Title
An AMR All-Speed Projection Algorithm in Proto

Author
Gebhart, Christopher Lee

Publication Date
2021

Peer reviewed|Thesis/dissertation

An AMR All-Speed Projection Algorithm in *Proto*

by

Christopher Lee Gebhart

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Phillip Colella, Co-chair
Professor Panayiotis Papadopoulos, Co-chair
Professor Oliver O'Reilly
Professor Per-Olof Persson

Spring 2021

An AMR All-Speed Projection Algorithm in *Proto*

Abstract

An AMR All-Speed Projection Algorithm in *Proto*

by

Christopher Lee Gebhart

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Phillip Colella, Co-chair

Professor Panayiotis Papadopoulos, Co-chair

In this study, we examine the all-speed projection approach to computational gas dynamics in the low-Mach limit. This limit is characterized by slow, non-stiff advective flow accompanied by fast, stiff acoustic motions. The all-speed projection method uses a Helmholtz projection to derive redundant equations in the stiff variables: pressure and potential velocity. The resulting equations are discretized using a high-order finite-volume method and integrated in time using a method of lines formulation and an implicit-explicit (IMEX) additive Runge-Kutta (ARK) method. The all-speed projection approach differs from the "zero-Mach" approach of deriving equations of motion in the asymptotic limit as the Mach number becomes small. The zero-Mach strategy yields a differential/algebraic system which is difficult to manage and also explicitly forbids simulation of acoustic waves.

We introduce adaptive mesh refinement (AMR) to the existing all-speed projection algorithm. In so doing, we needed to address a number of challenges. Previous AMR multigrid methods took advantage of simplifications in order to make use of residual correction multigrid. We have instead used the Full Approximation Scheme (FAS) version of multigrid in our AMR framework. Doing so gives the framework more flexibility, allowing us to solve more complicated operators with inhomogeneous boundary conditions.

Previous AMR frameworks have been written using *C++* for high level abstractions, and with *Fortran* used to implement single patch operators. This approach is not productive because *Fortran* is not an expressive language for representing high-level mathematics and manually programming with low-level abstractions does not scale well with current trends in scientific computing. In light of these observations, we have written our AMR framework mostly from the ground up using *Proto*, a new *C++* embedded domain-specific language for high-performance computing. This study represents the first AMR application written using *Proto*.

For Kina
You believed in me like no one else has, or likely ever will

For Leona and her siblings
I hope you are inspired to go out and do what you believe is worth doing

Contents

List of Figures

List of Tables

Acknowledgments

It is with genuine pleasure that I express my boundless gratitude to my mentor **Dr. Phillip Colella**. During the course of the past five and a half years, he has provided support for my endeavors in the form of sage guidance and meticulous scrutiny, not to mention an incomprehensible degree of patience. If it weren't for Phil, not only would this piece of research not exist, but I would not be the quality of person that I am today. For that I will eternally be grateful.

This work is the product of many others as well. Practically every member of the **Advanced Numerical Algorithms Group at Lawrence Berkeley** Laboratory has made an impact, either personally or academically. In particular I would like to thank **Brian Van Straalen** and **Daniel Graves** for not only their merciless propensity for code-slinging, but also their grounding realism which is a trait sometimes absent in the field of academia.

I am thankful for a long line of educators for their contributions not only to my instruction, but to the quality of my character. It is because of **Dr. Panos Papadopoulos** that I considered graduate education as an option at a time when I was truly at a loss for what path I should take. **Dr. Oliver O'Reilly** was the first to impress upon me the utility of mathematics, but more importantly he taught me the power of learning to teach others. **Leigh-Anne Gobel**, **Lissa Loeffler Jones**, and **Pete Simoncini** taught me how to laugh at myself and think for myself when others would rather do both of those things for me. I have been unbelievably lucky to have had the pleasure of knowing and learning from each these wonderful people.

My personal life is also filled with exemplary humans without which this accomplishment would not have been tractable. My mother **Rose Gebhart**; my acquired family, **the Bosmans**; **David** and other **various Eliahus**; my friends **Avi Hecht** and **Conrad Park**; the good people in ***Panazea***, and many others both near and far have always believed in me and been able to find an encouraging word when necessary. I wish I had the space to thank all of you, but I think this dissertation has enough reading material as it is.

Of course, no one has been more irritatingly supportive of me than **Kina Bosman**, the woman who made the questionable decision to marry me. Through the agonizing days, weeks, and months lost to the *COVID-19* epidemic, she alone was able keep me pressing on. A sizable percentage of the effort detailed herein can be traced back to her obstinate support of me and everything I do. I am and always will be indescribably grateful for her; I could not be a luckier man.

Chapter 1

Introduction

In this study we examine a computational fluid dynamics algorithm which operates in the low-Mach limit. This limit is typically defined by $M < 0.1$ and is characterized by very fast, low amplitude acoustic waves which accompany the advective motion. Here M is the Mach number $M = ||\mathbf{v}||/c$ where $\mathbf{v}$ is the characteristic advective velocity and c is the sound speed. In this regime, both acoustic and compressible phenomena may be considered important, though the weak acoustic waves have vanishingly small impact on the advective dynamics. The inherent challenge in simulating this limit is that these two dynamic ranges occur at very different spatial and temporal scales. Stability of an explicit time discretization restricts the time step to $\Delta t < \Delta x/c$ according to the *acoustic* CFL (Courant-Friedrich-Lewy) condition. In order to accurately simulate the advective motions, the time step is only constrained by the *advective* CFL condition $\Delta t < \Delta x/||\mathbf{v}||$ which is an order of $1/M$ less restrictive.

A proven strategy for simulating the low-Mach regime is to derive new equations of motion for the zero-Mach limit. These new systems are derived by writing the state variables as asymptotic expansions in powers of M in the limit $M \to 0$. The result is a different set of evolution equations which are accompanied by constraints on the velocity divergence. An example of this limit is incompressible flow which has the simple constraint $\nabla \cdot \mathbf{v} = 0$.

The approach we take in this work is based on the all-speed projection approach first introduced by Colella and Pao [16]. In the original version, the velocity field was split into its stiff potential component and its non-stiff solenoidal component using a Helmholtz projection. The stiff variables were then treated implicitly while the remaining variables were treated explicitly. In [12], the approach was refined by opting instead to retain the full compressible equations of motion and introduce redundant equations for the stiff pressure and potential velocity variables. The resulting method is a consistent discretization of the compressible flow equations that is valid for all Mach numbers in the absence of shocks. Evolution in time is computed using an implicit-explicit (IMEX) additive Runge-Kutta (ARK) method with the full compressible equations treated explicitly and the redundant equations treated implicitly. The resulting method has no acoustic CFL limitation on the time step and successfully captures the $M \to 0$ limits for a variety of applications including both viscous

and reacting flows.

In the present work we extend this all-speed projection method to include adaptive mesh refinement (AMR) in space using nested block-structured grids. Advective dynamics can have multiple length scales (*e.g.* mixing or combustion). Additionally, small scale advective processes can generate low-amplitude long-wavelength acoustic waves in the low-Mach limit. Adaptive refinement allows the algorithm to resolve all of these processes in an economical fashion.

In the process of carrying out this endeavor, we have needed to address a number of difficulties. Finite volume methods, particularly at high-order (> 2) require complicated stencil operators at refinement boundaries [31]. Using a formulation in which averaging between levels is *exact* makes these complications manageable but at the expense of requiring a different iterative solver implementation than what has been typically used. Specifically, we have abandoned the residual-correction formulation of multigrid that has been used in AMR methods for unsteady incompressible flow in favor of the more general Full Approximation Scheme (FAS) version. This alteration introduces some additional complexity to the algorithm, but allows us to solve very general operators with inhomogeneous boundary conditions.

Implementation of structured-grid AMR methods has been carried out using frameworks based on MPI distributed-memory parallelization of serial single-patch physics code written in *Fortran*. Current trends in scientific computing have rendered this strategy less and less effective at abstracting away the tedious implementation details that are necessary for high performance. Mixed-language programming is not ideal, and *Fortran* is not an expressive language for representing the mathematical algorithms which arise in high-order methods. As hardware targets become more sophisticated, additional execution parameters must be specified regarding threading, data layout, and so on. From a productivity standpoint, this is not a scalable programming model.

Our approach uses a different model implemented through *Proto*, a new high-performance computing library written in *C++*. *Proto*'s principle design goals are to furnish an expressive "math-like" syntax which hides the details of high-performance implementation from the application developer. In our AMR framework, *Proto* replaces the single-patch implementation of problem specific mathematics previously encoded in *Fortran* with convenient high-level abstractions that dramatically enhance productivity.

The outline of this thesis will proceed as follows. In Chapter 2 we will discuss the specification of the all-speed projection algorithm and in Chapter 3 we present the associated discretizations on a single level. In Chapter 4 we will provide the preliminaries for discussing AMR algorithms as well as our novel implementations of AMR and multigrid. Finally, Chapter 5 briefly discusses *Proto* and Chapter 6 summarizes our progress with applying the all-speed projection algorithm to a variety of relevant problems.

Chapter 2

All-Speed Projection Formulation

In this section we derive the system of equations that we will later discretize in our implementation of the all-speed projection algorithm. We start with the inviscid, non-reacting compressible Euler equations and augment them with redundant evolution equations for the pressure and potential velocity. The new system has the same solution as the compressible Euler equations assuming a set of initial condition constraints are satisfied. Finally, we motivate this strategy by examining the properties of the stiff variables in the acoustic limit.

2.1 All-Speed Equations of Motion

Our starting point for deriving the equations of motion for the all-speed projection algorithm is the compressible Euler equations,

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v}\cdot\nabla)\mathbf{v} - \frac{1}{\rho}\nabla p, \tag{2.1}$$

$$\frac{\partial \rho}{\partial t} = -\nabla\cdot(\rho\mathbf{v}), \tag{2.2}$$

$$\frac{\partial \rho h}{\partial t} = -\nabla\cdot(\rho h\mathbf{v}) - \rho c^2(\nabla\cdot\mathbf{v}), \tag{2.3}$$

where ρ is the density, $\mathbf{v}$ is the velocity vector, p is the pressure, h is the enthalpy, and c is the speed of sound. To complete the compressible Euler equations we need to add an equation of state,

$$p = p(\rho, h), \tag{2.4}$$

here written in terms of p, ρ, and h. The speed of sound based on Equation (2.4) is

$$c^2 = \frac{\frac{\partial p}{\partial \rho}}{1 - \frac{1}{\rho}\frac{\partial p}{\partial h}}. \tag{2.5}$$

Together, Equations (2.1) to (2.5) form a closed system for $\mathbf{v}, \rho, h$, p, and c.

We consider the case of a closed domain Ω with solid walls on $\partial\Omega$ such that the normal velocity at the boundary is zero as shown in Equation (2.6)

$$\mathbf{v} \cdot \mathbf{n} \;=\; 0 \quad on \quad \partial\Omega. \tag{2.6}$$

Periodic boundary conditions will also be used as needed, and the initial conditions will be functions of space defined later on:

$$\rho(\mathbf{x}, t = 0) \;=\; \rho_0(\mathbf{x}), \tag{2.7}$$
$$\mathbf{v}(\mathbf{x}, t = 0) \;=\; \mathbf{v}_0(\mathbf{x}), \tag{2.8}$$
$$\rho h(\mathbf{x}, t = 0) \;=\; E_0(\mathbf{x}). \tag{2.9}$$

In order to isolate the stiff and non-stiff terms in the equations of motion, we make use of the Helmholtz decomposition [19]. This decomposition splits smooth vector fields that satisfy appropriate boundary conditions into their solenoidal and potential components as shown in Equation (2.10),

$$\mathbf{u} = \mathbf{u}_d + (\nabla\phi = \mathbf{u}_p), \tag{2.10}$$

where we represent the potential component of $\mathbf{u}$ as the gradient of a scalar as is customary. We can show that the solenoidal and potential components are orthogonal by

$$\int_\Omega (\mathbf{u}_d \cdot \nabla\phi)\, d\mathbf{x} \;=\; \int_\Omega \nabla \cdot (\mathbf{u}_d\phi)\, d\mathbf{x} \;=\; \int_{\partial\Omega} \phi\mathbf{u}_d \cdot \mathbf{n} dA \;=\; 0, \tag{2.11}$$

where we have assumed periodic boundary conditions. Noting that $\mathbf{u}_d$ by definition has zero divergence, we solve for ϕ using

$$\Delta\phi \;=\; \nabla \cdot \mathbf{u} \quad on \quad \Omega. \tag{2.12}$$

The boundary condition given by

$$\frac{\partial\phi}{\partial n} \;=\; 0 \quad on \quad \partial\Omega, \tag{2.13}$$

represents a solid wall, but in principle other boundaries could be considered. Combining Equations (2.10) and (2.12) allows us to form an expression for the the solenoidal and potential projection operators $\mathbb{P}$ and $\mathbb{Q}$ respectively:

$$\mathbb{Q}(\mathbf{u}) \;=\; \nabla(\Delta^{-1})\nabla \cdot \mathbf{u}, \tag{2.14}$$
$$\mathbb{P} \;=\; \mathbb{I} - \mathbb{Q}. \tag{2.15}$$

The $\mathbb{P}$ and $\mathbb{Q}$ operators are symmetric and idempotent *i.e.*,

$$\mathbb{Q} \;=\; \mathbb{Q}^T, \qquad \mathbb{P} \;=\; \mathbb{P}^T, \tag{2.16}$$
$$\mathbb{Q}^2 \;=\; \mathbb{Q}, \qquad \mathbb{P}^2 \;=\; \mathbb{P}, \tag{2.17}$$

from which it follows that,

$$||\mathbb{Q}|| \;=\; ||\mathbb{P}|| \;=\; 1. \tag{2.18}$$

We derive an equation for the potential velocity $\mathbf{v}_p$ by applying $\mathbb{Q}$ to the full momentum equation,

$$\frac{\partial \mathbf{v}_p}{\partial t} \;=\; \mathbb{Q}\left[\frac{\partial \mathbf{v}}{\partial t}\right] \;=\; -\mathbb{Q}\left[(\mathbf{v}\cdot\nabla)\mathbf{v} + \frac{1}{\rho}\nabla p)\right]. \tag{2.19}$$

We also need an evolution equation for the pressure,

$$\frac{\partial p}{\partial t} \;=\; -(\mathbf{v}\cdot\nabla)p - \rho c^2(\nabla\cdot\mathbf{v}_p). \tag{2.20}$$

The pressure equation is easy to derive using Equations (2.3) and (2.4).

We add Equations (2.19) and (2.20) to a modified version of the original compressible Euler equations (2.1) through (2.3) to obtain

$$\frac{\partial \mathbf{v}}{\partial t} \;=\; -\mathbf{A}_\mathbf{v} - \frac{1}{\rho}\nabla p, \tag{2.21}$$

$$\frac{\partial \rho}{\partial t} \;=\; -\nabla\cdot(\rho\mathbf{v}_a), \tag{2.22}$$

$$\frac{\partial \rho h}{\partial t} \;=\; -\nabla\cdot(\rho h\mathbf{v}_a) - \rho c^2(\nabla\cdot\mathbf{v}_p), \tag{2.23}$$

$$\frac{\partial \mathbf{v}_p}{\partial t} \;=\; \mathbb{P}\left[\frac{1}{\rho}\nabla p + \mathbf{A}_\mathbf{v}\right] - \mathbf{A}_\mathbf{v} - \frac{1}{\rho}\nabla p, \tag{2.24}$$

$$\frac{\partial p}{\partial t} \;=\; -\nabla\cdot(\mathbf{v}_a p) + p(\nabla\cdot\mathbf{v}_p) - \rho c^2(\nabla\cdot\mathbf{v}_p). \tag{2.25}$$

The term $\mathbf{A}_\mathbf{v}$ is defined as

$$\mathbf{A}_\mathbf{v} \;=\; \nabla\cdot(\mathbf{v}\mathbf{v}_a) - \mathbf{v}(\nabla\cdot\mathbf{v}_p), \tag{2.26}$$

and the advection velocity $\mathbf{v}_a$ is

$$\mathbf{v}_a \;=\; \mathbb{P}(\mathbf{v}) + \mathbf{v}_p. \tag{2.27}$$

It is important to emphasize that $\mathbf{v}_p$ and p are evolved variables; they are not obtained from the projection and the equation of state.

We have made some conscious decisions in how we have written these equations. We have written the velocity advection term $(\mathbf{v}_a\cdot\nabla)\mathbf{v}$ as $\mathbf{A}_\mathbf{v}$ which decomposes the term into the divergence of a flux and a source term. The $(\mathbf{v}_a\cdot\nabla)p$ term in the pressure equation has gotten a similar treatment. Finally, we have chosen to write the equation for $\mathbf{v}_p$ in terms of $\mathbb{P}$ instead of $\mathbb{Q}$. These choices will facilitate adapting our discretization for the IMEX ARK and AMR frameworks later on.

The system defined by Equations (2.21) through (2.25) is distinct from the original system of Equations (2.1) through (2.3). However, the solutions to the new system are the same as

the solutions to the compressible Euler equations assuming the initial conditions satisfy the constraints

$$p = p(\rho, h), \tag{2.28}$$
$$\mathbb{Q}(\mathbf{v}) = \mathbf{v}_p. \tag{2.29}$$

This restriction arises due to the presence of the two new redundant evolution equations for p and $\mathbf{v}_p$. Because these are initial value constraints, they are analytically satisfied for all time if they are satisfied at $t = 0$. We will see that we must keep these constraints in mind when the time comes to write down our discretization of the algorithm.

2.2 Acoustic Equations

We may gain some insight into how Equations (2.21) through (2.25) may be solved by examining their form in the absence of advection and stratification. Removing these effects and focusing on the pressure and potential velocity equations yields

$$\frac{\partial \mathbf{v}_p}{\partial t} = -\frac{1}{\rho}\nabla p, \tag{2.30}$$
$$\frac{\partial p}{\partial t} = -\rho c^2(\nabla \cdot \mathbf{v}_p). \tag{2.31}$$

We can combine these statements to eliminate $\mathbf{v}_p$ resulting in

$$\frac{\partial^2 p}{\partial t^2} = c^2 \Delta p, \tag{2.32}$$

which is a wave equation in p. We can get some idea of how this system will behave when solved implicitly by discretizing Equations (2.30) and (2.31) in time using the implicit backward Euler method. The semi-discrete equations are

$$\mathbf{v}_p^{n+1} = \mathbf{v}_p^n - (\delta t)\frac{1}{\rho}\nabla p^{n+1}, \tag{2.33}$$
$$p^{n+1} = p^n - (\delta t)\rho c^2 \nabla \cdot \mathbf{v}_p^{n+1}. \tag{2.34}$$

where n refers to the discrete time and δt is the time step. Combining these results to eliminate $\mathbf{v}_p^{n+1}$ yields

$$\left[\mathbb{I} - (\delta t)^2 c^2 \Delta\right] p^{n+1} = p^n - (\delta t)\rho c^2 \nabla \cdot \mathbf{v}^n, \tag{2.35}$$

which is in the form of an elliptic Helmholtz equation. This is a perfectly tractable equation to solve, and this simple result motivates our decision to treat these variables implicitly.

