaluating an arbitrary set of Lagrangian points at a distributed-memory partitioned Chebyshev octree is given in Algorithm 2. In a distributed-memory context, in addition to determining the octant containing the Lagrangian particle, we also need to determine the MPI process the particle is located in its domain's partition. To do this, similar to the approach discussed in Section 4.1.2, we exploit the order-preserving properties of Morton IDs. More precisely, we first compute the Morton ID m_i for each local evaluation point x_i and then sort the points by their Morton ID to $\{m_{k_1}, \cdots, m_{k_n}\}$. This requires $\mathcal{O}(n \log n)$ work. Then, we determine the minimum Morton ID of the current partition which is the Morton ID of the first leaf octant $M_p^{\min}$ (also called the *breakpoint* of the partition). Then, we



(a) (b) (c)

Figure 10: *Illustration of the separation of the local and remote query points for the parallel tree evaluation: (a) The query points in the yellow rank are indicated with black dots. (b) The local and remote points are indicated with red and blue dots, respectively. (c) The positions for the remote points need to be scattered to corresponding processes. After Chebyshev evaluation of the remote points, their values will be gathered to the original process (rank yellow).*

determine the Morton ID of the next partition $M_{\mathcal{P}'}^{\min}$ by global communication of the minimum Morton ID of all partitions. The values of all the points which have a Morton ID in the range $M_{\mathcal{P}}^{\min} \leqslant m_{k_i} < M_{\mathcal{P}'}^{\min}$ are locally available. In Algorithm 2, we denote these points with $\mathcal{X}_{\text{local}}$. Determining this range for each partition requires just two binary searches in the sorted array of point Morton IDs. We call the remaining points *remote points* because the positions of these points need to be *scattered* to the MPI processes of the corresponding partitions. Each MPI process evaluates the local points and other processes' remote points whose values are available at the partition's domain of the current MPI process. In Algorithm 2, the later is denoted by $\tilde{\mathcal{X}}_{\text{local}}$. Finally, we *gather* the remote points values to the original process.

The global sort for assigning Lagrangian points to octants and MPI partitions can be quite costly. Since the locations of the evaluation points are related to the Chebyshev grid points, we expect the values of the majority of the semi-Lagrangian points to be locally available. Thus, we reduce the communication by first separating all the local points and performing the global sort only on the remote points. For the global sorting, we use an efficient parallel hypercube sorting algorithm [87].

In Figure 10, we illustrate the separation of the local and remote query points for the parallel tree evaluation. In Figure 10a, the query points in the yellow rank are indicated with black dots. In Figure 10b, the local and remote points are indicated with red and blue dots, respectively. The positions and values of the remote points need to be exchanged with corresponding MPI processes as depicted in Figure 10c.

Algorithm 2 Evaluate Parallel Chebyshev Octree at Arbitrary Target Points

Input:
$\mathcal{T}$: Parallel input tree
$\mathcal{X}$: List of target points
Output:
$\mathcal{V}$: Evaluated tree values at target points

1: **procedure** EVALUATEPARALLELTREE($\mathcal{T}, \mathcal{X}$)
2: $\mathcal{M}_\mathcal{X} \leftarrow$ MORTONID($\mathcal{X}$)
3: $\mathcal{M}_\mathcal{X} \leftarrow$ MERGESORT($\mathcal{M}_\mathcal{X}$)
4: $\mathcal{N} \leftarrow$ LEAFNODES($\mathcal{T}$)
5: $\mathcal{M}_\mathcal{P}^{min} \leftarrow$ MORTONID($\mathcal{N}[0]$) ▷ Local breakpoint
6: $\mathcal{M}^{min} \leftarrow$ MPI_ALLGATHER($\mathcal{M}_\mathcal{P}^{min}$) ▷ Global breakpoint list
7: $\mathcal{M}_{\mathcal{P}'}^{min} \leftarrow \mathcal{M}^{min}[\mathtt{myrank}+1]$ ▷ Next partition's breakpoint
8: $\mathcal{I}_{min} \leftarrow$ BINARYSEARCH($\mathcal{M}_\mathcal{X}, \mathcal{M}_\mathcal{P}^{min}$)
9: $\mathcal{I}_{max} \leftarrow$ BINARYSEARCH($\mathcal{M}_\mathcal{X}, \mathcal{M}_{\mathcal{P}'}^{min}$)
10: $\mathcal{X}_{local} \leftarrow \mathcal{M}_\mathcal{X}[\mathcal{I}_{min} \cdots \mathcal{I}_{max}]$
11: $\mathcal{V}_{local} \leftarrow$ EVALUATESEQUENTIALTREE($\mathcal{T}, \mathcal{X}_{local}$) ▷ Algorithm 1
12: $\mathcal{X}_{remote} \leftarrow \mathcal{M}_\mathcal{X} \setminus \mathcal{X}_{local}$
13: $\tilde{\mathcal{X}}_{local} \leftarrow$ PARALLELSORT($\mathcal{M}^{min}, \mathcal{X}_{remote}$)
14: $\tilde{\mathcal{V}}_{remote} \leftarrow$ EVALUATESEQUENTIALTREE($\mathcal{T}, \tilde{\mathcal{X}}_{local}$) ▷ Algorithm 1
15: $\mathcal{V}_{remote} \leftarrow$ MPI_ALLTOALL($\tilde{\mathcal{V}}_{remote}$)
16: $\mathcal{V} \leftarrow \mathcal{V}_{local} \cup \mathcal{V}_{remote}$
17: **end procedure**

4.2.3 *Semi-Lagrangian on Parallel Chebyshev Octrees*

Once an efficient distributed-memory algorithm for octree evaluation is available (see Algorithm 2), we can extend the sequential semi-Lagrangian scheme to distributed-memory systems by constructing the trees in a parallel manner and integrating the parallel tree evaluation algorithm in the two main components of the semi-Lagrangian, namely computing the characteristics and interpolating the concentration values at the departure points. This procedure is given in Algorithm 3.

In analogy to the sequential algorithm, we initialize a new tree for the next time step with the same refinement as the concentration tree of the previous time step. Each process iterates over each local leaf-octant to construct a single array of the interpolation points of all the local octants in each process. Then we solve the characteristics backward in time by using the positions of the interpolation points as the initial value. This requires two parallel velocity tree evaluations for each interpolation point. In the next step we evaluate the concentration values at the departure points and based on these values we construct the Chebyshev coefficients for each leaf-

octant. Finally, we adapt our mesh to new values by performing our adaptive mesh refinement/coarsening procedure (see Section 4.4).

In general velocity and concentration values are represented by separate octrees. This allows us to resolve each field with any desired accuracy. In addition, by using separate velocity and concentration fields, coupling the semi-Lagrangian solver to any other solver becomes straightforward.

However, since the trees are partitioned independently across MPI tasks, the load balancing, minimization of communication, and efficient memory utilization for tree evaluations in the naive implementation of the semi-Lagrangian become quite challenging. In the next section, we address these issues by introducing a novel tree partitioning scheme.

Algorithm 3 Semi-Lagrangian on Parallel Chebyshev Octree Data-Structure

Input:
$\mathcal{T}_c, \mathcal{T}_v$: Input tree for c and v
Output:
Updated $\mathcal{T}_c$

1: **procedure** SolveSemiLag($\mathcal{T}_c, \mathcal{T}_v$)
2: $\mathcal{N} \leftarrow$ LeafNodes($\mathcal{T}_c$)
3: **for each** $\mathcal{B} \in \mathcal{N}$ **do**
4: $\mathcal{X} \leftarrow \mathcal{X} \cup$ InterpPoints($\mathcal{B}$)
5: **end for**
6: $\mathcal{X}_{departure} \leftarrow$ Trajectory($\mathcal{T}_v, \mathcal{X}$) ▷ Multiple Algorithm 2
7: $c \leftarrow$ EvaluateParallelTree($\mathcal{T}_c, \mathcal{X}_{departure}$) ▷ Algorithm 2
8: **for each** $\mathcal{B} \in \mathcal{N}$ **do**
9: $\alpha^{\mathcal{B}} \leftarrow M^{-1} c^{\mathcal{B}}$
10: **end for**
11: RefineTree($\mathcal{T}_c$) ▷ See Section 4.4
12: **end procedure**

4.3 DISTRIBUTED-MEMORY PARTITIONING SCHEMES

If we allow the velocity tree to be partitioned independently of the concentration tree (Separate-Trees (ST) approach, see Figures 11 and 12a), the following two issues arise:

- Firstly, the evaluation points may be partitioned completely differently from the partitioning of the velocity tree, requiring a very high communication load to send evaluation coordinates to remote velocity tree partitions and to bring back the evaluated velocity data to the concentration tree partition. As a result, the communication cost is independent of the CFL number and depends solely on the partitioning of the trees.

- Secondly, for regions with a very fine concentration mesh and a coarse velocity mesh, a very large number of evaluation points will be assigned to a single velocity partition causing imbalances in terms of computation and memory. If the spatial features of the velocity and concentration trees are very different, such imbalances can be quite significant to the extent that they can exhaust the memory in a hardware node and cause crashes.

Both issues would severely impair the scalability and robustness of our solver. Hence, a preprocessing step is necessary to address this problem.

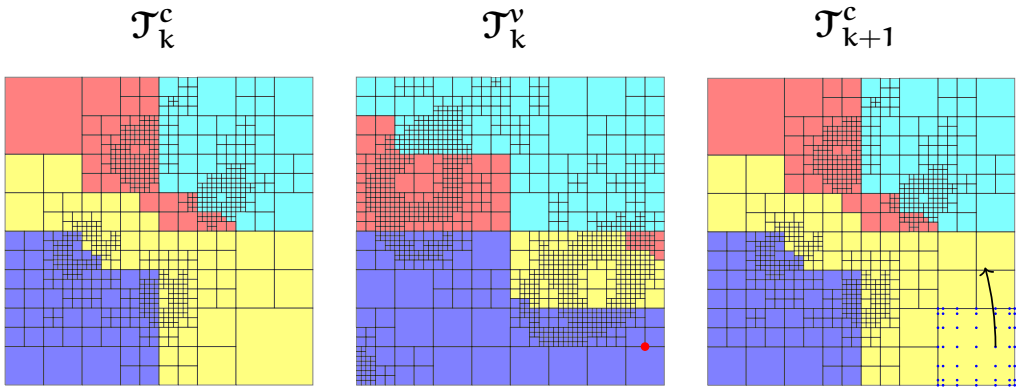
In Figure 11, we illustrate the Semi-Lagrangian scheme with two separate trees $\mathcal{T}^c$ and $\mathcal{T}^v$ for the concentration and velocity values, respectively. The two trees are constructed and partitioned independently. The blue dots present the Chebyshev grid in the concentration tree. Starting with this grid, we solve the characteristics with the mid-point method. The evaluation of concentration and velocity trees along the characteristics and the departure points are indicated with red dots. For simplicity, we present the procedure for only one particular octant located in the domain's partition of the yellow rank. For this octant, the grid positions and the evaluated velocity values need to be exchanged between blue and yellow ranks (see Figures 11a and 11b). Once the departure points are determined, an evaluation of the concentration tree is required which also needs internode communication with the green rank (see Figure 11c).

As is shown in Figure 11, computing the characteristics in the Semi-Lagrangian scheme with two separate trees which are partitioned independently requires high communication cost. This cost is independent of the CFL number and has no upper-bound. Conventional methods such as creating *Ghost Layers* and communicating the overlapping domains before proceeding with the semi-Lagrangian procedure can not address this issue, since it might lead to complete replication of the velocity partitions in the concentration tree's processes. This results in excessive memory consumption. In addition, the procedure for predicting ghost layers introduces an additional overhead.

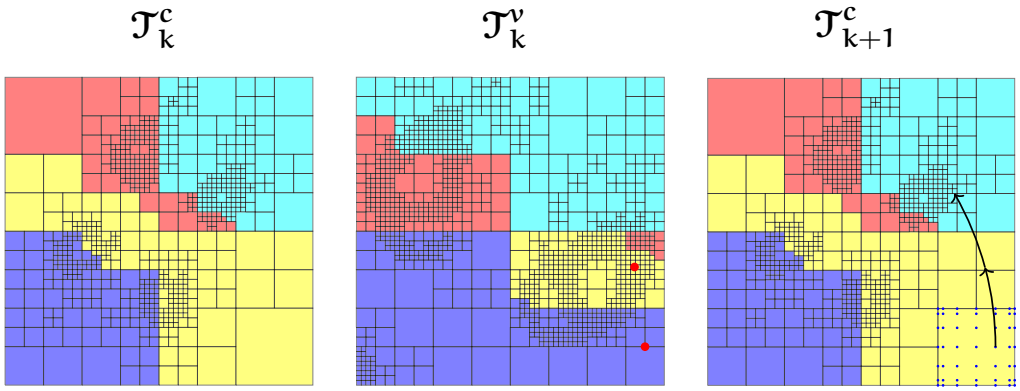
Typically, for reasonable CFL numbers, the displacement of the Lagrangian points in one time-step is small. Therefore, to a large extent, the evaluation points are distributed according to the discretization of the concentration tree. Based on this observation, in Sections 4.3.1 and 4.3.2, we propose two possible solutions to the above stated issues.

4.3.1 *Complete-Merge Scheme*

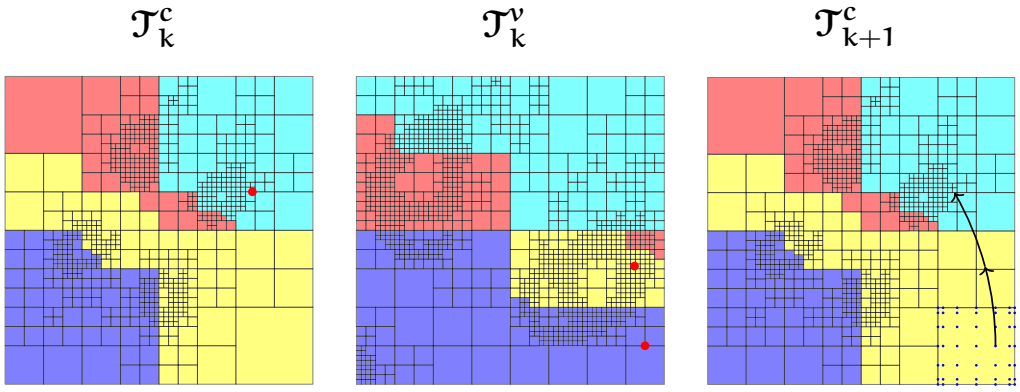
Having the same partitioning for both concentration and velocity values, such that the same domain areas of the concentration and velocity fields are assigned to the same MPI process, is a desirable property. Since the



(a) *Evaluation of velocity tree along the characteristics: First Evaluation*



(b) *Evaluation of velocity tree along the characteristics: Second Evaluation*



(c) *Evaluation of concentration tree at departure points*

Figure 11: *Illustration of the Semi-Lagrangian scheme with two separate trees for the concentration ($\mathcal{T}^c$) and velocity ($\mathcal{T}^v$) values. Computing the characteristics in the Semi-Lagrangian scheme with two separate independently partitioned trees requires high communication cost. This cost is independent of the CFL number and has no upper-bound: (a) (First row) The two trees are constructed and partitioned independently. The blue dots represent the Chebyshev grid in the concentration tree. Starting with this grid, we solve the characteristics with the mid-point method. The evaluation points in the velocity tree are indicated with red dots. (b) (Second row) For this particular octant, the grid positions and the evaluated velocity values need to exchanged between yellow and blue ranks. (c) (Third row) Once the departure points are determined, an evaluation of the concentration tree is required which also requires internode communication with green rank.*

displacement of the Lagrangian particles in one time-step is small, this improves the data locality and therefore reduces the communication cost for computing the characteristics for Lagrangian particles.

In the *Complete-Merge (CM)* scheme, we achieve the same partitioning of the trees by enforcing the same refinement for both trees. That is, we refine the velocity tree in regions where it is coarser than the concentration tree and similarly refine the concentration tree in regions where it is coarser than the velocity tree. The resulting trees, which now have the same refinement, are partitioned uniformly across the processes. As a result both trees have the same partitioning. The Complete-Merge scheme is depicted in Figure 12b.

Recall that the computational complexity of the semi-Lagrangian scheme is proportional to the number of leaf-octants of the concentration tree. That is, the refinement introduced in the concentration tree due to the Complete-Merge scheme increases the computational cost without contributing to the numerical accuracy. This increase in computational cost can be severe when the two trees refinement varies significantly. As a result, a major disadvantage of the Complete-Merge approach is that depending on the spatial features of the velocity and concentration trees, it requires a larger number of unknowns for a fixed target accuracy. A performance comparison between the Complete-Merge scheme and the original Separate-Trees scheme is given in Section 4.3.3.

4.3.2 *Semi-Merge Scheme*

The *Semi-Merge (SM)* approach enforces the same partitioning for both concentration and velocity trees with the help of some additional refinement but allows for the two trees to have non-identical refinements. That is, an MPI task is responsible for the same spatial region in both trees but the number of octants in the concentration and velocity trees in the region can differ. An example of the Semi-Merge approach is depicted in Figure 12c.

The procedure used to establish the Semi-Merge partitioning scheme is given in Algorithm 4. To determine the new partitioning, we merge the Morton IDs of the leaf octants of both trees and then we sort them and partition them such that each partition has the same number of leaves. The partitioning of these Morton IDs gives us the optimal partitioning such that the combined number of local leaf octants of both trees is roughly the same across all processes. However, depending on the trees' original refinement, some of the leaf nodes corresponding to the Morton IDs at the new partition boundary may not exist on one of the two trees. Therefore, we introduce these new leaf nodes by refining the trees at the new partition boundary. Then, we can impose the new partitioning to the original trees.

This approach adds only a very small number of octants to either tree and in general would have significantly fewer octants compared to the

Complete-Merge approach. The main memory costs are associated with the leaves. Therefore, partitioning the merged leaves helps with memory load balance and makes the overall scheme robust.

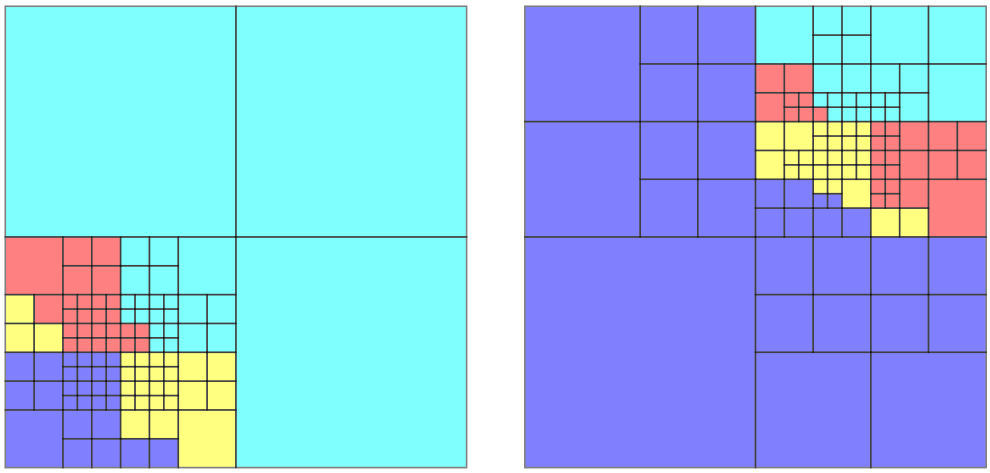
The strength of the Semi-Merge method comes from the ability of the method to enforce the same partitioning on the two trees and hence improves the data locality of the velocity and concentration, without noticeably increasing the computational cost due to additional refinement as in the Complete-Merge scheme. Of course this doesn't ensure optimal work load balance, but in practice it performs reasonably well for communication bounded problems such as the distributed-memory semi-Lagrangian scheme.

Algorithm 4 Semi-Merge Partitioning Scheme

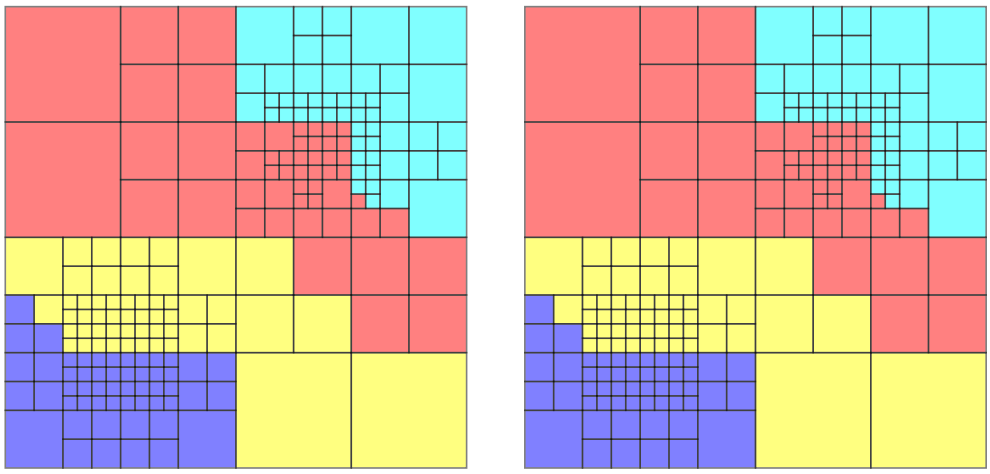
Input:
$\mathcal{T}_c, \mathcal{T}_v$: input trees for c and v
Output:
Semi-Merged $\mathcal{T}_c$ and $\mathcal{T}_v$

```
1:  procedure SemiMerge(T_c, T_v)
2:      M ← MortonID(T_v) ∪ MortonID(T_c)
3:      M ← ParallelSort(M)
4:      M ← Redistribute(M)
5:      b ← M[0]                                  ▷ New local breakpoint
6:      if b ∉ T_v then
7:          RefineTree(T_v, b)
8:      end if
9:      if b ∉ T_c then
10:         RefineTree(T_c, b)
11:     end if
12:     RepartitionTree(T_v, b)
13:     RepartitionTree(T_c, b)
14: end procedure
```

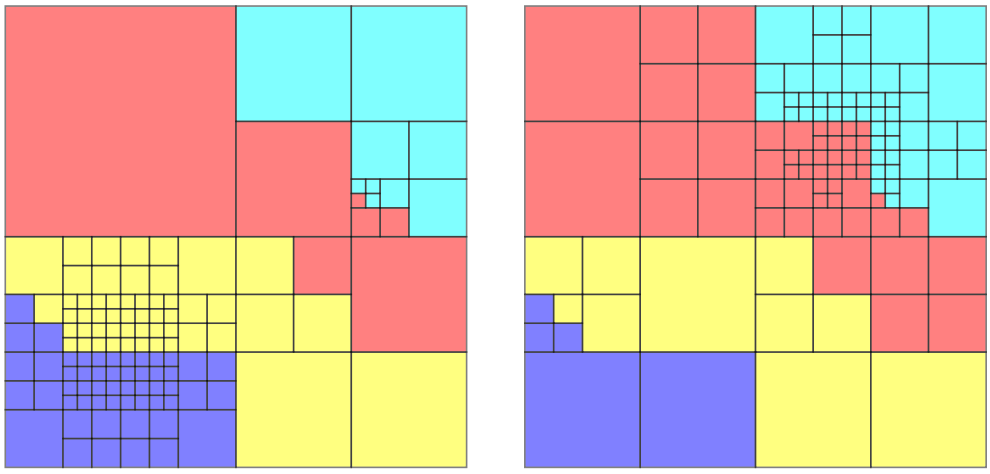
For small CFL numbers we can prove that our code will not crash due to insufficient memory caused by a large load imbalance. However, for a semi-Lagrangian scheme, it is hard to balance the memory for arbitrarily large CFL numbers and complex velocity fields since the particle distribution during the interpolation phase can be different from both concentration and velocity fields. Also, it is worth noting that it would be possible to introduce some kind of weighted partitioning to improve performance but we do not adopt this approach in this thesis. With suitable weighting factors, the partitioning of these Morton IDs yeilds the optimal partitioning required to ensure that the combined work of local leaf octants of both trees is roughly the same across all processes.



(a) ***Separate-Trees (ST):*** *The concentration and velocity trees are constructed and partitioned independently.*



(b) ***Complete-Merge (CM):*** *The trees are forced to have same the refinement and thus also the same partitioning.*



(c) ***Semi-Merge (SM):*** *The trees are forced to have the same partitioning with the help of some additional refinement but are allowed to have non-identical refinements.*

Figure 12: *Illustration of different partitioning schemes for concentration and velocity trees.*

4.3.3 *Comparison of Partitioning Schemes*

In this section we present a detailed comparison of the total runtime breakdown for the three schemes: the original Separate-Trees approach in which we do not attempt to merge the trees (denoted by ST), the Complete-Merge scheme (denoted by CM) and the Semi-Merge scheme (denoted by SM). In this test scenario, the initial concentration is given by one Gaussian function with amplitude of 1.0 and variance $\sigma = 5\text{E-}2$ placed at $r = (0.7, 0.7, 0.7)$ in a unit cube. We use the Hopf field as our velocity field:

$$\boldsymbol{v}(x, y, z, t) = \frac{A}{(r^2 + x^2 + y^2 + z^2)^2} \begin{pmatrix} 2(-ry + xz) \\ 2(rx + yz) \\ r^2 - x^2 - y^2 - z^2 \end{pmatrix},$$

where r is the constant radius to the inner coil positioned at the center of unit cube and $A = 0.05$. We solve the advection problem for one time step with $\delta t = 2.5\text{E-}2$ and low-order discretization ($q = 5$).

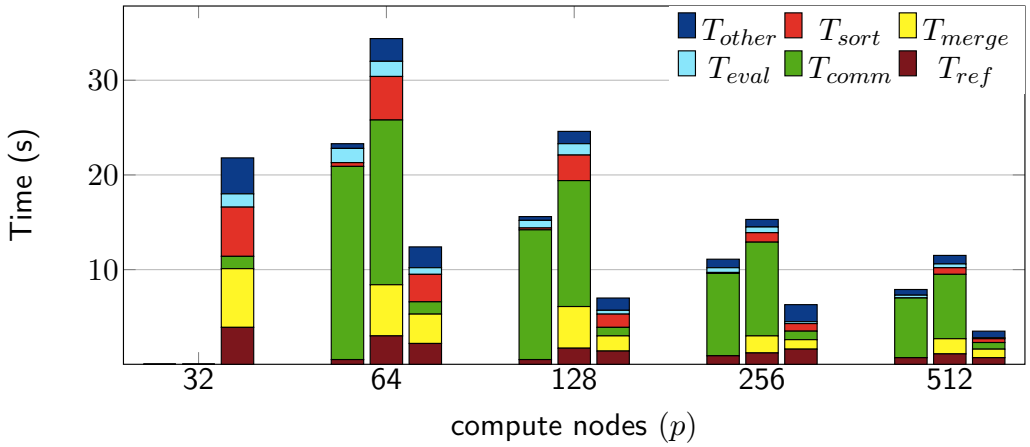
The experiments are carried out on the Stampede system at the Texas Advanced Computing Center. Stampede nodes have two 8-core Intel Xeon E5-2680 (2.8GHz) processors and 32GB RAM. All of our runs were done using ***1 MPI task / node*** and ***16 OpenMP threads***.

PERFORMANCE COMPARISON: STRONG SCALING In Figure 13 we study the fixed-sized (strong) scaling results, varying the number of MPI tasks from 32 to 512 processes for a problem size of 40 Million unknowns. We report the detailed breakdown of the total time (T_{solve}) into tree merging (T_{merge}), sorting of local points (T_{sort}), communicating of points with other processes (T_{comm}), Chebyshev evaluation (T_{eval}) and tree refinement (T_{ref}).

The communication cost is the time for communicating point coordinates and returning the evaluated values for points which have to be evaluated on a remote processor. The communication cost in Separate-Trees scheme accounts for $78\% - 87\%$ of total time while in Complete-Merge scheme this is reduced to $50\% - 64\%$ of the total time. In the Semi-Merge scheme, the communication accounts for only $5\% - 20\%$ of total time which is a relatively small percentage. We observe a reduction of communication cost in Semi-Merge scheme by a factor of $9\times$ to $15.6\times$ compared to Separate-Trees scheme and $9.7\times$ to $14.7\times$ compared to Complete-Merge scheme.

The time spent in partitioning of the trees is denoted by T_{merge}. In the Complete-Merge scheme this accounts for $11\% - 17\%$ of the total time while it increases to $15\% - 28\%$ of the total time in the Semi-Merge scheme. However, due to additional refinement required in the Complete-Merge scheme, the partitioning cost in Complete-Merge is more expensive than the Semi-Merge scheme by a factor of $1.7\times$ to $2.75\times$.

We observe that for all the runs of the Complete-Merge scheme, T_{ref}, T_{sort} and T_{eval} are more expensive than both Separate-Trees and Semi-Merge. A



(a) *Comparision of the strong scaling results for different partitioning schemes. In each column group, we present the results of Separate-Tree, Complete-Merge and Semi-Merge schemes, respectively.*

Merging	p	q	N_{dof}	Semi-Lagrangian					
				T_{ref}	T_{merge}	T_{comm}	T_{sort}	T_{eval}	T_{solve}
ST	32	5	–	–	–	–	–	–	–
CM			–	–	–	–	–	–	–
SM			4.0E+7	3.9	6.2	1.3	5.2	1.4	21.8
ST	64	5	4.0E+7	0.5	–	20.4	0.4	1.5	23.3
CM			4.3E+8	3.0	5.4	17.4	4.6	1.6	34.4
SM			4.0E+7	2.2	3.1	1.3	2.9	0.7	12.4
ST	128	5	4.0E+7	0.5	–	13.7	0.2	0.8	15.6
CM			4.3E+8	1.7	4.4	13.3	2.7	1.2	24.6
SM			4.0E+7	1.4	1.6	0.9	1.4	0.4	7.0
ST	256	5	4.0E+7	0.9	–	8.7	0.1	0.5	11.1
CM			4.3E+8	1.2	1.8	9.9	1.0	0.6	15.3
SM			4.0E+7	1.6	1.0	0.9	0.8	0.2	6.3
ST	512	5	4.0E+7	0.7	–	6.3	0.0	0.3	7.9
CM			4.3E+8	1.1	1.6	6.8	0.7	0.4	11.5
SM			4.0E+7	0.7	0.9	0.7	0.4	0.1	3.5

(b) *Detailed breakdown of the total time of the strong scaling experiment for different partitioning schemes. Here, the number of processes and unknowns are denoted by* p *and* N_{dof}*, respectively.*

Figure 13: *Performance comparison of merging schemes for a strong scaling test of an advection problem with a Hopf field as the velocity field for one time step with* $\delta t = 2.5E\text{-}2$ *and low-order discretization* ($q = 5$)*. All of our runs were executed using 1 MPI task / node and 16 OpenMP threads. We report the detailed breakdown of the total time* (T_{solve}) *into tree merging* (T_{merge})*, sorting of local points* (T_{sort})*, communicating of points with other processes* (T_{comm})*, Chebyshev evaluation* (T_{eval}) *and tree refinement* (T_{ref})*. In this test case, the Semi-Merge algorithm is more than* $2\times$ *faster than the original approach and more than* $3\times$ *faster than the Complete-Merge approach.*

major disadvantage of the Complete-Merge approach is that it requires a larger number of unknowns. Both velocity and concentration fields have to be represented on a finer mesh than what would be required if both quantities were represented independently on separate trees. Therefore, the Complete-Merge scheme has a larger memory footprint and requires more work and more communication when compared to our Semi-Merge scheme. These drawbacks can be clearly observed in the results presented in Figure 13, where the Complete-Merge scheme is over 3x slower than the Semi-Merge scheme. While the original approach requires a similar number of unknowns, it is over 2x slower than the Semi-Merge scheme due to the high communication overhead. In this test case, both the original and the Complete-Merge approaches failed for 32 processes due to exceeding memory consumption while the Semi-Merge approach performed well. The success of the Semi-Merge method is due in part to its ability to load balance the memory footprint required for the evaluation of Lagrangian particles.

PERFORMANCE COMPARISON: SEPARATION OF LOCAL AND REMOTE POINTS As we discussed in Section 4.2.2, to evaluate a set of arbitrary points on a distributed-memory tree and to locate the corresponding partition for each evaluation point, we perform a global search over the Morton ID of the points.

In Table 1, we compare the tree evaluation cost for the three partitioning schemes discussed in the previous section. For each partitioning scheme, we study two cases:

1. We perform the global search on all of the evaluation points. In Table 1, the evaluation cost for this procedure is denoted by T_G.

2. We first separate the evaluation points whose values are locally available and we perform the global search only for the remote points, so reducing the communication cost for the global search procedure. The evaluation cost for this case is denoted by T_L.

We use the same velocity and concentration field as in Figure 13 but to increase the number of remote points the CFL number is increased by a factor of 10. We report the time spent in the tree evaluation with different merging schemes in a one step semi-Lagrangian with RK2 for various numbers of processes (strong scaling). The first two evaluations compute the velocity values at the positions of the Lagrangian particles along the characteristics, while the third evaluation computes the concentration values at the departure points.

For the first evaluation, separating the local points and remote points for the Complete-Merge scheme results in a speedup of 1.6× to 2.57×, while for the Semi-Merge scheme, we achieve a speedup of 3.46×. However, the

speedup for the Separate-Trees scheme is negligible. The first tree evaluation with local points separation for the Semi-Merge scheme is up to $4\times$ faster than the Complete-Merge scheme, when the number of processes is increased, the speedup decreases. In comparison with the Separate-Trees scheme, the Semi-Merge is up to $4.15\times$ faster, however, when the number of processes is increased, the speedup also increases.

For the second evaluation in the velocity field, the separation of local points for the Separate-Trees, Complete-Merge and Semi-Merge schemes results in speedups of up to $1.5\times$, $1.29\times$ and $2.7\times$, respectively. Again, the Semi-Merge scheme outperforms the Separate-Trees scheme by up to a factor of $2.7\times$ and the Complete-Merge scheme by up to a factor of $3\times$.

For the evaluation of the concentration tree at the departure points (third evaluation in Table 1), the separation of the local points and remote points for all three schemes results in better performance. However, we observe that the Separate-Trees scheme outperforms the other schemes. The reason for this is that synchronizing the partitioning of the velocity and concentration trees improves only the velocity tree evaluations and does not necessarily contribute to the concentration tree evaluations at the departure points.

4.4 DYNAMIC ADAPTIVE MESH REFINEMENT/COARSENING

Many problems require dynamic changes in spatio-temporal scales while evolving over time. Accurate solving of these problems can be achieved by using higher spatial resolution or by using higher order methods both in space and time. Problems such as mantle convection, which describes the thermal and geological evolution of the earth [18] or climate models to study the local effects of climate change [69] are two examples of scientific and engineering problems, which in practice require higher resolutions to capture the phenomena of interest in sufficient detail.

In general, problems in scientific simulations scale as N^{α}. The power α can be reduced by using fast methods such as Fast Multipole Method for N-body problems and integral equations as we exploit in this work and is discussed in Chapter 3. A further work reduction can only be obtained by reducing N itself [19]. This can be achieved by using AMR methods.

AMR dynamically adapts the mesh to resolve spatio-temporal features of interest during the simulation. For many problems, high spatial resolutions are required only for particular regions of the simulation domain. For such problems, AMR can reduce the number of unknowns by orders of magnitude by invoking higher resolution only for the area of interest in the simulation domain. For instance, the grid refinement follows the gradient of the values or the local error estimates (see Chapter 3) and other regions of the domain will be approximated on coarser grids. However, designing an AMR algorithm that is dynamic, scalable, and that supports

Table 1: *Comparison of the time spent in tree evaluation by performing global sort on all evaluation points (denoted by T_G) vs. separating the local points and performing the global sort only on remote points (denoted by T_L). The experiment was conducted for different merging schemes for one time-step semi-Lagrangian with RK2 (three evaluations). The first two evaluations compute the velocity values at the positions of Lagrangian particles along the characteristics while the third evaluation computes the concentration values at the departure points.*

Merging	p	Tree Evaluation					
		First		Second		Third	
		T_L	T_G	T_L	T_G	T_L	T_G
ST	1	7.59	7.66	7.74	7.76	2.57	2.60
CM	1	27.86	28.61	28.25	28.79	9.55	10.01
SM	1	7.77	8.12	7.04	7.92	2.52	2.64
ST	8	2.94	4.22	3.16	4.38	0.49	0.52
CM	8	3.92	10.09	3.74	4.84	1.35	1.75
SM	8	0.97	3.13	1.42	3.87	0.51	1.26
ST	16	1.79	2.17	2.05	2.29	0.28	0.32
CM	16	1.79	3.08	2.28	2.95	0.95	1.14
SM	16	0.48	1.63	0.75	1.78	0.32	0.78
ST	32	1.08	1.17	1.19	1.80	0.20	0.22
CM	32	0.91	1.52	1.46	1.81	0.49	0.62
SM	32	0.26	0.90	0.54	1.09	0.25	0.40

high-order methods is challenging. In order to gain better performance by using AMR methods, the time needed for AMR components should remain small compared to the solver time, so that inefficiencies of the algorithms for adaptivity does not balance out the gains accrued by using AMR. Developing efficient parallel dynamic load-balancing for AMR that does not require large volumes of communication is an active area of research. Efficient dynamic data-structures for multiscale problems are also essential.

In general the AMR methods fall into two categories: structured (SAMR) and unstructured (UAMR). In SAMR method, the domain is decomposed by adaptive, hierarchical, logically-rectangular or structured triangular [4], [66] grid regions. In contrast to SAMR, the UAMR methods deploy conforming unstructured elements like tetrahedra in three dimensions. Unstructured mesh algorithms and their adaptive strategies are out of the scope of this thesis. For a comprehensive treatment and for references to the extensive literature on the subject one may refer to [10], [23], [48].

The SAMR methods are first developed in [7], [8] for shock hydrodynamics problems but they have proved to be popular technique in other areas such as CFD [2], [4], [52], [74], [90], [92] and cosmology and astrophysics [14], [32]. A comprehensive review and references of SAMR methods are provided in [23]. In this work, we take a great advantage of features of tree-based SAMR method such as higher performance due to regular grid structure and being able to reuse the structured grid code in nested domain components (see Section 3.3.1). In the following, we explain the essential components of our dynamic AMR, namely our *tree refining/coarsening* algorithm and our *dynamic mesh adaptation* approach.

4.4.1 *Tree Coarsening/Refining Procedure*

The tree *refining* operation is designed to be local. Each process first builds a list of all local leaf-octants by a post order traversal of the local tree. Then we proceed with adaptive refinement of leaf-octants in the list. More precisely, we iterate over each leaf-octant in the list and check if it satisfies the refinement criteria. That is, we estimate the truncation error per octant by computing the absolute sum of the highest order coefficients (see Equation (34)) and in the event that the estimated error is bigger than the predefined tree error tolerance, we subdivide the octant into eight childeren and continue with the same procedure for the children, by adding them to the leaf-octants list. This procedure continues until all the leaf-octants reach the desired accuracy or the maximum permissible tree depth is reached. This is done directly on the octree and consists of completely local operations requiring no internode communication.

The Chebyshev coefficients of the newly created children are constructed from the values of the parent octants. That is, we evaluate the values at the Chebyshev node points of the new child octant. Then, obtaining the coefficients from the points is accomplished using a tensor product rule as it is explained in Section 3.3.1 (by applying 1D Chebyshev approximation in each direction).

The independent local refinement of the tree can lead to load-imbalance. To balance the load and avoid exceeding local memory, the mesh must be redistributed uniformly among all processes. Consequently in the next step we determine the load-imbalance by counting the number of octants in each process (require `MPI_Allreduce` communication) and repartition the octants if the load-imbalance has exceeded a predefined threshold (requires point-to-point communication). The tree *coarsening* procedure follows a similar approach, examining the local partition of the octree for eight leaves from the same parent that satisfy the coarsening criteria.

4.4.2 *Dynamic Mesh Adaptation*

Since the spatial features of the solution evolve over time, we need to dynamically adapt the mesh to the changing distribution of the concentration values. In our semi-Lagrangian implementation in Algorithm 3, we first initialize a tree with the same refinement as in the previous time-step. Once we have computed the values of the new time-step and stored them in this tree, we call the Coarsen/Refine procedure to adapt the mesh to the new field values. In our implementation, we adapt the mesh to the new values after each time-step. However, depending on the speed of the propagation of information in our domain and the CFL number, the AMR procedure can be performed every few time-steps.

In Figure 14 we present the visualization of a slotted Zalesak disk in a rotating velocity field and a sequence of meshes that are adapted to the Zalesak disk as it changes over time. In this example, the maximum permissible tree depth is $D_{depth} = 7$ and we use high order spatial discretization $q = 14$. This is an effective resolution of nearly 1 billion unknowns with a uniform mesh.

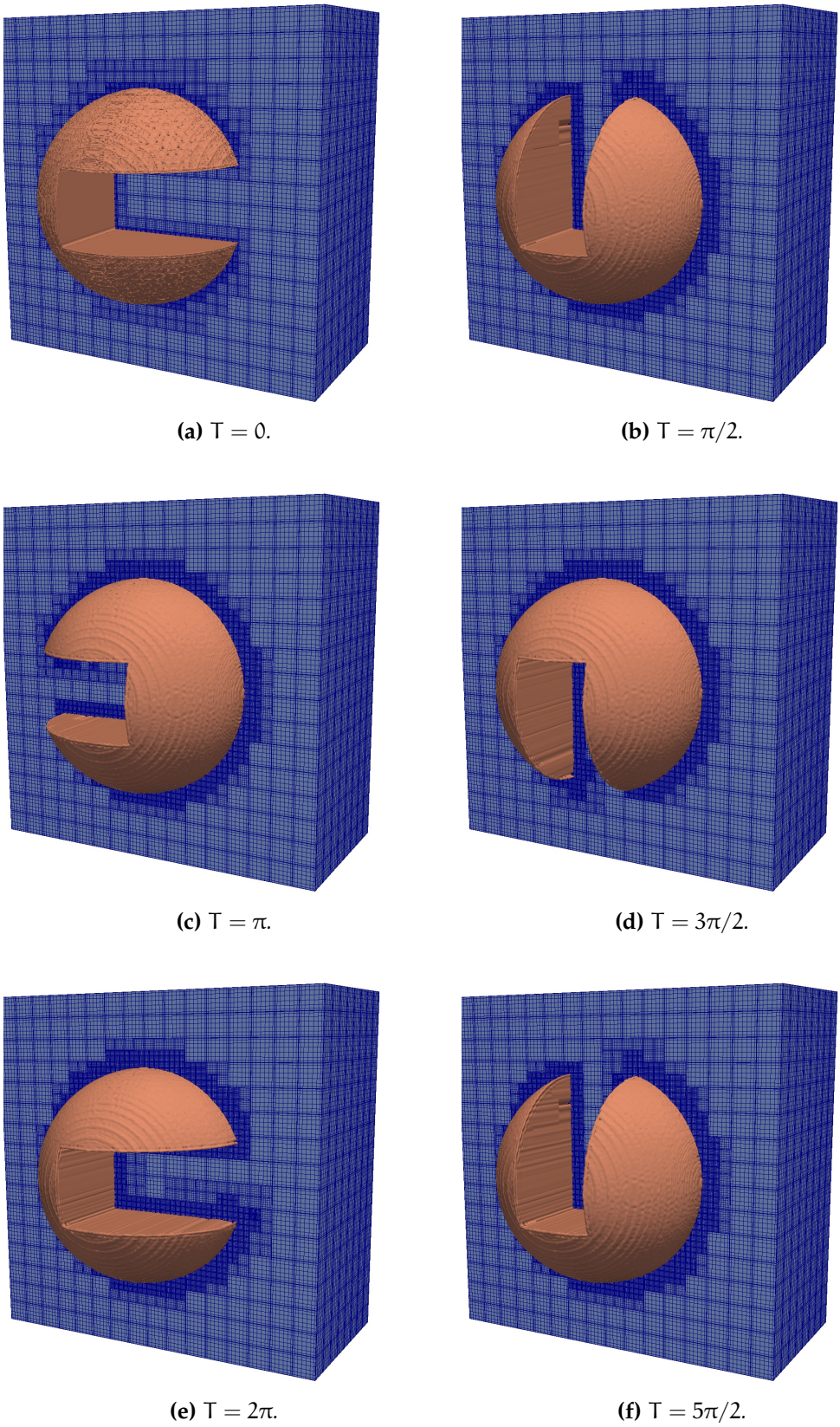
4.5 SEMI-LAGRANGIAN WITH UNSTEADY VELOCITY FIELDS

So far we have considered only transport problems with steady velocity $\boldsymbol{v}(\boldsymbol{x})$ where the velocity does not change over time. Computing the trajectory for Lagriangian particles using the explict second order midpoint rule requires the evaluation of velocity fields at times t_{k+1} and $t_k + \delta t/2$ along the characteristics (see Equations (40) and (41)). For steady velocity fields, no temporal interpolation of discrete velocity values is required and therefore we can exploit the same parallel tree evaluation described in Section 4.2.2. However, since the velocity values are normally given at discrete points in time, the evaluation for unsteady velocity fields at time $t_k + \delta t/2$ requires additional treatment.

In this section, we propose two possible solutions to off-grid temporal evaluation for unsteady velocity values represented with a tree data structure at discrete points in simulation time. We consider the following two scenarios:

1. The velocity field is available for any arbitrary number of future time-steps and is not affected by the features of the concentration field. For example, the coupling of velocity and concentration values is one directional and the velocity values are obtained independently from a separate solver.

2. The velocity values are not available in future time-steps. For example, the velocity itself is the quantity of interest we are computing. Solving the incompressible Navier–Stokes equations by treating the advective



(a) $T = 0$. **(b)** $T = \pi/2$.

(c) $T = \pi$. **(d)** $T = 3\pi/2$.

(e) $T = 2\pi$. **(f)** $T = 5\pi/2$.

Figure 14: *Illustration of rotating slotted Zalesak's disc in 3D in a vorticity velocity field with dynamic AMR. The figure shows the snapshots of meshes that are adapted to the Zalesak disk as it changes over time. The orange area represent the Zalesak disc and the blue area shows the corresponding adapted meshes.*

term with the semi-Lagrangian scheme (see Chapter 7) falls into this category.

In the next section, we address the first scenario by using a cubic interpolation procedure for time-dependent fields, while for the second scenario, we deploy a second-order extrapolation scheme which is discussed in Section 4.5.2.

4.5.1 *Temporal Interpolation Scheme*

We would like to interpolate a time-dependent field $\boldsymbol{v}(\boldsymbol{x}, t)$ at a given query time t_q for a given set of N interpolation points $\mathcal{X} = \{\boldsymbol{x}_j : j \in \{0, \cdots, N-1\}\}$. We assume the field values are given at a set of m discrete points in time $\{t_{k-m/2+1}, \cdots, t_{k+m/2}\}$. The field values at each of the time snapshots are represented with a piecewise Chebyshev octree data structure $\{\mathcal{T}^{t_{k-m/2+1}}, \cdots, \mathcal{T}^{t_{k+m/2}}\}$.

To interpolate the tree values at time $\{t_q : t_k < t_q < t_{k+1}\}$ at positions $\boldsymbol{x}_j \in \mathcal{X}$ we first evaluate each of the trees $\mathcal{T}^i$ and $i \in \{t_{k-m/2+1}, \cdots, t_{k+m/2}\}$ at all the evaluation points in $\mathcal{X}$ by using our parallel tree evaluation procedure described in Algorithm 2. In this way, we construct m interpolation data points $\{\boldsymbol{v}^{t_{k-m/2+1}}(\boldsymbol{x}_j), \cdots, \boldsymbol{v}^{t_{k+m/2}}(\boldsymbol{x}_j)\}$ for each evaluation point $\boldsymbol{x}_j$. By using these interpolation data points and a $(m-1)$-order interpolation scheme, we approximate the field values at time t_q. This procedure is given in Algorithm 5.

In our semi-Lagrangian implementation, we use the above stated algorithm to interpolate the time-dependent velocity values at off-grid points in a space-time mesh. We use cubic interpolation, which requires 4 snapshots of the velocity values $\{\mathcal{T}_{\boldsymbol{v}}^{t_{k-1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+2}}\}$. We evaluate the trees at evaluation points and construct the 4 interpolation data points $\{\boldsymbol{v}^{t_{k-1}}(\boldsymbol{x}_j), \cdots, \boldsymbol{v}^{t_{k+2}}(\boldsymbol{x}_j)\}$, which we use to compute the cubic interpolation at time $\{t_q : t_k < t_q < t_{k+1}\}$. This procedure is depicted in Figure 15. We have also vectorized the cubic interpolation by using AVX vector intrinsics such that we interpolate four evaluation points together.

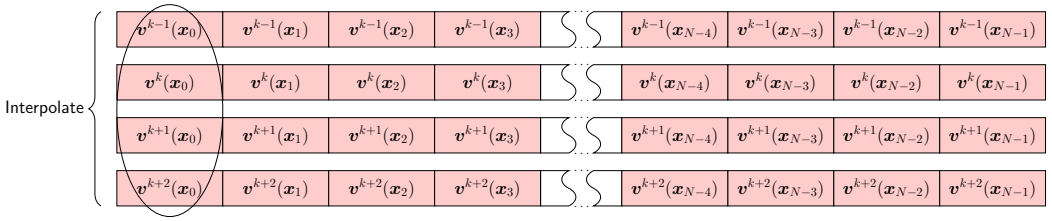


Figure 15: *Temporal interpolation of a time-dependent velocity field represented with piecewise Chebyshev Octree data structures at multiple discrete points in time. We construct interpolation data points* $\{\boldsymbol{v}^{t_{k-1}}(\boldsymbol{x}_j), \cdots, \boldsymbol{v}^{t_{k+2}}(\boldsymbol{x}_j)\}$ *by evaluating the velocity snapshots* $\{\mathcal{T}_{\boldsymbol{v}}^{t_{k-1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+2}}\}$ *at* N *evaluation points* $\boldsymbol{x}_j$.

By replacing the parallel tree evaluation in our trajectory computation algorithm with the parallel tree interpolation algorithm stated above, we extend our transport solver for unsteady velocity fields. However, this method is only applicable to problems where the velocity field is decoupled form the concentration field and its values are available in $m/2$ time-steps in advance, where m is our desired interpolation order.

Algorithm 5 Temporal Interpolation for Unsteady Fields.

Input:

$\{\mathcal{T}_{\boldsymbol{v}}^{t_{k-m/2+1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+m/2}}\}$: velocity trees at multiple discrete time snapshots

$\mathcal{X}, t_q$: queried space and time

Output:

$\mathcal{V}$: interpolated values at time t_q and position $\mathcal{X}$

1: **procedure** InterpolateParallelTrees($\{\mathcal{T}_{\boldsymbol{v}}^{t_{k-m/2+1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+m/2}}\}, \mathcal{X}, t_q$)
2: **for** $i \in \{t_{k-m/2+1}, \cdots, t_{k+m/2}\}$ **do**
3: $m[i] \leftarrow$ EvaluateParallelTree$(\mathcal{T}_{\boldsymbol{v}}^{i}, \mathcal{X})$ ▷ Algorithm 2
4: **end for**
5: $\mathcal{V} \leftarrow$ Interpolate(m, t_q)
6: **end procedure**

4.5.2 *Temporal Extrapolation Scheme*

In the previous section, the velocity at time $t_k + \delta t/2$ was computed by interpolating the velocity values using data points at times $\{t_{k-m/2+1}, \cdots, t_{k+m/2}\}$ where m is a positive integer number and depends on the order of the interpolation scheme. However, for the equations where the semi-Lagrangian method is used to compute the velocity itself (such as the Navier–Stokes equations in Chapter 7), the trajectory computing scheme described in Section 4.5.1 can not be applied because the velocity field is not given in advance at any future time-steps. For this category of problems, we apply a modified version of the explicit second order midpoint rule described in [102]:

$$\hat{\boldsymbol{x}} = \boldsymbol{x}_{\text{grid}} - \frac{\delta t}{2}\boldsymbol{v}(\boldsymbol{x}_{\text{grid}}, t_k), \tag{42}$$

$$\boldsymbol{x}_d = \boldsymbol{x}_{\text{grid}} - \delta t \boldsymbol{v}(\hat{\boldsymbol{x}}, t_k + \delta t/2). \tag{43}$$

This differs slightly from the classical Runge-Kutta method (Equations (40) and (41)) because the first stage (Equation (42)) uses the velocity $\boldsymbol{v}(\boldsymbol{x}_{\text{grid}}, t_k)$

rather than $\boldsymbol{v}(\boldsymbol{x}_{grid}, t_{k+1})$. In the second stage (Equation (43)) the velocity at $t_k + \delta t/2$ is approximated using the second-order extrapolation:

$$\boldsymbol{v}_{k+\frac{1}{2}} = \frac{3}{2}\boldsymbol{v}_k - \frac{1}{2}\boldsymbol{v}_{k-1}. \tag{44}$$

In the next section, we deploy both interpolation and extrapolation schemes to compute the trajectory in our semi-Lagrangian implementation and compare the convergence and stability features of both methods for complex unsteady velocity fields like *Taylor-Green vortex* flow.

4.6 NUMERICAL RESULTS

We conduct numerical experiments to demonstrate the correctness of our semi-Lagrangian advection solver discussed in this chapter. All experiments were carried out on the Stampede system at the Texas Advanced Computing Center. In Section 4.6.1, we consider the steady Taylor-Green vortex velocity field to demonstrate the convergence of the advection solver. We conduct the experiment for both uniform and adaptive tree data structures and also report the single-node time-to-solution results to compare the performance results of the AMR scheme. In Sections 4.6.2 and 4.6.3, we study the convergence behavior of our solver for simple and complex velocity fields. Moreover, for unsteady velocity fields, we compare the error of the scheme for both temporal interpolation and extrapolation methods.

4.6.1 *Convergence Study: Steady Taylor-Green Vortex Flow*

To demonstrate the convergence of the advection solver, we consider the steady Taylor-Green vortex velocity (see Figure 16):

$$\boldsymbol{v}(x, y, z, t) = \begin{pmatrix} \cos(2\pi x)\sin(2\pi y)\sin(2\pi z) \\ \sin(2\pi x)\cos(2\pi y)\sin(2\pi z) \\ \sin(2\pi x)\sin(2\pi y)\cos(2\pi z) \end{pmatrix}. \tag{45}$$

A Gaussian function is used for the initial concentration field. Since no analytical solution for this problem is available, to compute the error, we advect the concentration field for the second half of the time horizon with the reversed velocity field and compare our solution with the initial condition. We present convergence results for this test case for uniform and adaptive trees in Table 2. For the uniform case, we increase the depth of the tree L and the temporal resolution and report the error for a fixed time-horizon ($T = 1.0$). For the case of the adaptive tree we control the error by reducing the tree refinement tolerance ϵ_{tree}. We present results for two different discretization orders ($q = 8, 14$). In each case we conduct experiments to show convergence as we reduce the time-step size δt while keeping the time horizon for

the simulation fixed. We report the relative L^∞ and L^2 norm of the error at $t = T$. The refinement tolerance for the Chebyshev tree is chosen experimentally to minimize the number of unknowns N_{dof} while keeping the final solution error unchanged. We also report the total time to solution T_{solve}.

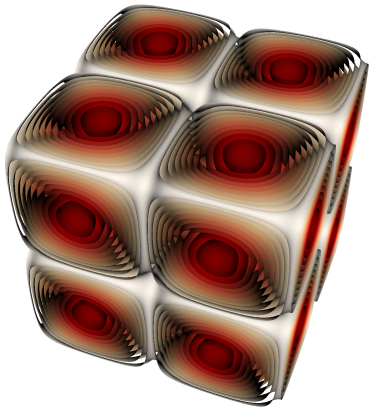


Figure 16: *Visualization of the steady Taylor-Green velocity field.*

We show the results for different time-step sizes. Starting with $\delta t = 0.01$ and 100 time steps, we achieve about two digits of accuracy for the uniform tree and three digits of accuracy for the adaptive tree with moderate discretization of order $q = 8$. For high discretization of order $q = 14$ with the same setup, we achieve better accuracy by up to one order of magnitude. In the next step, we reduce the time-step size by $2\times$ to $\delta t = 5\text{E-}3$ (200 time steps) while decreasing the spatial tolerance. For the uniform case we increase the maximum level of the tree by one and for the adaptive case we reduce the tree tolerance by a factor of 10. As the temporal and spatial resolution is increased in this way, we observe in the solution with moderate discretization for both the uniform and adaptive trees that the L^∞ error drops approximately by a factor of $28\times$. For high order discretization, we observe a factor of $164\times$ reduction in error for the uniform tree and a factor of $26\times$ reduction for the adaptive tree. We achieve 7 digits of accuracy, as we reduce the time-step size to $\delta t = 2.5\text{E-}3$ (400 time steps). For the same accuracy, an adaptive mesh requires significantly fewer degrees-of-freedom and hence, has lower computation costs. For 7 digits of accuracy, using an adaptive mesh is about $10\times$ faster compared to a uniform mesh.