2.3 Low-Mach Methods

By way of providing additional context for the all-speed projection algorithm, we briefly examine other methods that have been used to simulate time-dependent low-Mach flows. Chiefly among these are a category of methods which are derived through expanding the state variables in powers of M and deriving a new set of equations in the limit as $M \rightarrow 0$. We refer to these methods collectively as "zero-Mach" methods. Much of the progress in using this kind of approach has been made in the field of combustion starting with [40] and [26] and elaborated on by [22] and [17]. Other versions of these methods have been applied to atmospheric flows [4] and astrophysical processes [5].

There are various forms of zero-Mach methods including the incompressible Euler equations, the anelastic equations, and various configurations that have been derived for reacting fluid flow (see for example [26], equations 1.18a-d). In each of these applications, a system is formed which amounts to a set of differential/algebraic equations containing a set of evolution equations for the state variables and a constraint of the form $\nabla \cdot (\phi \mathbf{v}) = ...$ which serves to eliminate acoustic waves. The differential/algebraic formulation constitutes a major difficulty. These systems are fundamentally different from systems of pure evolution equations and it is difficult to evolve them in time using traditional methods if it is possible to do so at all [37, 6, 10]. It is sometimes possible to use a projection to eliminate the constraint yielding a system of evolution equations [14, 8], but this approach only works if the constraint is homogeneous and independent of time, *e.g.* $\nabla \cdot \mathbf{v} = 0$. Constraints also cause problems when combined with AMR refinement in time [2, 28, 29].

The all-speed projection method first formulated in [16], elaborated on in [12], and further improved upon herein is an alternative to the zero-Mach approach. This method still produces a set of evolution equations and *initial value* constraints (Equations (2.28) and (2.29)) which are fundamentally different from the constraints required of zero-Mach methods. These constraints are analytically satisfied for all time if they are satisfied at $t = 0$ and are relatively simple to enforce numerically. The all-speed method also makes use of a projection, but here it is used to split the compressible dynamics instead of eliminate them. Even in the presence of stiff physics, the system generated in the all-speed formulation is simply better understood than the differential/algebraic specification of zero-Mach methods.

Chapter 3

Single Level Discretization

This chapter contains the details for implementing the all-speed projection algorithm described in the previous chapter on a single level. The definitions in this chapter will need to be expanded upon when we introduce local refinement in the following chapter, but most of the discretizations used herein will still be used in the AMR formulation.

3.1 Finite Volume Discretization

Our general approach in discretizing Equations (2.21) through (2.25) is very similar to the fourth-order finite-volume method outlined in [31]. On the domain Ω, the control volumes $V_{\boldsymbol{i}}$ and the faces $A_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}$ which bound them are defined as

$$V_{\boldsymbol{i}} = [\boldsymbol{i}h, (\boldsymbol{i}+\mathbf{e})h] \qquad \boldsymbol{i} \in \mathbb{Z}^{\mathrm{D}}, \tag{3.1}$$

$$A_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = [\boldsymbol{i}h, (\boldsymbol{i}+\mathbf{e}-\mathbf{e}^d)h] \qquad \boldsymbol{i} \in \mathbb{Z}^{\mathrm{D}}. \tag{3.2}$$

Here $\mathbf{e}^d$ is the unit vector in the direction d and

$$\mathbf{e} = [1, 1, ..., 1]. \tag{3.3}$$

Note that despite the syntax $\boldsymbol{i} - \frac{1}{2}\mathbf{e}^d$ in Equation (3.2), our discretization includes both the upper and lower faces of each control volume in the domain. That is to say for every control volume, there are $2D$ faces with each internal face being shared by two control volumes.

Our state variables will be represented as averages over control volumes with the exception of $\mathbf{v}_p$ which is specified by averages over faces,

$$\langle \phi \rangle_{\boldsymbol{i}}(t) = \frac{1}{h^{\mathrm{D}}} \int_{V_{\boldsymbol{i}}} \phi(\mathbf{x}, t) d\mathbf{x}, \tag{3.4}$$

$$\left\langle u^d \right\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}(t) = \frac{1}{h^{\mathrm{D}-1}} \int_{A_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}} u^d(\mathbf{x}, t) dA. \tag{3.5}$$

$$\tag{3.6}$$

In the exposition to follow, we use the notation $\langle\phi\rangle_{\mathbf{f}}$ to designate the vector $[\langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}] \quad d \in [1, \mathrm{D}]$. In the case of a vector-valued variable, $\langle\mathbf{u}\rangle_{\mathbf{f}}$ is understood to refer to the vector $[\langle u^d\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}] \quad d \in [1, \mathrm{D}]$ which is composed of the d-directional components averaged over the faces normal to d. In the case where we want to refer to only the d component of $\langle\phi\rangle_{\mathbf{f}}$ or $\langle\mathbf{u}\rangle_{\mathbf{f}}$, we write $\langle\phi\rangle_d$ or $\langle\mathbf{u}\rangle_d$ respectively.

There are two desirable properties of finite-volume methods when combined with the discrete divergence operator. First, advective transport of thermodynamic quantities is locally conservative; the flux out of one cell is automatically the same as the flux into the adjacent cell. Without this property, grid-based methods can give unphysical results even in the low-Mach number limit [36]. The second convenient property of these methods is that elliptic problems with solvability conditions have natural discrete analogues.

3.2 Time Discretization

In order to achieve the desired splitting of the stiff and non-stiff dynamics, we use a semi-discrete approach to advance the state. In space, we use second or fourth-order finite volume discretizations. Time integration is achieved using a method of lines formulation and an Implicit-Explicit (IMEX) Additive Runge-Kutta (ARK) method. Specifically, we elect to use the fourth-order, 6 stage ARK4(3)6L[2]SA scheme detailed in [20].

To integrate a system of differential equations using an IMEX ARK method, the equations must be written in the form

$$\frac{\partial\langle\mathbf{U}\rangle}{\partial t} \;=\; \mathbf{F}^E(\langle\mathbf{U}\rangle) + \mathbf{F}^I(\langle\mathbf{U}\rangle). \tag{3.7}$$

This form designates implicit and explicit contributions to the right-hand side ($\mathbf{F}^I$ and $\mathbf{F}^E$ respectively).

Based on the insights gained from the acoustic limiting case, we choose to treat the pressure gradient and velocity divergence terms in the pressure and velocity equations implicitly. The forms of $\mathbf{F}^E$ and $\mathbf{F}^I$ are

$$\frac{\partial}{\partial t}\left\langle\begin{bmatrix}\mathbf{v}\\ \rho\\ \rho h\\ p\end{bmatrix}\right\rangle = \left\langle\begin{bmatrix}-\mathbf{A}_{\mathbf{v}}\\ -\nabla\cdot(\rho\mathbf{v}_a)\\ -\nabla\cdot(\rho h\mathbf{v}_a) - \rho c^2(\nabla\cdot\mathbf{v}_p)\\ -\nabla\cdot(\mathbf{v}_a p) + p(\nabla\cdot\mathbf{v}_a)\end{bmatrix}\right\rangle + \left\langle\begin{bmatrix}-\frac{1}{\rho}\nabla p\\ 0\\ 0\\ -\rho c^2(\nabla\cdot\mathbf{v}_p)\end{bmatrix}\right\rangle, \tag{3.8}$$

$$\frac{\partial}{\partial t}\langle\mathbf{v}_p\rangle_{\mathbf{f}} = \left\langle\mathbb{P}\left[\frac{1}{\rho}\nabla p + \mathbf{A}_{\mathbf{v}}\right] - \mathbf{A}_{\mathbf{v}}\right\rangle_{\mathbf{f}} + \left\langle-\frac{1}{\rho}\nabla p\right\rangle_{\mathbf{f}}. \tag{3.9}$$

The first set of terms on the right-hand side make up $\mathbf{F}^E$ and the second set form $\mathbf{F}^I$.

The time update is defined by

$$\langle\mathbf{U}\rangle^{n+1} \;=\; \langle\mathbf{U}\rangle^n + \Delta t\sum_{k=1}^{s} b_k\left[\mathbf{F}^E(\langle\mathbf{U}^k\rangle) + \mathbf{F}^I(\langle\mathbf{U}^k\rangle)\right], \tag{3.10}$$

where we assume an s-stage Runge-Kutta method. The states are defined at each time step as $\langle \mathbf{U} \rangle^n = \langle \mathbf{U} \rangle (t = t_n)$ and $\langle \mathbf{U} \rangle^{n+1} = \langle \mathbf{U} \rangle (t = t^n + \Delta t)$. The intermediate states for each Runge-Kutta stage are given by

$$\langle \mathbf{U} \rangle^k = \langle \mathbf{U} \rangle^n + \Delta t \left[\sum_{j=1}^{s} a^E_{jk} \mathbf{F}^E(\langle \mathbf{U} \rangle^j) + a^I_{jk} \mathbf{F}^I(\langle \mathbf{U} \rangle^j) \right], \tag{3.11}$$

and are equivalent to $\langle \mathbf{U} \rangle (t = t^n + c_k \Delta t)$. All of the coefficients a^E_{jk}, a^I_{jk}, b_k, and c_k are given by the implicit and explicit Butcher tableaus of the IMEX ARK method.

In general, Equation (3.11) is implicit with respect to $\langle \mathbf{U} \rangle^k$. In practice, because we choose a diagonally implicit scheme for the implicit half of the IMEX ARK method, there is only one implicit term on the right hand side of Equation (3.11): namely, the term involving a^I_{kk}. In light of this observation, we split Equation (3.11) into

$$\langle \mathbf{U} \rangle^k = \langle \mathbf{U} \rangle^n + \chi^k + \Delta t a^I_{kk} \mathbf{F}^I(\langle \mathbf{U} \rangle^k), \tag{3.12}$$

$$\chi^k = \Delta t \sum_{j=1}^{k-1} \left[a^E_{kj} \mathbf{F}^E(\langle \mathbf{U} \rangle^j) + a^I_{kj} \mathbf{F}^I(\langle \mathbf{U} \rangle^j) \right]. \tag{3.13}$$

Three pieces of information must be provided to the IMEX ARK method to advance $\langle \mathbf{U} \rangle$. The first two are the forms of $\mathbf{F}^E$ and $\mathbf{F}^I$ (see Equations (3.8) and (3.9)), and the third is a strategy for computing the implicit solve in Equation (3.12).

3.3 Spatial Discretization

With the time discretization specified, we now turn our attention to the spatial terms. In this section we specify how to compute $\mathbf{F}^E$ and $\mathbf{F}^I$ while in the next section we discuss the implicit solve in Equation (3.12) and address the constraints.

Operators

First we must define a number of operators to facilitate the exposition. Our discretization consistently expresses variables either as averages over cells or faces. Occasionally we will need to explicitly convert between averaged and non-averaged quantities. We do so through a convolution operators $\mathbb{C}$ and $\mathbb{C}_{\mathbf{f}}$ defined as

$$\langle \phi \rangle = \mathbb{C}(\phi), \tag{3.14}$$

$$\langle \phi \rangle_d = \mathbb{C}_d(\phi_d), \tag{3.15}$$

$$\langle \phi \rangle_{\mathbf{f}} = \mathbb{C}_{\mathbf{f}}(\phi_{\mathbf{f}}) = [\mathbb{C}_d(\phi_d)] \quad \forall d. \tag{3.16}$$

Likewise, we also find ourselves needing to convert between cell and face averaged quantities. We do this using the interpolation operators $\mathcal{I}^d_{cell}$ and $\mathcal{I}^d_{face}$. $\mathcal{I}_{cell}$ and $\mathcal{I}_{face}$ are convenient

definitions that also take into account the mappings between $\mathbb{R}^1$ and $\mathbb{R}^{\mathrm{D}}$. These operators are defined:

$$\langle\phi\rangle = \mathcal{I}^d_{cell}(\langle\phi\rangle_d), \tag{3.17}$$
$$\langle\phi\rangle_d = \mathcal{I}^d_{face}(\langle\phi\rangle), \tag{3.18}$$
$$\langle\phi\rangle = \mathcal{I}_{cell}(\langle\phi\rangle_{\mathbf{f}}) = \mathcal{I}^{d=1}_{cell}(\langle\phi\rangle_{d=1}), \tag{3.19}$$
$$\langle\phi\rangle_{\mathbf{f}} = \mathcal{I}_{face}(\langle\phi\rangle) = \left[\mathcal{I}^d_{face}\left(\langle\phi\rangle\right)\right] \quad \forall d, \tag{3.20}$$
$$\langle\mathbf{u}\rangle = \mathcal{I}_{cell}(\langle\mathbf{u}\rangle_{\mathbf{f}}) = \left[\mathcal{I}^d_{cell}\left(\langle u_d\rangle_d\right)\right] \quad \forall d, \tag{3.21}$$
$$\langle\mathbf{u}\rangle_{\mathbf{f}} = \mathcal{I}_{face}(\langle\mathbf{u}\rangle) = \left[\mathcal{I}^d_{face}\left(\langle u_d\rangle\right)\right] \quad \forall d. \tag{3.22}$$

In addition to the above, we make use of a third-order upwind interpolation operator,

$$\langle\phi\rangle^u_d = \mathcal{I}^d_u(\langle\phi\rangle), \tag{3.23}$$

when computing the advection terms.

Products and quotients are another area where a disciplined approach must be used. At second order, the product of averages is equal to the average of the product, but for higher-order implementations there are additional correction terms. In light of this complication, we explicitly define product and quotient operations in terms of the $\mathcal{P}$ and $\mathcal{Q}$ operators respectively:

$$\langle\phi\psi\rangle = \mathcal{P}\left(\langle\phi\rangle, \langle\psi\rangle\right), \tag{3.24}$$
$$\langle\phi\psi\rangle_d = \mathcal{P}_d\left(\langle\phi\rangle_d, \langle\psi\rangle_d\right), \tag{3.25}$$
$$\left\langle\frac{\phi}{\psi}\right\rangle = \mathcal{Q}\left(\langle\phi\rangle, \langle\psi\rangle\right), \tag{3.26}$$
$$\left\langle\frac{\phi}{\psi}\right\rangle_d = \mathcal{Q}_d\left(\langle\phi\rangle_d, \langle\psi\rangle_d\right). \tag{3.27}$$

An assortment of derivative operators are needed. As we will see, in this particular finite-volume discretization it is natural for differentiation to be coupled with interpolation between cell averages and face averages and *vice versa*. The derivative of a cell-averaged quantity evaluated as a face average is written as

$$\left\langle\frac{\partial\phi}{\partial x_d}\right\rangle_d = \mathbb{D}_d\left(\langle\phi\rangle\right). \tag{3.28}$$

Using $\mathbb{D}$ we can define the gradient operator

$$\left\langle\nabla(\phi)\right\rangle = \mathcal{G}(\langle\phi\rangle) = \left[\mathbb{D}_d(\langle\phi\rangle)\right] \quad \forall d. \tag{3.29}$$

In our discretization, the gradient naturally maps cell-averaged scalars to face-averaged fluxes. The divergence,

$$\langle \nabla \cdot \mathbf{u} \rangle \;=\; \mathcal{D}(\langle \mathbf{u} \rangle_{\mathbf{f}}), \tag{3.30}$$

also has a natural mapping: from face-averaged fluxes to a cell-averaged scalar.

Finally, we examine the projection operators $\mathbb{P}$ and $\mathbb{Q}$. $\left\langle \mathbb{Q}(\mathbf{u_f}) \right\rangle$ is computed in three steps:

$$\langle \phi \rangle \;=\; \mathcal{D}(\langle \mathbf{u} \rangle_{\mathbf{f}}), \tag{3.31}$$

$$\langle \psi \rangle \;=\; \left\langle \Delta^{-1}(\phi) \right\rangle, \tag{3.32}$$

$$\left\langle \mathbb{Q}(\mathbf{u_f}) \right\rangle_{\mathbf{f}} \;=\; \mathcal{G}(\langle \psi \rangle). \tag{3.33}$$

We have already defined $\mathcal{D}$ and $\mathcal{G}$. The Δ operator is the constant coefficient Laplace operator which we discretize as

$$\left\langle \Delta\left(\phi\right) \right\rangle \;=\; \mathcal{D}\left[\mathcal{G}\left(\langle \phi \rangle\right)\right]. \tag{3.34}$$

We defer the discussion of inverting this operator until the next chapter. With $\mathbb{Q}$ defined, $\mathbb{P}$ follows easily as

$$\left\langle \mathbb{P}(\mathbf{u}) \right\rangle \;=\; \left[\mathbb{I} - \mathcal{I}_{cell}\mathbb{Q}\mathcal{I}_{face}\right](\langle \mathbf{u} \rangle), \tag{3.35}$$

$$\left\langle \mathbb{P}(\mathbf{u_f}) \right\rangle \;=\; \left[\mathbb{I} - \mathbb{Q}\right](\langle \mathbf{u} \rangle_{\mathbf{f}}). \tag{3.36}$$

When discretized in this manner, the projection operator in Equation (3.35) no longer has the properties described in Equations (2.16) through (2.18) because of the additional interpolation steps. Nevertheless, the efficacy of this approximate projection has been well documented [2, 23, 28].