4.6.2 *Convergence Study: Unsteady Vorticiy Flow*

In this section, we show convergence results for an advection problem in a simple unsteady vorticity velocity field defined by $\boldsymbol{v}(x, y, z) = (1 + \sin(2\pi t))(-x\hat{\mathbf{i}} + y\hat{\mathbf{j}})$. The initial concentration is given by a Gaussian function placed at location $R_0 = (0.6, 0.5, 0.5)$ in a unit cube $\Omega = [0, 1]^3$ with variance $\sigma = (6\text{E-}2, 6\text{E-}2, 6\text{E-}2)$ and amplitude 1.0. The analytic solution for the concentration at a point r and time t is given by,

$$c(r, t) = \exp\left(-\frac{|r - R(t)|^2}{2\sigma^2}\right), \tag{46}$$

where

$$R(t) = |R|\left(\cos\left(\theta_0 + t - \frac{\cos(2\pi t) - 1}{2\pi}\right)\hat{\mathbf{i}} + \sin\left(\theta_0 + t - \frac{\cos(2\pi t) - 1}{2\pi}\right)\hat{\mathbf{j}}\right).$$

Table 2: *Convergence of the advection solver for steady Taylor-Green vortex flow with the Gaussian function as the concentration field with uniform and adaptive trees. We present results for two different discretization orders ($q = 8, 14$). In each case, we conduct experiments to show convergence as we reduce the time-step size δt while keeping the time horizon for the simulation fixed. T_{solve} is the overall time in seconds at a single node. L is the maximum level of the tree for the uniform case. The tree refinement tolerance for the adaptive trees is denoted by ϵ_{tree}. We observe that for 7 digits of accuracy, using an adaptive mesh is about $10\times$ faster compared to a uniform mesh.*

q	δt	N_{iter}	Uniform Tree				Adaptive Tree			
			L	L^2	L^∞	T_{solve}	ϵ_{tree}	L^2	L^∞	T_{solve}
8	1.0E-2	100	3	3.5E-2	2.5E-2	7.79	1E-3	3.2eE-3	2.7E-3	12.83
8	5.0E-3	200	4	1.3E-3	8.7E-4	93.78	1E-4	2.9eE-4	9.9E-5	57.03
14	1.0E-2	100	3	1.7E-3	1.3E-3	38.52	1E-3	1.7eE-3	6.0E-4	20.49
14	5.0E-3	200	4	7.5E-6	7.9E-6	525.48	1E-5	2.6eE-5	2.3E-5	133.88
14	2.5E-3	400	5	1.6E-7	2.8E-7	8059.58	1E-7	3.2eE-7	2.0E-7	809.80

In Table 3, we show the convergence results for a fixed time horizon $T = [0, 1.0]$. We use a high order discretization $q = 14$ and we set the tree refinement tolerance to $\epsilon_{tree} = 1E\text{-}9$. By using high-order accurate spatial discretization and keeping the spatial accuracy fixed for all runs, we ensure that our time integrator is the main source of error. We report the results for two time integrators for unsteady fields, namely, the integrator using the interpolation scheme and the one using the extrapolation scheme.

Starting with $\delta t = 2.5E\text{-}1$ and 4 time-steps, we achieve two digits of accuracy with the interpolation time integrator, while the extrapolation time integrator delivers only one digit of accuracy. We observe a $4\times$ reduction in error as we continue to reduce the time-step size by a factor of 2 for both interpolation and extrapolation schemes. This confirms second-order convergence in time. We also observe that for higher temporal resolution and simple velocity fields such as the vorticity field in this example, the difference in error for the interpolation and extrapolation time integrators is negligible and the errors of both schemes are of the same order.

In Table 9, we study the convergence results of our solver for the same setup described above, however, we conduct the experiment with a longer time horizon $T = [0, 2\pi]$ and as we reduce the time-step size by a factor of 2, we also reduce the tree error tolerance by a factor of 10. This is essential due to the convergence behavior of the semi-Lagrangian schemes (see Section 2.1.2), since the upper bound of the L^∞ error grows proportionally to the number of time-steps, while its magnitude depends on the spatial error. We report the results for low and high order discretization ($q = 3, 14$).

Table 3: *Convergence study of advection solver for unsteady vorticity field for a fixed time horizon* $T = [0, 1.0]$*. By using accurate spatial discretization (*$q = 14, \epsilon_{tree} =$ 1E-9*), we ensure that our time integrator is the main source of error. By reducing the time-step size by a factor of 2, we observe a second-order convergence in time. We also report the results for both interpolation and extrapolation time integrators. We observe that both schemes result in temporal error of the same order for simple velocity fields.*

q	δt	N_{iter}	L^∞	
			Interpolation	**Extrapolation**
14	2.50E-1	4	3.24E-2	1.22E-1
	1.25E-1	8	8.54E-3	1.87E-2
	6.25E-2	16	1.92E-3	2.93E-3
	3.12E-2	32	4.42E-4	5.46E-4
	1.56E-2	64	1.06E-4	1.18E-4
	7.81E-3	128	2.61E-5	2.71E-5
	3.91E-3	256	6.47E-6	6.60E-6
	1.95E-3	512	1.61E-6	1.63E-6

For low order discretization, starting with a $\epsilon_{tree} =$ 1E-2 and a time resolution $\delta t =$ 6.28E-2 (100 time-steps), both the interpolation and extrapolation schemes achieve two digits of accuracy. As we increase the temporal and spatial resolution, the extrapolation scheme error reduces by a factor of $4\times$. The error reduction in interpolation scheme varies between factor of $8\times$ and $4\times$.

For high order discretization, we start with a temporal resolution $\delta t =$ 1,57E-2 (400 time-steps). As we decrease the time-step size to 3,92E-3 (1600 time-steps) and the tree error tolerance to $\epsilon_{tree} =$ 1E-4, we observe for both interpolation and extrapolation schemes a $16\times$ drop in L^∞ error, which corresponds to second-order convergence.

4.6.3 *Convergence Study: Unsteady Taylor-Green Vortex Flow*

In this section, we study the convergence behavior of our solver for complex unsteady velocity fields, in particular, a cosinusoidal time-dependent Taylor-Green vortex velocity field. For the concentration field, we use a Gaussian function placed at $R_0 = (0.6, 0.5, 0.5)$ in a unit cube $\Omega = [0, 1]^3$

Table 4: *Convergence study of advection solver for unsteady vorticity field with varying temporal and spatial resolution. As we reduce the time-step size by a factor of 2, due to the convergence behavior of Semi-Lagrangian schemes, we also reduce the tree error tolerance by a factor of* 10*. We conduct the experiment for both low and high order discretization (*$q = 3, 14$*) for a time horizon* $T = [0, 2\pi]$*. For both choices of discretization order and unsteady time integrators (interpolation and extrapolation), we observe a second-order convergence.*

q	ϵ_{tree}	L	δt	N_{iter}	L^∞	
					Interpolation	**Extrapolation**
3	1E-2	5	6.28E-2	100	3.08E-2	3.32E-2
	1E-3	7	3.14E-2	200	3.82E-3	7.18E-3
	1E-4	8	1.57E-2	400	8.23E-4	1.57E-3
14	1E-2	3	1.57E-2	400	7.83E-4	1.57E-3
	1E-3	3	7.85E-3	800	1.99E-4	3.83E-4
	1E-4	4	3.92E-3	1600	5.02E-5	9.60E-5

with variance $\sigma = (6E\text{-}2, 6E\text{-}2, 6E\text{-}2)$ and amplitude 1.0. We place this Gaussian function in the time-dependent *Taylor-Green* vortex flow:

$$\boldsymbol{v}(x, y, z, t) = \begin{pmatrix} \cos(2\pi x)\sin(2\pi y)\sin(2\pi z) \\ \sin(2\pi x)\cos(2\pi y)\sin(2\pi z) \\ \sin(2\pi x)\sin(2\pi y)\cos(2\pi z) \end{pmatrix} \cos(\omega t). \tag{47}$$

No analytical solution exists for this problem, however, due to cosinusoidal nature of the velocity field, the transporting substance oscillates around its initial location. By exploiting this fact, we advect the concentration field for the time horizon $T = [0, 2\pi/\omega]$ and to compute the error, we compare the results with the initial condition of the concentration field.

In Table 5, we show the temporal convergence of the scheme by decreasing the time-step size by a factor of 2 while we keep the spatial accuracy fixed ($q = 14, \epsilon_{tree} = 1E\text{-}5$). Depending on the temporal resolution, we observe an $8\times$ to $4\times$ reduction in the error, as we continue to reduce the time-step size. Notice that for this complex velocity field, the extrapolation scheme results in order of magnitude higher error for the same setup as the interpolation scheme.

Table 5: *Convergence study of the advection solver for unsteady cosinusoidal Taylor-Green velocity field for a fixed time horizon* $T = [0, 1.0]$*. We use a fixed spatial accuracy* ($q = 14, \epsilon_{tree} = 1E\text{-}5$) *and reduce the time-step size by a factor of 2. Depending on the temporal resolution, we observe an* $8\times$ *to* $4\times$ *reduction in the error as we continue to reduce the time-step size. We also observe that for this complex velocity field, the extrapolation scheme results in order of magnitude higher error for the same setup as the interpolation scheme.*

q	δt	N_{iter}	L^∞	
			Interpolation	**Extrapolation**
14	2.50E-1	4	1.66E-1	9.38E-1
	1.25E-1	8	9.35E-3	1.11E-1
	6.25E-2	16	1.11E-3	1.44E-2
	3.12E-2	32	1.38E-4	1.83E-3
	1.56E-2	64	1.70E-5	2.30E-4
	7.81E-3	128	4.28E-6	2.87E-5
	3.91E-3	256	6.07E-6	6.80E-6

4.7 SUMMARY

In this chapter we introduced a semi-Lagrangian advection solver with an aribitary order accurate piecewise Chebyshev octree spatial discretization. We explained the algorithms we developed to extend the scheme to parallel octrees, where we use Morton IDs to determine the location of the Lagrangian particles in a dynamic adaptive hierarchical domain decomposition. In our implementation, we allow for different parallel adaptive trees for the velocity and concentration fields and propose a novel partitioning scheme for parallel trees to define an upper-bound for inter-node communication cost in distributed-memory semi-Lagrangian schemes without increasing the computational cost. We also discussed our dynamic AMR algorithms and showed that by using AMR the number of unknowns for a computation with a fixed target accuracy can be reduced by orders of magnitude compared to the same computation using a uniform grid. To extend our scheme to unsteady velocity fields, we introduced two interpolation/extrapolation-based methods for off-grid temporal evaluation of unsteady velocity values represented with tree-based data structure at discrete points in time. Finally, we study the convergence of our scheme in various scenarios for complex steady and unsteady velocity fields.

Part II

APPLICATIONS

In this part we present how to apply our novel Semi-Lagrangian/ Volume-Integral methodology to three well-known problems in CFD, the diffusion equation, the advection-diffusion equation and the incompresible Navier–Stokes equations.

As the first application of our methodology we solve the unsteady diffusion equation in Chapter 5, where we validate the volume integral method as an attractive alternative to common PDE-based approaches.

The advection-diffusion equation is considered in Chapter 6. In this chapter we introduce three different temporal discretization schemes for the advection-diffusion equation and study the stability, convergence, single-node performance, strong and weak scalability of the scheme for various challenging examples.

Finally, in Chapter 7 we extend our Semi-Lagrangian/Volume-Integral method to the incompresible Navier–Stokes equations.

5

THE DIFFUSION EQUATION

In this chapter, as the first application for our methodology, we solve the parabolic diffusion equation by applying the volume integral method described in Chapter 2. Our goal is to validate the arbitrary-order accurate volume integral solver introduced in Chapter 3 as an accurate and fast alternative to common PDE-based solvers.

The diffusion equation is a PDE describing the concentration fluctuations due to the diffusion process, which states that regions of high concentration tend to spread into regions of low concentration. The diffusion equation can be obtained from the *continuity* equation:

$$\frac{\partial c}{\partial t} + \boldsymbol{\nabla} \cdot \mathbf{j} = 0,$$

where $\mathbf{j}$ is the flux of diffusive material. The continuity equation states that a change in concentration c in any closed system is due to outflow and inflow of concentration (flux) in the system. *Fick's first law* states that the flux of the material is proportional to the gradient of the local concentration:

$$\mathbf{j} = -\mathcal{D}(c, \boldsymbol{x}) \boldsymbol{\nabla} c(\boldsymbol{x}, t),$$

where $\mathcal{D}(c, \boldsymbol{x})$ is the collective diffusion coefficient for concentration c at location $\boldsymbol{x}$. The diffusion equation can be easily obtained from the continuity equation combined with Fick's first law such that the general diffusion equation can be written as:

$$\frac{\partial c(\boldsymbol{x}, t)}{\partial t} - \boldsymbol{\nabla} \cdot (\mathcal{D}(c(\boldsymbol{x}, t), \boldsymbol{x}) \boldsymbol{\nabla} c(\boldsymbol{x}, t)) = 0, \quad \boldsymbol{x} \in \Omega. \tag{48}$$

In homogeneous materials $\mathcal{D}$ does not vary in space. So when $\mathcal{D}$ is independent of the concentration, *e. g.* it is constant, the diffusion equation reduces to the following linear equation (also known as the *heat equation*):

$$\frac{\partial c(\boldsymbol{x}, t)}{\partial t} = \mathcal{D} \Delta c(\boldsymbol{x}, t), \ \boldsymbol{x} \in \Omega. \tag{49}$$

In this chapter we propose fast algorithms for solving the linear diffusion problem (Equation (49)) for the *concentration* $c(\boldsymbol{x}, t)$ with initial condition $c(\boldsymbol{x}, 0) = c_0(\boldsymbol{x})$ and with either free-space or periodic boundary conditions at the boundary $\partial\Omega$ of $\Omega = [0, 1]^3$.

There are many time integration methods available for systems of Ordinary Differential Equations (ODEs) and PDEs including Runge-Kutta methods and linear multistep methods [45]. Since our goal in this chapter is to

demonstrate the applicability of our volume integral formulation, we use only a first-order time integrator, namely the backward Euler method. In Chapter 6, we apply higher order temporal methods such as *Strang splitting* and *Stiffly-Stable* methods as the temporal discretization of the advection-diffusion equation. Applying the first-order backward Eular temporal discretization on Equation (49) leads to a steady elliptic problem, the modified Poisson equation. What remains to be done is to solve the modified Poisson equation in each time-step. We deploy the volume integral formulation described in Section 2.2 to solve the steady elliptic modified Poisson equation, where we solve the convolution integral by using the PVFMM library (see Chapter 3). In Section 5.2, we conduct numerical experiments to show the convergence of the scheme.

5.1 THE FIRST-ORDER BACKWARD EULER TEMPORAL INTEGRATION

Given the concentration field $c(\mathbf{x}, t_k)$, we would like to compute the concentration values $c(\mathbf{x}, t_{k+1})$, where $t_{k+1} = t_k + \delta t$. By using the backward Euler method as the time integrator, we discretize the unsteady diffusion problem described in Equation (49) in time as follows:

$$\frac{c(\mathbf{x}, t_{k+1}) - c(\mathbf{x}, t_k)}{\delta t} - \mathcal{D}\Delta c(\mathbf{x}, t_{k+1}) = 0. \tag{50}$$

We transform Equation (50) by bringing the known values to the right-hand side and dividing the both sides by the constant diffusion coefficient $\mathcal{D}$

$$\frac{1}{\mathcal{D}\delta t} c(\mathbf{x}, t_{k+1}) - \Delta c(\mathbf{x}, t_{k+1}) = \frac{1}{\mathcal{D}\delta t} c(\mathbf{x}, t_k). \tag{51}$$

The resulting equation corresponds to a modified Poisson equation (see Section 2.2.3)

$$\alpha c_{k+1} - \Delta c_{k+1} = f, \tag{52}$$

with $\alpha = 1/(\mathcal{D}\delta t)$ and $f = \alpha c_k$. Here, we have suppressed the explicit notation in $\mathbf{x}$ and t (compare with Equation (51)). To solve the Equation (52) for c_{k+1}, we use the volume integral formulation described in Section 2.2. In this formulation, the solution of the modified Poisson problem is given as a convolution of the right-hand side $f = \alpha c_k$ with the fundamental solution of the modified Poisson PDE

$$c_{k+1}(\mathbf{x}) = \int_y \frac{e^{-\lambda|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|} f(\mathbf{y}) d\mathbf{y} = \int_y \frac{1}{4\pi\mathcal{D}\delta t} \frac{e^{-\lambda|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|} c_k(\mathbf{y}) d\mathbf{y}, \tag{53}$$

where $\lambda = \sqrt{1/(\mathcal{D}\delta t)}$. Note that in this formulation, the convolution kernel function depends on the temporal resolution δt.

The process of applying the volume integral method to the diffusion equation with the temporal discretization given in Equation (51) can thus be

summarized as follows: To step forward from the computed approximation c_k at times t_k to new approximation at c_{k+1} at the forward time t_{k+1}, we first construct the right-hand side of the modified Poisson by multiplying c_k with the factor $1/(\mathcal{D}\delta t)$. Then, we solve the modified Poisson equation by computing the integral in Equation (53). In our implementation, the concentration value c_k is approximated with a parallel piecewise Chebyshev octree data structure described in Sections 3.3.1 and 4.2.1. To compute the convolution in Equation (53), we use the PVFMM library which is a KIFMM-based volume integral solver (see Chapter 3). The PVFMM library takes the modified c_k tree as an input and updates the tree values to c_{k+1} by computing the FMM with the modified Poisson kernel. The machinery of the PVFMM library is comprehensively described in Chapter 3. The procedure of solving the unsteady diffusion equation by using the first-order backward Euler method for temporal discretization and applying the volume integral method is given in Algorithm 6.

Algorithm 6 Solving the diffusion equation by using the backward Euler method and the FMM-based volume integral solver.

Input:
$\mathcal{T}_c^{t_k}$: Concentration tree at time t_k.
Output:
$\mathcal{T}_c^{t_{k+1}}$: Concentration tree at time t_{k+1}.

1: **procedure** SOLVEDIFFUSION($\mathcal{T}_c^{t_k}$)
2: $\mathcal{N} \leftarrow$ LEAFNODES($\mathcal{T}_c^{t_k}$)
3: **for each** $\mathcal{B} \in \mathcal{N}$ **do**
4: $\alpha^{\mathcal{B}} \leftarrow M^{-1}(\alpha_k^{\mathcal{B}}/(\mathcal{D}\delta t))$
5: **end for**
6: $\mathcal{T}_c^{t_{k+1}} \leftarrow$ RUNFMM($\mathcal{T}_c^{t_k}$) ▷ FMM with modified Poisson kernel
7: **end procedure**

5.2 NUMERICAL RESULTS

In this section, we show convergence results for a diffusion problem with a Gaussian function as the initial concentration. The Gaussian function is placed at location $R_0 = (0.5, 0.5, 0.5)$ in a unit cube $\Omega = [0, 1]^3$. The analytic solution for the concentration at a point r and time t is given by,

$$c(r, t) = \frac{A}{\sigma(t)^3} \exp\left(-\frac{|r - R_0)|^2}{2\sigma^2(t)}\right), \tag{54}$$

where $\sigma(t) = \sqrt{\sigma_0^2 + 2\mathcal{D}t}$. Table 6 gives the temporal convergence results for a sequence of decreasing time-step size δt for a fixed time horizon $T = [0, 1.0]$. To render the spatial error negligible, we use a high order dis-

cretization $q = 14$ and we set the tree refinement tolerance to $\epsilon_{tree} = 1E\text{-}5$ for all the runs. We present the relative L^2 and L^∞ error for three values of $\mathtt{diffusivity}$ ($\mathcal{D} = 1E\text{-}3, 1E\text{-}4, 1E\text{-}5$) as we increase the temporal resolution.

For all three diffusivity values, by increasing the temporal resolution by a factor of $2\times$, we observe a $2\times$ drop in both L^2 and L^∞ errors, which confirms the expected first-order temporal convergence with the backward Euler time discretization.

Table 6: *Convergence of the diffusion solver for a diffusive Gaussian function with three different diffusivity constants ($\mathcal{D} = 1E\text{-}3, 1E\text{-}4, 1E\text{-}5$). We present results for high order discretization($q = 14$). In each case, we conduct experiments to show convergence as we reduce the time-step size δt while keeping the time horizon for the simulation fixed ($T = [0, 1.0]$). The tree refinement tolerance for the adaptive trees is fixed to $\epsilon_{tree} = 1E\text{-}5$. We observe the expected first-order convergence for all three diffusivity constants.*

q	δt	N_{iter}	$\mathcal{D} = 1E\text{-}3$		$\mathcal{D} = 1E\text{-}4$		$\mathcal{D} = 1E\text{-}5$	
			L^2	L^∞	L^2	L^∞	L^2	L^∞
14	1.00E-0	1	6.77E-2	1.15E-1	1.33E-3	2.56E-3	1.83E-5	$2.95e-05$
	5.00E-1	2	3.85E-2	6.54E-2	6.87E-4	1.33E-3	9.21E-6	$1.48e-05$
	2.50E-1	4	2.07E-2	3.52E-2	3.49E-4	6.79E-4	4.62E-6	$7.42e-06$
	1.25E-1	8	1.08E-2	1.83E-2	1.76E-4	3.43E-4	2.32E-6	$3.71e-06$
	6.25E-2	16	5.49E-3	9.36E-3	8.84E-5	1.72E-4	1.19E-6	$1.86e-06$
	3.12E-2	32	2.78E-3	4.73E-3	4.43E-5	8.64E-5	6.42E-7	$9.29e-07$
	1.56E-2	64	1.40E-3	2.38E-3	2.21E-5	4.32E-5	3.94E-7	$4.65e-07$
	7.81E-3	128	7.00E-4	1.19E-3	1.10E-5	2.16E-5	2.82E-7	$2.32e-07$

The convergence results in Table 6 demonstrate the effectiveness of our approach to apply the volume integral formulation to elliptic problems. In the next chapter, we extend our method to the advection-diffusion equation, where we treat the advective term with an explicit semi-Lagrangian scheme and the parabolic diffusion term with an implicit volume integral formulation similar to the one deployed in this chapter.

6

THE ADVECTION-DIFFUSION EQUATION

In this chapter we consider the advection-diffusion problem as the second application for our methodology. The advection-diffusion equation describes transport phenomena where a physical quantity such as energy, substances or particles transport in a flowing medium such as water or air. The applications include porous media flows simulation [81], transport phenomena in complex fluids [56], and multiphysics simulations [3], [24]. The general equation describing the advection-diffusion is:

$$\frac{\partial c(\mathbf{x},t)}{\partial t} + \boldsymbol{\nabla}\cdot(\boldsymbol{v}(\mathbf{x},t)c(\mathbf{x},t)) - \boldsymbol{\nabla}\cdot(\mathcal{D}(c(\mathbf{x},t),\mathbf{x})\boldsymbol{\nabla}c(\mathbf{x},t)) = 0, \tag{55}$$

where $c(\mathbf{x},t)$ is the quantity of interest (e. g. concentration for mass or heat transfer). $\boldsymbol{v}(\mathbf{x},t)$ is a given time-dependent velocity field and $\mathcal{D}(c,\mathbf{x})$ is the collective diffusion coefficient.

For common situations where the diffusion coefficient is constant and the velocity field describes an incompressible flow (i. e. $\boldsymbol{\nabla}\cdot\boldsymbol{v} = 0$), the Equation (55) simplifies to:

$$\frac{\partial c(\mathbf{x},t)}{\partial t} + \boldsymbol{v}(\mathbf{x},t)\cdot\boldsymbol{\nabla}c(\mathbf{x},t) - \mathcal{D}\Delta c(\mathbf{x},t) = 0. \tag{56}$$

Developing algorithms for solving the Equation (56) with initial condition $c(\mathbf{x},0) = c_0(\mathbf{x})$ and with either free-space or periodic boundary conditions at the boundary of a unit cube is the subject of this chapter. In particular, we would like to develop a fast, unconditionally stable, parallel, scalable and high-order advection-diffusion solver. For this purpose we exploit the semi-Lagrangian technology described in Chapter 4 and the KIFMM-based volume integral solver discussed in Chapter 3.

In Section 6.1 we introduce three various time discretizations for the advection-diffusion problem starting with the *splitting* methods. The main idea of the splitting methods is to breakdown a complicated problem into smaller subproblems and solve each part with dedicated numerical methods. The splitting methods allows to solve complex PDEs by splitting the PDE into its simpler constituent differential operators and treat each differential operator individually using specialized numerical solver. This results in significant reduction of the algorithmic complexity of the overall method to evolve in time for the fully coupled problem. However, in addition to the numerical error of each subproblem, the splitting methods introduce the so called *splitting error*. In this chapter we study two different splitting methods: the first-order Lie and the second-order *symmetric Strang* splitting

method which we discuss in Sections 6.1.1 and 6.1.2, respectively. Moreover, in Section 6.1.3 we discuss the *stiffly-stable* method [35], which is a variation of second-order backward multistep method. In Section 6.2 we conduct various numerical experiments to show the convergence of our schemes. We study the single-node performance and the cost of the main components of the advection-diffusion solver for low and high order discretizations in Section 6.3. The isogranular and fixed-sized scaling results of the solver are discussed in Chapter 8. Finally, in Section 6.4 we simulate a realistic scenario, namley the transport of substance with stationary Stokes flow through a porous medium with highly complex pore structure. This has applications in many areas of science and engineering such as petroleum engineering, geosciences and material sciences.

6.1 TEMPORAL INTEGRATION METHODS

In this section our goal is to develop a stable, high-order accurate temporal discretization scheme for the advection-diffusion problem. We propose three different approaches: (1) first-order Lie operator splitting method, which we use later as a low order method to construct additional initial values for high-order multi-step methods (2) second-order symmetric Strang operator splitting method, and (3) the second-order multi-step stiffly-stable method.

In problems consisting of different components, one subproblem might be very stiff and require an implicit approach, while an explicit method might be much more suitable for the other subproblem of the PDE of interest. Using the same implicit method for the whole problem often can lead to a large, complex, nonlinear algebraic problem. Therefore it is in general infeasible to apply the same numerical integration approach to the different parts of the system. For such cases, the operator splitting method is more favorable because it allows to independently discretize and solve the subproblems. Moreover, different time-steps for different subproblems becomes feasible. As an example, splitting of the advection-diffusion problem has the advantage that the stiff diffusion operator can be treated implicitly while the advective term can be solved explicitly. However, splitting methods come with an additional numerical error, the *splitting error*. The splitting error hase been studied in various works for both ODEs [42] and PDEs [45]. For the advection-diffusion-reaction equation, the derivation of the splitting error is given in [55]. A high-order splitting method for the incompressible Navier-Stokes equations is discussed in [50]. In Section 6.1.1, we discuss the first-order *Lie* operator splitting method for the advection-diffusion problem. We briefly discuss the second-order *symmetric Strang* splitting method in Section 6.1.2. To obtain a higher-order temporal integration methods, in Section 6.1.3 we apply the second-order multi-step *stiffly-stable* method to the advection-diffusion problem.

6.1.1 *The First-Order Lie Operator Splitting Method*

We begin with the first-order Lie splitting approximation method, which is the simplest splitting method [94]. In this method we split Equation (56) into two operators

$$\frac{\partial c(\mathbf{x}, t)}{\partial t} + (\mathcal{L}_1 + \mathcal{L}_2)\, c(\mathbf{x}, t) = 0, \tag{57}$$

where $\mathcal{L}_1$ and $\mathcal{L}_2$ represent the advective term $\boldsymbol{v}(\mathbf{x}, t) \cdot \boldsymbol{\nabla}$ and the diffusive term $-\mathcal{D}\Delta$, respectively. To compute the solution of Equation (57) at time $t_k + \delta t$ from a given approximated solution c_k at time t_k, we first march in time from t_k to $t_k + \delta t$ by applying only the $\mathcal{L}_1$ operator. That is, we first solve the subproblem

$$\frac{\partial c(\mathbf{x}, t)}{\partial t} + \mathcal{L}_1 c(\mathbf{x}, t) = 0, \tag{58}$$

with $c(\mathbf{x}, t_k) = c_k$ as the initial condition. We denote the solution of the first subproblem by $\tilde{c}_{k+1}$. Once the $\tilde{c}_{k+1}$ is obtained, we solve the second subproblem

$$\frac{\partial c(\mathbf{x}, t)}{\partial t} + \mathcal{L}_2 c(\mathbf{x}, t) = 0, \tag{59}$$

from t_k to $t_k + \delta t$ and using $\tilde{c}_{k+1}$ as the initial condition instead of c_k. This gives the solution of the coupled advection-diffusion problem with overall first order temporal accuracy $\mathcal{O}(\delta t)$. In our implementation, we solve the first subproblem (the advective term) by using our explicit semi-Lagrangian advection solver introduced in Chapter 4 and the second subproblem (the diffusion term) with the implicit volume integral diffusion solver described in Chapter 5.

For our spatial discretization, we use the parallel, aribitary-order accurate, piecewise Chebyshev octree data structure (see Sections 3.3.1 and 4.2.1). The procedure of solving the advection-diffusion problem with this temporal and spatial discretization is given in Algorithm 7. The concentration tree at time t_k is denoted by $\mathcal{T}_c^{t_k}$. Starting with this concentration tree, we compute the semi-Lagrangian step for advection by using Algorithm 3. The updated concentration values $\tilde{\mathcal{T}}_c^{t_{k+1}}$ are plugged into Algorithm 6, which solves the diffusion equation by using a volume integral formulation.