Computing All Speed Terms

Now that each of our individual operators have been defined, we can use them to piece together the definition of each of the terms in $\mathbf{F}^E$ and $\mathbf{F}^I$. We start with the advective terms. The advective velocity $\mathbf{v}_a$,

$$\langle \mathbf{v} \rangle_{\mathbf{f}} \;=\; \mathcal{I}_{face}\left(\langle \mathbf{v} \rangle\right), \tag{3.37}$$

$$\langle \mathbf{v}_a \rangle_{\mathbf{f}} \;=\; \mathbb{P}\left(\langle \mathbf{v} \rangle_{\mathbf{f}}\right) + \left\langle \mathbf{v}_p \right\rangle_{\mathbf{f}}, \tag{3.38}$$

is discretized on cell faces. In the second-order case, we compute $\langle \phi \rangle_{\mathbf{f}}$ with

$$\langle \phi \rangle_{\mathbf{f}} = \mathcal{I}_u\left(\langle \phi \rangle\right) \tag{3.39}$$

for the advection terms. Note that here the upwind interpolation operator is used. Using the advection velocity and interpolated scalars, we can compute the advective terms:

$$\left\langle \nabla \cdot (\phi \mathbf{v}_a) \right\rangle = \mathcal{D}\left[\mathcal{P}_{\mathbf{f}}\left(\langle\phi\rangle_{\mathbf{f}}, \langle \mathbf{v}_a \rangle_{\mathbf{f}}\right)\right], \tag{3.40}$$

$$\left\langle \nabla \cdot (\mathbf{v}\mathbf{v}_a) \right\rangle = \left[\mathcal{D}\left[\mathcal{P}_{\mathbf{f}}\left(\langle v_{d'}\rangle_{\mathbf{f}}, \langle \mathbf{v}_a \rangle_{\mathbf{f}}\right)\right]\right] \quad \forall d', \tag{3.41}$$

$$\left\langle \phi \left(\nabla \cdot \mathbf{v}_p\right) \right\rangle = \mathcal{P}\left(\langle\phi\rangle, \mathcal{D}\left(\left\langle \mathbf{v}_p \right\rangle_{\mathbf{f}}\right)\right), \tag{3.42}$$

$$\left\langle \mathbf{v} \left(\nabla \cdot \mathbf{v}_p\right) \right\rangle = \left[\mathcal{P}\left(\langle v_{d'}\rangle, \mathcal{D}\left(\left\langle \mathbf{v}_p \right\rangle_{\mathbf{f}}\right)\right)\right] \quad \forall d'. \tag{3.43}$$

Turning our attention to the pressure gradient term, it is simplest to compute on cell faces,

$$\left\langle \frac{1}{\rho}\nabla p \right\rangle_{\mathbf{f}} = \mathcal{Q}\left(\mathcal{G}\left(\langle p \rangle\right), \mathcal{I}_{face}\left(\langle\rho\rangle\right)\right). \tag{3.44}$$

Finally we have the projection

$$\left\langle \mathbb{P}\left[\frac{1}{\rho}\nabla p + \mathbf{A}_{\mathbf{v}}\right] \right\rangle_{\mathbf{f}} = \mathbb{P}_{\mathbf{f}}\left[\left\langle \frac{1}{\rho}\nabla p \right\rangle_{\mathbf{f}} + \langle \mathbf{A}_{\mathbf{v}} \rangle_{\mathbf{f}}\right], \tag{3.45}$$

and the remaining terms can be computed by interpolation:

$$\langle \mathbf{A}_{\mathbf{v}} \rangle = \left\langle \nabla \cdot (\mathbf{v}\mathbf{v}_a) \right\rangle - \left\langle \mathbf{v}\left(\nabla \cdot \mathbf{v}_p\right) \right\rangle, \tag{3.46}$$

$$\langle \mathbf{A}_{\mathbf{v}} \rangle_{\mathbf{f}} = \mathcal{I}_{face}\left(\langle \mathbf{A}_{\mathbf{v}} \rangle\right), \tag{3.47}$$

$$\left\langle \frac{1}{\rho}\nabla p \right\rangle = \mathcal{I}_{cell}\left(\left\langle \frac{1}{\rho}\nabla p \right\rangle_{\mathbf{f}}\right). \tag{3.48}$$

3.4 Solvers

We have specified how we are to compute all of the terms in $\mathbf{F}^E$ and $\mathbf{F}^I$. All that remains are to discuss how we implicitly solve for the stiff terms and how we enforce the constraints. Each of these procedures involves either a Poisson or Helmholtz solve.

Pressure Solve

First we discuss the pressure solve. We start with Equation (3.12) and reorganize yielding,

$$\langle\phi\rangle^k = \langle\phi\rangle^* + \Delta t a^I_{kk} \mathbf{F}^I(\langle\phi\rangle^k), \tag{3.49}$$

$$\langle\phi\rangle^* = \langle\phi\rangle^n + \left\langle \chi_\phi \right\rangle, \tag{3.50}$$

where χ_ϕ is the component of χ (Equation (3.13)) associated with ϕ. Since χ is completely known at the time of the implicit solve and $\mathbf{F}^I_\rho$ and $\mathbf{F}^I_{\rho h}$ are identically zero, we can conclude that $\rho^k = \rho^*$ and $(\rho h)^k = (\rho h)^*$ are also known. The implicit update function $\mathbf{F}^I$ for $\mathbf{v}$, $\mathbf{v}_p$, and p does not depend implicitly on $\mathbf{v}$, hence we can focus on the equations

$$\begin{aligned} \langle \mathbf{v} \rangle_p^k &= \langle \mathbf{v} \rangle_p^* - \Delta t a^I_{kk} \left\langle \frac{1}{\rho} \nabla p^k \right\rangle, \\ \langle p \rangle^k &= \langle p \rangle^* - \Delta t a^I_{kk} \left\langle \rho c^2 \nabla \cdot \mathbf{v}_p^k \right\rangle, \end{aligned} \tag{3.51}$$

alone and compute the update for $\mathbf{v}$ through back substitution. This is precisely the logic we developed in Chapter 2 when analyzing the acoustic limit.

Eliminating all references to $\mathbf{v}_p^k$ in Equations (3.51) yields

$$\begin{aligned} \left[\langle \zeta \rangle - \Delta t a^I_{kk} \Delta_\rho \right] \langle p \rangle^k &= \langle \zeta p^* \rangle - \mathcal{D} \left(\left\langle \mathbf{v}_p^* \right\rangle \right), \\ \langle \zeta \rangle &= \frac{1}{\Delta t a^I_{kk}} \left\langle \frac{1}{(\rho c^2)^*} \right\rangle, \end{aligned} \tag{3.52}$$

where everything in sight is known (or easily computed) except for $\langle p \rangle^k$. The operator Δ_ρ is the variable coefficient Laplacian given by

$$\left\langle \Delta_\rho \left(\phi \right) \right\rangle = \mathcal{D} \left[\mathcal{P}_{\mathbf{f}} \left(\left\langle \frac{1}{\rho} \right\rangle_{\mathbf{f}}, \mathcal{G} \left(\langle \phi \rangle \right) \right) \right]. \tag{3.53}$$

Equations (3.52) are the final form that will be solved inside of the IMEX ARK method.

This is a convenient point to discuss why we have elected to use a staggered approach in our discretization. To this end, consider

$$\Delta \phi_{\boldsymbol{i}} = \frac{1}{4h^2} \left[\phi_{\boldsymbol{i}-2} - 2\phi_{\boldsymbol{i}} + \phi_{\boldsymbol{i}+2} \right], \tag{3.54}$$

which is a one-dimensional second-order discretization of the Δ operator with variables discretized purely on cell centers. When this operator is inverted using an iterative method, points with even values of $\boldsymbol{i}$ will be updated based on other even values of $\boldsymbol{i}$ and similarly for the odd values. No information flows between these two components of the solution, and hence they can diverge from one another as numerical error accumulates. This is a well documented phenomenon related to iterative solution of Poisson's Equation [38]. On the other hand, if we write the same operator using face-centered gradients we arrive at

$$\Delta \phi_{\boldsymbol{i}} = \frac{1}{h^2} \left[\phi_{\boldsymbol{i}-1} - 2\phi_{\boldsymbol{i}} + \phi_{\boldsymbol{i}+1} \right]. \tag{3.55}$$

which clearly does not exhibit the same problem as Equation (3.54).

Constraints

The constraints defined by the velocity projection and equation of state (Equations (2.28) and (2.29)) are initial value constraints and are satisfied for all time if they are satisfied by the initial data. However, numerical error will cause the state to drift and the constraints will cease to be satisfied. to prevent this, we reimpose the constraints at the end of each time step.

First we consider how to enforce the pressure constraint. The goal is to adjust the full pressure field such that p is the same as the pressure obtained from the equation of state (referred to in this section as p_0). We must also be careful to adjust the potential velocity to reflect any changes made to p.

This logic motivates us to return to the acoustic Equations (2.30) and (2.31). We rewrite this system as

$$\frac{\partial p}{\partial t} + \rho c^2 \left(\nabla \cdot \mathbf{v}_p\right) = \xi \left(p_0 - p\right), \tag{3.56}$$

$$\frac{\partial \mathbf{v}_p}{\partial t} + \frac{1}{\rho}\nabla p = 0, \tag{3.57}$$

where we have added a penalty term which forces p towards p_0. Here the parameter ξ is positive and scales inversely with Δt. Next, we transform Equations (3.56) and (3.57) into the difference equations

$$\langle \delta p \rangle = \Delta t \xi (p_0 - \langle p \rangle) - \Delta t \left\langle \rho c^2 \nabla \cdot (\delta \mathbf{v}_p) \right\rangle, \tag{3.58}$$

$$\left\langle \delta \mathbf{v}_p \right\rangle = -\Delta t \left\langle \frac{1}{\rho} \nabla \delta p \right\rangle, \tag{3.59}$$

where δp and $\delta \mathbf{v}_p$ are the increments that we will use to enforce the constraint. Reiterating the same logic that we used to derive the pressure solve in the previous section, we can eliminate references to $\delta \mathbf{v}_p$ and produce the elliptic Helmholtz equation,

$$\left[\left\langle \frac{1}{\rho c^2} \right\rangle \mathbb{I} - (\Delta t)^2 \Delta_\rho \right] \left(\langle \delta p \rangle\right) = \Delta t \xi \left\langle \frac{(p_0 - p)}{\rho c^2} \right\rangle. \tag{3.60}$$

Once δp is known, the correction to $\mathbf{v}_p$ can be computed using Equation (3.59). Using these increments, we update the state variables using

$$\langle \phi \rangle^{n+1} = \langle \phi \rangle + \langle \delta \phi \rangle. \tag{3.61}$$

In addition to the constraint implied by the equation of state, we must also ensure that $\mathbf{v}_p$ is truly the potential component of $\mathbf{v}$. This constraint is easy to enforce using the Helmholtz projection,

$$\left\langle \mathbf{v}_p \right\rangle_{\mathbf{f}}^{n+1} = \mathbb{Q}\left(\left\langle \mathbf{v}_p \right\rangle_{\mathbf{f}}\right), \tag{3.62}$$

$$\langle \mathbf{v} \rangle^{n+1} = \mathcal{I}_{face}\left(\left\langle \mathbf{v}_p \right\rangle_{\mathbf{f}}^{n+1}\right) + \mathbb{P}(\langle \mathbf{v} \rangle). \tag{3.63}$$

Note that for these projections to have the desired effects, the velocity constraints must be enforced *after* the pressure constraints. Otherwise, $\mathbf{v}_p$ may not be a discrete gradient due to baroclinic variations in ρ.

3.5 Operator Definitions

At this point the only details that remain unaccounted for are how we discretize the convolution, interpolation, multiplication, division, and differentiation operators.

Finite Volume Averages

As discussed above, we take special care to distinguish between *cell-averaged* quantities (*e.g.* $\langle\phi\rangle_{\boldsymbol{i}}$) and *cell-centered* quantities (*e.g.* $\phi_{\boldsymbol{i}}$). Cell-centered quantities are defined at points $\mathbf{x}_{\boldsymbol{i}}$ in the center of a control volume $V_{\boldsymbol{i}}$,

$$\phi_{\boldsymbol{i}} \;=\; \phi(\mathbf{x}_{\boldsymbol{i}}) \qquad \mathbf{x}_i \;=\; h\boldsymbol{i} + \frac{h}{2}\mathbf{e}. \tag{3.64}$$

Cell-averaged quantities are defined as the average of the quantity over $V_{\boldsymbol{i}}$,

$$\langle\phi\rangle_{\boldsymbol{i}} \;=\; \frac{1}{h^{\mathrm{D}}}\int_{\mathbf{x}_l}^{\mathbf{x}_h} \phi(\mathbf{x})d\mathbf{x} \qquad \mathbf{x}_l \;=\; h\boldsymbol{i} \qquad \mathbf{x}_h \;=\; h(\boldsymbol{i}+\mathbf{e}). \tag{3.65}$$

We use the convolution operator $\mathbb{C}$ to convert between cell-centered and cell-averaged values. This operator is defined by

$$\mathbb{C}\left(\phi\right)_{\boldsymbol{i}} = \phi_{\boldsymbol{i}} = \langle\phi\rangle_{\boldsymbol{i}} + \mathcal{O}\left(h^2\right), \tag{3.66}$$

$$\mathbb{C}\left(\phi\right)_{\boldsymbol{i}} = \phi_{\boldsymbol{i}} + \frac{h^2}{24}\sum_{d=1}^{\mathrm{D}}\left(\frac{\partial^2\phi}{\partial x_d^2}\right)_{\boldsymbol{i}} = \langle\phi\rangle_{\boldsymbol{i}} + \mathcal{O}\left(h^4\right), \tag{3.67}$$

$$\mathbb{C}^{-1}\left(\langle\phi\rangle\right)_{\boldsymbol{i}} = \langle\phi_{\boldsymbol{i}}\rangle - \frac{h^2}{24}\sum_{d=1}^{\mathrm{D}}\left(\frac{\partial^2\phi}{\partial x_d^2}\right)_{\boldsymbol{i}} = \phi_{\boldsymbol{i}} + \mathcal{O}(h^4). \tag{3.68}$$

Similar expressions hold for face-averaged data:

$$\mathbb{C}_d\left(\phi\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \phi_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}\left(h^2\right), \tag{3.69}$$

$$\mathbb{C}_d\left(\phi\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \phi_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \frac{h^2}{24}\sum_{d'\neq d}^{\mathrm{D}}\left(\frac{\partial^2\phi}{\partial x_{d'}^2}\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}\left(h^4\right), \tag{3.70}$$

$$\mathbb{C}_d^{-1}\left(\langle\phi\rangle_{\mathbf{f}}\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} - \frac{h^2}{24}\sum_{d'\neq d}^{\mathrm{D}}\left(\frac{\partial^2\phi}{\partial x_{d'}^2}\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \phi_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}(h^4). \tag{3.71}$$

Cell-Face Interpolation

To convert between cell- and face-averaged quantities, we use the interpolation stencils

$$\mathcal{I}^d_{face}\left(\langle\phi\rangle\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \frac{1}{2}\langle\phi\rangle_{\boldsymbol{i}-\mathbf{e}^d} + \frac{1}{2}\langle\phi\rangle_{\boldsymbol{i}} = \langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}(h^2), \tag{3.72}$$

$$\begin{aligned}\mathcal{I}^d_{face}\left(\langle\phi\rangle\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} &= \frac{7}{12}\left(\langle\phi\rangle_{\boldsymbol{i}-\mathbf{e}^d} + \langle\phi\rangle_{\boldsymbol{i}}\right) \\ &\quad - \frac{1}{12}\left(\langle\phi\rangle_{\boldsymbol{i}-2\mathbf{e}^d} + \langle\phi\rangle_{\boldsymbol{i}+\mathbf{e}^d}\right) = \langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}(h^4),\end{aligned} \tag{3.73}$$

$$\mathcal{I}^d_{cell}\left(\langle\phi\rangle_{\mathbf{f}}\right)_{\boldsymbol{i}} = \frac{1}{2}\langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \frac{1}{2}\langle\phi\rangle_{\boldsymbol{i}+\frac{1}{2}\mathbf{e}^d} = \langle\phi\rangle_{\boldsymbol{i}} + \mathcal{O}(h^2), \tag{3.74}$$

$$\begin{aligned}\mathcal{I}^d_{cell}\left(\langle\phi\rangle_{\mathbf{f}}\right)_{\boldsymbol{i}} &= \frac{13}{24}\left(\langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \langle\phi\rangle_{\boldsymbol{i}+\frac{1}{2}\mathbf{e}^d}\right) \\ &\quad - \frac{1}{24}\left(\langle\phi\rangle_{\boldsymbol{i}-\frac{3}{2}\mathbf{e}^d} + \langle\phi\rangle_{\boldsymbol{i}+\frac{3}{2}\mathbf{e}^d}\right) = \langle\phi\rangle_{\boldsymbol{i}} + \mathcal{O}(h^4).\end{aligned} \tag{3.75}$$

In addition to the operators above, we define the third-order upwind interpolation operator $\mathcal{I}_u$. This operator is discretized

$$\mathcal{I}_u\left(\langle\phi\rangle\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = -\frac{1}{6}\langle\phi\rangle_{\boldsymbol{i}-2\mathbf{e}^d} + \frac{5}{6}\langle\phi\rangle_{\boldsymbol{i}-\mathbf{e}^d} + \frac{1}{3}\langle\phi\rangle_{\boldsymbol{i}} \qquad \left\langle v^d_a\right\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} > 0, \tag{3.76}$$

$$= \frac{1}{3}\langle\phi\rangle_{\boldsymbol{i}-\mathbf{e}^d} + \frac{5}{6}\langle\phi\rangle_{\boldsymbol{i}} - \frac{1}{6}\langle\phi\rangle_{\boldsymbol{i}+\mathbf{e}^d} \qquad \left\langle v^d_a\right\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} . < 0. \tag{3.77}$$

Derivatives

The appropriate stencils for computing derivatives of cell-averaged data as face-averages are

$$\mathbb{D}\left(\langle\phi\rangle\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \frac{1}{2h}\left(\langle\phi\rangle_{\boldsymbol{i}} - \langle\phi\rangle_{\boldsymbol{i}-\mathbf{e}^d}\right) = \left\langle\frac{\partial\phi}{\partial x_d}\right\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}(h^2), \tag{3.78}$$

$$\begin{aligned}\mathbb{D}\left(\langle\phi\rangle\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} &= \frac{5}{4h}\left(\langle\phi\rangle_{\boldsymbol{i}} - \langle\phi\rangle_{\boldsymbol{i}-\mathbf{e}^d}\right) \\ &\quad - \frac{1}{12h}\left(\langle\phi\rangle_{\boldsymbol{i}+\mathbf{e}^d} - \langle\phi\rangle_{\boldsymbol{i}-2\mathbf{e}^d}\right) = \left\langle\frac{\partial\phi}{\partial x_d}\right\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}(h^4).\end{aligned} \tag{3.79}$$

The face-averaged derivatives above are used to compute the components of the gradient operator $\mathcal{G}$. The divergence $\mathcal{D}$ is defined

$$\mathcal{D}\left(\langle\mathbf{u}\rangle_{\mathbf{f}}\right)_{\boldsymbol{i}} = \frac{1}{h}\sum_{d=1}^{\mathrm{D}}\left[\langle\mathbf{u}\rangle_{\boldsymbol{i}+\frac{1}{2}\mathbf{e}^d} - \langle\mathbf{u}\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}\right], \tag{3.80}$$

which is the standard discretization used in finite-volume methods to achieve discrete conservation.