6.1.2 *The Symmetric Strang Operator Splitting Method*

A second-order splitting approximation can be obtained by using symmetric Strang formulas [86]. Here, we briefly describe the method for a given PDE consisting of two operators $\mathcal{L}_1$ and $\mathcal{L}_2$. Given approximated value c_k at time t_k, to step forward to new approximation c_{k+1} at the forward time

Algorithm 7 Solving the advection-diffusion equation with Lie splitting method and Chebyshev octree data structures.

Input:
$\mathcal{T}_c^{t_k}$: Concentration tree at time t_k.
$\{\mathcal{T}_{\boldsymbol{v}}^{t_{k-m/2+1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+m/2}}\}$: Velocity trees at m discrete points in time.
Output:
$\mathcal{T}_c^{t_{k+1}}$: Concentration tree at time t_{k+1}.

1: **procedure** LIESPLITTING($\mathcal{T}_c^{t_k}, \{\mathcal{T}_{\boldsymbol{v}}^{t_{k-m/2+1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+m/2}}\}$)
2: $\quad\tilde{\mathcal{T}}_c^{t_{k+1}} \leftarrow$ SOLVESEMILAG($\mathcal{T}_c^{t_k}, \{\mathcal{T}_{\boldsymbol{v}}^{t_{k-m/2+1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+m/2}}\}$) ▷ Algorithm 3
3: $\quad\mathcal{T}_c^{t_{k+1}} \leftarrow$ SOLVEDIFFUSION($\tilde{\mathcal{T}}_c^{t_{k+1}}$) ▷ Algorithm 6
4: **end procedure**

$t_k + \delta t$, we first solve the first subproblem with the initial value c_k from time t_k to $t_k + \delta t/2$. That is, we solve

$$\frac{\partial c(\boldsymbol{x}, t)}{\partial t} + \mathcal{L}_1 c(\boldsymbol{x}, t) = 0, \quad c(\boldsymbol{x}, t_k) = c_k, \quad t_k \rightarrow t_{k+1/2}. \tag{60}$$

We denote the solution of Equation (60) by $\tilde{c}_{k+1/2}$. In the next step, we solve the second subproblem from t_k to $t_k + \delta t$ by using $\tilde{c}_{k+1/2}$ as the initial value:

$$\frac{\partial c(\boldsymbol{x}, t)}{\partial t} + \mathcal{L}_2 c(\boldsymbol{x}, t) = 0, \quad c(\boldsymbol{x}, t_k) = \tilde{c}_{k+1/2}, \quad t_k \rightarrow t_{k+1}. \tag{61}$$

We denote the solution of Equation (61) by $c_{k+1/2}$. As the final step, we solve again the first subproblem, however, we march in time from $t_k + \delta t/2$ to $t_k + \delta t$ and use $c_{k+1/2}$ as the initial value:

$$\frac{\partial c(\boldsymbol{x}, t)}{\partial t} + \mathcal{L}_1 c(\boldsymbol{x}, t) = 0, \quad c(\boldsymbol{x}, t_{k+1/2}) = c_{k+1/2}, \quad t_{k+1/2} \rightarrow t_{k+1}. \tag{62}$$

If time-stepping of at least second-order is used for all subproblems, this procedure results in a second-order time integration scheme to compute for c_{k+1}. Since our volume integral diffusion solver discussed in Chapter 5 is only first-order, we can not obtain a second-order time integrator scheme by using the procedure stated above. To overcome this issue, in the next section we introduce a second-order backward multistep scheme, which does not require additional technologies other than the ones we have introduced and implemented so far.

6.1.3 *The Second-Order Stiffly-Stable Method*

In this section, to obtain a second-order time integration method for the advection-diffusion problem, we employ a variation of the backward mul-

tistep scheme [35]. We consider the advection-diffusion equation in the Lagrangian form by using the material derivative (see Section 2.1) :

$$\frac{Dc(\mathbf{x},t)}{Dt} - \mathcal{D}\Delta c(\mathbf{x},t) = 0. \tag{63}$$

To discretize Equation (63), we follow the time-stepping scheme described in [102]. Given the concentrations $c(\mathbf{x},t_k)$ and $c(\mathbf{x},t_{k-1})$, we compute the concentration $c(\mathbf{x},t_{k+1})$, where $t_{k+1} = t_k + \delta t$. A second-order time discretization is

$$\frac{\frac{3}{2}c(\mathbf{x},t_{k+1}) - 2c(\mathbf{X}_k,t_k) + \frac{1}{2}c(\mathbf{X}_{k-1},t_{k-1})}{\delta t} - \mathcal{D}\Delta c(\mathbf{x},t_{k+1}) = 0, \tag{64}$$

where $\mathbf{X}_k$ and $\mathbf{X}_{k-1}$ are points in the trajectory of a (virtual) Lagrangian particle passively moving due to the velocity $\boldsymbol{v}$ and passing through $\mathbf{x}$ at time t_{k+1}. These positions are solved by the following equation (characteristics) backward in time

$$\frac{d\mathbf{X}(t)}{dt} = -\boldsymbol{v}(\mathbf{X}(t),t), \quad \mathbf{X}(0) = \mathbf{x}. \tag{65}$$

We define the *semi-Lagrangian departure points* by $\mathbf{X}_k = \mathbf{X}(\delta t)$ and $\mathbf{X}_{k-1} = \mathbf{X}(2\delta t)$ (see Figure 17). Equation (65) can be solved using an explicit time stepping scheme. In our implementation, we use a second-order Runge-Kutta scheme with the same δt as in Equation (64). This way, we construct

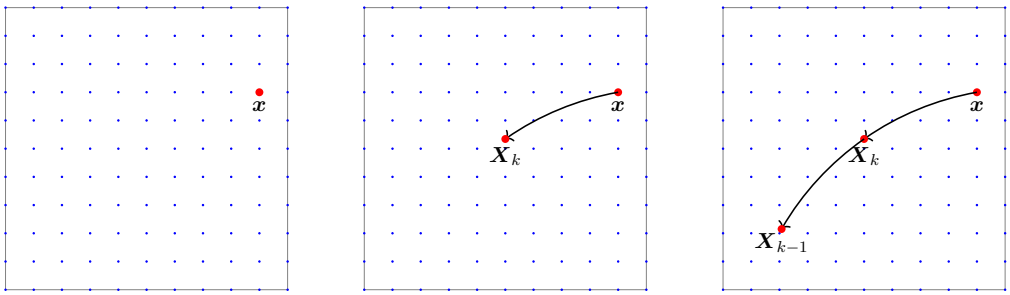


Figure 17: *Illustration of the departure points in stiffly-stable method. Starting from grid points $\mathbf{x}$ at time t_{k+1}, we compute the characteristics to obtain the departure points of the Lagrangian particles at time t_k and t_{k-1}, which are denoted by $\mathbf{X}_k$ and $\mathbf{X}_{k-1}$, respectively.*

an optimal upwind scheme for Equation (56) by numerically computing the backward characteristics. Solving Equation (65) requires interpolation for $\boldsymbol{v}(\mathbf{X}(t),t)$. Furthermore, once we have the semi-Lagrangian points, we need to interpolate $c(\cdot,t_k)$ and $c(\cdot,t_{k+1})$ at these points. This interpolation is critical for the performance of the scheme since it can introduce artificial diffusion and overshooting. The algorithm for the interpolation is discussed in Sections 4.1.2 and 4.2.2. Once we have the interpolated values, by bringing the known values of Equation (64) to the right-hand side, we obtain the

following equation (here, we have suppressed the explicit notation in $\mathbf{x}$ and t):

$$\frac{3}{2\delta t}c_{k+1} - \mathcal{D}\Delta c_{k+1} = \frac{2}{\delta t}c_k - \frac{1}{2\delta t}c_{k-1}. \tag{66}$$

This equation corresponds to the modified Poisson equation (see Section 2.2.3)

$$\alpha c_{k+1} - \Delta c_{k+1} = f, \tag{67}$$

with $\alpha = 3/(2\mathcal{D}\delta t)$ and $f = 2c_k/(\mathcal{D}\delta t) - c_{k-1}/(2\mathcal{D}\delta t)$. The solution of Equation (67) for c_{k+1} as a volume integral is

$$\begin{aligned} c_{k+1}(\mathbf{x}) &= \int_y \frac{e^{-\lambda|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|} f(\mathbf{y})\,dy \\ &= \int_y \frac{1}{4\pi\mathcal{D}\delta t}\frac{e^{-\lambda|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|}\left(2c_k(\mathbf{y}) - \frac{1}{2}c_{k-1}(\mathbf{y})\right) dy, \end{aligned} \tag{68}$$

where $\lambda = \sqrt{3/(2\mathcal{D}\delta t)}$. To solve this convolution integral, we use the PVFMM library described in Chapter 3. Note that in this formulation, similar to the volume integral formulation of the diffusion equation described in Section 5.1, the convolution kernel function depends on the temporal resolution δt. That is, for an adaptive time-stepping scheme, the convolution kernel function needs to be modified accordingly. We would like to mention that we have not implemented adaptive time-stepping in this thesis, but there is nothing in our formulation that prevents it. Adaptive time stepping would improve time-to-solution by reducing the number of time-steps for time-varying velocity fields. Since the stiffly-stable scheme belongs to the category of multistep methods, we keep and use the information from previous time-steps. In particular, to compute the solution at time t_{k+1}, the previous two time-steps of the concentration trees $\mathcal{T}_c^{t_{k-1}}$ and $\mathcal{T}_c^{t_k}$ needs to be stored for later use. With each additional previous time-step values that the multistep method requires, comes the need for an additional initial value which is not normally provided. In our implementation, the $\mathcal{T}_c^{t_0}$ is given analytically and we construct the $\mathcal{T}_c^{t_1}$ by using a lower order method like the first-order splitting method explained in Section 6.1.1.To construct additional initial values with low-order methods, we use a higher temporal resolution compared to our simulation time-step size δt. However, constructing the initial values with low-order methods, can be a potential additional source of numerical error. We don't have a mathematical stability investigation of this approach, only empirical evidence. Algorithm 8 shows the stiffly-stable method in the context of Chebyshev octree data structures. For unsteady velocity fields, we use the temporal interpolation scheme described in Section 4.5.1, which requires the velocity trees for m snapshots in

Algorithm 8 Solving the advection-diffusion equation with the stiffly-stable method and the Chebyshev octree data structure.

Input:
$\mathcal{T}_c^{t_k}, \mathcal{T}_c^{t_{k-1}}$: Concentration trees at time t_k and t_{k-1}.
$\{\mathcal{T}_{\boldsymbol{v}}^{t_{k-m/2+1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+m/2}}\}$: Velocity trees at m time steps.
Output:
$\mathcal{T}_c^{t_{k+1}}$: Concentration tree at time t_{k+1}.

procedure AdvDiffStifflyStable($\mathcal{T}_c^{t_{k-1}}, \mathcal{T}_c^{t_k}, \{\mathcal{T}_{\boldsymbol{v}}^{t_{k-m/2+1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+m/2}}\}$)
  $\mathcal{N} \leftarrow$ LeafNodes($\mathcal{T}_c^{t_{k-1}}$)
  for each $\mathcal{B} \in \mathcal{N}$ **do**
    $\mathbf{X}_{k+1} \leftarrow \mathbf{X}_{k+1} \cup$ InterpPoints($\mathcal{B}$)
  end for
  $\mathbf{X}_k \leftarrow$ Trajectory($\{\mathcal{T}_{\boldsymbol{v}}^{t_{k-m/2+1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+m/2}}\}, \mathbf{X}_{k+1}$)
  $c_k \leftarrow$ EvaluateParallelTree($\mathcal{T}_c^{t_k}, \mathbf{X}_k$) ▷ Algorithm 2
  $\mathbf{X}_{k-1} \leftarrow$ Trajectory($\{\mathcal{T}_{\boldsymbol{v}}^{t_{k-m/2+1}}, \cdots, \mathcal{T}_{\boldsymbol{v}}^{t_{k+m/2}}\}, \mathbf{X}_k$)
  $c_{k-1} \leftarrow$ EvaluateParallelTree($\mathcal{T}_c^{t_{k-1}}, \mathbf{X}_{k-1}$) ▷ Algorithm 2
  for each $\mathcal{B} \in \mathcal{N}$ **do**
    $\alpha^{\mathcal{B}} \leftarrow M^{-1}\left(\frac{2}{\mathcal{D}\delta t} c_k^{\mathcal{B}} - \frac{1}{2\mathcal{D}\delta t} c_{k-1}^{\mathcal{B}}\right)$
  end for
  $\mathcal{T}_c^{t_{k+1}} \leftarrow$ RunFMM($\mathcal{T}_c^{t_{k-1}}$) ▷ Modified Poisson kernel
  RefineTree($\mathcal{T}_c^{t_{k+1}}$) ▷ AMR algorithm in Section 4.4
end procedure

time from $t_{k-m/2+1}$ to $t_{k+m/2}$. In our implementation, we use a cubic interpolation in time ($m = 4$). Each process, by iterating over all leaf-octants of its local concentration tree of the previous time-step ($\mathcal{T}_c^{t_{k-1}}$), constructs an array of Chebyshev interpolation points. In Algorithm 8, these are denoted by $\mathbf{X}_{k+1}$. Using the local array of the interpolation points as the initial value, we compute the departure points at time t_k (denoted by $\mathbf{X}_k$). The trajectory computation with unsteady velocity fields, requires multiple parallel tree evaluation operations (Algorithm 5). Once the departure points $\mathbf{X}_k$ are determined, an array of the concentration values c_k at time t_k at the departure points, can be obtained by parallel evaluation of $\mathcal{T}_c^{t_k}$ at $\mathbf{X}_k$. To obtain c_{k-1}, we proceed with the same procedure explained above, however, our initial values for the trajectory computation will be departure points at time t_k. Once, the two arrays of c_k and c_{k-1} are available, we store the linear combination of these values in $\mathcal{T}_c^{t_{k-1}}$. By reusing the $\mathcal{T}_c^{t_{k-1}}$, we avoid constructing a new tree for the computed values in each time-step. In the next step, we compute the convolution integral with modified Poisson kernel and finally we apply our AMR algorithm described in Section 4.4 to adapt the tree refinement to the updated values.

6.2 NUMERICAL RESULTS

In this section we conduct various numerical experiments with both steady and unsteady velocity fields to demonstrate the convergence of our numerical scheme for the advection-diffusion problem.

In Section 6.2.1 we first consider a set of diffusive Gaussian functions in a steady vorticity velocity field with known analytical solution and study the convergence behavior of the our scheme. In 6.2.2, we study the temporal as well as spatial convergence behavior of our scheme for unsteady vorticity velocity fields.

6.2.1 *Convergence Study: Steady Vorticity Flow*

In this section we show convergence results for an advection-diffusion problem with a known analytical solution, which we solve by applying the stiffly-stable temporal discretization method. The initial concentration is given by a set of five Gaussian functions randomly placed at locations $r_i \in (0, 0.2)$ with variance $\sigma_i \in (1\text{E-}2, 3\text{E-}2)$ and amplitude $a_i \in (-0.5, 0.5)$. This concentration is placed in a vorticity velocity field defined by $\boldsymbol{\nu}(x, y, z) = -x\hat{\mathbf{i}} + y\hat{\mathbf{j}}$. For diffusivity $\mathcal{D}$, the analytic solution for the concentration at a point $r = x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}}$ and time t is given by

$$c(r, t) = \sum_i a_i \frac{\sigma_i^3(0)}{\sigma_i^3(t)} \exp\left(-\frac{|r - r_i(t)|^2}{2\sigma_i^2(t)}\right) \tag{69}$$

where, $r_i(t) = |r_i|\left(\cos(\theta_i + t)\hat{i} + \sin(\theta_i + t)\hat{j}\right)$, $\sigma_i(t) = \sqrt{\sigma_i^2 + 2\mathcal{D}t}$. In Table 7, we present convergence results for two different values of the diffusivity ($\mathcal{D} = 1\text{E-}4, 1\text{E-}5$) and varying discretization orders ($q = 4, 8, 14$). In each case, we conduct experiments to show convergence as we reduce the time-step size δt while keeping the time horizon for the simulation fixed at $T = 1.6$. We report the relative L^∞ norm of the error at $t = T$. The refinement tolerance for the Chebyshev tree is chosen experimentally to minimize the number of unknowns N_{dof} while keeping the final solution error unchanged. We also report the number of levels of refinement for the adaptive octree, the total time to solution T_{solve} and the overall flop-rate in GFLOPS. In addition, we report the breakdown of the time spent in different stages of the algorithm: tree refinement (T_{ref}), semi-Lagrangian advection ($T_{sort} + T_{eval}$), and diffusion computation using FMM (T_{fmm}). For the Chebyshev evaluation and FMM computation stages, we also report the flop-rates.

For each choice of diffusivity and discretization order, we show results for three different time-step sizes. Starting with $\delta t = 0.1$ and 16 time steps, we achieve about two digits of accuracy. As we reduce the time step size by $4\times$ to $\delta t = 2.5\text{E-}2$ (64 time steps), we observe that the L^∞ error in the solution

drops by 16×. We observe a further 16× drop in error as we reduce the time step size to $\delta t = 6.25\text{E-}3$ (256 time steps). This confirms the quadratic convergence with δt.

6.2.2 *Convergence Study: Unsteady Vorticiy Flow*

In this section we show convergence results for the same test case as in Section 4.6.2. However, we solve an advection-diffusion problem instead of only an advection equation and use the stiffly-stable temporal discretization. Recall that the unsteady vorticity velocity field is defined by $\boldsymbol{v}(x, y, z) = (1 + \sin(2\pi t))(-x\hat{\mathbf{i}} + y\hat{\mathbf{j}})$ and the initial concentration is given by a Gaussian function placed at location $R_0 = (0.6, 0.5, 0.5)$ in a unit cube $\Omega = [0, 1]^3$. The analytic solution for the concentration at a point r and time t is given by

$$c(r, t) = \frac{A}{\sigma(t)^3} \exp\left(-\frac{|r - R(t)|^2}{2\sigma^2(t)}\right), \tag{70}$$
$$R(t) = |R|\left(\cos(\theta_0 + t - \frac{\cos(2\pi t) - 1}{2\pi})\hat{\mathbf{i}} + \sin(\theta_0 + t - \frac{\cos(2\pi t) - 1}{2\pi})\hat{\mathbf{j}}\right).$$

where $\sigma(t) = \sqrt{\sigma_0^2 + 2\mathcal{D}t}$. In Table 8, we show convergence results for a fixed time horizon $T = [0, 1.0]$. We use a high order discretization $q = 14$ and we set the tree refinement tolerance to $\epsilon_{tree} = 1\text{E-}5$ and study the temporal convergence behaivor of our scheme for both interpolation- and extrapolation-based temproal integrators. Starting with temporal resolution $\delta t = 2.5\text{E-}1$ and 4 time-steps, we obtain two digits of accurcay for the interpolation and and one digit of accurcay for the extrapolation schemes. For the interpolation scheme, we observe a factor of 4× drop in relative L^∞ error, as we decrease the time-step size by a factor of 2, which confirms the second-order convergence rate. For the extrapolaiton scheme, we also observe the second-order convergence for higher temporal resolutions. However, for small time-step sizes the convergence order fluctuates.
In Table 9, we study the convergence and stability of the scheme for the same test case as in Table 8, but we conduct the experiment for longer time horizon, namely, for an integration over one period ($T = [0, 2\pi]$). We vary the number of time-steps from 100 to 1600, which corresponds to an increase in temporal resolution by a factor of 16×. As we increase our temporal accuracy by increasing the number of time-steps for a fixed time horizon, we also increase our spatial accuracy, to avoid the accumulation of spatial error in semi-Lagrangian-based schemes for a high number of time-steps (see Section 2.1.2). We use both low and high order discretization ($q = 3, 14$). For a fixed tree error tolerance ϵ_{tree}, this does not affect the spatial accuracy but rather the computational cost for a fixed problem.

For low discretization order, as we decrease the time-step size by a factor of 2, from $6.28\text{E-}2$ (100 time-steps) to $3.14\text{E-}2$ (200 time-steps), we observe

Table 7: *Convergence and single-node performance results for an advection-diffusion problem with different values of diffusivity ($\mathcal{D} = 1\text{E-}4, 1\text{E-}5$) and discretization orders ($q = 4, 8, 14$) for a fixed time-horizon ($T = 1.6$s). In each case, we show convergence in relative L^∞-norm at $t = 1.6$ as we reduce the time step size (δt) and the refinement tolerance (ϵ_{tree}). We report the total time to solution (T_{solve}) and the overall performance in* GFLOPS. *We have also presented a breakdown of the cost of various stages in the algorithm.*

$\mathcal{D}$	q	δt	ϵ_{tree}	L	N_{dof}	L_T^∞	T_{solve} (GFLOPS)	T_{ref}	T_{sort}	T_{eval} (GFLOPS)	T_{fmm} (GFLOPS)
	4	1.0E-1	2.5E-4	7	5.2E+5	6.9E-3	14.1 (44)	2.3	2.0	0.5 (38)	7.9 (76)
1E-4	4	2.5E-2	1.0E-5	8	5.0E+6	3.4E-4	569.0 (44)	43.0	112.7	19.7 (42)	312.4 (78)
	4	6.3E-3	2.5E-6	8	1.3E+7	3.3E-5	8185.7 (33)	301.9	2160.4	251.7 (35)	4562.4 (57)
	8	1.0E-1	2.5E-4	5	2.8E+5	5.2E-3	4.1 (41)	0.2	1.0	0.5 (91)	1.6 (74)
1E-4	8	2.5E-2	5.0E-5	5	4.0E+5	3.3E-4	22.6 (43)	0.8	6.4	2.8 (94)	8.4 (82)
	8	6.3E-3	2.5E-6	6	9.6E+5	2.1E-5	269.5 (38)	5.1	90.1	32.1 (96)	70.7 (97)
	14	1.0E-1	1.0E-3	4	3.0E+5	5.2E-3	6.2 (58)	0.3	1.2	1.5 (134)	1.8 (81)
1E-4	14	2.5E-2	5.0E-5	4	5.4E+5	3.3E-4	34.0 (71)	1.0	8.3	9.1 (143)	9.3 (108)
	14	6.3E-3	2.5E-6	5	8.4E+5	2.2E-5	229.1 (72)	4.6	62.3	59.2 (146)	53.5 (136)
	4	1.0E-1	1.0E-4	8	9.5E+5	9.3E-3	41.8 (43)	5.8	6.9	1.5 (39)	21.7 (80)
1E-5	4	2.5E-2	1.0E-5	9	5.0E+6	5.9E-4	864.0 (45)	54.4	178.6	28.7 (44)	475.5 (78)
	4	6.3E-3	2.5E-6	9	1.3E+7	3.7E-5	12241.0 (35)	426.5	3401.3	342.1 (40)	6878.3 (60)
	8	1.0E-1	2.5E-4	6	2.8E+5	9.3E-3	5.9 (43)	0.3	1.5	0.7 (93)	2.3 (81)
1E-5	8	2.5E-2	5.0E-5	6	4.0E+5	6.0E-4	30.6 (44)	1.0	9.2	3.8 (95)	10.9 (87)
	8	6.3E-3	2.5E-6	7	9.6E+5	3.7E-5	361.8 (37)	6.2	124.7	42.0 (96)	91.8 (98)
	14	1.0E-1	1.0E-3	4	3.0E+5	9.1E-3	7.7 (60)	0.3	1.6	1.8 (138)	2.2 (88)
1E-5	14	2.5E-2	5.0E-5	5	5.4E+5	6.0E-4	43.4 (76)	1.2	10.7	11.2 (143)	12.5 (125)
	14	6.3E-3	2.5E-6	5	8.4E+5	3.7E-5	310.4 (72)	5.4	86.6	78.5 (146)	67.6 (147)

a $4\times$ drop in relative L^∞ error. We also observe a $15\times$ drop in error, for high order discretization, as we increase the spatial and temporal resolution from $\epsilon_{tree} = 1\text{E-}3$ and $\delta t = 1.57\text{E-}2$ (400 time-steps) to $\epsilon_{tree} = 1\text{E-}5$ and $\delta t = 3.92\text{E-}3$ (1600 time-steps).

6.3 SINGLE-NODE PERFORMANCE ANALYSIS

In this section we study the single-node runtime and performance results of the advection-diffusion solver. All experiments were carried out on a single node of the Stampede system at the Texas Advanced Computing Center. Stampede nodes have two 8-core Intel Xeon E5-2680 (2.8GHz) processors and 32GB RAM. Stampede has a theoretical peak performance of 345GFLOPS per node. While the discretization order does not affect the accuracy of the solution (for fixed ϵ_{tree}), it can have a significant effect on the cost of the algorithm.

In general, a higher discretization order also results in a higher cost per unknown. For example, in the case of Chebyshev evaluation, the cost of each evaluation is $\mathcal{O}(q^3)$. Similarly, for the volume FMM, the cost per unknown depends on the Chebyshev degree q and the multipole order m. However, when approximating smooth functions, a higher discretization order can result in significantly fewer unknowns. In Table 7, for $\delta t = 0.1$, we require about 5E+5 unknowns and 7 levels of octree refinement with low-order discretization. For the same case, with high-order discretization, we require about 3E+5 unknowns and 4 levels of octree refinement. For higher accuracy (with $\delta t = 6.25\text{E-}3$), the difference is even more significant with higher order discretization requiring $15\times$ fewer unknowns. We observe that for low accuracy, a moderate discretization order ($q = 8$) works best and for higher accuracy, $q = 14$ gives a faster time to solution. In Table 7 and Figure 18, we have also presented a detailed breakdown of the total solve time. T_{ref} is the time spent in refinement and coarsening of the Chebyshev octree. This makes up a small percentage (approximately $2\% - 5\%$) of the total runtime. For semi-Lagrangian advection, the most time consuming part of the computation is evaluating the piecewise Chebyshev representation of the velocity and the concentration. It has the following two main components. T_{sort} is the time for sorting

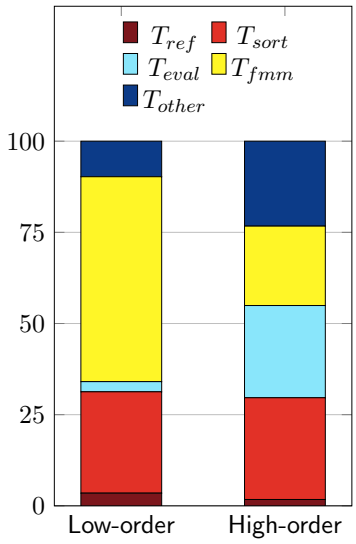


Figure 18: *Comparison of the cost of various components of the stiffly-stable algorithm depending on the discretization order in percentage of the total cost.*

Table 8: *Convergence study of advection-diffusion solver for unsteady vorticity field for a fixed time horizon* $T = [0, 1.0]$*. By using accurate spatial discretization* ($q = 14, \epsilon_{tree} = 1E\text{-}5$)*, we assure that our time integrator is the main source of the error. By reducing the time-step size by a factor of* 2*, we observe a second-order convergence in time. We also report the results for both interpolation and extrapolation time integrators. We observe that the both scheme, result in temporal error of the same order for the unsteady vorticity velocity field.*

q	δt	N_{iter}	L^∞	
			Interpolation	**Extrapolation**
14	2.50E-1	4	4.67E-2	1.35E-1
	1.25E-1	8	1.13E-2	1.78E-2
	6.25E-2	16	1.98E-3	1.46E-3
	3.12E-2	32	4.06E-4	6.08E-4
	1.56E-2	64	9.27E-5	1.69E-4
	7.81E-3	128	2.22E-5	4.30E-5

Table 9: *Convergence study of advection-diffusion solver for unsteady vorticity field for varying temporal and spatial resolution. To avoid the accumulation of the spatial error in the semi-Lagrangian scheme, we reduce the time-step size by a factor of* 2*, as we reduce the tree error tolerance by a factor of* 10*. We conduct the experiment for both low and high order discretization* ($q = 3, 14$) *for a time horizon* $T = [0, 2\pi]$*. For both choices of discretization order and unsteady time integrators (interpolation and extrapolation), we observe a second-order convergence.*

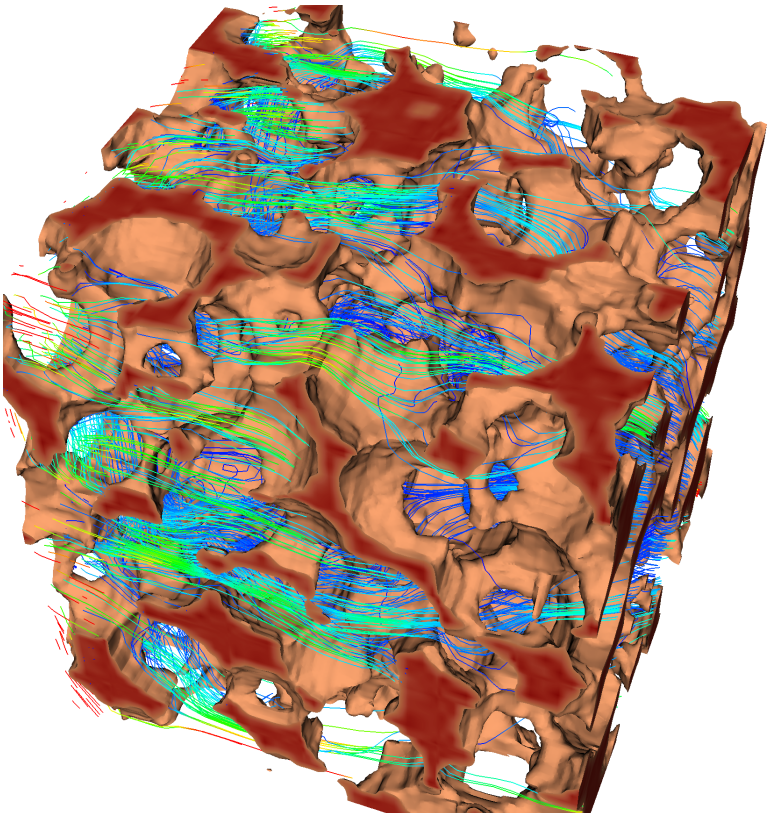
q	ϵ_{tree}	L	δt	N_{iter}	L^∞	
					Interpolation	**Extrapolation**
3	1E-2	5	6.28E-2	100	1.48E-2	1.24E-2
	1E-3	6	3.14E-2	200	3.84E-3	3.80E-3
14	1E-3	3	1.57E-2	400	5.80E-4	4.46E-4
	1E-4	3	7.85E-3	800	1.51E-4	9.18E-5
	1E-5	4	3.92E-3	1600	3.85E-5	2.04E-5

the evaluation points by their Morton IDs to determine the octant in which each point resides and rearranging the evaluated values in the original ordering of the points. T_{eval} is the time spent in evaluating the Chebyshev representation of each leaf node at the points belonging to that octant. The Chebyshev evaluation has $\mathcal{O}(q^3)$ cost per evaluation point and therefore, for low-order discretizations it is significantly less expensive than the sorting. However, for $q = 14$ they have comparable cost. For high order discretizations, the Chebyshev evaluation has very high arithmetic intensity and we are able to achieve about 146GFLOPS or 42% of the theoretical peak performance. This is due to careful vectorization and optimization for data reuse in cache. T_{sort} and T_{eval} together account for $20\% - 30\%$ of the solve time for low order discretization and about $44\% - 53\%$ of the solve time for high order discretization. The FMM evaluation time T_{fmm} accounts for $52\% - 56\%$ of the solve time for low order discretizations and $22\% - 29\%$ of the solve time for high order discretizations. For high discretization orders and sufficiently large problem sizes, the volume FMM can achieve high flop rates. However, for low order discretization or small problem sizes, the performance degrades quickly. In the results presented here, we observe a performance in the range of 57GFLOPS to 147GFLOPS.