Product and Quotient Rules

Second order finite volume discretizations lack any need for product or quotient rules. At second-order, the cell average of a product or quotient is the product or quotient of the averages respectively. At higher-order we must be a bit more careful. The appropriate expressions for the product and quotient rules are:

$$\mathcal{P}\left(\langle\phi\rangle, \langle\theta\rangle\right)_{\boldsymbol{i}} = \langle\phi\rangle_{\boldsymbol{i}} \langle\theta\rangle_{\boldsymbol{i}} = \langle\phi\theta\rangle_{\boldsymbol{i}} + \mathcal{O}(h^2), \tag{3.81}$$

$$\mathcal{P}\left(\langle\phi\rangle, \langle\theta\rangle\right)_{\boldsymbol{i}} = \langle\phi\rangle_{\boldsymbol{i}} \langle\theta\rangle_{\boldsymbol{i}} + \frac{h^2}{24} \sum_{d=1}^{\mathrm{D}} \left[\left(\frac{\partial \phi}{\partial x_d} \right)_{\boldsymbol{i}} \left(\frac{\partial \theta}{\partial x_d} \right)_{\boldsymbol{i}} \right] = \langle\phi\theta\rangle_{\boldsymbol{i}} + \mathcal{O}(h^4), \tag{3.82}$$

$$\mathcal{Q}\left(\langle\phi\rangle, \langle\theta\rangle\right)_{\boldsymbol{i}} = \frac{\langle\phi\rangle_{\boldsymbol{i}}}{\langle\theta\rangle_{\boldsymbol{i}}} = \left\langle \phi/\theta \right\rangle_{\boldsymbol{i}} + \mathcal{O}(h^2), \tag{3.83}$$

$$\mathcal{Q}\left(\langle\phi\rangle, \langle\theta\rangle\right)_{\boldsymbol{i}} = \frac{\langle\phi\rangle_{\boldsymbol{i}}}{\langle\theta\rangle_{\boldsymbol{i}}} - \frac{h^2}{12\theta_{\boldsymbol{i}}^2} \sum_{d=1}^{D} \left[\left(\frac{\partial \phi}{\partial x_d} \right)_{\boldsymbol{i}} \left(\frac{\partial \theta}{\partial x_d} \right)_{\boldsymbol{i}} - \frac{\phi_{\boldsymbol{i}}}{\theta_{\boldsymbol{i}}} \left| \frac{\partial \theta}{\partial x_d} \right|_{\boldsymbol{i}}^2 \right] = \left\langle \phi/\theta \right\rangle_{\boldsymbol{i}} + \mathcal{O}(h^4). \tag{3.84}$$

Similar relations hold for face-averaged quantities:

$$\mathcal{P}_{\mathbf{f}}\left(\langle\phi\rangle_{\mathbf{f}}, \langle\theta\rangle_{\mathbf{f}}\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} \langle\theta\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \langle\phi\theta\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}(h^2), \tag{3.85}$$

$$\begin{aligned} \mathcal{P}_{\mathbf{f}}\left(\langle\phi\rangle_{\mathbf{f}}, \langle\theta\rangle_{\mathbf{f}}\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} &= \langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} \langle\theta\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \frac{h^2}{24} \sum_{d' \neq d}^{D} \left[\left(\frac{\partial \phi}{\partial x_{d'}} \right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} \left(\frac{\partial \theta}{\partial x_{d'}} \right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} \right] \\ &= \langle\phi\theta\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}(h^4), \end{aligned} \tag{3.86}$$

$$\mathcal{Q}_{\mathbf{f}}\left(\langle\phi\rangle_{\mathbf{f}}, \langle\theta\rangle_{\mathbf{f}}\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \frac{\langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}}{\langle\theta\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}} = \left\langle \phi/\theta \right\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}(h^2), \tag{3.87}$$

$$\begin{aligned} \mathcal{Q}_{\mathbf{f}}\left(\langle\phi\rangle_{\mathbf{f}}, \langle\theta\rangle_{\mathbf{f}}\right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} &= \frac{\langle\phi\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}}{\langle\theta\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}} - \frac{h^2}{12\theta_d^2} \sum_{d' \neq d}^{D} \left[\left(\frac{\partial \phi}{\partial x_{d'}} \right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} \left(\frac{\partial \theta}{\partial x_{d'}} \right)_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} - \frac{\phi_{\boldsymbol{i}}}{\theta_{\boldsymbol{i}}} \left| \frac{\partial \theta}{\partial x_{d'}} \right|_d^2 \right] \\ &= \left\langle \phi/\theta \right\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} + \mathcal{O}(h^4). \end{aligned} \tag{3.88}$$

In the fourth-order corrections in the above relations, the multiplication by h^2 allows us to substitute *centered* quantities for *averaged* ones as well as the use of simple second-order finite differences to compute the necessary derivatives.

Chapter 4

AMR and Multigrid

Up until this point, we have presented a single-level approach to discretizing the all-speed projection algorithm. One of our key contributions in this study, however, is the addition of local refinement. We have also deferred our discussion of how we compute the solves for the projection, pressure, and pressure constraint.

In this chapter we will introduce the fundamentals for discussing both multigrid and AMR and discuss how the discretizations from the previous chapter will need to be adapted to suit the needs of these multi-level algorithms. We will discuss multigrid in both it's residual-correction and full approximation forms, and motivate our use of the latter. Finally, we will present our full AMR FAS-Multigrid algorithm and some details regarding its performance.

4.1 Terminology for Multi-Level Algorithms

Before we discuss the AMR and multigrid algorithms, we must introduce some definitions that are useful for discussing algorithms which employ multiple levels of data. We start by discussing the levels themselves. Each level ℓ consists of a grid Γ^ℓ which covers the entire problem domain. The grid $\Gamma^\ell \subset \mathbb{Z}^D$ has associated with it a grid spacing h^ℓ. The region $\Omega^\ell \subseteq \Gamma^\ell$ defines the part of the grid that contains data on level ℓ. Grids associated with different levels have different spacings with smaller values of ℓ associated with coarser grids. Two successive grids' spacings are related by a refinement ratio $r = h^\ell / h^{\ell+1}$. For the sake of practicality, we limit r to be a power of 2, and generally it takes the value of 2, 4, or 8. r is the same between all levels in the same algorithm, but as we will see r can and often will be different in the context of different algorithms. In particular, in this study we refer to r_{MG} and r_{AMR} as the refinement ratios used in multigrid and AMR respectively. Points $\boldsymbol{i}$ in a domain Ω^ℓ can be represented on coarser grids through the use of a coarsening operator $\mathcal{C}_r(\boldsymbol{i}) = \lfloor \frac{\boldsymbol{i}}{r} \rfloor$. In practice, we often use this operator to refer to the region Ω^ℓ represents on $\Gamma^{\ell-1}$. We write this as $\mathcal{C}_r\left(\Omega^\ell\right) \subseteq \Omega^{\ell-1}$.

Up to this point we have introduced terminology that is germane to both AMR and multigrid. To this we add definitions used to describe locally refined grids which only apply

to AMR. Each point $\boldsymbol{i}$ in Ω^ℓ has an associated cell $\Omega^\ell_{\boldsymbol{i}}$. (These domains are essentially the same the control volumes $V_{\boldsymbol{i}}$ that we defined in the previous chapter). We restrict the manner in which finer levels are built such that each control volume $\Omega^\ell_{\boldsymbol{i}}$ is either completely refined by cells in $\Omega^{\ell+1}$ or not refined at all. We also stipulate that no cell in level $\ell+1$ represents a region in space directly adjacent to a cell in level $\ell-1$. That is to say, at least one cell on level ℓ lies in between. This collection of constraints on locally refined grids we refer to as *proper nesting* [1, 9].

We refer to the entire array of locally refined grids as an AMR hierarchy. Data defined on an AMR hierarchy is defined on all of Ω^ℓ for each level ℓ. Data that is defined on the region of Ω^ℓ that is not covered by $\Omega^{\ell+1}$ is said to be *valid* because it is the highest resolution data for that region of the problem domain. Data which lies in $\mathcal{C}\left(\Omega^{\ell+1}\right) \subseteq \Omega^\ell$ is termed *invalid*. These terms apply both to cell-averaged quantities and face-averaged ones. The *composite* value of data defined on an AMR hierarchy is the union of all valid data with all redundancies resolved by choosing the finest resolution available. In contrast, the *level* value of a variable is the union of all data on a given level.

4.2 Multi Level Discretization

Our general approach in discretizing Equations (2.21) through (2.25) is essentially the same as it was for the single level case. The finite-volumes $V_{\boldsymbol{i}}$ and areas $A_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}$ are defined the same way by

$$V_{\mathbf{i}} = \left[\mathbf{i}h, (\mathbf{i}+\mathbf{e})h\right] \qquad \mathbf{i} \in \Omega^\ell, \tag{4.1}$$

$$A_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} = \left[\mathbf{i}h, (\mathbf{i}+\mathbf{e}-\mathbf{e}^d)h\right] \qquad \mathbf{i} \in \Omega^\ell. \tag{4.2}$$

Averages over control volumes and faces are computed by

$$\langle\phi\rangle_{\boldsymbol{i}}(t) = \frac{1}{h^D}\int_{V_{\mathbf{i}}} \phi(\mathbf{x},t)d\mathbf{x} \qquad V_{\boldsymbol{i}} \in \Omega^0, ..., \Omega^{\ell_{max}}, \tag{4.3}$$

$$\left\langle u^d \right\rangle_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}(t) = \frac{1}{h^{D-1}}\int_{A_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d}} u^d(\mathbf{x},t)dA \qquad A_{\boldsymbol{i}-\frac{1}{2}\mathbf{e}^d} \in \Omega^0, ..., \Omega^{\ell_{max}}, \tag{4.4}$$

and are likewise unchanged in this paradigm except that now we have multiple levels worth of data. In refined regions, the area covered by one coarse cell $V^\ell_{\boldsymbol{i}}$ with side length h^ℓ will be covered by r^D finer volumes $[V^{\ell+1}_{r\boldsymbol{i}}...]$ with side length $h^{\ell+1} = h^\ell/r$. Similarly a coarse level surface A^ℓ is covered on the finer level by r^{D-1} finer surfaces $[A^{\ell+1}_{\boldsymbol{i}-\frac{r}{2}\mathbf{e}^d}...]$. Both the coarse and fine data are maintained and will be used in further computations as needed.

In terms of notation, we refer to a variable on an AMR hierarchy as $\mathbf{U}^{comp}$ where we allude to the hierarchy representing a *composite* variable as previously described. A variable defined on a single level is written $\mathbf{U}^\ell$. In this chapter, all variables are assumed to be cell or face averaged, and hence the $\langle\cdot\rangle$ will be omitted except when it is needed for clarity.

The temporal discretization of the problem is completely unchanged when multiple levels are involved. We still elect to use a semi-implicit solver and a method of lines formulation to advance the solution and as such we must provide definitions for the implicit and explicit spatial terms $\mathbf{F}^I$ and $\mathbf{F}^E$ respectively as well as the implicit solve and the constraints. The only difference is that we must compute these quantities for each Ω^ℓ in the hierarchy.

4.3 AMR Operators

Now that our discretization includes multiple levels of data, we must rethink how spatial operators are computed. On a given level the operators defined in Chapter 3 continue to be valid. However, additional steps must be taken which add dependencies on possible coarse data below and fine data above. Some of these considerations only apply to the case of local refinement, and others are valid for all data hierarchies.

Averaging and Interpolation

We first define new operators for transferring both cell-averaged and face-averaged data between levels. The fine-to-coarse operators,

$$\langle\phi\rangle^{\ell-1} = \mathcal{A}_r\left(\langle\phi\rangle^{\ell}\right) \quad on \quad \mathcal{C}_r\left(\Omega^\ell\right), \tag{4.5}$$

$$\langle\phi\rangle_d^{\ell-1} = \mathcal{A}_{d,r}\left(\langle\phi\rangle_d^{\ell}\right) \quad on \quad \mathcal{C}_r\left(\Omega^\ell\right), \tag{4.6}$$

are simple arithmetic averages. These operators have the very useful property of contributing no additional truncation error when operating on averaged data.

Moving in the other direction, we now define a set of interpolation operators. The first of these,

$$\langle\phi\rangle^{\ell} = \mathcal{I}_r^0\left(\langle\phi\rangle^{\ell-1}\right) \quad on \quad \Omega^\ell, \tag{4.7}$$

is piecewise-constant. We will also need a higher-order interpolation operator,

$$\langle\phi\rangle^{\ell} = \mathcal{I}_r^{ho}\left(\langle\phi\rangle^{\ell-1}\right) \quad on \quad \Omega^\ell, \tag{4.8}$$

$$\langle\phi\rangle_d^{\ell} = \mathcal{I}_{d,r}^{ho}\left(\langle\phi\rangle_d^{\ell-1}\right) \quad on \quad \Omega^\ell, \tag{4.9}$$

for transferring boundary condition data from coarse to fine levels. The stencils for these operators are much more involved, and are derived at the start of the computation using the least-squares methodology outlined in [31].

General Multi Level Operators

We refer to a general operator $\mathcal{L}^{comp}$ acting on an AMR hierarchy as an AMR or composite operator. The composite operator is computed from a series of level operators

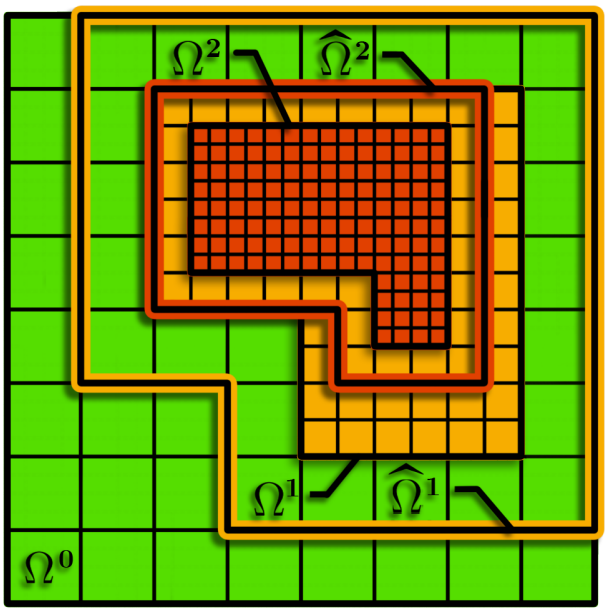


Figure 4.1: *Nested grid relationships between* $\mathbf{U}^\ell$, $\hat{\mathbf{U}}^\ell$, *and* $\mathbf{U}^{\ell-1}$ *used to interpolate boundary conditions in a nested grid hierarchy*

$\mathcal{L}^\ell(\hat{\mathbf{U}}^\ell, \mathbf{U}^{\ell-1}, \mathbf{U}^{\ell+1})$. These operators are written

$$\begin{aligned} \mathcal{L}^\ell(\mathbf{U}^\ell, \mathbf{U}^{\ell-1}, \mathbf{U}^{\ell+1}) &= \mathcal{J}(\hat{\mathbf{U}}^\ell) + \mathcal{R}(\mathbf{U}^\ell, \mathbf{U}^{\ell+1}) \quad on \quad \Omega^\ell \\ \hat{\mathbf{U}}^\ell &= \mathcal{I}^{ho}_{r_{AMR}}\left(\mathbf{U}^{\ell-1}\right) \quad on \quad \hat{\Omega}^\ell - \Omega^\ell, \end{aligned} \tag{4.10}$$

$$\begin{aligned} \mathcal{L}^\ell_{\mathbf{f}}(\mathbf{U}^\ell, \mathbf{U}^{\ell-1}, \mathbf{U}^{\ell+1}) &= \mathcal{J}_{\mathbf{f}}(\hat{\mathbf{U}}^\ell) + \mathcal{R}_{\mathbf{f}}(\mathbf{U}^\ell, \mathbf{U}^{\ell+1}) \quad on \quad \Omega^\ell \\ \hat{\mathbf{U}}^\ell &= \mathcal{I}^{ho}_{r_{AMR}}\left(\mathbf{U}^{\ell-1}\right) \quad on \quad \hat{\Omega}^\ell - \Omega^\ell. \end{aligned} \tag{4.11}$$

Here $\hat{\mathbf{U}}^\ell$ is $\mathbf{U}^\ell$ evaluated on $\hat{\Omega}^\ell$, an extension of Ω^ℓ that includes a region of "ghost" or "halo" cells on all sides. The region $\hat{\Omega}^\ell$ satisfies

$$\Omega^\ell \subseteq \hat{\Omega}^\ell, \qquad \mathcal{C}(\hat{\Omega}^\ell) \subseteq \Omega^{\ell-1} \; on \; \ell > 0, \qquad \hat{\Omega}^0 = \Omega^0. \tag{4.12}$$

These relationships are illustrated by Figure 4.1.

The operator $\mathcal{L}^\ell$ executes a uniform-grid operator $\mathcal{J}$ on level ℓ using input data on level ℓ as well as the levels above and below if available. The $\mathbf{U}^{\ell-1}$ dependence arises from the need to furnish boundary conditions on levels with $\ell > 0$. Boundary conditions are interpolated from the next coarser level into the "ghost" cells of $\hat{\Omega}^\ell$ for refined regions or computed based on the problem domain boundary as in the single level case on the coarsest level. The $\mathcal{R}$ operators account for a coarse-fine boundary correction which we will discuss in the following section.

The operators $\mathcal{J}$ or $\mathcal{J}_{\mathbf{f}}$ represent a fairly general single-level function that conforms with the structure of

$$\mathcal{J}\left(\hat{\mathbf{U}}^{\ell}\right) = \mathcal{I}\hat{\mathbf{U}}^{\ell} - \beta\left[\mathcal{D}\left(\mathcal{F}\left(\hat{\mathbf{U}}^{\ell}\right)\right) + \mathcal{S}\left(\hat{\mathbf{U}}^{\ell}\right)\right] \quad on \quad \Omega^{\ell}, \tag{4.13}$$

$$\mathcal{J}_{\mathbf{f}}\left(\hat{\mathbf{U}}^{\ell}\right) = \mathcal{F}\left(\hat{\mathbf{U}}^{\ell}\right) \quad on \quad \Omega^{\ell}. \tag{4.14}$$

These forms decompose the function into a diagonal term $\mathcal{I}$, a divergence of a flux $\mathcal{D}\mathcal{F}$ and source term $\mathcal{S}$ for $\mathcal{J}$ and a pure flux for $\mathcal{J}_{\mathbf{f}}$. All of the operators that we used to compute $\mathbf{F}^E$ and $\mathbf{F}^I$ in the previous chapter as well as the pressure solve and the constraints have been consciously written in this form.

Refluxing

We now turn our attention to the operators $\mathcal{R}$ and $\mathcal{R}_{\mathbf{f}}$ in Equations (4.10) and (4.11). Part of the need for developing the special form for $\mathcal{J}$ in Equation (4.13) stems from the special treatment needed by fluxes in the presence of a coarse-fine boundary $\delta\Omega^{\ell}$ in an AMR hierarchy.

Physically, flux terms represent the flow of mass, momentum, energy, *etc.* to or from adjacent cells. If the flux used to compute the divergence is different on each side of the boundary than a non-physical amount of information is added or lost resulting in an error. In other words, the finite-volume method will cease to be conservative. On a single grid, this never happens because there is no ambiguity in choosing the correct flux. However, on a coarse-fine boundary fluxes are multiply defined.

To motivate this special treatment, consider the computation of the a flux on the coarse and fine sides of a coarse-fine boundary as illustrated in Figure 4.2. Here we have a coarse grid half covered by a fine grid with a refinement ratio of 2. We seek to compute the horizontal contribution to the divergence of a flux $\mathcal{F}$ which is defined at each surface.

First consider the divergence at point A. The divergence can be written as

$$\mathcal{D}(\mathcal{F})_A \;=\; \frac{1}{h^c}\left[\mathcal{F}_2 - \mathcal{F}_1\right], \tag{4.15}$$

or as

$$\mathcal{D}(\mathcal{F})_A \;=\; \frac{1}{h^c}\left[\mathcal{A}\left(\mathcal{F}_{4,0}, \mathcal{F}_{4,1}\right) - \mathcal{A}\left(\mathcal{F}_{3,0}, \mathcal{F}_{3,1}\right)\right], \tag{4.16}$$

where h^c is the coarse-level grid spacing. Because the fine-level fluxes are presumably more accurate, we elect to compute the divergence at A using Equation (4.16).

At point C things are simpler. the only fluxes available are $\mathcal{F}_3$ and $\mathcal{F}_4$ so the answer is simply

$$\mathcal{D}(\mathcal{F})_C \;=\; \frac{1}{h^c}\left[\mathcal{F}_4 - \mathcal{F}_3\right]. \tag{4.17}$$

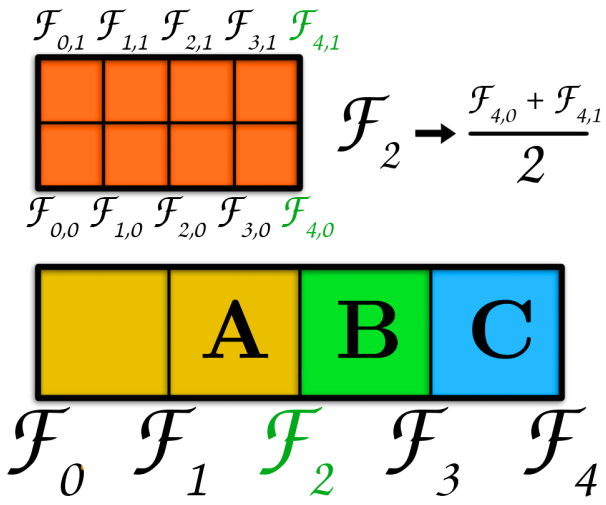


Figure 4.2: *Computing the coarse-level divergence at three points near the coarse-fine boundary*

The divergence at point B is where the ambiguity arises. The only option for the high side is $\mathcal{F}_3$, but the low-side flux is multiply defined. What should be done? The answer is that to maintain the conservation property of the finite-volume method on the coarse level, the flux used on the high side of point A must be the same as the flux used on the low side of point B. Hence we must be sure that the divergence at B is

$$\mathcal{D}(\mathcal{F})_B \;=\; \frac{1}{h^c}\left[\mathcal{F}_3 - \mathcal{A}\left(\mathcal{F}_{4,0}, \mathcal{F}_{4,1}\right)\right]. \tag{4.18}$$

In practice, we often do not have knowledge of the locations of the coarse-fine boundary locations when computing $\mathcal{J}$. Because of this, we compute $\mathcal{D}(\mathcal{F})$ assuming there is no finer level and then add a correction in the form of $\mathcal{R}$. These corrections are

$$\mathcal{R}\left(\mathbf{U}^\ell, \mathbf{U}^{\ell+1}\right) = \frac{1}{h^\ell}\left[\mathcal{A}\left(\mathcal{F}(\mathbf{U}^{\ell+1})\right) - \mathcal{F}(\mathbf{U}^\ell)\right] \quad on \quad \partial\Omega^\ell, \quad \ell < \ell_{max}, \tag{4.19}$$

$$\mathcal{R}_\mathbf{f}\left(\mathbf{U}^\ell, \mathbf{U}^{\ell+1}\right) = \mathcal{A}\left(\mathcal{F}(\mathbf{U}^{\ell+1})\right) - \mathcal{F}(\mathbf{U}^\ell) \quad on \quad \partial\Omega^\ell, \quad \ell < \ell_{max}. \tag{4.20}$$

These corrections have the effect of replacing coarse fluxes with the average of overlying fine fluxes on the coarse-fine boundary. Note the factor of $1/h^\ell$ which is absent in $\mathcal{R}_\mathbf{f}$. This difference stems from the lack of a divergence operator in $\mathcal{J}_\mathbf{f}$. Note also that because $\mathcal{R}$ uses the flux function $\mathcal{F}$, $\mathcal{R}$ also implicitly depends on $\mathcal{J}$.

Finest and Coarsest Level Cases

$\mathcal{L}^\ell(\hat{\mathbf{U}}^\ell, \mathbf{U}^{\ell-1}, \mathbf{U}^{\ell+1})$ is the most general version of the level operator, but we must consider the cases where $\ell = 0$ and $\ell = \ell_{max}$. The first case corresponds to the coarsest level in the hierarchy. The boundaries of Ω^0 are the boundaries of the problem domain, so the boundary conditions are computed using whichever method would be used in the case of a single-level problem. This version of $\mathcal{L}^\ell$ we write as $\mathcal{L}^\ell(\hat{\mathbf{U}}^\ell, \mathbf{U}^{\ell+1})$.

The case where $\ell = \ell_{max}$ corresponds to the finest level in the hierarchy. There are no coarse-fine boundaries on the interior of $\Omega^{\ell_{max}}$ and hence there is no refluxing to be done and $\mathcal{R}$ is identically zero. This version of $\mathcal{L}^\ell$ we write as $\mathcal{L}^\ell(\hat{\mathbf{U}}^\ell, \mathbf{U}^{\ell-1})$.

Both of these corner cases apply to the face-averaged operator $\mathcal{L}^\ell_{\mathbf{f}}$ as well.

In the pseudocode and discussion of these algorithms below, $\mathcal{L}^\ell(\mathbf{U}^\ell)$ is understood to mean the operator defined in Equation (4.10) or (4.11) with the appropriate coarse and fine level arguments depending on the value of ℓ.

4.4 Multigrid

Up until this point, we have said nothing at all about solvers for either single-level or multi-level problems. Here we finally introduce multigrid, a multi-level solver that complements the local refinement structure of AMR. We discuss the general approach of multigrid as well as two different implementations: the residual-correction scheme and the full approximation scheme.

Multigrid is a framework that leverages data defined recursively on N_{MG} grids refined with respect to one another by a ratio r_{MG}. As we will see, one critical difference between a multigrid hierarchy and an AMR hierarchy is that the former has no local refinement; each grid in the hierarchy covers the same region of space. That is to say, $\Omega^\ell = \Gamma^\ell$ for all ℓ. Multigrid takes advantage of the error smoothing properties that are innate to iterative relaxation methods like point-Jacobi or Gauss-Seidel to solve for $\mathbf{U}$ in the problem $\mathcal{L}(\mathbf{U}) = \mathbf{G}$ [38, 11].

The general approach of the multigrid algorithm is as follows. After smoothing the solution error with an iterative scheme for N_{down} iterations (typically 2-4) on the finest grid, the problem is then transferred to a coarser grid using the averaging operator $\mathcal{A}_{MG}$. Because relaxation damps high frequency modes in the error, the data is smooth and the coarse problem is a good approximation of the fine one [11]. The critical notion of multigrid is that smooth discrete Fourier modes become more oscillatory when expressed on a coarser grid. This phenomenon means that on the coarser grid, relaxation will be effective in problems where the convergence of the same iterative solver may have stalled on the finer grid. What's more, there is little need to be concerned about aliased high frequency modes because, again, these modes are preferentially damped by the iterative method [11, 38].

Once the problem is coarsened down to Ω^0, $\mathbf{U}^0$ is relaxed for N_{bottom} (≈ 20) iterations and interpolated back to the previous fine grid where it is used to correct the value of $\mathbf{U}^1$.

Interpolation tends to reintroduce high frequency modes, but this side effect is easily remedied with N_{up} ($\approx N_{down}$) more relaxation steps. This process continues until the algorithm returns to $\mathbf{U}^{N_{MG}-1}$ where it terminates with the final solution. This whole cycle from the finest level, down to the coarsest level, and back up again is termed a *v-cycle* after the pattern of recursive calls. It is usually the case that more than one v-cycle is needed to obtain the desired accuracy.

We introduce one more piece of context before we examine concrete multigrid implementations. Namely, we must discuss the role of the residual. If $\mathbf{U}^\star$ is an approximation to the true solution $\mathbf{U}$ to $\mathcal{L}\mathbf{U} = \mathbf{G}$, then we can write the error in $\mathbf{U}^\star$ as

$$\mathbf{E} = \mathbf{U} - \mathbf{U}^\star, \tag{4.21}$$

and the residual in $\mathbf{U}^\star$ as

$$\mathbf{R} = \mathbf{G} - \mathcal{L}(\mathbf{U}^\star). \tag{4.22}$$

If we assume $\mathcal{L}$ is linear, we can derive an auxiliary problem,

$$\mathcal{L}(\mathbf{E}) = \mathbf{R}, \tag{4.23}$$

which is in terms of $\mathbf{E}$ instead of $\mathbf{U}$. If we solve the residual equation for $\mathbf{E}$, then we can trivially determine $\mathbf{U} = \mathbf{U}^\star + \mathbf{E}$.

We now examine two specific implementations of multigrid. The first is the residual-correction formulation detailed in Algorithm 1. This is the formulation of multigrid that has been used for systems of equations arising in AMR computations since the 1990s [3, 33, 27].

Algorithm 1 Residual Correction Scheme Multigrid

```
1:  procedure RCSMultigridVCycle(U^ℓ, G^ℓ, ℓ)
2:      if ℓ > 0 then
3:          U^ℓ ← Relax(U^ℓ, G^ℓ, N_down)
4:          U^{ℓ-1} ← 0
5:          G^{ℓ-1} ← A_MG(G^ℓ − L(U^ℓ))  on  C_MG(Ω^ℓ)
6:          FASMultigridVCycle(U^{ℓ-1}, G^{ℓ-1}, ℓ − 1)
7:          U^ℓ += I_MG(U^{ℓ-1})
8:          U^ℓ ← Relax(U^ℓ, G^ℓ, N_up)
9:      else
10:         U^ℓ ← Relax(U^ℓ, G^ℓ, N_bottom)
11:     end if
12: end procedure
```

The inputs to RCSMultigridVCycle on a given level are the level solution $\mathbf{U}^\ell$, the right hand side forcing $\mathbf{G}^\ell$, and the level itself ℓ. An important feature of residual-correction multigrid is that only on the finest level do $\mathbf{U}^\ell$ and $\mathbf{G}^\ell$ represent the solution and forcing

of the problem $\mathcal{L}(\mathbf{U}) = \mathbf{G}$. At all coarser levels, these values are the error and residual respectively of the problem on the next finer level.

Starting on level ℓ, the algorithm first relaxes the error in the input $\mathbf{U}^\ell$ (line 3). The error on the next coarser level is then initialized as zero (line 4) and the coarse level residual is computed (line 5). The function is then called recursively. After the recursive call, $\mathbf{U}^{\ell-1}$ actually holds $\mathbf{E}^{\ell-1}$, the error in $\mathbf{U}^\ell$ resolved on the coarser grid. To update $\mathbf{U}^\ell$, the error is interpolated and added to $\mathbf{U}^\ell$ (line 7). Finally, the error on level ℓ is relaxed again to remove the high frequency error introduced by the interpolation operation (line 8). On the coarsest level where there is no recursive call to make, multigrid simply relaxes the error for a longer time, usually for about 20-30 iterations, before passing the result back up to the finer level.

There are two limitations to this algorithm as posed. The first is that $\mathcal{L}$ must be linear because the residual equation is used. Additionally, the problem must be posed with homogeneous boundary conditions because the initial value of the error is uniformly zero.

Algorithm 2 Full Approximation Scheme Multigrid

1: **procedure** FASMULTIGRIDVCYCLE($\mathbf{U}^\ell$, $\mathbf{U}_{BC}$, $\mathbf{G}^\ell$, ℓ)
2: **if** $\ell > 0$ **then**
3: $\mathbf{U}^\ell \leftarrow \text{RELAX}(\mathbf{U}^\ell, \mathbf{G}^\ell, N_{down})$
4: $\mathbf{U}^{\ell-1} \leftarrow \mathcal{A}_{\text{MG}}(\mathbf{U}^\ell)$
5: $\mathbf{U}_0^{\ell-1} \leftarrow \mathbf{U}^{\ell-1}$
6: $\mathbf{G}^{\ell-1} \leftarrow \mathcal{A}_{\text{MG}}(\mathbf{G}^\ell - \mathcal{L}(\mathbf{U}^\ell, \mathbf{U}_{BC}))$ *on* $\mathcal{C}_{\text{MG}}(\Omega^\ell)$
7: $\mathbf{G}^{\ell-1} \mathrel{+}= \mathcal{L}^{\ell-1}(\mathbf{U}^{\ell-1}, \mathbf{U}_{BC})$ *on* $\Omega^{\ell-1}$
8: FASMULTIGRIDVCYCLE($\mathbf{U}^{\ell-1}, \mathbf{G}^{\ell-1}, \ell-1$)
9: $\mathbf{U}^\ell \mathrel{+}= \mathcal{I}_{\text{MG}}\left(\mathbf{U}^{\ell-1} - \mathbf{U}_0^{\ell-1}\right)$
10: $\mathbf{U}^\ell \leftarrow \text{RELAX}(\mathbf{U}^\ell, \mathbf{G}^\ell, N_{up})$
11: **else**
12: $\mathbf{U}^\ell \leftarrow \text{RELAX}(\mathbf{U}^\ell, \mathbf{G}^\ell, N_{bottom})$
13: **end if**
14: **end procedure**

As an alternative, we examine the implementation of FAS-multigrid detailed in Algorithm 2. The inputs are the level solution $\mathbf{U}^\ell$, the level forcing $\mathbf{G}^\ell$, and optionally $\mathbf{U}_{BC}$, which is a coarse grid representation of $\mathbf{U}$ used to interpolate boundary conditions. This is different from the residual-correction version which assumes homogeneous boundary conditions, and only comes into play when this algorithm is used in tandem with AMR as we will see in the next section.

The FAS algorithm also departs from the residual-correction one at lines 4 and 5 where $\mathbf{U}^{\ell-1}$ is computed explicitly using averaging, and a copy of this quantity is saved in $\mathbf{U}_0^{\ell-1}$. The coarse level forcing is also different. Instead of computing the residual, the algorithm

computes a new forcing by setting the residual on levels ℓ and $\ell - 1$ to be equal,

$$\mathcal{A}\left(\mathbf{R}^{\ell}\right) = \mathbf{R}^{\ell-1}, \tag{4.24}$$

$$\mathcal{A}\left(\mathbf{G}^{\ell} - \mathcal{L}^{\ell}(\mathbf{U}^{\ell})\right) = \mathbf{G}^{\ell-1} - \mathcal{L}^{\ell-1}(\mathbf{U}^{\ell-1}), \tag{4.25}$$

$$\mathbf{G}^{\ell-1} = \mathcal{A}\left(\mathbf{G}^{\ell} - \mathcal{L}^{\ell}(\mathbf{U}^{\ell})\right) + \mathcal{L}^{\ell-1}(\mathbf{U}^{\ell-1}), \tag{4.26}$$

as shown on lines 6 and 7. With $\mathbf{U}^{\ell-1}$ and $\mathbf{G}^{\ell-1}$ computed, the function can be called recursively on the next coarser level. The output of the recursive call is not an error in $\mathbf{U}^{\ell}$ like in the residual-correction formulation, but is instead an updated value of $\mathbf{U}^{\ell}$ itself on the coarser grid. Hence, the algorithm must explicitly compute the error with which to update $\mathbf{U}^{\ell}$ (line 8). This is the reason why a copy of $\mathbf{U}^{\ell-1}$ needed to be stored before the recursive call.

The key takeaway from this comparision is that FAS-multigrid does not solve a residual equation nor does it place any restriction on the boundary conditions of the operator $\mathcal{L}$. Removing this restriction provides a more flexible algorithm which can solve affine or even non-linear operators with inhomogeneous boundary conditions [11].

In all of the above, no mention has been made of a specific relaxation routine to implement Relax in Algorithms 1 and 2. Our implementation uses point-Jacobi iteration, but in principle any routine with the high-frequency smoothing properties described above would suffice.