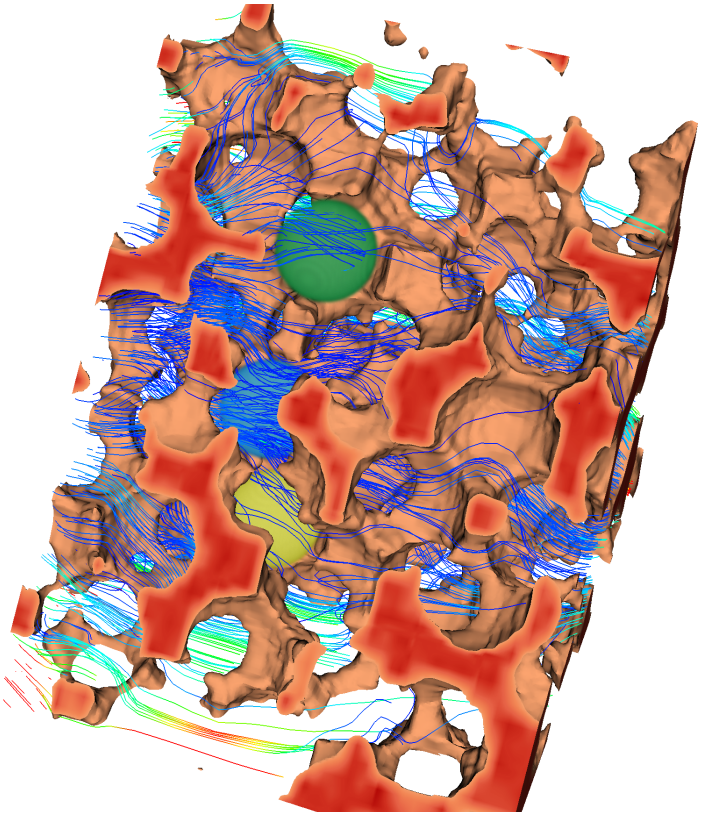
6.4 POROUS MEDIA SIMULATION AND VISUALIZATION SHOWCASE

To demonstrate the capabilities of our solver for more reallistic and complex geometries, we simulate the transport of substances in a porous medium with highly complex pore structure. A porous medium is a solid material containing voids. Many substances such as rocks, petroleum reservoir, bones, cements and ceramics can be considered as porous media. The simulation of porous media has applications in many areas of science and engineering such as filtration, rock mechanics, petroleum engineering, geosciences, material science, etc.

The Stokes flow in the porous medium is computed by using a volume integral equation solver and penalty formulation to enforce the no slip condition. The Stokes flow solver is not part of this work and the interested reader is referred to [63] for further information. The advection-diffusion problem is solved using the stiffly-stable scheme described in this chapter. In Figure 19 the red and orange areas represent the solid material. The streamlines visualize the velocity field, which corresponds to a stationary Stokes flow through this porous medium microstructure. We solve Equation (56) using this velocity field. The initial condition is the linear superposition of three Gaussians indicated by light blue, light green, and yellow. In Figure 19b, we clip the geometry to better visualize the velocity streamlines and initial concentrations fields. In Figure 19c and Figure 19d, we show two different snapshots in time, as the three color-coded substances flow through the porous medium.

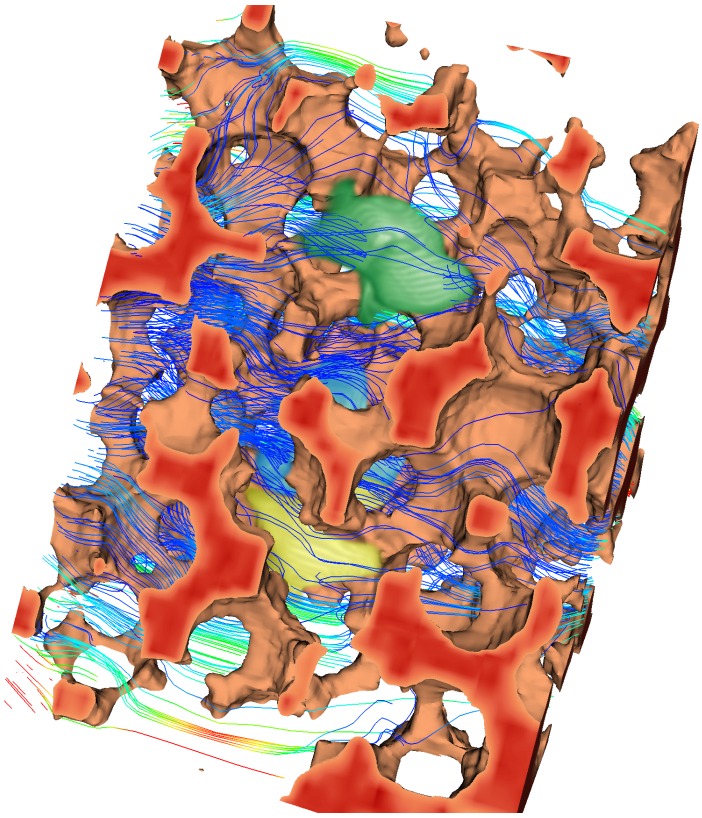


(a)

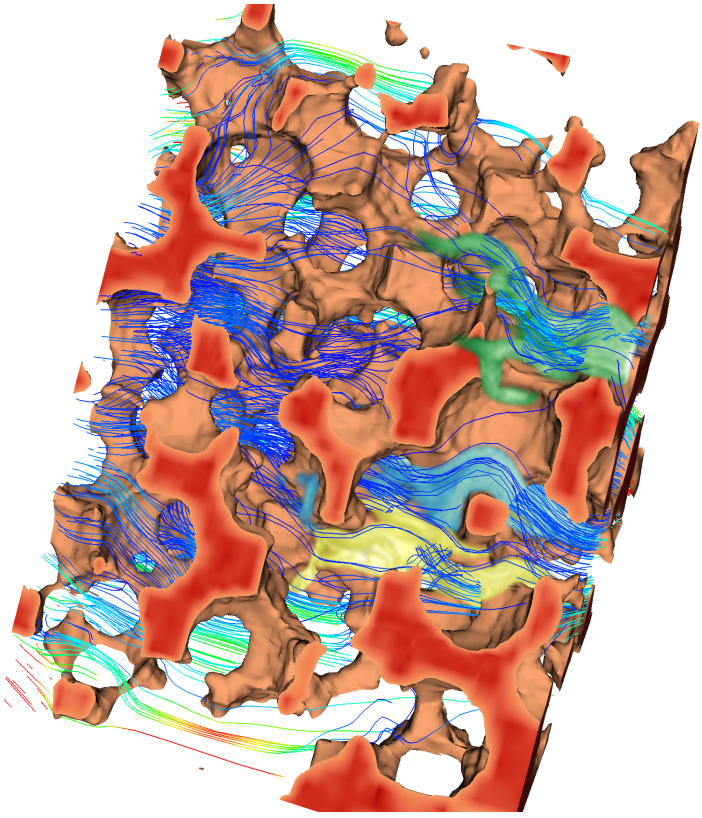


(b)

Figure 19: *Advection-diffusion problem in a porous medium. The red and orange areas represent the solid phase. The streamlines visualize the velocity field, which corresponds to a stationary Stokes flow through this porous medium microstructure. We solve Equation* (56) *using this velocity field. The velocity is calculated using a volume integral equation solver, whereas the advection-diffusion problem is solved using the stiffly-stable scheme described in this chapter.*



(c)



(d)

Figure 19: *The initial porous media and the stationary velocity field are visualized in (a). In (b), we clip the geometry to better visualize the velocity streamlines and initial concentrations fields. The initial condition is the linear superposition of three Gaussians indicated by light blue, light green, and yellow. In (c) and (d), we show two different snapshots in time, as the three color-coded substances flow through the porous medium.*

6.5 SUMMARY

In this chapter we introduced an explicit-implicit solver for the advection-diffusion problem, where we treat the linear advective term with the explicit, but unconditionally stable semi-Lagrangian method and the stiff parabolic diffusion term with the implicit volume integral method. We used the parallel aribitary-order accurate Chebyshev octree data structure as our spatial discretization. For the temporal integration, we introduced the first-order Lie splitting method, which we use to construct additional initial values required for higher order multistep methods and the second-order stiffly-stable multi-step method. We demonstrated the convergence of our scheme for complex steady and unsteady velocity fields for both interpolation- and extrapolation-based time integrators. In our single-node performance analysis, we observed that using higher order discretization results in significantly fewer unknowns and therefore faster time-to-solution. Finally, to demonstrate the capabilities of our solver we simulated a challenging realistic scenario, namely, the transport of substances in a porous medium with highly complex pore structure. In the next chapter we extend our Semi-Lagrangian/Volume-Integral approach to solve the incompressible Navier–Stokes equations.

7

THE INCOMPRESSIBLE NAVIER-STOKES EQUATION

In the previous chapter we introduced our novel Semi-Lagrangian/Volume-Integral method for the advection-diffusion equation. In this chapter our goal is to extend this method to the incompressible Navier-Stokes equations. The Navier–Stokes equations are a set of nonlinear PDEs, which describe the flow of viscous fluid substances and are therefore used in various applications in science and engineering such as modeling weather, the design of vehicles, blood flow simulation, etc. In this chapter we provide numerical and algorithmic details of our methodology to solve the incompressible Navier–Stokes equations. We use a similar formulation as for the advection-diffusion solver. However, since the velocity is not given at any future time-steps, we deploy the extrapolation-based trajectory computation method described in Section 4.5.2 in the semi-Lagrangian algorithm. Similar to the advection-diffusion solver, we use the arbitrary-order accurate Chebyshev octree spatial discretization combined with the explicit-implicit time-marching scheme, where we treat the nonlinear convective term with the second-order unconditionally stable semi-Lagrangian method and the stiff unsteady Stokes operator with an implicit volume integral formulation.

Section 7.1 gives a breif description of the incompressible Navier–Stokes equations with primitive variable formulation. We also discuss the non-primitive variable formulation of the incompressible Navier–Stokes equations such as the velocity-vorticity formulation. In Section 7.2 we discuss our two-step, second-order, stiffly-stable temporal discretization deployed for our time-marching approach. Finally, in Section 7.3 we study the stability and correctness of our proposed numerical scheme for a well-known benchmark problem, the Taylor-Green vortex flow for a wide range of Reynolds numbers. However, the stability of the nonlinear convective term for more complex flows requires further numerical investigation.

7.1 THE GOVERNING EQUATIONS

For liquids we can assume the density to be constant. We refer to the fluids with constant density as incompressible. For incompressible flows the continuity equation reduces to $\nabla \cdot \boldsymbol{v} = 0$. Thus the incompressible Navier–Stokes equations read

$$\rho \left(\frac{\partial \boldsymbol{v}(\boldsymbol{x}, t)}{\partial t} + \boldsymbol{v}(\boldsymbol{x}, t) \cdot \nabla \boldsymbol{v}(\boldsymbol{x}, t) \right) = -\nabla P(\boldsymbol{x}, t) + \mu \Delta \boldsymbol{v}(\boldsymbol{x}, t) + \mathcal{F}, \qquad (71)$$

where $\boldsymbol{v}(\boldsymbol{x}, t)$ is the fluid velocity field, $P(\boldsymbol{x}, t)$ is the pressure, μ is the dynamic viscosity and ρ is the fluid density.

These simplications reduces the complexity of the Navier–Stokes equations for different flow regimes; however, they come at a cost that the simplified equations might not be able to describe all physical phenomena. For instance, the incompressibility rules out physical phenomena such as sound or shock waves and therefore can not be used if these phenomena are of interest. However, the incompressibility assumption typically holds well for all fluid flows at low Mach numbers (the ratio of the flow velocity to the local speed of sound is low).

7.1.1 *The Non-Dimensional Formulation*

To obtain the non-dimensional formulation of the incompressible Navier–Stokes equation, we define a characteristic length L, a characteristic velocity U, and a characteristic time T based on the spatial and temporal features of the problem. By introducing the dimensionless variables $\boldsymbol{v}' = \boldsymbol{v}/U$, $\boldsymbol{x}' = \boldsymbol{x}/L$, $t' = t/T$ and $P' = PL/(\mu U)$, the incompressible Navier–Stokes equation in non-dimensional form in absence of external forces read

$$\beta \frac{\partial \boldsymbol{v}'}{\partial t'} + Re\boldsymbol{v}' \cdot \boldsymbol{\nabla}' \boldsymbol{v}' = -\boldsymbol{\nabla}' P' + \Delta' \boldsymbol{v}', \tag{72}$$

where $\beta = L^2/\nu T$ is the *frequency parameter*, $Re = UL/\nu$ is the *Reynolds number* and $\nu = \mu/\rho$ is the *kinematic viscosity*. The Reynolds number present the ratio of inertial convective forces to viscous forces, while the frequency parameter expresses the ratio of inertial acceleration forces relative to viscous forces.

Due to the complexity of the Navier–Stokes equations, for different flow regimes the simplified versions of the incompressible Navier–Stokes equations will be used. The flow regimes are categorized based on the Reynolds number and the frequency parameter. Flows with low Reynolds number are called *laminar* while flows with high Reynolds number are *turbulent*. When the Reynolds number and frequency parameters are very small $Re, \beta \ll 1$, then the inertial forces can be neglected. For these cases the flow is named *Stokes flow* or also known as *creeping flow*. The Stokes flow is described with the Stokes equation, which is a linearization of the Navier–Stokes equations (see also Section 2.2.4):

$$-\boldsymbol{\nabla} P(\boldsymbol{x}, t) + \mu \Delta \boldsymbol{v}(\boldsymbol{x}, t) + \mathcal{F} = 0. \tag{73}$$

Stokes equations arises in several geophysical areas such as mantle convection [68] and ice sheet dynamics [46]. The stokes equations can also model flows in biofuels, polymers and porous media flows (see Section 6.4). Due to importance of the Stokes equations, there is extensive literature available

using various numerical methods to solve the Stokes equations [91], [105]. The most common approaches to discretize the Equation (73) are finite element [20], finite-difference or finite volume methods. In [63] a Stokes solver is introduced with a volume integral formulation for problems with variable coefficients and complex geometries based on the same PVFMM library that we use in this dissertation. We used this Stokes solver in Section 6.4 to compute the Stokes flow inside a porous medium with a highly complex pore structure.

7.1.2 *The Velocity-Vorticity Formulation*

As an alternative to the more common velocity-pressure formulation of the incompressible Navier–Stokes equations, the so called vorticity formulation describes the motion of the local rotation of the fluid instead of the primitive quantities such as velocity and pressure. This formulation relies on the idea of eliminating the pressure variable by applying the curl operator and introducing the vorticity $\boldsymbol{\omega}$ as the curl of the velocity

$$\frac{D\boldsymbol{\omega}}{Dt} = \frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{v} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{v} + \nu \Delta \boldsymbol{\omega}. \tag{74}$$

A detailed discussion of the advantages and disadvantages of this formulation is given in [83].

Computing the numerical solution of the incompressible Navier–Stokes equations is challenging due to nonlinear advective term, the incompressibility condition and the coupling of the equations. In the next section we discuss our explicit-implicit Semi-Lagrangian/Volume-Integral approach to solve the incompressible Navier–Stokes equations.

7.2 THE SECOND-ORDER STIFFLY-STABLE SCHEME

We consider the incompressible Navier–Stokes equations in the Lagrangian form where the advective term in Equation (71) transforms to the material derivative

$$\frac{D\boldsymbol{v}(\boldsymbol{x}, t)}{Dt} = -\nabla p(\boldsymbol{x}, t) + \nu \Delta \boldsymbol{v}(\boldsymbol{x}, t), \tag{75}$$

$$\nabla \cdot \boldsymbol{v}(\boldsymbol{x}, t) = 0. \tag{76}$$

Here, we assume no external forces are applied to the fluid ($\mathcal{F} = 0$). We solve the Equations (75) and (76) with the initial condition $\boldsymbol{v}(\boldsymbol{x}, 0) = \boldsymbol{v}_0(\boldsymbol{x})$ with free-space and periodic boundary conditions at the boundary $\partial\Omega$ of a cubic domain Ω. A second-order temporal discretization is given by the stiffly-stable method (see also Section 6.1.3)

$$\frac{\frac{3}{2}\boldsymbol{v}(\boldsymbol{x}, t_{k+1}) - 2\boldsymbol{v}(\boldsymbol{X}_k, t_k) + \frac{1}{2}\boldsymbol{v}(\boldsymbol{X}_{k-1}, t_{k-1})}{\delta t} = \mu \Delta \boldsymbol{v}(\boldsymbol{x}, t_{k+1}) - \nabla p(\boldsymbol{x}, t_{k+1}), \tag{77}$$

where $\mathbf{X}_k$ and $\mathbf{X}_{k-1}$ are the departure points of the Lagrangian particles at time t_k and t_{k-1} that arrive at the grid points $\boldsymbol{x}$ at time t_{k+1}. The departure points are obtained by solving the characteristics backward in time. In contrast to the advection-diffusion problem, where the velocity field is normally given, here the velocity is the quantity we are computing. Therefore to solve the characteristics equation backward in time, we need to apply the extrapolation-based modified midpoint rule (see Section 4.5.2 and [102])

$$\hat{\mathbf{x}} = \mathbf{x} - \frac{\delta t}{2}\boldsymbol{v}(\mathbf{x}, t_k), \tag{78}$$

$$\mathbf{x}_d = \mathbf{x} - \delta t \boldsymbol{v}(\hat{\mathbf{x}}, t_k + \delta t/2). \tag{79}$$

Here, the velocity at t_k is known and the velocity at $t_k + \delta t/2$ is approximated by using the second-order extrapolation method

$$\boldsymbol{v}_{k+\frac{1}{2}} = \frac{3}{2}\boldsymbol{v}_k - \frac{1}{2}\boldsymbol{v}_{k-1}. \tag{80}$$

To compute $\mathbf{X}_k$, we solve Equations (78) and (79) with the same temporal resolution δt as in Equation (77). The initial condition of the characteristics is the Eulerian grid position $\boldsymbol{x}$ at time t_{k+1}. We require velocity values at times t_k and $t_k + \delta t/2$ at midpoint position $\tilde{\mathbf{x}}$. The latter is approximated by evaluating the known velocity trees at time t_k and t_{k-1} at the midpoint position $\tilde{\mathbf{x}}$ of each Lagrangian particle and extrapolating the values at time $t_k + \delta t/2$ according to Equation (80). Similarly, $\mathbf{X}_{k-1}$ is obtained by solving the characteristics starting at grid position x at time t_{k+1}, however with a temporal resolution of $2\delta t$ such that solving the modified midpoint equations requires the known velocity values at time t_k and t_{k-1}. By so doing, we avoid the interpolation of velocity values at $t_k - \delta t/2$. Once the departure points are computed, we can obtain the $\boldsymbol{v}_k = \boldsymbol{v}(\mathbf{X}_k, t_k)$ and $\boldsymbol{v}_{k-1} = \boldsymbol{v}(\mathbf{X}_{k-1}, t_{k-1})$ by interpolation. In the next step, by bringing the known values to the right-hand side, we transform Equation (75) to the following equation:

$$\frac{3}{2\delta t}\boldsymbol{v}_{k+1} - \mu\Delta\boldsymbol{v}_{k+1} + \boldsymbol{\nabla} p_{k+1} = \frac{2}{\delta t}\boldsymbol{v}_k - \frac{1}{2\delta t}\boldsymbol{v}_{k-1}. \tag{81}$$

This equation corresponds to the unsteady Stokes formulation (see Section 2.2)

$$\alpha\boldsymbol{v} - \mu\Delta\boldsymbol{v} + \nabla p = f,$$

with $\alpha = 3/2\delta t$. Given the velocity values $\boldsymbol{v}_k$ and $\boldsymbol{v}_{k-1}$, we solve for $\boldsymbol{v}_{k+1} = \boldsymbol{v}(\boldsymbol{x}, t_{k+1})$ by computing the convolution of the right-hand side $f = 2\boldsymbol{v}_k/\delta t - \boldsymbol{v}_{k-1}/(2\delta t)$ with the unsteady Stokes kernel

$$\begin{aligned}\boldsymbol{v}_{k+1}(\mathbf{x}) &= \frac{1}{8\pi\mu}\int_y \left(\frac{A(R)}{r}I + \frac{B(R)}{r^3}(\mathbf{r}\otimes\mathbf{r})\right) f(\mathbf{y})\,dy \\ &= \frac{1}{8\pi\mu}\int_y \left(\frac{A(R)}{r}I + \frac{B(R)}{r^3}(\mathbf{r}\otimes\mathbf{r})\right)\left(\frac{2}{\delta t}\boldsymbol{v}_k(\mathbf{y}) - \frac{1}{2\delta t}\boldsymbol{v}_{k-1}(\mathbf{y})\right) dy,\end{aligned} \tag{82}$$

where $\lambda = \sqrt{\frac{\alpha}{\mu}}$, $r = |\mathbf{x} - \mathbf{y}|$ and $R = \lambda r$. I is the identity operator and $\otimes$ denotes the tensor product. $A(R)$ and $B(R)$ are given in Equations (24) and (25) in Chapter 2. Similar to the advection-diffusion solver our field values are represented with parallel, piecewise Chebyshev octree data structures. We use the PVFMM library to solve the volume integral in Equation (82) on this spatial discretization. The procedure to solve the incompressible Navier–Stokes equations based on the Semi-Lagrangian/Volume-Integral scheme stated above is given in Algorithm 9. Notice in Algorithm 9 that for computing the departure points at t_{k-1}, in contrast to the advection-diffusion solver in Algorithm 8, we use the $\mathbf{X}_{k+1}$ as the initial value for the trajectory computation and solve backward in time for a time-step size $2\delta t$ by using the extrapolation scheme described in Equations (78) to (80).

Algorithm 9 Solving the incompressible Navier-Stokes equations with the stiffly-stable method and Chebyshev octree data structure.

Input:
$\mathcal{T}_{\boldsymbol{\nu}}^{t_k}, \mathcal{T}_{\boldsymbol{\nu}}^{t_{k-1}}$: Velocity trees at time t_k and t_{k-1}.
Output:
$\mathcal{T}_{\boldsymbol{\nu}}^{t_{k+1}}$: Velocity tree at time t_{k+1}.

procedure NSStifflyStable($\mathcal{T}_{\boldsymbol{\nu}}^{t_{k-1}}, \mathcal{T}_{\boldsymbol{\nu}}^{t_k}$)
  $\mathcal{N} \leftarrow$ LeafNodes($\mathcal{T}_{\boldsymbol{\nu}}^{t_{k-1}}$)
  for each $\mathcal{B} \in \mathcal{N}$ **do**
    $\mathbf{X}_{k+1} \leftarrow \mathbf{X}_{k+1} \cup$ InterpPoints($\mathcal{B}$)
  end for
  $\mathbf{X}_k \leftarrow$ Trajectory($\mathcal{T}_{\boldsymbol{\nu}}^{t_{k-1}}, \mathcal{T}_{\boldsymbol{\nu}}^{t_k}, \mathbf{X}_{k+1}, \delta t$)
  $\boldsymbol{\nu}_k \leftarrow$ EvaluateParallelTree($\mathcal{T}_{\boldsymbol{\nu}}^{t_k}, \mathbf{X}_k$) ▷ Algorithm 2
  $\mathbf{X}_{k-1} \leftarrow$ Trajectory($\mathcal{T}_{\boldsymbol{\nu}}^{t_{k-1}}, \mathcal{T}_{\boldsymbol{\nu}}^{t_k}, \mathbf{X}_{k+1}, 2\delta t$)
  $\boldsymbol{\nu}_{k-1} \leftarrow$ EvaluateParallelTree($\mathcal{T}_{\boldsymbol{\nu}}^{t_{k-1}}, \mathbf{X}_{k-1}$) ▷ Algorithm 2
  for each $\mathcal{B} \in \mathcal{N}$ **do**
    $\alpha^{\mathcal{B}} \leftarrow M^{-1}\left(\frac{2}{\delta t}\boldsymbol{\nu}_k^{\mathcal{B}} - \frac{1}{2\delta t}\boldsymbol{\nu}_{k-1}^{\mathcal{B}}\right)$
  end for
  $\mathcal{T}_{\boldsymbol{\nu}}^{t_{k+1}} \leftarrow$ RunFMM($\mathcal{T}_{\boldsymbol{\nu}}^{t_{k-1}}$) ▷ Unsteady Stokes kernel
  RefineTree($\mathcal{T}_{\boldsymbol{\nu}}^{t_{k+1}}$) ▷ AMR algorithm in Section 4.4
end procedure

7.3 TAYLOR-GREEN VORTEX

The classical Taylor-Green vortex flow is a benchmark problem to study the generation of small-scale vorticity by vortex stretching [13]. By having an initial smooth analytical solution with simple periodic boundary condition, the Taylor-Green vortex flow serves as a well-established benchmark test problem, which provides an easy way to validate the code.

The Taylor-Green initial condition in a periodic domain $\Omega = [0, 2\pi]^3$ is described by the following equations [98]:

$$\begin{aligned} \mathbf{u}_x(\mathbf{x}, t = 0) &= \frac{2}{\sqrt{3}} \sin(\theta + \frac{2\pi}{3}) \sin(x) \cos(y) \cos(z), \\ \mathbf{u}_y(\mathbf{x}, t = 0) &= \frac{2}{\sqrt{3}} \sin(\theta - \frac{2\pi}{3}) \cos(x) \sin(y) \cos(z), \\ \mathbf{u}_z(\mathbf{x}, t = 0) &= \frac{2}{\sqrt{3}} \sin(\theta) \cos(x) \cos(y) \sin(z). \end{aligned} \tag{83}$$

Here, we set the $\theta = 0$ in all of our runs. This initial field is visualized in Figure 20.