Optimizing Helmholtz Convergence

There are two Helmholtz solves present in our implementation of the all-speed projection algorithm, namely Equations (3.52) and (3.60). There is a potential problem in both of these formulations in that the diagonal coefficient of the Helmholtz operator scales like $1/p$. This value can be quite small, especially in the low-Mach limit since $M^2 \sim 1/p$. As written, the eigenvalue corresponding to the constant eigenmode will likewise become very small and hence the convergence of iterative methods will tend to stall for that mode. There is also no clear way to scale the equation which will not retain a large scale difference between the constant and non-constant eigenmodes.

We rectify this problem by making the bottom-most iterative solve of FAS-Multigrid semi-analytic. To do this, we wrap Algorithm 2 in Algrithm 3 as shown below.

Here we use the $\bar{(\cdot)}$ operator to denote the average value over the whole domain as opposed to the average over a cell. Line 3 of Algorithm 3 is the key to the fix; here we factor out the constant eigenmode of the problem $\mathcal{L}(\mathbf{U}) = \mathbf{G}$ yielding a well conditioned Helmholtz solve in all the other eigenmodes (line 4). Finally, in line 5 we analytically solve the constant eigenmode problem and update the solution accordingly.

Using this method, we solve for the constant eigenmode directly while separately solving for the non-constant modes iteratively.

Algorithm 3 Helmholtz Modification to FAS-Multigrid

procedure HELMHOLTZSOLVE($\mathbf{U}^{\ell_{MG}}$, $\mathbf{G}^{\ell_{MG}}$, ℓ_{MG}, ℓ_{AMR})
  if $\ell_{MG} = 0$ and $\ell_{AMR} = 0$ and $\mathcal{I} \neq 0$ **then**
    $\mathbf{G}_0^{\ell_{MG}} \leftarrow \mathbf{G}^{\ell_{MG}} + (\bar{\mathcal{I}} - \mathcal{I})\mathbf{U}^{\ell_{MG}}$
    RELAX on $\left[\bar{\mathcal{I}}\mathbb{I} - \beta\left(\mathcal{D}\mathcal{F} + \mathcal{S}\right)\right]\mathbf{U}^{\ell_{MG}} = \mathbf{G}_0^{\ell_{MG}}$
    $\mathbf{U}^{\ell_{MG}} \mathrel{+}= \frac{1}{\bar{\mathcal{I}}}\bar{\mathbf{G}}_0^{\ell_{MG}} - \bar{\mathbf{U}}^{\ell_{MG}}$
  else
    FASMULTIGRIDVCYCLE($\mathbf{U}^{\ell_{MG}}, \mathbf{G}^{\ell_{MG}}, \ell_{MG}$)
  end if
end procedure

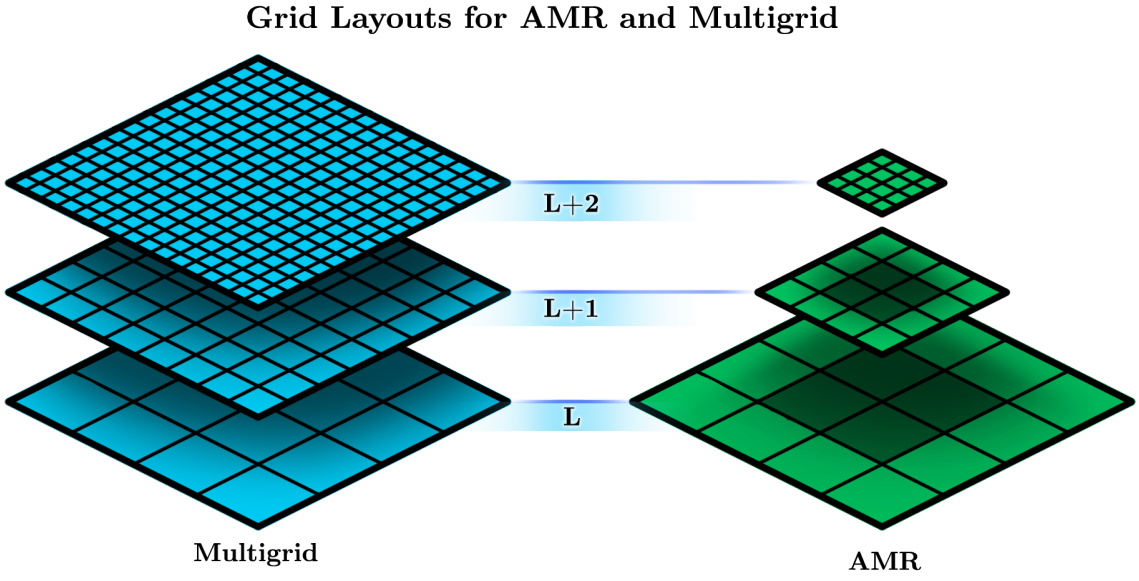


Figure 4.3: *Structures of AMR verusus multigrid grid layouts. Only AMR (right) implements local refinement. Both of the layouts shown use a refinement ratio of 2.*

4.5 AMR Algorithms

We now have established all the fundamentals needed to evaluate $\mathcal{L}^{comp}(\mathbf{U}^{comp})$ as well as to solve the problem $\mathcal{L}^{comp}(\mathbf{U}^{comp}) = \mathbf{G}^{comp}$ where $\mathbf{G}^{comp}$ is a known forcing and $\mathbf{U}^{comp}$ is unknown.

We have already spoken about multigrid, which makes use of successively refined versions of the same domain. The grid layout of AMR is distinct from multigrid in that more refined domains cover less and less of the problem domain Ω^0. This is where local refinement enters the picture. Figure 4.3 illustrates the distinction between AMR and multigrid grid layouts.

AMR Apply

First we present Algorithm 4, the procedure for evaluating $\mathcal{L}^{comp}$. Here the terminology $\mathcal{A}_{AMR}$ and $\mathcal{C}_{AMR}$ are used to represent inter-level operators which use r_{AMR} as their refinement ratio. The averaging operations are there to make sure that the most precise value of $\mathbf{U}^{\ell-1}$ is used to compute $\mathcal{L}^{\ell-1}$ and the boundary conditions for $\mathcal{L}^{\ell}$.

Algorithm 4 AMR Operator Apply

procedure AMRApply($\mathbf{L}^{comp}$, $\mathbf{U}^{comp}$, ℓ)
  if $\ell = 0$ **then**
    $\mathbf{L}^{\ell} \leftarrow \mathcal{L}^{\ell}(\mathbf{U}^{\ell}, \mathbf{U}^{\ell+1})$
  else if $\ell = \ell_{max}$ **then**
    $\mathbf{U}^{\ell-1} \leftarrow \mathcal{A}_{AMR}(\mathbf{U}^{\ell}) \quad on \quad \mathcal{C}_{AMR}(\Omega^{\ell})$
    AMRApply($\mathbf{L}^{comp}, \mathbf{U}^{comp}, \ell - 1$)
    $\mathbf{L}^{\ell} \leftarrow \mathcal{L}^{\ell}(\mathbf{U}^{\ell}, \mathbf{U}^{\ell-1})$
  else
    $\mathbf{U}^{\ell-1} \leftarrow \mathcal{A}_{AMR}(\mathbf{U}^{\ell}) \quad on \quad \mathcal{C}_{AMR}(\Omega^{\ell})$
    AMRApply($\mathbf{L}^{comp}, \mathbf{U}^{comp}, \ell - 1$)
    $\mathbf{L}^{\ell} \leftarrow \mathcal{L}^{\ell}(\mathbf{U}^{\ell}, \mathbf{U}^{\ell-1}, \mathbf{U}^{\ell+1})$
  end if
end procedure

AMR FAS-Multigrid

Solving $\mathcal{L}^{comp}(\mathbf{U}^{comp}) = \mathbf{G}^{comp}$ using AMR is a bit more involved. AMR itself is not a solver, and so we will need access to a subroutine that provides this utility. We choose to use FAS-multigrid as our solver.

Algorithm 5 describes the solution process. AMRVCycle is a subroutine which takes an initial guess for $\mathbf{U}^{comp}$ and the right hand side $\mathbf{G}^{comp}$ and improves upon the guess. The improvement is executed as follows. Starting at the finest level ℓ_{max}, the level solution $\mathbf{U}^{\ell}$ is relaxed using FAS-multigrid. Boundary conditions are provided using interpolated data from the next coarser level $\mathbf{U}^{\ell-1}$. A coarsened copy of the relaxed solution is then saved and an updated forcing $\hat{\mathbf{G}}^{\ell-1}$ for the next coarser level is computed using the same logic as in Equations (4.24) through (4.26). This updated forcing accounts for the progress that has been made by relaxing on this level. AMRVCycle is then called recursively on the next coarser level with $\hat{\mathbf{G}}^{\ell-1}$ as the forcing. On the coarsest level, the problem $\mathcal{L}^{0}(\mathbf{U}^{0}) = \hat{\mathbf{G}}^{0}$ is solved outright using FAS-multigrid.

After the recursive call on level ℓ, the coarse level error can be computed as $\mathbf{U}^{\ell-1} - \mathbf{U}_{0}^{\ell-1}$ where $\mathbf{U}_{0}^{\ell-1}$ is the copy that was saved prior to the recursive call. $\mathbf{U}^{\ell}$ can be updated using the interpolated value of this error. Finally, one last smoothing is done using FAS-multigrid

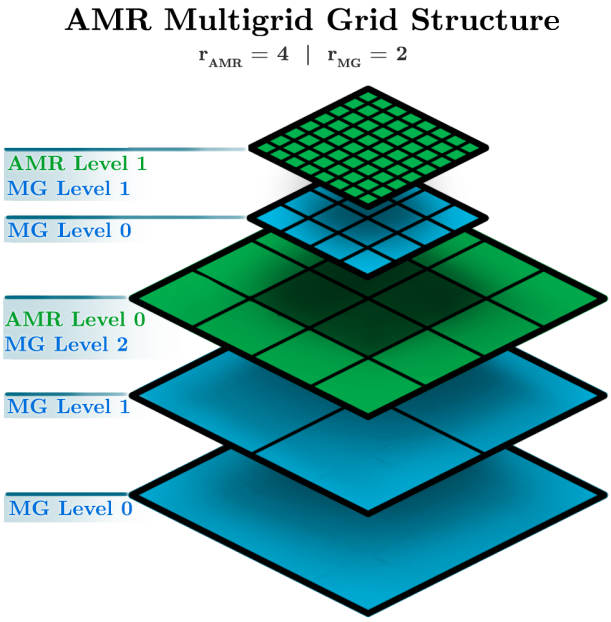


Figure 4.4: *AMR-Multigrid grid structure.* $r_{AMR} = 4$ *and* $r_{MG} = 2$

.

before $\mathbf{U}^\ell$ is passed back up to the next finer level. This process continues until the algorithm returns to the finest level.

This pattern of relaxation, averaging down, interpolation, and correction is almost identical to what is done in FAS-Multigrid. The salient differences are that AMRVCycle *uses* FASMultigridVCycle as it's relaxation routine and that additional steps must be taken when computing the coarse level forcing to account for local refinement. Finally, each call to the level operator $\mathcal{L}$ is making use of the refluxing correction $\mathcal{R}$ at coarse-fine boundaries, a step which is unnecessary in the pure FAS-multigrid algorithm.

At the end of the v-cycle, the solution may or may not be good enough. The AMRSolve function checks the solution using the norm of the residual, and repeatedly calls AMRVCycle until the norm of the residual is smaller than a provided tolerance. This iterative application of v-cycles to improve the solution is how all problems of the form $\mathbf{U} = \mathcal{L}^{-1}\mathbf{G}$ are solved. As a final note, we make the observation that if $\ell_{max} = 1$ then AMRVCycle simply reduces to solving a single level problem using FAS-multigrid.

Multigrid Subcycling

We must elaborate on a few points pertaining to AMR and FAS-multigrid operate together. As shown in Algorithm 5, FAS-multigrid is called by AMR in very much the same way that Relax is called by FAS-multigrid. On the coarsest AMR level, the process is fairly obvious. However we have not yet explained how multigrid should function on the finer AMR levels.

Algorithm 5 Full AMR algorithm using FAS-Multigrid

procedure AMRSolve($\mathbf{U}^{comp}, \mathbf{G}^{comp}, N_{solve}, \tau$)
  $\mathbf{R}^{comp} = \mathbf{G}^{comp} - \mathcal{L}^{comp}(\mathbf{U}^{comp})$
  $R_0 = ||\mathbf{R}^{comp}||$
  for $n \in [1, N_{solve}]$ **do**
    AMRVCycle($\mathbf{U}^{comp}, \mathbf{G}^{comp}, \mathbf{G}^{\ell_{max}}, \ell_{max}$)
    $\mathbf{R}^{comp} = \mathbf{G}^{comp} - \mathcal{L}^{comp}(\mathbf{U}^{comp})$
    $R_i \leftarrow ||\mathbf{R}^{comp}||$
    if $R_i/R_0 < \tau$ **then**
      return success
    end if
  end for
  return failure
end procedure
procedure AMRVCycle($\mathbf{U}^{comp}, \mathbf{G}^{comp}, \hat{\mathbf{G}}^{\ell}, \ell$)
  if $\ell = 0$ **then**
    $\mathbf{U}^{\ell} \leftarrow$ FASMultigridVCycle($\mathbf{U}^{\ell}, \hat{\mathbf{G}}^{\ell}, N^{\ell}_{MG} - 1$)
  else
    $\mathbf{U}^{\ell} \leftarrow$ FASMultigridVCycle($\mathbf{U}^{\ell}, \mathbf{U}^{\ell-1}, \hat{\mathbf{G}}^{\ell}, N^{\ell}_{MG} - 1$)
    $\mathbf{U}^{\ell-1} \leftarrow \mathcal{A}_{AMR}(\mathbf{U}^{\ell})$
    $\mathbf{U}^{\ell-1}_0 \leftarrow \mathbf{U}^{\ell-1}$
    $\hat{\mathbf{G}}^{\ell-1} \leftarrow \mathbf{G}^{\ell-1}$
    if $\ell < \ell_{max}$ **then**
      $\hat{\mathbf{G}}^{\ell-1} \mathrel{+}= \mathcal{A}_{AMR}\left(\mathbf{G}^{\ell} - \mathcal{L}^{\ell}(\mathbf{U}^{\ell}, \mathbf{U}^{\ell-1}, \mathbf{U}^{\ell+1})\right) \quad on \quad \mathcal{C}_{AMR}(\Omega^{\ell})$
    else
      $\hat{\mathbf{G}}^{\ell-1} \mathrel{+}= \mathcal{A}_{AMR}\left(\mathbf{G}^{\ell} - \mathcal{L}^{\ell}(\mathbf{U}^{\ell}, \mathbf{U}^{\ell-1})\right) \quad on \quad \mathcal{C}_{AMR}(\Omega^{\ell})$
    end if
    AMRVCycle($\mathbf{U}^{AMR}, \mathbf{G}^{AMR}, \hat{\mathbf{G}}^{\ell-1}, \ell - 1$)
    $\mathbf{U}^{\ell} \mathrel{+}= \mathcal{I}^{0}_{AMR}\left(\mathbf{U}^{\ell-1} - \mathbf{U}^{\ell-1}_0\right)$
    $\mathbf{U}^{\ell} \leftarrow$ FASMultigridVCycle($\mathbf{U}^{\ell}, \mathbf{U}^{\ell-1}, \hat{\mathbf{G}}^{\ell}, N^{\ell}_{MG} - 1$)
  end if
end procedure

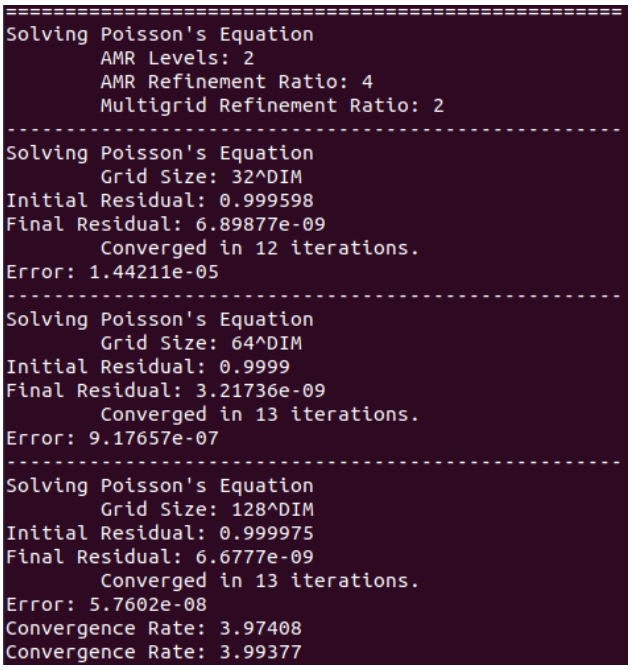


Figure 4.5: *Performance of AMR FAS-Multigrid for solving Poisson's Equation with* $r_{AMR} = 4$ *and* $r_{MG} = 2$ *on 2 AMR levels*

.

The answer to this question depends on the relative refinement ratios of the AMR and multigrid algorithms. When working in tandem, we enforce the restriction $r_{AMR} \geq r_{MG}$. When $r_{AMR} = r_{MG}$, then the calls to multigrid on the mid-levels of AMR have only a single multigrid level, and hence reduce down to calls to RELAX. However, if $r_{AMR} > r_{MG}$ then multigrid levels can exist at refinements *between* AMR levels. We disallow multigrid to have levels as coarse or coarser than the next coarsest AMR level, so there will only be $\log_2(r_{AMR}/r_{MG}) + 1$ multigrid sublevels for the calls on AMR levels greater than 0. This hybrid grid structure is illustrated by Figure 4.4.

These multigrid *subcycles* also obtain their boundary conditions in a different way. Instead of averaging the boundary conditions down from the input state, multigrid subcycles have access to the next coarsest AMR level and can interpolate boundary conditions from that data. This is what the argument $\mathbf{U}_{BC}$ in Algorithm 2 is used for.

AMR FAS-Multigrid Performance

Using FAS-multigrid in place of residual-correction multigrid represents a significant departure from previous AMR solver implementations. Figure 4.5 provides some results that illustrate the performance of our implementation. Here we use AMR FAS-multigrid to solve Poisson's equation on three different grid hierarchies of increasing resolution. Each hierarchy

consists of two AMR levels with $r_{AMR} = 4$ and $r_{MG} = 2$, the same configuration as that illustrated in Figure 4.4.

From these results we note that the residual is decreased in magnitude by about an order of magnitude by each v-cycle. The number of iterations required for the solver to converge is also essentially independent of the grid spacing. Finally, the Poisson solver furnishes a fourth-order accurate solution. These results are excellent and underscore our confidence in the decision to use FAS-multigrid for our AMR framework solver.

4.6 A Review of High-Order AMR Finite-Volume Methods

Block-structured AMR on uniform nested grids was first formulated in 1989 by Berger and Colella [9]. That study included a finite-volume discretization and implemented the refluxing procedure discussed herein, however only explicit methods were used to advance the system in time. Thompson and Ferziger introduced the idea of using multigrid to solve elliptic equations for AMR as applied to the steady Navier-Stokes equations in [42]. Implicit solver methods using residual-correction AMR multigrid were elaborated on by [3] who applied the paradigm to a nodal-point discretization for a vortex method, and later by [33] and [27] who applied it to a second-order finite-volume discretization.

High-order finite-volume methods for AMR are a relatively new idea. They were first examined by Barad and Colella in [7] using Mehrstellen stencils to solve Poisson's equation. These ideas were elaborated on by McCorquodale and Colella in [31] where they were used to simulate explicit gas dynamics in an AMR framework.

Our method uses the ideas of [31], though we have not implemented local refinement in time. We have replaced residual-correction multigrid with FAS-multigrid as our implicit solver. The impact of this choice is a modest amount of additional complexity in the implementation and greatly increased flexibility. The formulations in [33] and [27] use very specific stencils to put the single-level problem in residual-correction form. With FAS-multigrid, concerns like this are completely eliminated. Boundary conditions are factored out of the multigrid structure entirely and we can now apply AMR-multigrid to affine or even non-linear operators in a way not available in the residual-correction paradigm.

Chapter 5

Proto

In this chapter we take a detour away from the scientific computing component of this study to discuss software development. To the practiced reader this may feel like an unusual transition, but that sense of strangeness only serves to underscore a larger problem. For many years, the field of scientific computing has been able to proceed incrementally thanks to relatively stable hardware targets running well-implemented *Fortran*. However, current trends suggest that the composition of high-performance computers is rapidly changing in exotic ways, and software development will have adapt to keep up.

5.1 Trends in High Performance Computing

For many years prior to the time of writing, scientific computing has been a booming field. Between June 1993 and June 2001, the performance leaders in high-performance computing jumped from 59.7 GFlops to 7.23 TFlops. Perhaps even more surprising, the rank 500 machine in June 2001 also outpaced the rank 1 machine in 1993. As of November 2020, the industry leaders are clocking in at hundreds of TFlops with the number 1 machine - Supercomputer Fugaku - putting up numbers north of 0.5 *Exa*Flops and the number 500 machine reporting over 1 PetaFlops [35, 41].

Much of this dramatic growth was fueled by the advent of the manycore paradigm. Between June 1993 and June 2001, the number of cores used in the top machines jumped from a few hundred to a few thousand. Since then, that number has reached well into the millions or even tens of millions [41]. Cores themselves didn't change all that much, often remaining very similar to Intel X86 systems. New systems simply added more and more of them. Meanwhile, the software running on these systems was able to become increasingly tuned to the stationary target. Codes became very well optimized, but also increasingly reliant on the manycore model.

The last decade has seen a shift from the manycore mentality. We now see exotic architectures like Intel's Many Integrated Core (MIC) nodes, ARM chips, Graphical Procesing Units (GPUs) and Field Programmable Gate Arrays (FPGAs). These technologies are hardly

theoretical; Oak Ridge National Laboratory's Summit computer embraces the heterogenous computing model including 2 IBM POWER9 CPUs and 6 Nvidia Tesla GPUs in each of it's 4,608 nodes [30].

There are several causes for this shift. Physical limitations preclude the annual increases in CPU clock speed observed in the past. Thermodynamics also play a role; the current fastest machines consume power at a staggering rate of 10s of MW. Memory is also a concern, with more and more complex hierarchies of "fast" and "slow" memory in evidence in new technologies [25].

As it stands, programmers of simulation software must fill the role of "full stack" developers. They must have intimate knowledge of the relevant physics and mathematics and of the hardware layout of the machine on which their code will run. Deploying an application on a different hardware configuration often requires an entirely different implementation to take advantage of potential performance gains. Using GPU targets as an example, the developer must keep in mind software prefetching, register planning, CUDA streams, scatter stencil loops, texture memory directives, optimal shared memory use, and the list goes on and on [21]. All the while, the physical problems that we would like to solve are growing in scope and complexity.

5.2 Proto Design and Goals

Proto was designed with three traits in mind: performance, expressiveness, and portability. Of these, performance is the most self explanatory. We want to avoid the same kind of performance gap that existed after the transition to the manycore paradigm. The goal for *Proto* is at least 50% of theoretical peak algorithmic performance based on a faithful performance model for the algorithm and platform.

Expressiveness refers to how easy an algorithm can be translated from its defining mathematical expressions into high-level application code. This quality can be quantified in the expansion factor between lines of mathematics and lines of code. The languages used in most scientific computing applications today are not very expressive, and the ones that *are* either sacrifice performance or generality [24, 39]. With the tools available in C++11 and C++14 - variadic templates and move semantics to name a few - we believe that performant, expressive code is within reach. Our goal for *Proto* is a no more than a ten-fold expansion between algorithm specification and implementation.

Portability refers to the ability of an application to be run on different computing architectures without reimplementation. As demonstrated above, portability is likely going to be a much more important feature in the future than it has been. Algorithms will likely outlive several computer architectures now, and multiple architectures are already being developed in parallel. It is already the case that nearly every high-performance application needs to be rewritten to some degree. With *Proto* we would like to do the best we can to make sure we do it right the first time by providing a simple build path through a C++ header-only library and a single-source multiple-target implementation.

5.3 Proto Syntax

One of the core design principle of *Proto* is to provide an expressive syntax for implementing operators on a single data patch. This is achieved through a framework for implementing pointwise and stencil functions. Following is a brief description of the elementary objects in *Proto* accompanied by examples which illustrate the framework's expressiveness.

Basic Objects

Listing 5.1 Usage of basic Proto objects

```
#include "Proto.H"

int main(int argc, char** argv)
{
    int domainSize = 16;

    // Define the points (0, 0, ...) and (15, 15, ...)
    Point p0 = Point::Zeros();
    Point p1 = Point::Ones(domainSize - 1);

    // Build a Box from two Points:
    Box domain(p0, p1);

    // Build a BoxData with 3  components from a Box:
    BoxData<double, 3> phi(domain);

    // Initalize the data in phi to 0
    phi.setVal(0);
}
```

First we must very briefly describe the atomic elements of *Proto*: Point, Box, and BoxData. A Point represents an element $\mathbf{p} \in \mathbb{Z}^D$. Two Point objects which satisfy $p_1^d <= p_2^d \quad \forall d \in [0, D)$ define a Box, a rectangular region in $\mathbb{Z}^D$. BoxData<T,C,D,E> is the principal data holder object in *Proto* and defines a multidimensional array with values given by a $CxDxE$ component tensor of T-type data at each point. Usually D and E are equal to 1, leaving C to define the number of components. $C = 1$ corresponds to a scalar valued variable. Listing 5.1 demonstrates how these objects work together.

Listing 5.2 A simple Proto FORALL example

```
#include "Proto.H"

void f_initPhi(Point& a_p, Var<double, 2>& a_phi, double a_h)
{
    // Assume DIM >= 2
    double x = a_p[0]*a_h + a_h/2.0;
    double y = a_p[1]*a_h + a_h/2.0;
    a_phi(0) = sin(x)*sin(y);
    a_phi(1) = cos(x)*cos(y);
}

int main(int argc, char** argv)
{
    int domainSize = 16;
    double L = 2.0*M_PI;
    double dx = L/domainSize;

    Box domain = Box::Cube(domainSize);
    BoxData<double, 2> phi(domain);

    forallInPlace_p(f_initPhi, phi, dx);
}
```

Pointwise Operations

Pointwise functions account for any operators of the form

$$\psi_{\boldsymbol{i}} = \mathcal{F}(\alpha_{\boldsymbol{i}}, \beta_{\boldsymbol{i}}, \gamma_{\boldsymbol{i}}, ..., a, b, c, ...),$$

where $\alpha_{\boldsymbol{i}}$ etc are patches of data evaluated at a point $\boldsymbol{i}$ and a, b, c etc. are constants. This is a very general class of operators. The only restriction is that all array inputs are evaluated at the same point.

Proto implements pointwise functions with a set of variadic functions named FORALL* which accept user-defined functions as inputs. There are versions of FORALL which return a BOXDATA or operate in place. The POINT at which the function is being evaluated is also available. See Listing 5.2 for an example of application.

We must make special mention of the VAR<T,C,D,E> data type which appears in Listing 5.2. This object is associated with a BOXDATA<T,C,D,E> input to FORALL and facilitates the pointwise execution.

Listing 5.3 A simple Proto STENCIL example

```
#include "Proto.H"

int main(int argc, char** argv)
{
    // Inputs
    int domainSize = 16;
    Box domain = Box::Cube(domainSize);
    BoxData<double, 4> phi(domain);
    phi.setVal(1);
    double dx = 0.1;

    // Create the Stencil
    Stencil<double> S = (-2.0*DIM)*Shift::Zeros();
    for (int dir = 0; dir < DIM; dir++)
    {
        S += 1.0*Shift::Basis(dir, +1);
        S += 1.0*Shift::Basis(dir, -1);
    }
    S *= 1.0/(dx*dx);

    // Apply the Stencil
    BoxData<double, 4> psi = S(phi);
}
```

Stencils

A stencil is an operator,

$$\psi_{\mathbf{i}} = \sum_{\mathbf{s}} (w_{\mathbf{s}} \phi_{\mathbf{i}+\mathbf{s}}) \qquad \mathbf{s} \in \mathbb{Z}^{\mathbb{D}},$$

that computes an output value at $\mathbf{i}$ from a linear combination of input values near $\mathbf{i}$. Both ψ and ϕ are allowed to be arrays so long as they have the same number of components. STENCIL covers the principle use case that FORALL misses, and together they represent the vast majority of operations that an application developer would want to execute on a patch. Stencil operators are first-class objects in *Proto*, so they can be built once and executed as many times as desired. The syntax for creating and applying a STENCIL is demonstrated in Listing 5.3.

N	*Proto* (GFlops)	DGEMM (GFlops)
64	1.4	3.0
128	1.9	2.4
256	2.3	2.0
512	2.4	2.0

Table 5.1: *Proto stencil and pointwise operator performance as compared with matrix multiplication using DGEMM*

5.4 Proto Efficiency

We examine the effectiveness of *Proto* in terms of two of its three design principles: expressiveness and performance. The matter of portability is beyond the scope of the current study.

Performance

We examine the speed of *Proto* by comparing the performance of a test problem including both stencil and pointwise operations in *Proto* with the performance of a benchmark that multiplies two NxN matrices using DGEMM as a triply-nested for-loop. The *Proto* test problem is

$$L \leftarrow \frac{1}{4D}\Delta(\phi), \tag{5.1}$$

$$\phi \leftarrow \phi + L - R\frac{h^2}{4D}, \tag{5.2}$$

where L, ϕ, and R are N^3 sized arrays. Equation (5.1) utilizes *Proto*'s stencil operator and Equation (5.2) uses FORALL. This test problem consists of $15N^3$ Flops which is comparable to the $2N^3$ Flops of the DGEMM computation. These problems are also comparable because the stride-one access for both are of length N, so the degree of vectorization is also equivalent. This is also the reason why we have implemented DGEMM as a nested for-loop even though there are more efficient configurations for implementing matrix multiplication.

Table 5.1 summarizes the results of this test. The comparison was executed on an Apple MacBook Pro equipped with a 3.1 Ghz Intel Core i7 processor. Both examples were compiled using clang++ with -O3 optimization.

We note that the results are at worst within a factor of 2 of each other. The *Proto* result even appears to do better than DGEMM for larger problems. This is partially because the *Proto* test problem has a higher arithmetic intensity and also due to the fact that for larger problem sizes the matrices used in the DGEMM problem cease to fit in the L3 cache. We should not examine these results too rigorously; the major takeaway is that *Proto* shows similar performance to DGEMM when solving a roughly comparable problem.

Expressiveness

As a case study, we present Listing 5.4, an implementation of the STEP method from the fourth-order Euler solver used in [31]. The corresponding mathematical specification of the algorithm is given by Equations (12) through (20) in [31].

Listing 5.4 *Proto* implementation of the calculation of an explicit right-hand side for a fourth-order accurate method for Euler's equations

```
namespace Euler
{
    void step(BoxData<double, NUMCOMPS>& a_rhs,
        const BoxData<double, NUMCOMPS>& a_U)
    {
        a_rhs.setVal(0.0);
        auto W_bar = forall<double, NUMCOMPS>
            (f_consToPrim, a_U, s_gamma);
        auto U = m_deconvolve(a_U);
        auto W = forall<double, NUMCOMPS>
            (f_constToPrim, U, s_gamma);
        auto W_ave = m_laplacian(W_bar, 1.0/24.0);
        W_ave += W;
        for (int d = 0; d < DIM; d++)
        {
            auto W_ave_f = m_interp[d](W_ave);
            auto F_bar_f = forall<double, NUMCOMPS>
                (f_getFlux, W_ave_f, d, s_gamma);
            F_ave_f += m_laplacian_f[d](F_bar_f, 1.0/24.0);
            a_rhs += m_divergence[d](F_ave_f);
        }
        a_rhs *= -1.0/s_dx;
    } // close step
} // close namespace
```

Using *Proto*, we are able to represent 9 lines of mathematics in just 24 lines of code broken into 11 statements and a for-loop. Even if we ignore the implementation of the function inputs in the calls to FORALL, the equivalent *Fortran* would require an order of magnitude more lines of code. This is because in *Fortran*, the programmer needs to explicitly compute loop bounds and write loops themselves. In the case of writing an implementation for GPUs using *CUDA*, the necessary code increases by yet another order of magnitude.

This example reveals the utility of a few other aspects of *Proto* which have not yet been discussed. *Proto* implements domain inference for both pointwise operators and stencils.

This optimization automatically computes the maximum appropriate range that an operator can produce given the combined domains of the inputs. Domain inference obviates the need for many lines of code which would otherwise be required to define these ranges. Like most of the automated functionalities of *Proto*, domain inference is optional and can be overridden by the application developer if necessary. *Proto* also takes advantage of *C++11* move-semantics which allow the programmer to make use of the assignment (=) operator for BoxData objects without having to worry about losing efficiency to unnecessary data copying.

Lastly, we note that many of the stencil operations used in this implementation are common across a wide array of structured grid applications. The $2N + 1$ point Laplacian operator is a good example of a function that is used in a myriad of contexts. Stencils of this variety are included in the implementation of *Proto* as a library of operators which is constantly growing over time. This study has made its own contribution to *Proto*'s stencil library in the form of the various fourth-order operators described in Chapter 3.

Chapter 6

Results

The sections to follow demonstrate the effectiveness of the all-speed projection algorithm with AMR and FAS-multigrid. All computations in this chapter are performed in serial on a 3.5 Ghz Intel Core i5 processor running Ubuntu 18.08 and compiled using g+ version 7.5.0. We will show good agreement with the known behavior of canonical problems and showcase the power of AMR when applied to a problem with multiple length scales that differ by orders of magnitude.

For these problems, we specialize to the case of a polytropic gas. The equation of state in Equation (2.4) simplifies to

$$p(\rho, h) \;=\; \frac{\gamma - 1}{\gamma}\rho h, \tag{6.1}$$

and the speed of sound is

$$c^2 = \frac{\gamma p}{\rho}. \tag{6.2}$$

6.1 Vortex in Stratified Flow

For our initial validation, we examine the well known vortex in a box problem. Our version,

$$v_x = u_0 \sin^2(\pi x)\sin(\pi y)\cos(\pi y), \tag{6.3}$$

$$v_y = -u_0 \sin^2(\pi y)\sin(\pi x)\cos(\pi x), \tag{6.4}$$

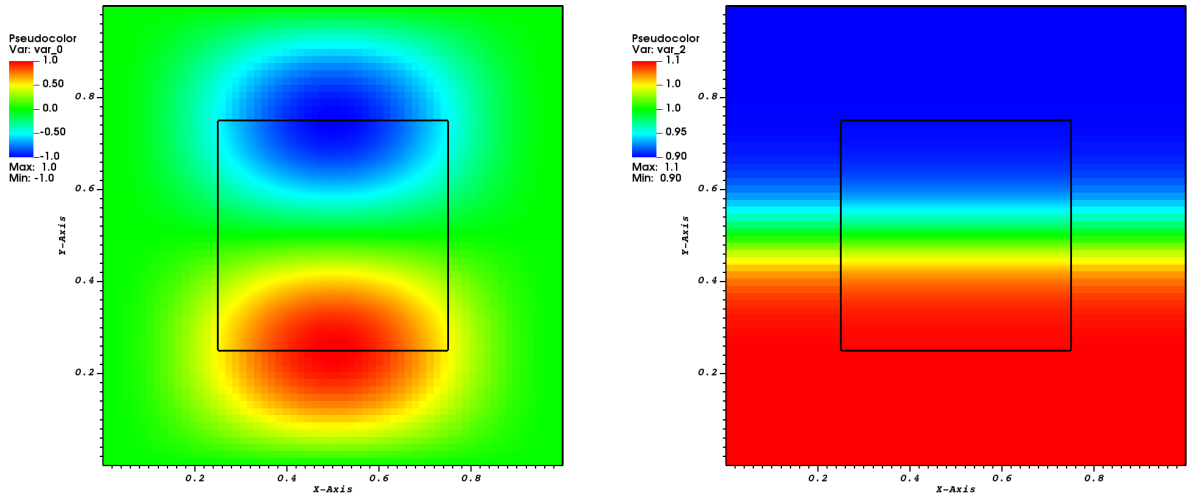
$$\rho = \rho_0 \left[1 - \epsilon \tanh\left(\frac{y - 0.5}{\delta}\right)\right], \tag{6.5}$$

$$p = p_0, \tag{6.6}$$

$$\rho h = \frac{p_0 \gamma}{\gamma - 1}, \tag{6.7}$$

includes density stratification in the vertical direction, but since we have neglected gravitational effects there is no restoring force to maintain stability and our initial condition is expected to simply advect with the flow.