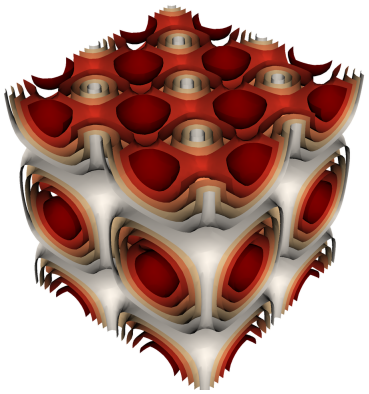
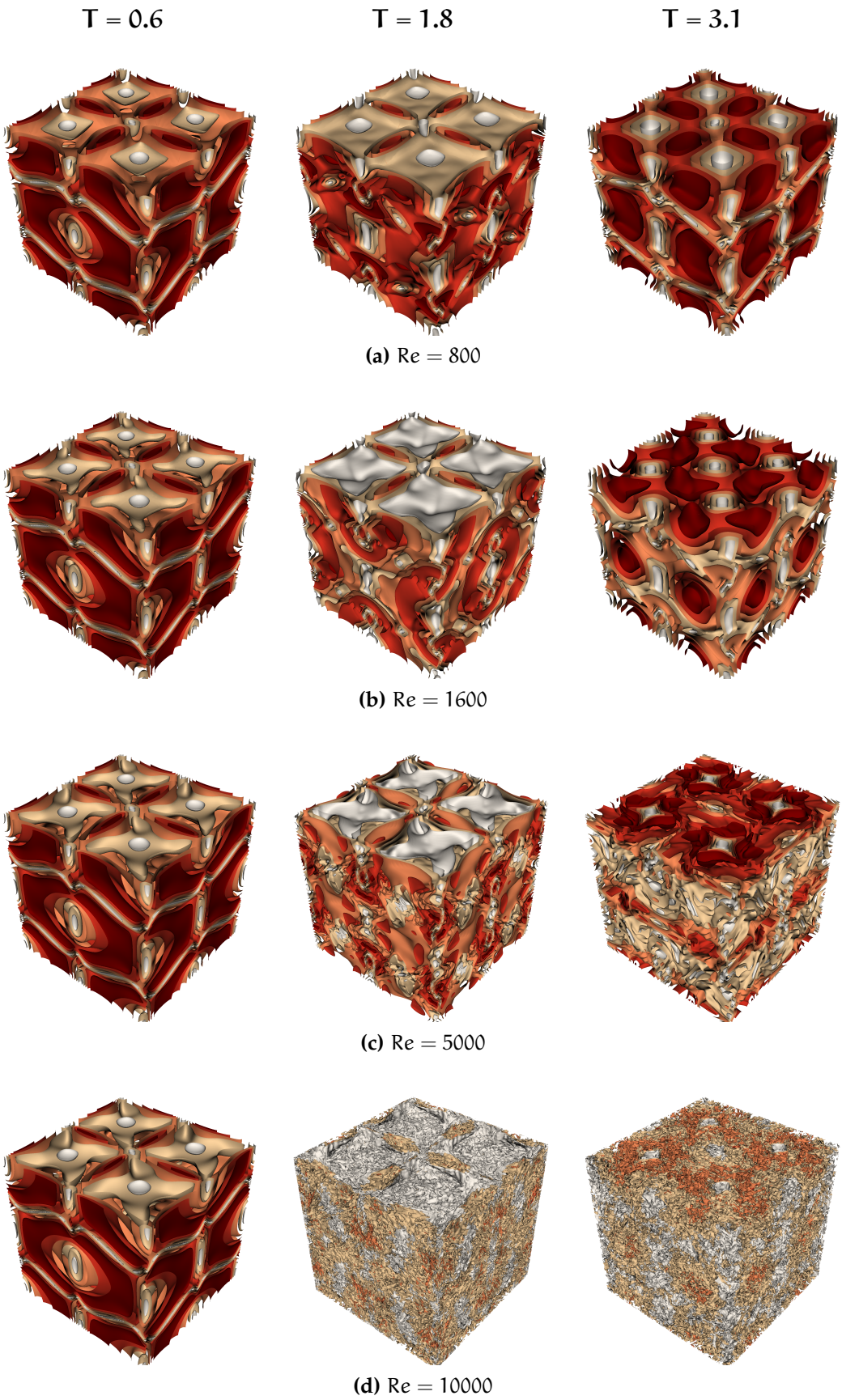


Figure 20: *Initial condition of the Taylor-Green benchmark problem visualized with multiple Isosurfaces plots of the velocity magnitude.*

In Figure 21, we show the soluton of the incompressible Navier–Stokes equations for the time horizon $T = 2\pi$ with this initial condition for four different Reynolds numbers varing from relatively low Reynolds numbers such as 800 and 1600 to very large Reynolds numbers in the turbulent flow regime such as 5000 and 10000. Here, the Reynolds number is defined as $Re = 1/\nu$. We use a high order discretization $q = 14$ and tree error tolerance $\epsilon_{tree} = 1E\text{-}3$ with a temporal resolution of $\delta t = 6.28E\text{-}2$. In Figure 21, to demonstrate the vortex dynamics of the flow for different Reynolds numbers, we visualize multiple snapshots of the isosurfaces of the velocity magnitude $|\nu|$ at times $T = 0.6, 1.8$, and 3.1. Notice the transition to a turbulent flow regime for high Reynolds numbers. In Figure 22 we also illustrate the contour plots of the vorticity magnitude $|\omega|$ of the Taylor-Green test case at plane $y = \pi$ for Reynolds number $Re = 1600$ for multiple snapshots in time, which has been studied extensively in the literature [13], [31], [34], [70], [80].



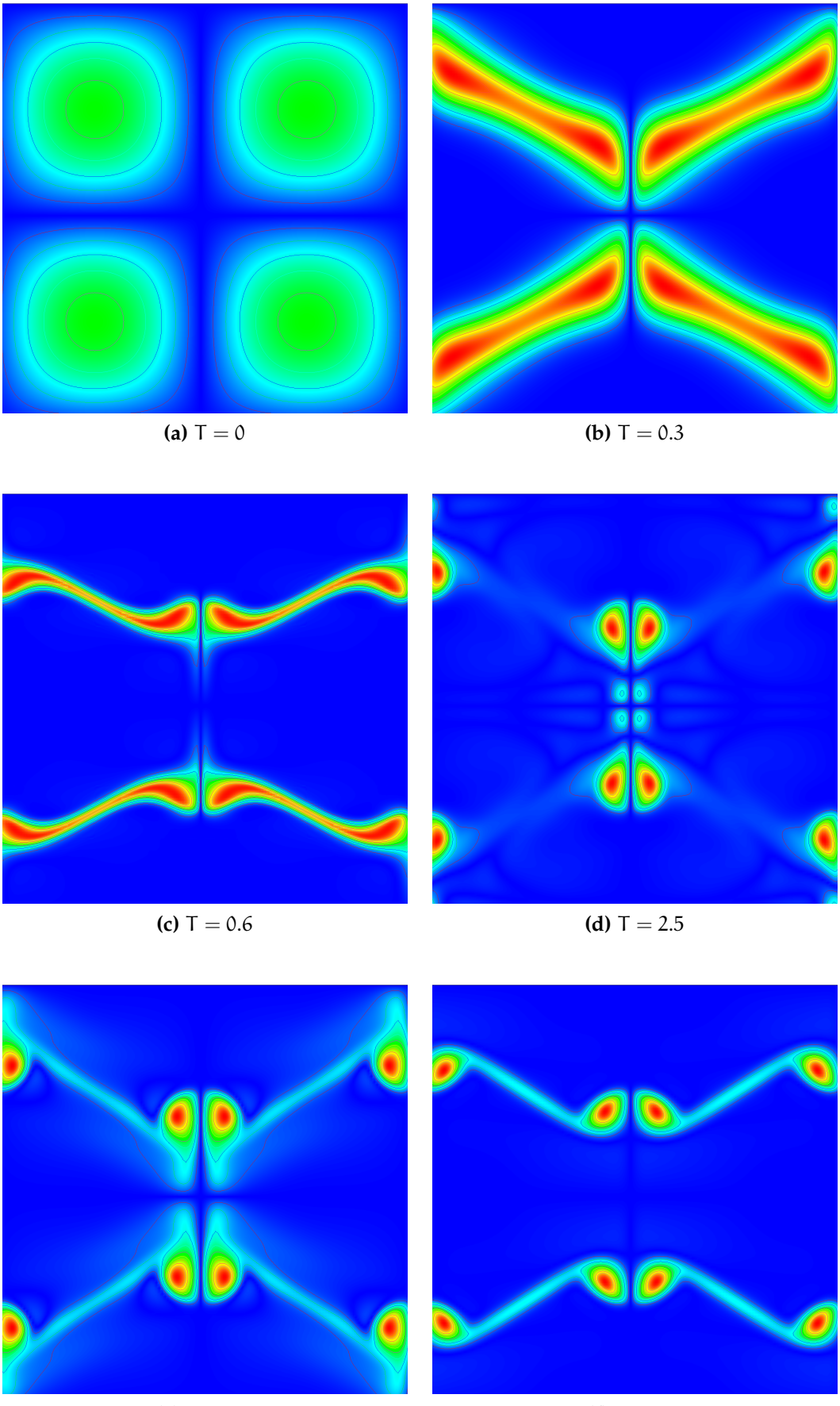
(a) Re = 800

(b) Re = 1600

(c) Re = 5000

(d) Re = 10000

Figure 21: *Solution of the incompressible Navier–Stokes equations with Taylor-Green vortex flow as the initial condition. Here, we visualize the isosurfaces of the velocity magnitude* $|\mathbf{v}|$ *for multiple snapshots of the solution at* $T = 0.6, 1.8,$ *and* 3.1 *for various Reynolds numbers (a)* $\mathrm{Re} = 800$*, (b)* $\mathrm{Re} = 1600$*, (c)* $\mathrm{Re} = 5000$*, and (d)* $\mathrm{Re} = 10000$*.*



(a) $T = 0$ **(b)** $T = 0.3$

(c) $T = 0.6$ **(d)** $T = 2.5$

(e) $T = 3.7$ **(f)** $T = 6.2$

Figure 22: *Contour plots of the vorticity magnitude* $|\omega|$ *of the Taylor-Green test case at plane* $y = \pi$ *for Reynolds number* $Re = 1600$ *at multiple snapshots in time.*

7.4 SUMMARY

In this chapter we extended our Semi-Lagrangian/Volume-Integral method introduced in previous chapters to the incompressible Navier–Stokes equations. Similar to the advection-diffusion solver, we used the arbitrary-order accurate Chebyshev octree spatial discretization and second-order stiffly-stable temporal discretization combined with the explicit-implicit Semi-Lagrangian/Volume-Integral time-marching scheme, where we treat the nonlinear advective term with the second-order unconditionally stable semi-Lagrangian method and the stiff unsteady Stokes operator with an implicit volume integral formulation. However, for the trajectory computation method in the semi-Lagrangian algorithm we used the extrapolation-based modified mid-point method. To demonstrate the correctness of our scheme, we conducted experiments for a wide range of Reynolds number varying from laminar to turbulant flow regimes with a classical benchmark problem for the incompressible Navier–Stokes solvers, namely the Taylor-Green vortex flow. This chapter was devoted to demonstrate an example of how our Semi-Lagrangian/Volume-Integral methodology can be extended to more complex problems such as the incompressible Navier–Stokes equations. However, the stability of the nonlinear term for more complex flows require further numerical investigation.

8

PARALLEL PERFORMANCE ANALYSIS

In this chapter we study the parallel performance of our AMR algorithm for the advection-diffusion solver. We analysed the single-node performance of our code in Section 6.2.1. Our goal in this chapter is to study the isogranular (weak) and fixed-sized (strong) scalability of our solvers. For the isogranular scalability experiments, we simultaneously increase the problem size and number of cores to keep the problem size per core constant. For the fixed-size scalability, we keep the total problem size constant as we increase the number of cores. By using these performance indicators, we are able to analyze our parallel implementation performance as well as our algorithmic scalability.

To study the scalability of our solvers, we consider different problems with various velocity and initial concentration fields with small, medium and large sized test problems. For some of these problems, we do not have an analytic solution so we only report timing and the breakdown for different parts of the algorithm. In particular:

- In Table 10 we report strong scaling results for an advection-diffusion problem with a modest number of unknowns. The initial condition is a linear combination of Gaussians and the velocity field is a rigid rotation; so we know the solution analytically. This problem is solved to 4 digits of accuracy.
- In Tables 11 and 12 we report strong scaling for two much larger problems. The initial condition is a Gaussian sphere and the velocity field is the Taylor-Green vortex flow. This configuration and multiple snapshots of the solution are visualized in Figure 23. The solution for this scenario develops sharp gradients as it evolves over time, which require significant refinement to be resolved accurately. In Figure 16 we also visualize the octree-mesh to highlight the dynamic mesh adaptation. We scale this problem from 16 compute nodes up to 1024 compute nodes (16 cores per compute node).
- In Table 13 we report weak scaling results from 32 to 1024 compute nodes for the same problem setup as in Tables 11 and 12.

Before presenting the results of our experiments, we summarize first the setting of the computing system we use to conduct the experiments.

HARDWARE All experiments were carried out on the Stampede system at TACC. Stampede is a Linux cluster consisting of 6400 compute nodes. The

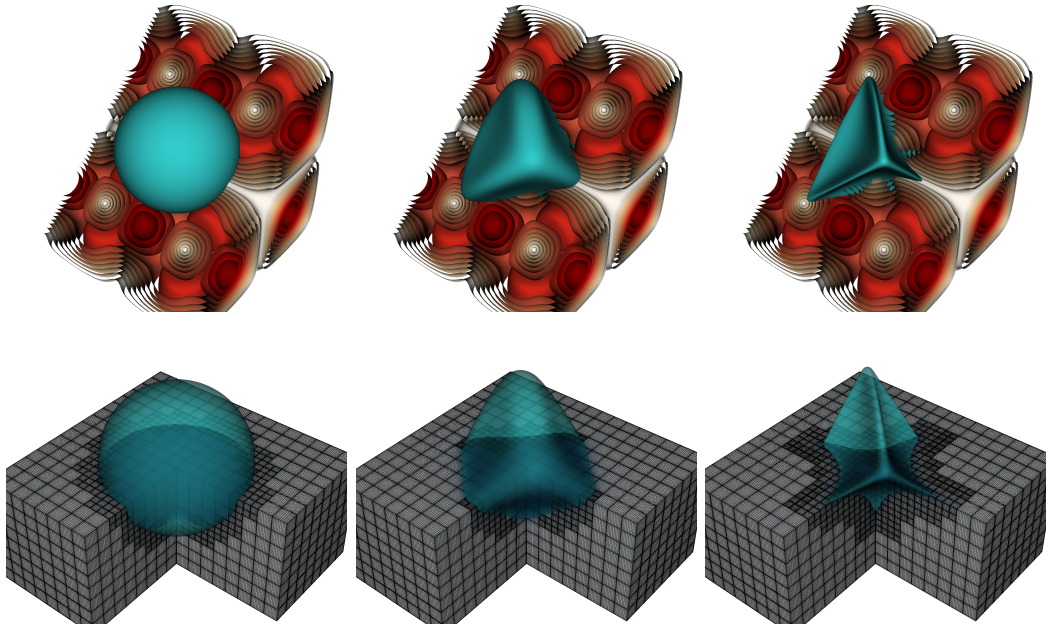


Figure 23: *In this figure we show the contours of the Taylor-Green vortex flow, which we use for the velocity field, and the Gaussian sphere that we use for the initial condition of the concentration. We also depict a few snapshots from the evolution of* c*. We can see that it develops sharp gradients that require significant refinement to resolve them accurately. We also observe that the spatial features of* c *are quite different from the spatial features of* $\boldsymbol{v}$*. In the second row, we visualize the octree mesh for the different time steps to highlight the dynamic mesh adaptation.*

compute nodes are connected with a fat tree topology [58] using a 56GB/s FDR Mellanox InfiniBand network. Each Stampede node has two 8-core Intel Xeon E5-2680 (2.8GHz) processors and 32GB RAM. The system has a theoretical peak performance of 345.5GFLOPS per node.

SOFTWARE Our code is written in C++ and uses OpenMP for shared memory parallelism, Intel MKL for high performance linear algebra operations, and Intel MPI for distributed memory parallelism. Our semi-Lagran gian interpolation is optimized with SSE2 and AVX. We use the PVFMM [62] library for the FMM. All our runs were done by using ***1 MPI task / node*** and ***16 OpenMP threads***.

In all of our runs, we adaptively refine or coarsen the octree ***at every time step***. All the velocity fields we use are scaled so that $\|\boldsymbol{v}\|_\infty \approx 1$. For an octree with maximum depth L and discretization order q, we can estimate the CFL number by

$$\text{CFL} = \frac{\delta t}{\delta x_{\text{cfl}}} \text{ and } \delta x_{\text{cfl}} = 2^{-L} q^{-2}, \tag{84}$$

since the spacing of our points is that of the Chebyshev points [93]. Using this formula, we see that the CFL number in most of our runs is quite large ($\mathcal{O}(100)$). In the tables, N_{dof} indicates the number of true degrees of freedom, roughly $q^3/6$ Chebyshev coefficients at each octant. Recall that when we perform the semi-Lagrangian advection, we use q^3 actual points so the problem size for the semi-Lagrangian, interpolation, sorting, and communication is six times larger than N_{dof}. We conduct the experiments by using Separate-Trees, Complete-Merge and Semi-Merge partitioning schemes. Regarding the Semi-Merge scheme, we can relate the memory costs and show the robustness of the code to the CFL number. More specifically, for a CFL number smaller than q^2, we would only need to move particles by distances smaller than the dimension of the smallest octant. So the extra memory needed would be bounded by the number of ghost octants. For a 2:1 balanced octree this is bounded.

8.1 TEST CASE: GAUSSIAN FUNCTIONS IN VORTICITY FLOW

In Table 10 and Figure 24 we present strong scaling results up to 64 compute nodes (1024 cores) of Stampede. Our test problem consists of 300 randomly distributed Gaussian functions (similar to the one discussed in Section 6.2.1) with variance $\sigma_i \in (5\text{E-}3, 1.5\text{E-}2)$ as intial conditions. We use the time step size $\delta t = 6.25\text{E-}3$ and 256 time steps. The relative L^∞ norm of error at the end of the simulation ($t = 1.6$) is 5.5E-5. We use the Semi-Merge scheme to partition the concentration and velocity trees across processors. We report results for moderate ($q = 8$) and high ($q = 14$) order discretizations. Overall the high-order scheme delivers nearly a $2\times$ speedup for the same accuracy.

For the $q = 8$ case, we require 8 levels of octree refinement with 122K leaf octants corresponding to 20 million unknowns. Scaling from 1 compute node to 8 nodes, we achieve $5\times$ speedup or nearly 63% parallel efficiency. The efficiency drops to 48% for 16 nodes. Scaling from 1 compute node to 64 compute nodes, we achieve nearly $18\times$ speedup or 28% parallel efficiency.

The time for adaptive refinement, repartitioning and merging of the velocity and concentration trees is reported as T_{ref}. On a single compute node, this accounts for just 2% of the total run time. On two compute nodes, we observe a $3.4\times$ increase in T_{ref} due to the communication between compute nodes. On 64 compute nodes, T_{ref} makes up for nearly 26% of the total time. The computation for the semi-Lagrangian advection is dominated by evaluation of the piecewise polynomial representation of concentration and velocity. We report a breakdown of the run time for the evaluation phase into: T_{comm} is the time for communicating point coordinates and bringing back the evaluated values for points which have to be evaluated on a remote processor, T_{sort} is the time for locally sorting the points by Morton ID to determine the leaf octant on which the points have to be evaluated, and T_{eval} is the time for evaluating the Chebyshev representation of each leaf node

Table 10: *We report strong scaling results for an advection-diffusion problem with diffusivity $\mathcal{D} = 1\text{E-}5$ solved to 4-digits of accuracy. We have used the time step size $\delta t = 6.25\text{E-}3$ and 256 time steps. We visualize the results for high order discretizations ($q = 14$). We present a detailed breakdown of the total runtime T_{solve} into T_{ref} (adaptive refinement, repartitioning and merging of trees), T_{fmm} (volume FMM computation), T_{comm}, T_{sort} and T_{eval} (communication, local sorting and Chebyshev evaluation at leaf nodes). For floating-point intensive parts of the algorithm we also report the performance in* GFLOPS ***per compute node***. *Here, p is the number of compute nodes, q is the degree of the approximation, L is the maximum tree level during the adaptive tree refinement, and N_{dof} is the total number of unknowns. As we scale from 1 compute node to 64 compute nodes, we achieve 17.9× speed up, 28% parallel efficiency for $q = 8$ and 13.4× speed up, 21% parallel efficiency for $q = 14$. The code scales reasonably well up to 16 nodes but then the efficiency drops.*

p	q	L	N_{dof}	Refinement	Semi-Lagrangian				FMM		Total
				T_{ref}	remote%	T_{comm}	T_{sort}	T_{eval} (GFLOPS)	T_{fsetup}	T_{fcomp} (GFLOPS)	T_{solve} (GFLOPS)
1	8	8	2.0E+7	204.3	0.0	0.0	4100.5	923.1 (97.2)	628.6	1080.0 (152)	9094.3 (29)
2	8	8	2.0E+7	687.7	0.2	24.3	1608.2	467.7 (96.0)	332.2	563.6 (145)	4816.3 (27)
4	8	8	2.0E+7	486.3	0.9	40.2	769.6	297.6 (75.4)	193.8	285.0 (144)	2646.9 (25)
8	8	8	2.0E+7	415.1	1.6	49.0	398.7	248.3 (45.2)	218.8	158.4 (130)	1786.8 (19)
16	8	8	2.0E+7	271.2	2.5	61.9	208.6	158.0 (35.5)	206.2	114.9 (90)	1184.4 (14)
32	8	8	2.0E+7	181.2	4.0	63.9	100.6	77.4 (36.2)	153.9	72.9 (71)	739.0 (11)
64	8	8	2.0E+7	129.8	6.0	81.6	52.1	42.5 (33.0)	111.4	53.0 (50)	507.7 (8)
1	14	6	9.4E+6	54.1	0.0	0.0	1607.6	1129.0 (150.1)	54.4	380.3 (253)	4277.3 (65)
2	14	6	9.4E+6	157.3	0.4	20.4	741.8	582.1 (145.6)	51.2	209.2 (230)	2314.1 (61)
4	14	6	9.4E+6	128.5	1.4	32.2	354.6	291.2 (145.5)	49.7	115.2 (209)	1244.5 (56)
8	14	6	9.4E+6	120.9	2.5	40.2	171.9	151.0 (140.4)	64.4	72.2 (167)	764.8 (46)
16	14	6	9.4E+6	115.1	4.1	44.1	82.4	79.1 (133.9)	68.5	49.5 (123)	516.6 (35)
32	14	6	9.4E+6	112.2	7.1	54.5	36.7	42.4 (125.0)	55.8	40.6 (75)	374.9 (24)
64	14	6	9.4E+6	99.4	13.4	68.7	16.9	24.3 (108.8)	52.0	37.2 (42)	319.7 (15)

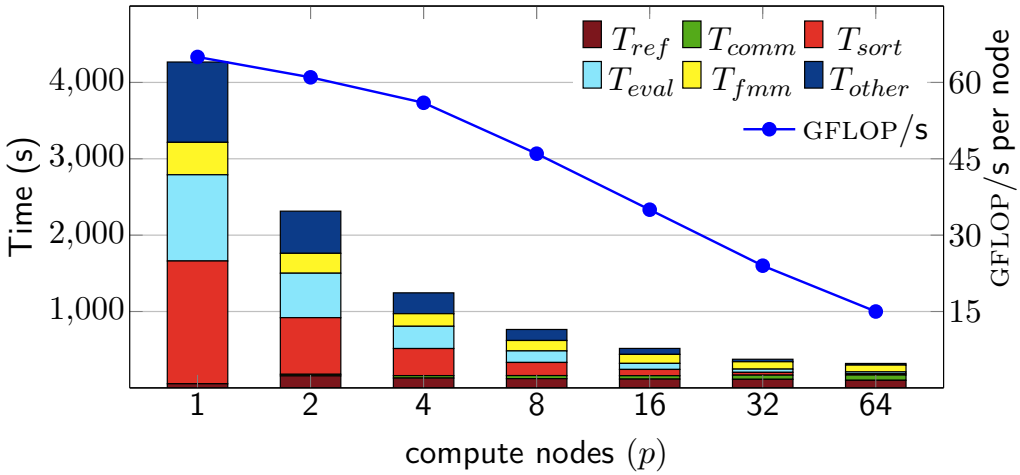


Figure 24: *We report strong scaling results for an advection-diffusion problem with diffusivity* $\mathcal{D} = 1\text{E-}5$ *solved to 4-digits of accuracy. We have used the time step size* $\delta t = 6.25\text{E-}3$ *and 256 time steps. We visualize the results for high order discretizations* ($q = 14$). *We present a detailed breakdown of the total runtime* T_{solve} *into* T_{ref} *(adaptive refinement, repartitioning and merging of trees),* T_{fmm} *(volume FMM computation),* T_{comm}, T_{sort} *and* T_{eval} *(communication, local sorting and Chebyshev evaluation at leaf nodes). As we scale from 1 compute node to 64 compute nodes, we achieve 17.9× speed up, 28% parallel efficiency for* $q = 8$ *and 13.4× speed up, 21% parallel efficiency for* $q = 14$. *The code scales reasonably well up to 16 nodes but then the efficiency drops.*

at the points belonging to that octant. Among these, the local sorting is the most expensive stage. As we scale from 1 node to 64 compute nodes, we observe a 79× speedup for the local sorting. The Chebyshev evaluation phase is the second most expensive stage in our semi-Lagrangian scheme. For this stage, we achieve about 34% parallel efficiency scaling up to 64 nodes. We also report the percentage (remote%) of points which have to be communicated to a different processor for evaluation. This communication cost is reported in T_{comm}. As we increase the number of compute nodes, the combined available network bandwidth increases. However, we also observe that the percentage of points which need to be communicated increases significantly. Consequently, the communication time T_{comm} increases as we increase the number of compute nodes. For 64 compute nodes, T_{comm} accounts for 16% for the total run time. We report the time for diffusion computation (using the volume FMM) in two parts: the FMM setup time (T_{fsetup}) and the actual FMM computation time T_{fcomp}. The FMM setup stage involves constructing the local essential tree, creating interaction lists and allocating memory buffers for the FMM computation. We need to setup FMM whenever the tree refinement changes. In our case, we have to do this in each time step. Some operations in the setup phase are communication intensive. Therefore, the FMM setup phase scales relatively poorly, giving a 5.6× speed up on 64 compute nodes. On the other hand, the computation

phase of FMM is highly optimized and achieves good performance. Overall, the FMM phase ($T_{fsetup} + T_{fcomp}$) of the algorithm accounts for $19\% - 32\%$ of the total time.

In the same table, we have reported similar strong scaling results for $q = 14$. Compared to $q = 8$, we require less than half the number of unknowns (9.4 million) and just 14K leaf octants. Consequently, the local sort time T_{sort} is smaller. The cost for the Chebyshev evaluation per unknown is higher. However, we also achieve higher flop-rates due to higher arithmetic intensity. The Chebyshev evaluation time T_{eval} shows $46.5\times$ speed up (73% parallel efficiency) when scaling to 64 compute nodes. The FMM computation stage also shows higher flop-rates and scales well up to 16 compute nodes. Because we have such a small number of octants and a relatively large CFL number, the tree refinement (T_{ref}), the ghost point communication (T_{comm}) and the FMM setup (T_{fsetup}) show poor scalability due to increasing communication costs.

8.2 TEST CASE: SPHERE IN TAYLOR-GREEN VORTEX FLOW

In this section we report strong and weak scaling results for the advection-diffusion solver up to 1024 compute nodes (16384 cores) of Stampede. The initial concentration is given by $c = \exp(-(r/R)^{\alpha})$, where R is the radius of a sphere and r is the distance from the evaluation point to the center of the sphere (see Figure 16). By increasing α, c develops a sharp gradient around $r = R$. This way, we adjust our problem size only by changing the value of α. Roughly speaking, by doubling α we increase the number of octants by 4. We present results for $\mathcal{D} = 1\text{E-}3$ and $R = 0.1$. The concentration field is placed in a Taylor-green vortex flow visualized in Figure 16.

This problem presents a challenging test for our AMR scheme for several reasons. First, when the advection process dominates, sharp gradients transport in domain which requires frequent refining/coarsening of the domain. After each mesh adaptation step, the work-load needs to be balanced, which requires internode communication. Second, by increasing the granularity as we increase the number of processes in a fixed-size scaling, for a fixed CFL number, the number of Lagrangian points whose values need to be communicated with the remote partitions increases. This is even more amplified in semi-Lagrangian problems with extremely localized refinement. Moreover, we stress our semi-Lagrangian solver by using extremely large CFL numbers.

8.2.1 *Strong Scaling Results*

In this section we present scaling results for two problems with 3.6E+8 and 7.4E+8 unknowns. As the number of cores increases, we keep the problem

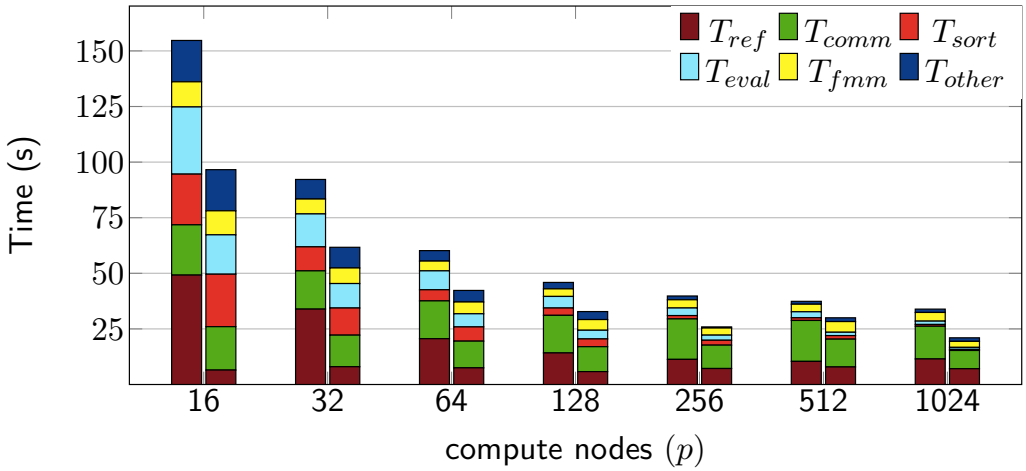


Figure 25: *Strong scaling results with Complete-Merge and Semi-Merge partitioning schemes for the advection-diffusion problem with diffusivity* $\mathcal{D} = 1\text{E-}3$ *with a Taylor-Green vortex flow as the velocity field. We show results for high order discretizations (*$q = 14$ *) for a problem size of* 3.6E+8 *unknowns. We present a detailed breakdown of the total runtime* T_{solve} *into* T_{ref} *(adaptive refinement and repartitioning of trees),* T_{fmm} *(volume FMM computation),* T_{comm}, T_{sort} *and* T_{eval} *(communication, local sorting and Chebyshev evaluation at leaf nodes). The time step* $dt = 6.25\text{E-}3$. *We show the results for one time step.*

size fixed (strong scaling). We require 528K and 1M leaf octants for the medium and large sized problem, respectively. This corresponds to approximately 17K octants per compute node for 32 nodes in the medium sized problem and 64 nodes for the large sized problem. Moreover, we provide scaling results for the Separate-Trees, the Complete-Merge and the Semi-Merge partitioning schemes.

For the medium sized problem, with high order discretizations ($q = 14$), we require 9 levels of refinement. With a time-step size $\delta t = 6.25\text{E-}3$, this corresponds to a very large CFL number of approximately 600. In Table 11, we report the breakdown of total runtime of the this problem into its components: T_{ref} (adaptive refinement and repartitioning of trees), T_{fmm} (volume FMM computation), T_{comm}, T_{sort} and T_{eval} (communication, local sorting and Chebyshev evaluation at leaf nodes). In Figure 26 we also report the percentage of the total runtime for each component of the solver.

On 16 compute nodes, the T_{ref} accounts for 32% of total runtime for the Complete-Merge partitioning scheme, while for the Semi-Merge scheme, the refinement cost is only 6.7%. The refinement and communication costs combined account for 46.5% of the total runtime for the Complete-Merge scheme on 16 compute nodes, while for the Semi-Merge scheme, this cost is reduced by a factor of 2 and accounts for 26%. As we scale from 16 compute nodes to 1024 compute nodes, we observe that the communicaiton cost dominates the runtime. This is due to the unavoidable communication of the positions and values of the Lagrangian particles for such a large CFL

Table 11: *Strong scaling results with complete merge and semi-merge partitioning schemes for the advection-diffusion problem with diffusivity* $\mathcal{D} =$ 1E-3 *with a Taylor-Green vortex flow as a velocity field. We show results for high order discretizations (*$q = 14$ *) for a problem size of* 3.6E+8 *unknowns. Here* p *is the number of processors. We also present a detailed breakdown of the total runtime* T_{solve} *into* T_{ref} *(adaptive refinement and repartitioning of trees),* T_{fmm} *(volume FMM computation),* T_{comm}*,* T_{sort} *and* T_{eval} *(communication, local sorting and Chebyshev evaluation at leaf nodes). The time step* $dt = 6.25E\text{-}3$*. We show the results for one time step.*

p	Merging	Refinement	Semi-Lagrangian			FMM	Total
		T_{ref}	T_{comm}	T_{sort}	T_{eval} (GFLOPS)	T_{fmm} (GFLOPS)	T_{solve} (GFLOPS)
16	CM	49.2	22.6	22.8	30.2 (73.0)	11.3 (97.0)	154.8 (22.7)
	SM	6.5	19.5	23.6	17.7 (124.4)	10.8 (101.8)	96.6 (36.0)
32	CM	33.9	17.2	10.8	14.8 (74.3)	6.7 (81.2)	92.2 (19.0)
	SM	8.0	14.2	12.2	11.0 (99.8)	7.0 (77.8)	61.7 (28.2)
64	CM	20.6	17.0	5.0	8.5 (64.3)	4.4 (62.1)	60.2 (14.6)
	SM	7.5	12.0	6.4	5.9 (92.2)	5.3 (51.6)	42.3 (20.6)
128	CM	14.2	16.9	3.3	5.2 (52.3)	3.4 (40.9)	45.9 (9.6)
	SM	5.8	11.2	3.5	3.9 (70.7)	4.8 (28.8)	32.8 (13.3)
256	CM	11.3	18.2	1.5	3.4 (40.1)	3.7 (18.6)	39.9 (5.5)
	SM	7.2	10.5	2.2	2.3 (58.3)	3.2 (21.7)	27.6 (7.9)
512	CM	10.4	18.4	1.2	2.7 (25.0)	3.4 (10.4)	37.5 (2.9)
	SM	7.9	12.5	1.6	1.5 (44.4)	4.8 (7.4)	30.0 (3.7)
1024	CM	11.5	14.7	0.8	1.5 (21.6)	3.9 (4.7)	34.0 (1.6)
	SM	7.1	8.2	0.5	0.9 (35.7)	2.7 (6.7)	21.0 (2.7)

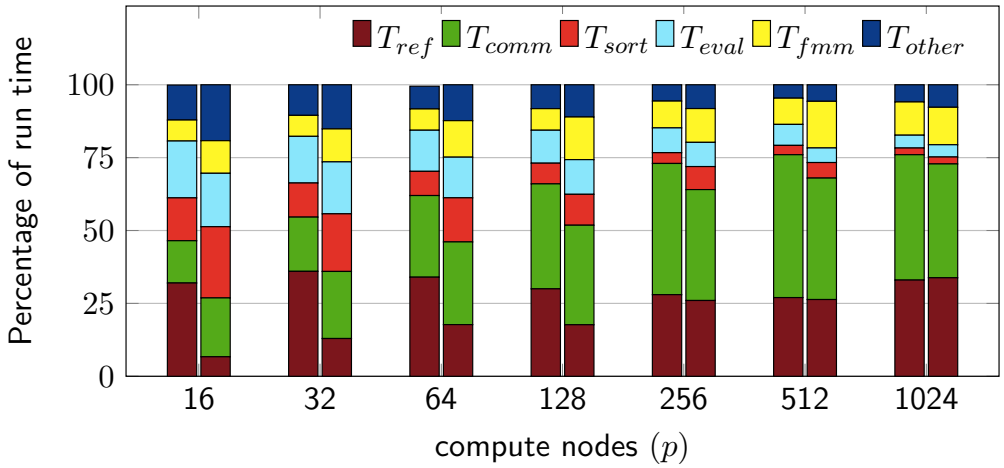


Figure 26: *Strong scalability for the advection-diffuion problem using Complete-Merge (left bars) and Semi-Merge (right bars) partitioning schemes. We report breakdown of the total runtime into different components with increasing number of cores from 256 to 16384. The overall runtime is dominated by communication cost, which is due to the unavoidable communication cost of the semi-Lagrangian scheme for very large CFL number.*

number. In other words, by increasing the partitioning granularity in strong scaling experiments, the number of semi-Lagrangian remote points whose values are not locally available increases. For a fixed time-step size, this results in higher internode communication for a larger number of compute nodes. Moreover, the communication pattern in semi-Lagrangian scheme is unstructured and depends on the features of the velocity field. Therefore for a distributed-memory semi-Lagrangian scheme, the communication pattern may significantly vary depending on the test problem.

The local sorting is the next expensive component of our scheme. However, this component scales reasonably well. As we scale from 16 to 1024 compute nodes, the cost of local sorting drops 28.5× for the Complete-Merge and 47.2× for the Semi-Merge scheme. This corresponds to 44% and 73% strong scaling parallel efficiency for the Complete-Merge and Semi-Merge partitioning schemes, respectively. Recall that we use q^3 actual points for the semi-Lagrangian, so the problem size for the local sorting, chebyshev interpolation and communication is six times larger than N_{dof}.

On 16 compute nodes the Chebyshev evaluation part accounts for 19.5% of the total runtime for the Complete-Merge scheme. This drops to 4.4% of the total time, as we scale to 1024 compute nodes (31% parallel efficiency). For the evaluation component of the Semi-Merge scheme, we observe similar parallel efficiency.

In Table 12 we report the strong scaling results for a 2× larger problem with 7.4E+8 unknowns. For this problem, we require 10 levels of refinement and scale from 32 to 1024 compute nodes. For 32 compute nodes, the Complete-Merge scheme fails due to excessive memory consumption

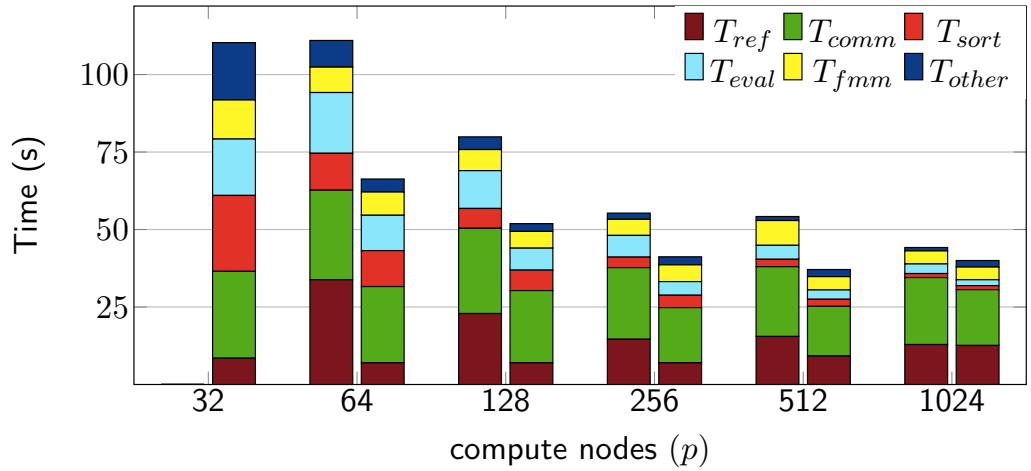


Figure 27: *Strong scaling results with Complete-Merge and Semi-Merge partitioning scheme for the advection-diffusion problem with diffusivity* $\mathcal{D} = 1\text{E-}3$ *with a Taylor-Green vortex flow as a velocity field. We show results for high order discretizations* ($q = 14$) *for a problem size of* 7.4E+8 *unknowns. Here* p *is the number of processors. We also present a detailed breakdown of the total runtime* T_{solve} *into* T_{ref} *(adaptive refinement and repartitioning of trees),* T_{fmm} *(volume FMM computation),* T_{comm}, T_{sort} *and* T_{eval} *(communication, local sorting and Chebyshev evaluation at leaf nodes). The time step* $dt = 6.25\text{E-}3$. *We show the results for one time step.*

while the Semi-Merge scheme performs well. We observe that for 64 compute nodes the Semi-Merge scheme is 1.6× faster than the Complete-Merge approach. However, by increasing the number of compute nodes, the performance gap between the two schemes decreases. This is due to the excessive communication volume of the semi-Lagrangian scheme with such a large CFL number.

Overall, considering the small number of octants per node, the extremely high CFL number and the challenging test for AMR, where the solution develops sharp gradients and requires very high locallized refinement, the scalability of our solver is quite good. As can be clearly observed from the results presented in Tables 11 and 12, the Semi-Merge partitioning is quite efficient compared to the Complete-Merge approach and drastically reduced the communication and refinement costs, while the Separate-Trees approach failed for most of the runs due to excessive memory consumption caused by severe load imbalance.

8.2.2 *Weak Scaling Results*

In this section we compare the isogranular scalability of the Complete-Merge and the Semi-Merge partitioning scheme for the same test case as above. As we increase the number of compute nodes from 32 to 1024, the problem size increases from 47 million unknowns to 1.4 billion unknowns,

Table 12: *Strong scaling results with Complete-Merge and Semi-Merge partitioning scheme for the advection-diffusion problem with diffusivity* $\mathcal{D} =$ 1E-3 *with a Taylor-Green vortex flow as a velocity field. We show results for high order discretizations* ($q = 14$) *for a problem size of* 7.4E+8 *unknowns. Here* p *is the number of processors. We also present a detailed breakdown of the total runtime* T_{solve} *into* T_{ref} *(adaptive refinement and repartitioning of trees),* T_{fmm} *(volume FMM computation),* T_{comm}*,* T_{sort} *and* T_{eval} *(communication, local sorting and Chebyshev evaluation at leaf nodes). The time step* $dt = 6.25\text{E-}3$*. We show the results for one time step.*

p	Merging	Refinement	Semi-Lagrangian			FMM	Total
		T_{ref}	T_{comm}	T_{sort}	T_{eval} (GFLOPS)	T_{fmm} (GFLOPS)	T_{solve} (GFLOPS)
32	CM	–	–	–	(–)	– (–)	– (–)
32	SM	8.5	28.0	24.5	18.2 (125.4)	12.6 (89.5)	110.3 (32.5)
64	CM	33.8	28.9	11.9	19.6 (57.9)	8.2 (69.0)	111.0 (16.3)
64	SM	7.0	24.6	11.6	11.4 (99.8)	7.5 (74.9)	70.4 (25.4)
128	CM	22.9	27.5	6.4	12.2 (46.4)	6.8 (41.8)	79.9 (11.3)
128	SM	7.0	23.3	6.6	7.1 (79.3)	5.4 (52.4)	53.2 (16.9)
256	CM	14.6	23.1	3.4	7.0 (40.4)	5.2 (27.3)	55.4 (8.2)
256	SM	7.0	17.8	4.0	4.4 (64.3)	5.4 (26.5)	41.2 (10.9)
512	CM	15.5	22.5	2.4	4.5 (31.1)	8.0 (9.0)	54.2 (4.2)
512	SM	9.2	16.0	2.3	3.0 (46.7)	4.3 (16.5)	37.1 (6.1)
1024	CM	12.9	21.6	1.3	3.1 (22.9)	4.2 (8.7)	44.2 (2.6)
1024	SM	12.6	18.0	1.3	1.9 (35.8)	4.1 (8.8)	40.0 (2.8)

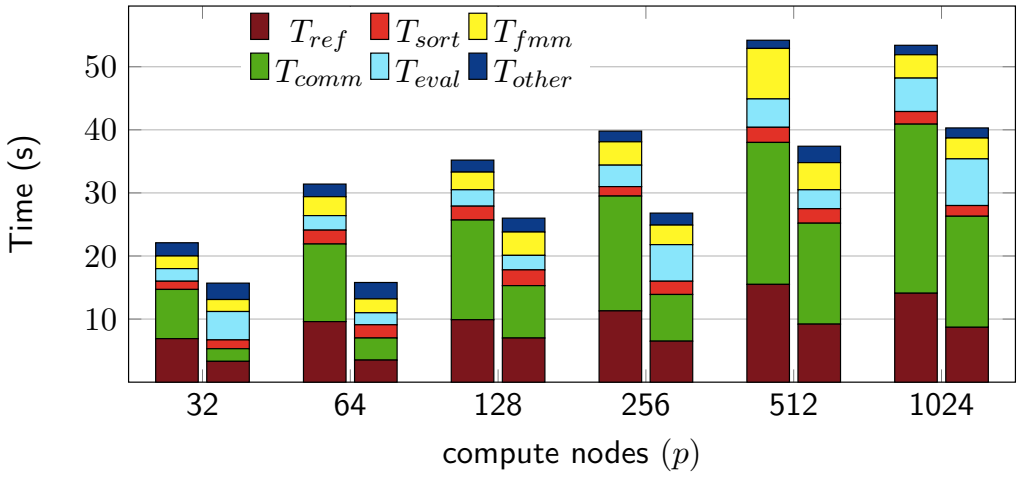


Figure 28: *Here we present weak scaling results for the same advection-diffusion problem as in Figure 27. We present a detailed breakdown of the total runtime* $\mathsf{T}_{\mathtt{solve}}$ *in to* $\mathsf{T}_{\mathtt{ref}}$ *(adaptive refinement and repartitioning of trees),* $\mathsf{T}_{\mathtt{fmm}}$ *(volume FMM computation),* $\mathsf{T}_{\mathtt{comm}}$, $\mathsf{T}_{\mathtt{sort}}$ *and* $\mathsf{T}_{\mathtt{eval}}$ *(communication, local sorting and Chebyshev evaluation at leaf nodes). We vary the problem size from* 4.7E+7 *unknowns on 32 compute nodes of Stampede to* 1.4E+9 *unknowns on 1024 compute nodes. The problem size per node remains roughly constant to about 2,000 octants per node. Recall that we use 1 MPI task per node and 16 OpenMP threads in all of our runs.*

an increase of almost 31×. Nevertheless, we show that the overall timings increase only by a factor of two. For the isogranular scaling, we keep the problem size approximately constant at 2100 octants per compute node. The time-step size is $\delta t = 6.25\text{E-}3$. That is, for the smallest problem, which requires 8 levels of tree refinement, the CFL number is approximately 300. For the biggest problem with 10 levels of refinement the CFL number exceeds 1000.

Figure 29 and Table 13 provide the breakdown of the overall runtime into the time consumed by adaptive refinement and repartitioning of trees ($\mathsf{T}_{\mathtt{ref}}$), volume FMM computation ($\mathsf{T}_{\mathtt{fmm}}$), communication ($\mathsf{T}_{\mathtt{comm}}$), local sorting ($\mathsf{T}_{\mathtt{sort}}$) and Chebyshev evaluation at leaf nodes ($\mathsf{T}_{\mathtt{eval}}$). The adaptive refinement and partitioning cost for the Complete-Merge is roughly 2× more expensive than the Semi-Merge scheme and accounts for $26\% - 31\%$ of the overall runtime for the Complete-Merge, while for the Semi-Merge this is reduced to $21\% - 24\%$. For 32 compute nodes, the communication cost for the Complete-Merge scheme accounts for 35% of the overall runtime while for the the Semi-Merge scheme, this is reduced to 12%. Overall, communication in the Complete-Merge scheme is up to 4× more expensive than in the Semi-Merge scheme. The cost of the local sorting varies from 3.7% to 7% of the total runtime for the Complete-Merge and 4% to 13% for the Semi-Merge scheme. On 32 compute nodes, the $\mathsf{T}_{\mathtt{fmm}}$ and $\mathsf{T}_{\mathtt{eval}}$ make up for approximately 9% of the of total runtime with Complete-Merge partitioning

Table 13: *Here we present weak scaling results for the same advection-diffusion problem as in Table 12. Here* p *is the number of processors,* q *is the degree of the approximation,* L *is the maximum tree level during the adaptive tree refinement, and* N_{dof} *is the total number of unknowns. We also present a detailed breakdown of the total runtime* T_{solve} *in to* T_{ref} *(adaptive refinement and repartitioning of trees),* T_{fmm} *(volume FMM computation),* T_{comm}*,* T_{sort} *and* T_{eval} *(communication, local sorting and Chebyshev evaluation at leaf nodes). We vary the problem size from* 4.7E+7 *unknowns on 32 compute nodes of Stampede to* 1.4E+9 *unknowns on 1024 compute nodes. The problem size per node remains roughly constant to about 2,000 octants per node. Recall that we use 1 MPI task per node and 16 OpenMP threads in all of our runs.*

p	Merging	L	N_{dof}	Refinement	Semi-Lagrangian			FMM	Total
				T_{ref}	T_{comm}	T_{sort}	T_{eval} (GFLOPS)	T_{fmm} (GFLOPS)	T_{solve} (GFLOPS)
32	CM	8	4.7E+7	6.9	7.8	1.3	2.0 (71.9)	2.0 (36.4)	22.1 (10.4)
	SM	8	4.7E+7	3.3	2.0	1.4	4.5 (32.0)	1.9 (38.0)	15.7 (14.6)
64	CM	9	9.3E+7	9.6	12.3	2.2	2.3 (61.9)	3.0 (24.1)	31.4 (7.3)
	SM	9	9.3E+7	3.5	3.5	2.1	1.9 (72.3)	2.2 (33.3)	15.8 (14.5)
128	CM	9	1.8E+8	9.9	15.8	2.2	2.6 (54.8)	2.8 (25.8)	35.2 (6.5)
	SM	9	1.8E+8	7.0	8.3	2.5	2.3 (60.6)	3.7 (19.5)	26.0 (8.7)
256	CM	9	3.6E+8	11.3	18.2	1.5	3.4 (40.1)	3.7 (18.6)	39.9 (5.5)
	SM	9	3.6E+8	6.5	7.4	2.1	5.8 (23.5)	3.1 (22.2)	26.8 (8.2)
512	CM	10	7.4E+8	15.5	22.5	2.4	4.5 (31.1)	8.0 (9.0)	54.2 (4.2)
	SM	10	7.4E+8	9.2	16.0	2.3	3.0 (46.7)	4.3 (16.5)	37.1 (6.1)
1024	CM	10	1.4E+9	14.1	26.8	2.0	5.3 (25.4)	3.7 (18.4)	53.5 (4.1)
	SM	10	1.4E+9	8.7	17.6	1.7	7.4 (18.3)	3.3 (20.6)	40.3 (5.4)

scheme. For the Semi-Merge scheme, T_{fmm} accounts for $8\% - 14\%$ and T_{eval} up to 28% of the total runtime. The Chebyshev evaluation and local sorting are slightly more costly in the Semi-Merge scheme compared to Complete-Merge scheme. However, as we explained above, the communication cost in the Complete-Merge is up to $4\times$ more expensive. As a result, the Semi-Merge scheme is overall up to $2\times$ faster for the weak scaling experiments we conducted in this section.

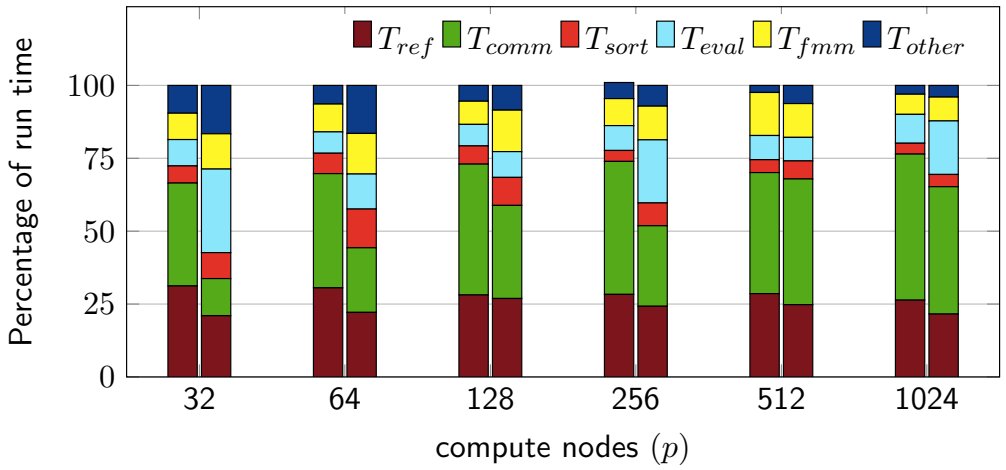


Figure 29: *Weak scalability for the advection-diffuion problem using Complete-Merge (left bars) and Semi-Merge (right bars) partitioning schemes. We report the breakdown of the total runtime into different components with increasing number of cores from* 512 *to* 16384*. The overall runtime is dominated by the communication cost. This is due to the very large CFL number.*

8.3 SUMMARY

In this chapter we studied fixed-size and isogranular scaling results of our AMR scheme with small, medium and large sized test problems for several challenging flows, which require extremely localized refinement to be resolved accurately. We showed in various experiments that our novel Semi-Merge partitioning scheme is quite efficient compared to the Complete-Merge and Separate-Trees approaches and can reduce the communication and refinement costs by orders of magnitude. For our largest run, we were able to solve a problem with 1.4 billion unknowns on a tree with maximum depth equal to 10 and using 14th-order elements on 16,384 cores.

9

CONCLUSION AND FUTURE WORK

In this dissertation we proposed parallel numerical algorithms for solving the advection-diffusion and the incompressible Navier–Stokes equations, which arise in many science and engineering applications simulating the fluid flows and transport phenomena in complex fluids. Solving these problems presents various challenges, which require novel approaches, both in terms of numerics and HPC. For example, in many problems adaptive non-uniform grids are required to accurately resolve spatio-temporal features of interest with minimum computational cost. Moreover, stability considerations impose severe constraints on the time-step sizes in regions that require high spatial resolution. Thus, large-scale and accurate simulation of these phenomena for complex realistic scenarios requires high-order numerical methods implemented with parallel and efficient algorithms with optimal complexity and support of scalable AMR methods.

In the following we summarize the main contributions of this dissertation, which address the above stated challenges, and their beneficial properties:

METHODOLOGY Our methodology involves combining the method of characteristics with the volume integral method by using an explicit-implicit time integration method. To alleviate the stability constraints imposed by the CFL condition, we treated the advective term with a second-order accurate and unconditionally stable semi-Lagrangian scheme. By so doing, we transformed the PDE of interest to a constant-coefficient elliptic PDE, which we implicitly solve with a volume integral formulation. Combing the semi-Lagrangian method with the volume integral method leads to an unconditionally stable scheme which allows large CFL numbers with no loss in accuracy. Moreover, adaptive quadrature rule to compute the volume integrals is relatively straightforward to implement by using a hierarchical domain decomposition.

OPTIMAL ALGORITHMS Discretization of volume integrals results in dense linear systems. We addressed this issue by using the PVFMM library, which computes the volume integrals with optimal complexity by exploiting the Kernel Independent Fast Multipole Method (KIFMM). By using KIFMM, we separate the near and far interactions and approximate the far interactions in a hierarchical manner with arbitrary precision in space with provable a priori error estimates.

HIGH-ORDER ACCURACY For our spatial discretization, we used an arbitrary-order accurate piecewise Chebyshev octree representation of the field

values on the computational domain. We used this spatial discretization for both the semi-Lagrangian and the volume integral solvers, thus simplifying the coupling of the two components.

ADAPTIVE MESH REFINEMENT/COARSENING For many problems, high spatial resolutions are required only for particular regions of the simulation domain. We extended our solver with dynamic AMR algorithms to accurately resolve the spatio-temporal features of interest at these regions with minimal number of unknowns. We showed examples, where by using the AMR algorithm, the number of unknowns for a computation with a fixed target accuracy was reduced by orders of magnitude compared to the same computation using a uniform grid and thus we obtained up to $10\times$ shorter time-to-solution.

NUMERICAL EXPERIMENTS By combining the semi-Lagrangian method with the volume integral method and using hierarchical domain decomposition and accelerated algorithms such as the KIFMM, we obtained an unconditionally stable, arbitrary-order accurate and fast alternative to conventional PDE-based approaches such as finite difference, finite element or the spectral methods. We showed the convergence of the scheme in various experiments using complex velocity fields and well-known benchmark problems. We found that it is critical for stability to have points right on the boundary between different octants and to filter the Chebyshev coefficients. We showed that once these modifications are in place the scheme is stable and accurate.

HIGH PERFORMANCE COMPUTING We developed efficient algorithms to extend the scheme to parallel octrees with dynamic load-balancing support. This was achieved by dynamically partitioning a Morton-ordered space-filling curve.

Novel Partitioning Scheme: For cases where the velocity field and the concentration field have different spatial scales and require separate discretization and thus independent distributed-memory partitioning, we proposed a novel partitioning scheme that efficiently merges the two independent trees to define an upper-bound for the communication cost with minimal increase in the computational cost. We showed examples, where we reduced the communication cost up to $20\times$ and achieved a $3\times$ speedup of the total runtime by just applying our novel partitioning scheme introduced in this dissertation.

Performance Analysis: We studied the single-node performance, strong scaling, and weak scaling of our scheme for several challenging flows that cannot be resolved efficiently without using high-order accurate discretizations. In our single-node performance analysis,

> we observed that using high order discretization results in significantly fewer unknowns and therefore, faster time-to-solution. We constructed simple examples in which a 4th-order discretization is at least two orders of magnitude slower than our 14th-order scheme. We demonstrated the scalability of our scheme for up to thousands of cores for hard cases with high levels of refinement, and with remeshing and repartitioning at every time step. For our largest run, we solve a problem with 1.4 billion unknowns in 40 seconds per time-step on a tree with maximum depth equal to 10 and using 14th-order elements on 16,384 cores. This is an effective resolution of nearly 100 billion unknowns with a uniform mesh.

By using the technologies developed in this dissertation, we were able to simulate compute-intensive realistic scenarios with complex geometries such as the transport phenomena in porous medium with highly complex pore structure.

FUTURE WORK The main focus of this dissertation was to develop parallel, scalable, arbitrary-order accurate and unconditionally stable solvers for the advection-diffusion and the incompressible Navier–Stokes equations by combining the method of the characteristics with the volume integral formulation of the elliptic PDEs. However, in order to be able to apply this machinery for more general problems, extending the scheme to complex geometries and more sophisticated boundary conditions are required.

Future directions of this thesis also include using higher order temporal integration methods, which can reduce the number of time-steps required for a fixed target accuracy. This can be achieved by extending our temporal integration method to higher order Backward Differentiation Formula (BDF) methods. However, by using the BDF method combined with the semi-Lagrangian scheme several issues might arise: First, the BDF method requires information from previous time steps at the departure points of the Lagrangian particles. Therefore depending on the order of the scheme, the solution at multiple time-steps need to be stored for later use. This can significantly increase the memory footprint of the solver for large problems. Second, to achieve higher order in time, we also require higher order backward time integration of the characteristics. Higher order time integrators for the characteristics require more off-grid evaluation of the velocity fields and thus increase the communication and the computation cost in a distributed-memory context. In order to gain better time-to-solution by using higher order time integrators, the time needed for higher order methods in each time step should remain small so that it does not balance out the gains accrued by solving with lower number of time-steps. This requires efficient and careful algorithmic design and implementation.

Besides implementing high order temporal integration methods, extending the scheme to adaptive time-stepping can also reduce the number of time-steps for time-varying velocity fields and thus further improve time-to-solution.

Moreover, the stability of the nonlinear convective term in the Navier–Stokes equations requires further numerical investigation. The scheme can also be extended to problems with variable coefficients, i. e. to allow for a variable (but smooth) diffusion coefficient. The main difference will be in the elliptic solve, where we would need to solve a volume integral equation instead of simply convolving with the Green's function.

BIBLIOGRAPHY

[1] A. Allievi and R. Bermejo, "Finite element modified method of characteristics for the navier-stokes equations," *International journal for numerical methods in fluids*, vol. 32, no. 4, pp. 439–463, 2000, ISSN: 1097-0363.

[2] A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome, "A conservative adaptive projection method for the variable density incompressible navier-stokes equations," *Journal of computational physics*, vol. 142, no. 1, pp. 1–46, 1998, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.1998.5890.

[3] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. Van Andel, "Nyx: A massively parallel amr code for computational cosmology," *The astrophysical journal*, vol. 765, no. 1, p. 39, 2013.

[4] M. Bader, C. Boeck, J. Schwaiger, and C. Vigh, "Dynamically adaptive simulations with minimal memory requirement-solving the shallow water equations using sierpinski curves," *Siam journal on scientific computing*, vol. 32, no. 1, pp. 212–228, 2010. DOI: 10.1137/080728871. eprint: http://dx.doi.org/10.1137/080728871. [Online]. Available: http://dx.doi.org/10.1137/080728871.

[5] J. R. Bates and A. McDonald, "Multiply-upstream, semi-lagrangian advective schemes: Analysis and application to a multi-level primitive equation model," *Monthly weather review*, vol. 110, no. 12, 1982. DOI: 10.1175/1520-0493(1982)110<1831:MUSLAS>2.0.CO;2. eprint: http://dx.doi.org/10.1175/1520-0493(1982)110<1831:MUSLAS>2.0.CO;2. [Online]. Available: http://dx.doi.org/10.1175/1520-0493(1982)110%3C1831:MUSLAS%3E2.0.CO;2.

[6] J. Benqué, G. Labadie, and J. Ronat, "A new finite element method for navier-stokes equations coupled with a temperature equation," in *Finite element flow analysis*, 1982, pp. 295–302.

[7] M. J. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," *Journal of computational physics*, vol. 82, no. 1, pp. 64–84, 1989.

[8] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of computational physics*, vol. 53, no. 3, pp. 484–512, 1984, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/0021-9991(84)90073-1. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0021999184900731.

[9] M. Bern, D. Eppstein, and S.-H. Teng, "Parallel construction of quadtrees and quality triangulations," in *Workshop on algorithms and data structures*, Springer, 1993, pp. 188–199.

[10] M. Bern and P. Plassmann, "Mesh generation," in *Handbook of computational geometry. elsevier science*, 2000, pp. 291–332.

[11] G. Biros and D. Malhotra, "PVFMM: A parallel kernel independent FMM for particle and volume potentials," *Communications in computational physics*, vol. 18, no. 3, pp. 808–830, 2015. DOI: 10.4208/cicp.020215.150515sw.

[12] G. Biros, L. Ying, and D. Zorin, "The embedded boundary integral method (ebi) for the incompressible navier-stokes equations," 2002.

[13] M. E. Brachet, D. I. Meiron, S. A. Orszag, B. Nickel, R. H. Morf, and U. Frisch, "Small-scale structure of the taylor–green vortex," *Journal of fluid mechanics*, vol. 130, pp. 411–452, 1983.

[14] G. L. Bryan, T. Abel, and M. L. Norman, "Achieving extreme resolution in numerical cosmology using adaptive mesh refinement: Resolving primordial star formation," in *Supercomputing, acm/ieee 2001 conference*, Nov. 2001, pp. 13–13. DOI: 10.1145/582034.582047.

[15] G. L. Bryan, M. L. Norman, *et al.*, "Enzo: An adaptive mesh refinement code for astrophysics," *The astrophysical journal supplement series*, vol. 211, no. 2, p. 19, 2014.

[16] H.-J. Bungartz and M. Schäfer, *Fluid-structure interaction: Modelling, simulation, optimisation*. Springer Science & Business Media, 2006, vol. 53.

[17] C. Burstedde, O. Ghattas, M. Gurnis, T. Isaac, G. Stadler, T. Warburton, and L. Wilcox, "Extreme-scale amr," in *Proceedings of the 2010 acm/ieee international conference for high performance computing, networking, storage and analysis*, IEEE Computer Society, 2010, pp. 1–12.

[18] C. Burstedde, O. Ghattas, M. Gurnis, G. Stadler, E. Tan, T. Tu, L. C. Wilcox, and S. Zhong, "Scalable adaptive mantle convection simulation on petascale supercomputers," in *Proceedings of the 2008 acm/ieee conference on supercomputing*, IEEE Press, 2008, p. 62.

[19] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox, "Towards adaptive mesh pde simulations on petascale computers," *Proceedings of teragrid*, vol. 8, 2008.

[20] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox, "Parallel scalable adjoint-based adaptive solution of variable-viscosity stokes flow problems," *Computer methods in applied mechanics and engineering*, vol. 198, no. 21-26, pp. 1691–1700, 2009, Advances in Simulation-Based Engineering Sciences - Honoring J. Tinsley Oden, ISSN: 0045-7825. DOI: http://dx.doi.org/10.1016/j.cma.2008.12.015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045782509000085.

[21] P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco, "Dynamic octree load balancing using space-filling curves," 2003.

[22] B. A. Cipra, "The best of the 20th century: Editors name top 10 algorithms," *Siam news*, vol. 33, no. 4, pp. 1–2,

[23] L. Diachin, R. Hornung, P. Plassmann, and A. Wissink, "Parallel adaptive mesh refinement," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2005.

[24] A. Dubey, A. Almgren, *et al.*, "A survey of high level frameworks in block-structured adaptive mesh refinement packages," *Journal of parallel and distributed computing*, vol. 74, no. 12, pp. 3217–3227, 2014.

[25] M. G. Duffy, "Quadrature over a pyramid or cube of integrands with a singularity at a vertex," *Siam journal on numerical analysis*, vol. 19, no. 6, pp. 1260–1262, 1982. DOI: 10.1137/0719090. eprint: http://dx.doi.org/10.1137/0719090. [Online]. Available: http://dx.doi.org/10.1137/0719090.

[26] G. K. El Khoury, P. Schlatter, A. Noorani, P. F. Fischer, G. Brethouwer, and A. V. Johansson, "Direct numerical simulation of turbulent pipe flow at moderately high Reynolds numbers," *Flow, turbulence and combustion*, vol. 91, no. 3, pp. 475–495, 2013.

[27] F. Ethridge and L. Greengard, "A new fast-multipole accelerated poisson solver in two dimensions," *Siam j. sci. comput*, pp. 741–760, 2001.

[28] M. Falcone and R. Ferretti, "Convergence analysis for a class of high-order semi-lagrangian advection schemes," *Siam journal on numerical analysis*, vol. 35, no. 3, pp. 909–940, 1998.

[29] ——, *Semi-lagrangian approximation schemes for linear and hamilton-jacobi equations*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2013. DOI: 10.1137/1.9781611973051. eprint: http://epubs.siam.org/doi/pdf/10.1137/1.9781611973051. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611973051.

[30] G. Folland, *Introduction to partial differential equations*. Princeton University Press, 1995, ISBN: 9780691043616. [Online]. Available: https://books.google.de/books?id=c0WFd3X%5C_R20C.

[31] H. Foysi and S. Sarkar, "The compressible mixing layer: An les study," *Theoretical and computational fluid dynamics*, vol. 24, no. 6, pp. 565–588, 2010, ISSN: 1432-2250. DOI: 10.1007/s00162-009-0176-8. [Online]. Available: http://dx.doi.org/10.1007/s00162-009-0176-8.

[32] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo, "Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes," *The astrophysical journal supplement series*, vol. 131, no. 1, p. 273, 2000. [Online]. Available: http://stacks.iop.org/0067-0049/131/i=1/a=273.

[33] Y. Fu and G. J. Rodin, "Fast solution method for three-dimensional stokesian many-particle problems," *Communications in numerical methods in engineering*, vol. 16, no. 2, pp. 145–149, 2000, ISSN: 1099-0887. DOI: 10.1002/(SICI)1099-0887(200002)16:2<145::AID-CNM323>3.0.CO;2-E.

[34] G. J. Gassner and A. D. Beck, "On the accuracy of high-order discretizations for underresolved turbulence simulations," *Theoretical and computational fluid dynamics*, vol. 27, no. 3, pp. 221–237, 2013, ISSN: 1432-2250. DOI: 10.1007/s00162-011-0253-7. [Online]. Available: http://dx.doi.org/10.1007/s00162-011-0253-7.

[35] C. W. Gear, *Numerical initial value problems in ordinary differential equations*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1971, ISBN: 0136266061.

[36] E. Germaine, L. Mydlarski, and L. Cortelezzi, "3D FLUX: A high-order fully three-dimensional flux integral solver for the scalar transport equation," *Journal of computational physics*, vol. 240, pp. 121–144, 2013.

[37] F. X. Giraldo, "The lagrange-galerkin spectral element method on unstructured quadrilateral grids," *Journal of computational physics*, vol. 147, no. 1, pp. 114–146, 1998, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.1998.6078. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999198960788.

[38] L. Greengard, *The rapid evaluation of potential fields in particle systems*. MIT press, 1988.

[39] L. Greengard and M. C. Kropinski, "An integral equation approach to the incompressible navier–stokes equations in two dimensions," *Siam journal on scientific computing*, vol. 20, no. 1, pp. 318–336, 1998. DOI: 10.1137/S1064827597317648. eprint: http://dx.doi.org/10.1137/S1064827597317648. [Online]. Available: http://dx.doi.org/10.1137/S1064827597317648.

[40] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of computational physics*, vol. 135, no. 2, pp. 280–292, 1997, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.1997.5706. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999197957065.

[41] A. Guittet, M. Theillard, and F. Gibou, "A stable projection method for the incompressible navier-stokes equations on arbitrary geometries and adaptive quad/octrees," *Journal of computational physics*, vol. 292, pp. 215–238, 2015.

[42] E. Hairer, C. Lubich, and G. Wanner, *Geometric numerical integration: Structure-preserving algorithms for ordinary differential equations*, ser. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2006, ISBN: 9783540306665. [Online]. Available: https://books.google.de/books?id=T1TaNRLmZv8C.

[43] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji, "42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence," in *Proceedings of the conference on high performance computing networking, storage and analysis*, ser. SC '09, Portland, Oregon: ACM, 2009, 62:1–62:12, ISBN: 978-1-60558-744-8. DOI: 10.1145/1654059.1654123. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654123.

[44] J. P. Huffenus and D. Khaletzky, "A finite element method to solve the navier-stokes equations using the method of characteristics," *International journal for numerical methods in fluids*, vol. 4, no. 3, pp. 247–269, 1984, ISSN: 1097-0363. DOI: 10.1002/fld.1650040304. [Online]. Available: http://dx.doi.org/10.1002/fld.1650040304.

[45] W. Hundsdorfer and J. Verwer, *Numerical solution of time-dependent advection-diffusion-reaction equations*, ser. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2010. [Online]. Available: https://books.google.de/books?id=l5kEkgAACAAJ.

[46] K. Hutter, *Theoretical glaciology: Material science of ice and the mechanics of glaciers and ice sheets*, ser. Mathematical Approaches to Geophysics. Springer, 1983, ISBN: 9789027714732. [Online]. Available: https://books.google.de/books?id=75kqTGNKV9wC.

[47] J. Jim Douglas and T. F. Russell, "Numerical methods for convection-dominated diffusion problems based on combining the method of characteristics with finite element or finite difference procedures," *Siam journal on numerical analysis*, vol. 19, no. 5, 1982. DOI: 10.1137/0719063. eprint: http://dx.doi.org/10.1137/0719063. [Online]. Available: http://dx.doi.org/10.1137/0719063.

[48] M. T. Jones and P. E. Plassmann, "Adaptive refinement of unstructured finite-element meshes," *Finite elements in analysis and design*, vol. 25, no. 1, pp. 41–60, 1997, ISSN: 0168-874X. DOI: http://dx.doi.org/10.1016/S0168-874X(96)00039-X. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168874X9600039X.

[49] B. Kaoui, G. Biros, and C. Misbah, "Why do red blood cells have asymmetric shapes even in a symmetric flow?" *Phys. rev. lett.*, vol. 103, p. 188 101, 18 Oct. 2009. DOI: 10.1103/PhysRevLett.103.188101. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevLett.103.188101.

[50] G. E. Karniadakis, M. Israeli, and S. A. Orszag, "High-order splitting methods for the incompressible navier-stokes equations," *Journal of computational physics*, vol. 97, no. 2, pp. 414–443, 1991, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/0021-9991(91)90007-8. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0021999191900078.

[51] D. E. Keyes, L. C. McInnes, C. Woodward, *et al.*, "Multiphysics simulations," *The international journal of high performance computing applications*, vol. 27, no. 1, pp. 4–83, 2013. DOI: 10.1177/1094342012468181. eprint: http://dx.doi.org/10.1177/1094342012468181. [Online]. Available: http://dx.doi.org/10.1177/1094342012468181.

[52] R. I. Klein, J. Bell, R. Pember, and T. Kelleher, "Three dimensional hydrodynamic calculations with adaptive mesh refinement of the evolution of rayleigh taylor and richtmyer meshkov instabilities in converging geometry: Multi-mode perturbations," Lawrence Livermore National Lab., CA (United States), Tech. Rep., 1993.

[53] J.-B. Lagaert, G. Balarac, and G.-H. Cottet, "Hybrid spectral-particle method for the turbulent transport of a passive scalar," *Journal of computational physics*, vol. 260, pp. 127–142, 2014.

[54] H. Langston, L. Greengard, and D. Zorin, "A free-space adaptive fmm-based pde solver in three dimensions," *Communications in applied mathematics and computational science*, vol. 6, no. 1, pp. 79–122, 2011.

[55] D. Lanser and J. Verwer, "Analysis of operator splitting for advection-diffusion-reaction problems from air pollution modelling," *Journal of computational and applied mathematics*, vol. 111, no. 1-2, pp. 201–216, 1999, ISSN: 0377-0427. DOI: http://dx.doi.org/10.1016/S0377-0427(99)00143-0. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042799001430.

[56] R. Larson, "The structure and rheology of complex fluids," 1999.

[57] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, "A massively parallel adaptive fast multipole method on heterogeneous architectures," *Commun. acm*, vol. 55, no. 5, pp. 101–109, May 2012, ISSN: 0001-0782. DOI: 10.1145/2160718.2160740. [Online]. Available: http://doi.acm.org/10.1145/2160718.2160740.

[58] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *Ieee transactions on computers*, vol. C-34, no. 10, pp. 892–901, Oct. 1985, ISSN: 0018-9340. DOI: 10.1109/TC.1985.6312192.

[59] K. Lindsay and R. Krasny, "A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow," *Journal of computational physics*, vol. 172, no. 2, pp. 879–907, 2001, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.2001.6862.

[60] J. Makino, T. Fukushige, and M. Koga, "A 1.349 tflops simulation of black holes in a galactic center on grape-6," in *Supercomputing, acm/ieee 2000 conference*, Nov. 2000, pp. 43–43. DOI: 10.1109/SC.2000.10042.

[61] A. V. Malevsky and S. J. Thomas, "Parallel algorithms for semi-lagrangian advection," *International journal for numerical methods in fluids*, vol. 25, no. 4, pp. 455–473, 1997, ISSN: 1097-0363. DOI: 10.1002/(SICI)1097-0363(19970830)25:4<455::AID-FLD572>3.0.CO;2-H. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1097-0363(19970830)25:4%3C455::AID-FLD572%3E3.0.CO;2-H.

[62] D. Malhotra and G. Biros, "PVFMM home page," 2016, [Online]. Available: http://www.pvfmm.org.

[63] D. Malhotra, A. Gholami, and G. Biros, "A volume integral equation stokes solver for problems with variable coefficients," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, IEEE Press, 2014, pp. 92–102.

[64] A. Mang, A. Gholami, and G. Biros, "Distributed-memory large deformation diffeomorphic 3d image registration," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, ser. SC '16, Salt Lake City, Utah: IEEE Press, 2016, 72:1–72:12, ISBN: 978-1-4673-8815-3. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014904.3015001.

[65] A. McDonald, "Accuracy of multiply-upstream semi-lagrangian advective schemes ii," *Monthly weather review*, vol. 115, no. 7, 1987.

[66] O. Meister, K. Rahnema, and M. Bader, "Parallel memory-efficient adaptive mesh refinement on structured triangular meshes with billions of grid cells," *Acm trans. math. softw.*, vol. 43, no. 3, 19:1–19:27, Sep. 2016, ISSN: 0098-3500. DOI: 10.1145/2947668. [Online]. Available: http://doi.acm.org/10.1145/2947668.

[67] Q. Meng and M. Berzins, "Scalable large-scale fluid–structure interaction solvers in the uintah framework via hybrid task-based parallelism algorithms," *Concurrency and computation: Practice and experience*, vol. 26, no. 7, pp. 1388–1407, 2014.

[68] L. Moresi, S. Zhong, and M. Gurnis, "The accuracy of finite element solutions of stokes's flow with strongly varying viscosity," *Physics of the earth and planetary interiors*, vol. 97, no. 1, pp. 83–94, 1996, ISSN: 0031-9201. DOI: http://dx.doi.org/10.1016/0031-9201(96)03163-9. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0031920196031639.

[69] R. L. Orbach, Orbach, and D. E. Keyes, "A science-based case for large-scale simulation," 2003.

[70] C. Pantano and S. Sarkar, "A study of compressibility effects in the high-speed turbulent shear layer using direct simulation," *Journal of fluid mechanics*, vol. 451, p. 329 371, 2002. DOI: 10.1017/S0022112001006978.

[71] O. Pironneau, "On the transport-diffusion algorithm and its applications to the navier-stokes equations," *Numerische mathematik*, vol. 38, no. 3, pp. 309–332, 1982.

[72] P. Ploumhans, G. Winckelmans, J. Salmon, A. Leonard, and M. Warren, "Vortex methods for direct numerical simulation of three-dimensional bluff body flows: Application to the sphere at re=300, 500, and 1000," *Journal of computational physics*, vol. 178, no. 2, pp. 427–463, 2002, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.2002.7035. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002199910297035X.

[73] C. Pozrikidis, *Boundary integral and singularity methods for linearized viscous flow*, ser. Cambridge Texts in Applied Mathematics. Cambridge University Press, 1992, ISBN: 9780521406932. [Online]. Available: https://books.google.de/books?id=qXt5bOqDEgQC.

[74] R. M. Propp, P. Colella, W. Y. Crutchfield, and M. S. Day, "A numerical model for trickle bed reactors," *Journal of computational physics*, vol. 165, no. 2, pp. 311–333, 2000, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.2000.6604. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002199910096604X.

[75] A. Rahimian, I. Lashuk, S. Veerapaneni, *et al.*, "Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures," in *Proceedings of the 2010 acm/ieee international conference for high performance computing, networking, storage and analysis*, ser. SC '10, Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11, ISBN: 978-1-4244-7559-9. DOI: 10.1109/SC.2010.42. [Online]. Available: https://doi.org/10.1109/SC.2010.42.

[76] M. Restelli, L. Bonaventura, and R. Sacco, "A semi-Lagrangian discontinuous Galerkin method for scalar advection by incompressible flows," *Journal of computational physics*, vol. 216, no. 1, pp. 195–215, 2006.

[77] A. Robert, "A semi-lagrangian and semi-implicit numerical integration scheme for the primitive meteorological equations," *J. meteor. soc. japan*, vol. 60, no. 1, 1982.

[78] A. Robert, "A stable numerical integration scheme for the primitive meteorological equations," *Atmosphere-ocean*, vol. 19, no. 1, 1981.

[79] J. S. Sawyer, "A semi-lagrangian method of solving the vorticity advection equation," *Tellus*, vol. 15, no. 4, pp. 336–342, 1963, ISSN: 2153-3490. DOI: 10.1111/j.2153-3490.1963.tb01396.x. [Online]. Available: http://dx.doi.org/10.1111/j.2153-3490.1963.tb01396.x.

[80] F. S. Schranner, J. A. Domaradzki, S. Hickel, and N. A. Adams, "Assessing the numerical dissipation rate and viscosity in numerical simulations of fluid flows," *Computers and fluids*, vol. 114, pp. 84–97, 2015, ISSN: 0045-7930. DOI: http://doi.org/10.1016/j.compfluid.2015.02.011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045793015000481.

[81] P. K. Smolarkiewicz and C. L. Winter, "Pores resolving simulation of darcy flows," *Journal of computational physics*, vol. 229, no. 9, pp. 3121–3133, 2010, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/j.jcp.2009.12.031.

[82] J. Song, C.-C. Lu, and W. C. Chew, "Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects," *Ieee transactions on antennas and propagation*, vol. 45, no. 10, pp. 1488–1493, Oct. 1997, ISSN: 0018-926X. DOI: 10.1109/8.633855.

[83] C. G. Speziale, "On the advantages of the vorticity-velocity formulation of the equations of fluid dynamics," *Journal of computational physics*, vol. 73, no. 2, pp. 476–480, 1987, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/0021-9991(87)90149-5. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0021999187901495.

[84] V. Springel, S. D. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, *et al.*, "Simulations of the formation, evolution and clustering of galaxies and quasars," *Nature*, vol. 435, no. 7042, pp. 629–636, 2005.

[85] A. Staniforth and J. Côté, "Semi-lagrangian integration schemes for atmospheric models-a review," *Monthly weather review*, vol. 119, no. 9, pp. 2206–2223, 1991.

[86] G. Strang, "On the construction and comparison of difference schemes," *Siam journal on numerical analysis*, vol. 5, no. 3, pp. 506–517, 1968. DOI: 10.1137/0705041. eprint: http://dx.doi.org/10.1137/0705041. [Online]. Available: http://dx.doi.org/10.1137/0705041.

[87] H. Sundar, D. Malhotra, and G. Biros, "Hyksort: A new variant of hypercube quicksort on distributed memory architectures," in *Proceedings of the 27th international acm conference on international conference on supercomputing*, ACM, 2013, pp. 293–302.

[88] H. Sundar, R. S. Sampath, and G. Biros, "Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel," *Siam journal on scientific computing*, vol. 30, no. 5, pp. 2675–2708, 2008.

[89] H. Sundar, G. Stadler, and G. Biros, "Comparison of multigrid algorithms for high-order continuous finite element discretizations," *Numerical linear algebra with applications*, vol. 22, no. 4, pp. 664–680, 2015.

[90] M. Sussman, A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome, "An adaptive level set approach for incompressible two-phase flows," *Journal of computational physics*, vol. 148, no. 1, pp. 81–124, 1999, ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.1998.6106. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002199919896106X.

[91] "Subject index," in *Finite element methods for viscous incompressible flows*, ser. Computer Science and Scientific Computing, M. D. Gunzburger, Ed., San Diego: Academic Press, 1989, pp. 265–269, ISBN: 978-0-12-307350-1. DOI: http://dx.doi.org/10.1016/B978-0-12-307350-1.50040-5. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123073501500405.

[92] J. A. Trangenstein, "Multi-scale iterative techniques and adaptive mesh refinement for flow in porous media," *Advances in water resources*, vol. 25, no. 812, pp. 1175–1213, 2002, ISSN: 0309-1708. DOI: http://dx.doi.org/10.1016/S0309-1708(02)00053-2. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0309170802000532.

[93] L. N. Trefethen, *Spectral methods in MATLAB*. Society for Industrial Mathematics, 2000.

[94] H. F. Trotter, "On the product of semi-groups of operators," *Proceedings of the american mathematical society*, vol. 10, no. 4, pp. 545–551, 1959.

[95] P. Tsuji and L. Ying, "A fast directional algorithm for high-frequency electromagnetic scattering," *J. comput. phys.*, vol. 230, no. 14, pp. 5471–5487, Jun. 2011, ISSN: 0021-9991. DOI: 10.1016/j.jcp.2011.02.013.

[96] T. Tu, D. R. O'Hallaron, and O. Ghattas, "Scalable parallel octree meshing for terascale applications," in *Supercomputing, 2005. proceedings of the acm/ieee sc 2005 conference*, Nov. 2005, pp. 4–4. DOI: 10.1109/SC.2005.61.

[97] T. Tu, D. R. O'Hallaron, and O. Ghattas, "Scalable parallel octree meshing for terascale applications," in *Supercomputing, 2005. proceedings of the acm/ieee sc 2005 conference*, IEEE, 2005, pp. 4–4.

[98] W. M. Van Rees, A. Leonard, D. Pullin, and P. Koumoutsakos, "A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high reynolds numbers," *Journal of computational physics*, vol. 230, no. 8, pp. 2794–2805, 2011.

[99] S. Verma, Y. Xuan, and G. Blanquart, "An improved bounded semi-lagrangian scheme for the turbulent transport of passive scalars," *Journal of computational physics*, vol. 272, pp. 1–22, 2014. DOI: http://dx.doi.org/10.1016/j.jcp.2014.03.062.

[100] M. S. Warren and J. K. Salmon, "A parallel hashed oct-tree n-body algorithm," in *Proceedings of the 1993 acm/ieee conference on supercomputing*, ser. Supercomputing '93, Portland, Oregon, USA: ACM, 1993, pp. 12–21, ISBN: 0-8186-4340-4. DOI: 10.1145/169627.169640. [Online]. Available: http://doi.acm.org/10.1145/169627.169640.

[101] A. Wiin-Nielsen, "On the application of trajectory methods in numerical forecasting," *Tellus*, vol. 11, no. 2, pp. 180–196, 1959.

[102] D. Xiu and G. E. Karniadakis, "A semi-Lagrangian high-order method for Navier-Stokes equations," *Journal of computational physics*, vol. 172, no. 2, pp. 658–684, 2001.

[103] L. Ying, G. Biros, and D. Zorin, "A kernel-independent adaptive fast multipole algorithm in two and three dimensions," *Journal of computational physics*, vol. 196, no. 2, pp. 591–626, 2004.

[104] L. Ying, G. Biros, D. Zorin, and H. Langston, "A new parallel kernel-independent fast multipole method," in *Proceedings of the 2003 acm/ieee conference on supercomputing*, ser. SC '03, Phoenix, AZ, USA: ACM, 2003, pp. 14–, ISBN: 1-58113-695-1. DOI: 10.1145/1048935.1050165. [Online]. Available: http://doi.acm.org/10.1145/1048935.1050165.

[105] S. Zhong, D. A. Yuen, and L. N. Moresi, "Numerical methods for mantle convection," *Treatise on geophysics*, vol. 7, pp. 227–252, 2007.