The domain is $x, y \in [0, 1]$ and the parameters are $\rho_0 = u_0 = 1$, $\epsilon = 0.1$, and $\delta = 0.1$. p_0 is used to control M, and we examine the problem for values of p_0 ranging from 10 to 10^5. The boundary conditions are solid walls in the y-direction and periodic in the x-direction. Figure 6.1 shows what this initial condition looks like for the horizontal velocity and density.



(a) *Horizontal velocity (v_x)* ***(b)*** *Density (ρ)*

Figure 6.1: *This is the initial condition for the stratified vortex problem. The black line designates the region of refinement.*

Here we will show second-order accuracy of the algorithm. Though the original goal was to show fourth-order accuracy, progress on the algorithm has not reached that point at the time of writing. Because of the extensible nature of the implementation, there is no reason to believe that high-order accuracy is not possible. Tables 6.1 and 6.2 summarize the error in each state variable in both the L_1 and L_∞ norms and the convergence behavior respectively for the low-Mach case ($M = 0.004$). Clearly we do not observe anything resembling convergence when it comes to the acoustic variables, but we also do not expect to. The domain of this problem is far too small to resolve any possible long wavelength acoustics. However we do observe good convergence for the advective variables, specifically the divergence-free velocity and the density.

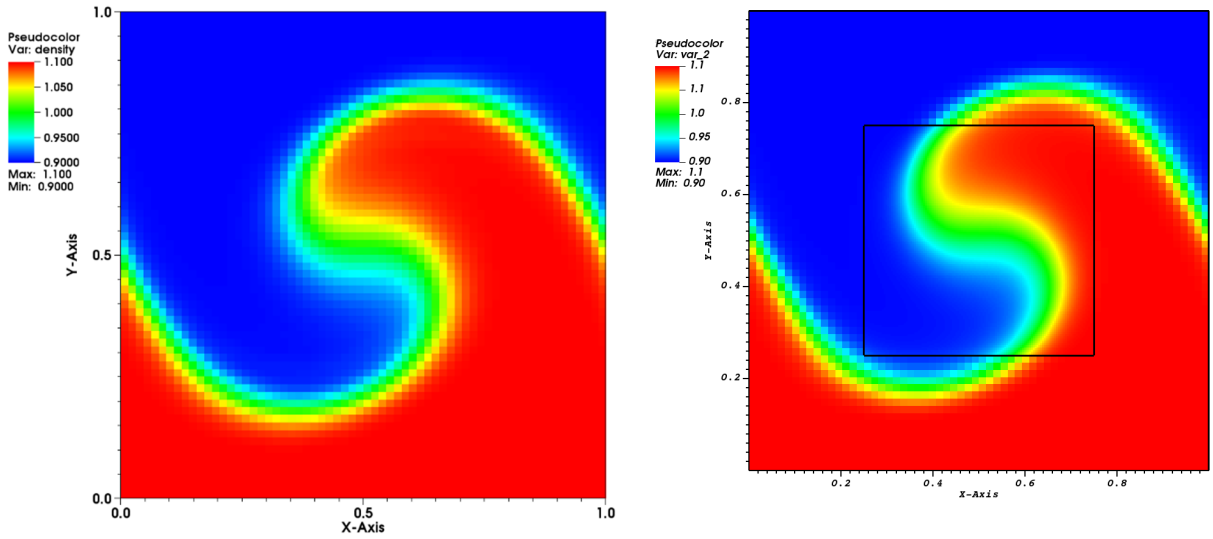
As an additional performance check, we observe the qualitative behavior of our outputs as compared to those in [12]. Figures 6.2 and 6.3 compare the solutions for the density, divergence-free velocity, and vorticity respectively. The agreement is good, though the peak-values are different as we would expect considering the single-level method is fourth-order while the AMR method is only second-order.

Variable Name	Richardson Error			
Variable	L_1 Coarse	L_1 Fine	L_∞ Coarse	L_∞ Fine
v_x	3.75×10^{-4}	3.72×10^{-4}	1.64×10^{-3}	9.69×10^{-4}
v_y	3.38×10^{-4}	3.04×10^{-4}	1.25×10^{-3}	9.75×10^{-4}
$v_{d,x}$	2.77×10^{-4}	7.59×10^{-5}	1.26×10^{-3}	4.08×10^{-4}
$v_{d,y}$	2.14×10^{-4}	6.57×10^{-5}	1.17×10^{-3}	3.09×10^{-4}
$v_{p,x}$	2.15×10^{-4}	3.67×10^{-4}	6.98×10^{-4}	9.06×10^{-4}
$v_{p,y}$	1.82×10^{-4}	2.76×10^{-4}	5.58×10^{-4}	9.32×10^{-4}
ρ	1.23×10^{-5}	3.23×10^{-6}	3.72×10^{-5}	1.46×10^{-5}
p	4.26×10^{-2}	3.73×10^{-2}	1.91×10^{-1}	1.20×10^{-1}
ρh	1.49×10^{-1}	1.31×10^{-1}	6.65×10^{-1}	4.21×10^{-1}

Table 6.1: *Low Mach error for the stratified vortex problem. ($M = 0.004$)*

Variable Name	Richardson Convergence	
Variable	L_1 Convergence	L_∞ Convergence
v_x	0.01	0.76
v_y	0.15	0.36
$v_{d,x}$	1.86	1.62
$v_{d,y}$	1.70	1.90
$v_{p,x}$	-0.77	-0.38
$v_{p,y}$	-0.60	-0.74
ρ	1.92	1.35
p	0.19	0.67
ρh	0.19	0.66

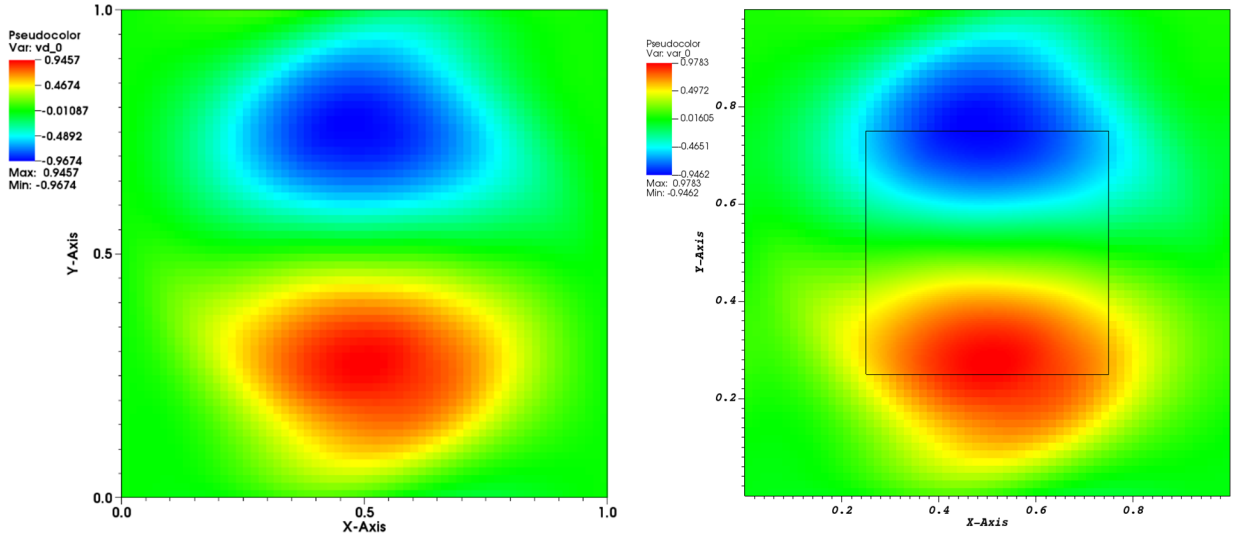
Table 6.2: *Low Mach Convergence for the stratified vortex problem. ($M = 0.004$)*



(a) *Single level data (ρ)*

(b) *AMR data (ρ)*

Figure 6.2: *Comparision of density fields between the single level algorithm in [12] and the AMR algorithm.*

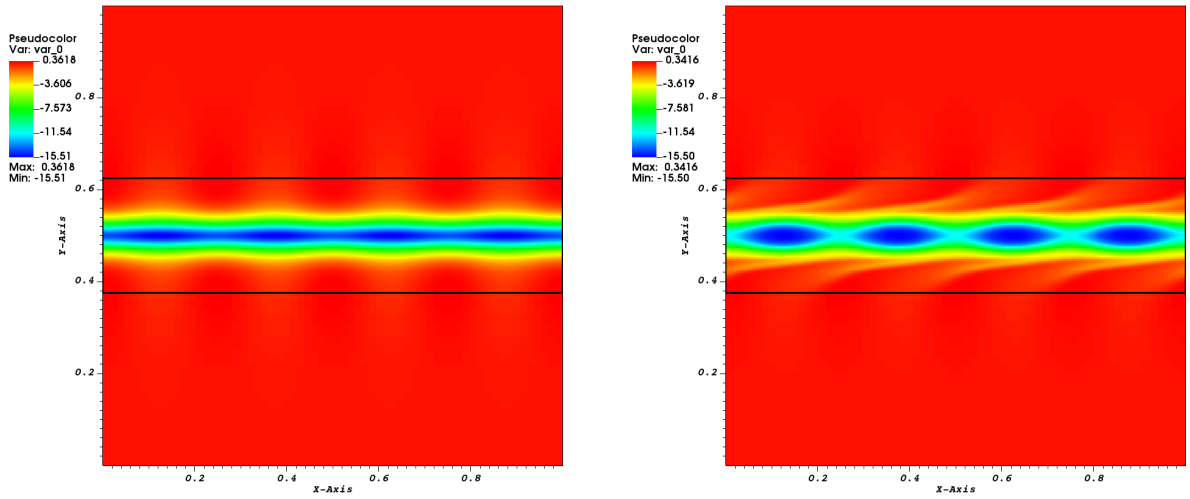


(a) *Single level data ($\mathbf{v}_d$)*

(b) *AMR data ($\mathbf{v}_d$).*

Figure 6.3: *Comparision of divergence-free velocity fields between the single level algorithm in [12] and the AMR algorithm.*

6.2 Shear Layer



(a) *Shear layer vorticity initial condition.* ***(b)*** *Shear layer vorticity at $t = 5$*

Figure 6.4: *Computation of vorticity roll-up in a lightly perturbed shear layer.*

We now turn our attention to the problem of a slightly perturbed shear layer. Unlike the stratified vortex, this problem actually leverages local refinement in a useful way by refining the region of concentrated vorticity. The initial condition is

$$v_x = u_0 \tanh\left(\frac{y - 0.5}{\delta_1}\right), \tag{6.8}$$

$$v_y = u_0 \delta_2 \sin(k\pi x) \exp\left[-\frac{(y - 0.5)^2}{\delta_1}\right], \tag{6.9}$$

$$\rho = \rho_0, \tag{6.10}$$

$$p = p_0, \tag{6.11}$$

$$\rho h = \frac{p_0 \gamma}{\gamma - 1}. \tag{6.12}$$

The domain is $x, y \in [0, 1]$ and the parameters are $\rho_0 = u_0 = 1$, $\delta_1 = 1.0/30$, and $\delta_2 = 0.05$. p_0 is again used to control Ma, and we examine the same values as we did for the stratified vortex problem. We also reuse the boundary conditions: periodic in the x-direction and solid walls in the y-direction.

As before, we examine the convergence of the second-order algorithm. Here we observe reasonable convergence results for the non-acoustic variables in the low-Mach case for both

Variable Name	Richardson Error			
Variable	L_1 Coarse	L_1 Fine	L_∞ Coarse	L_∞ Fine
v_x	1.14×10^{-4}	3.96×10^{-5}	7.19×10^{-4}	1.90×10^{-4}
v_y	1.44×10^{-4}	2.41×10^{-5}	8.79×10^{-4}	1.29×10^{-4}
$v_{d,x}$	1.17×10^{-4}	2.60×10^{-5}	7.34×10^{-4}	1.88×10^{-4}
$v_{d,y}$	1.44×10^{-4}	2.17×10^{-5}	8.72×10^{-4}	1.28×10^{-4}
$v_{p,x}$	6.01×10^{-6}	2.13×10^{-5}	1.69×10^{-5}	6.60×10^{-5}
$v_{p,y}$	1.07×10^{-6}	5.43×10^{-5}	7.43×10^{-6}	1.92×10^{-5}
ρ	4.36×10^{-6}	2.62×10^{-6}	2.08×10^{-5}	9.91×10^{-6}
p	1.04×10^{-4}	3.40×10^{-4}	3.73×10^{-4}	1.17×10^{-3}
ρh	2.13×10^{-1}	1.29×10^{-1}	1.02	4.90×10^{-1}

Table 6.3: *Low Mach error for shear layer:* $Ma = 0.004$

Variable Name	Richardson Convergence	
Variable	L_1 Convergence	L_∞ Convergence
v_x	1.52	1.92
v_y	2.57	2.77
$v_{d,x}$	2.18	1.97
$v_{d,y}$	2.73	2.77
$v_{p,x}$	-1.82	-1.97
$v_{p,y}$	-2.34	-1.37
ρ	0.73	1.07
p	-1.71	-1.64
ρh	0.72	1.05

Table 6.4: *Low Mach Convergence for shear layer:* $Ma = 0.004$

norms. Again, we do not observe convergence for the acoustic variables and we do not expect to for the same reasons as the previous problem. Here we do not observe convergence for the density, but that is because the initial density field is constant and any perturbations are the result of acoustic effects. By contrast, the stratified vortex initial condition has a non-trivial density distribution.

6.3 Sound Generation From a Vortex Pair

Now that we have established the baseline effectiveness of the algorithm, we can demonstrate its power by applying it to a problem that would not be reasonably possible to solve without

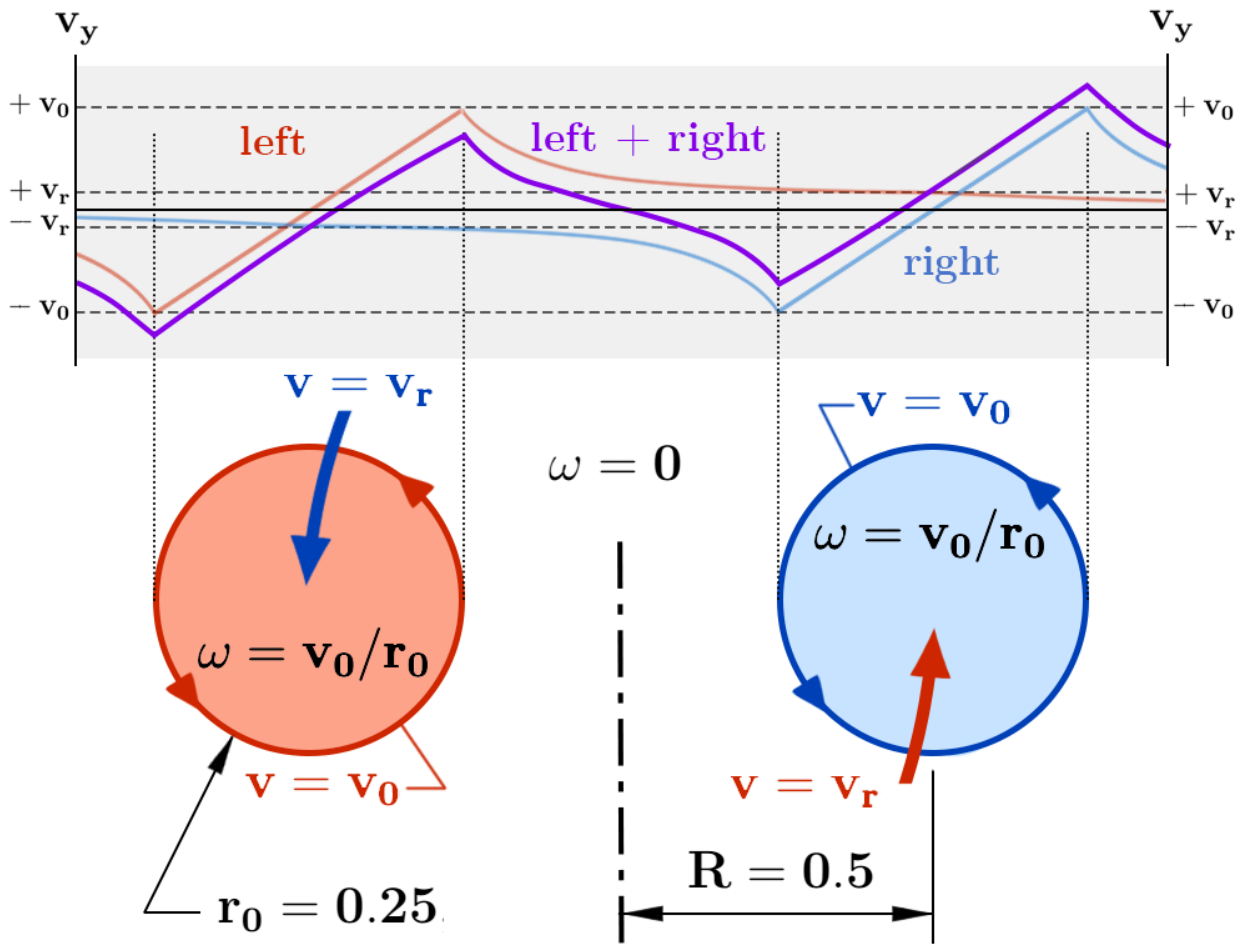


Figure 6.5: *A description of the dual vortex initial condition and relevant parameters. v_y is plotted as a function of x in the cross section of the two vortices.*

AMR. Namely, we examine the generation of acoustic waves from pair of corotating vortices. This problem has well documented behavior (see e.g. [32] [34]) which we seek to replicate.

We initialize each vortex as

$$v_x = -v_0 \frac{y}{r_0} \eta(r), \tag{6.13}$$

$$v_y = v_0 \frac{x}{r_0} \eta(r), \tag{6.14}$$

$$\eta(r) = \begin{cases} 1, & \text{if } r < r_0 \\ (\frac{r_0}{r})^2 & \text{otherwise,} \end{cases} \tag{6.15}$$

which is a constant, circular patch of vorticity with radius r_0. The quantity $r^2 = x^2 + y^2$ is the squared distance from the center of the vortex. As before, $v_0 = \rho_0 = 1.0$ and the enthalpy

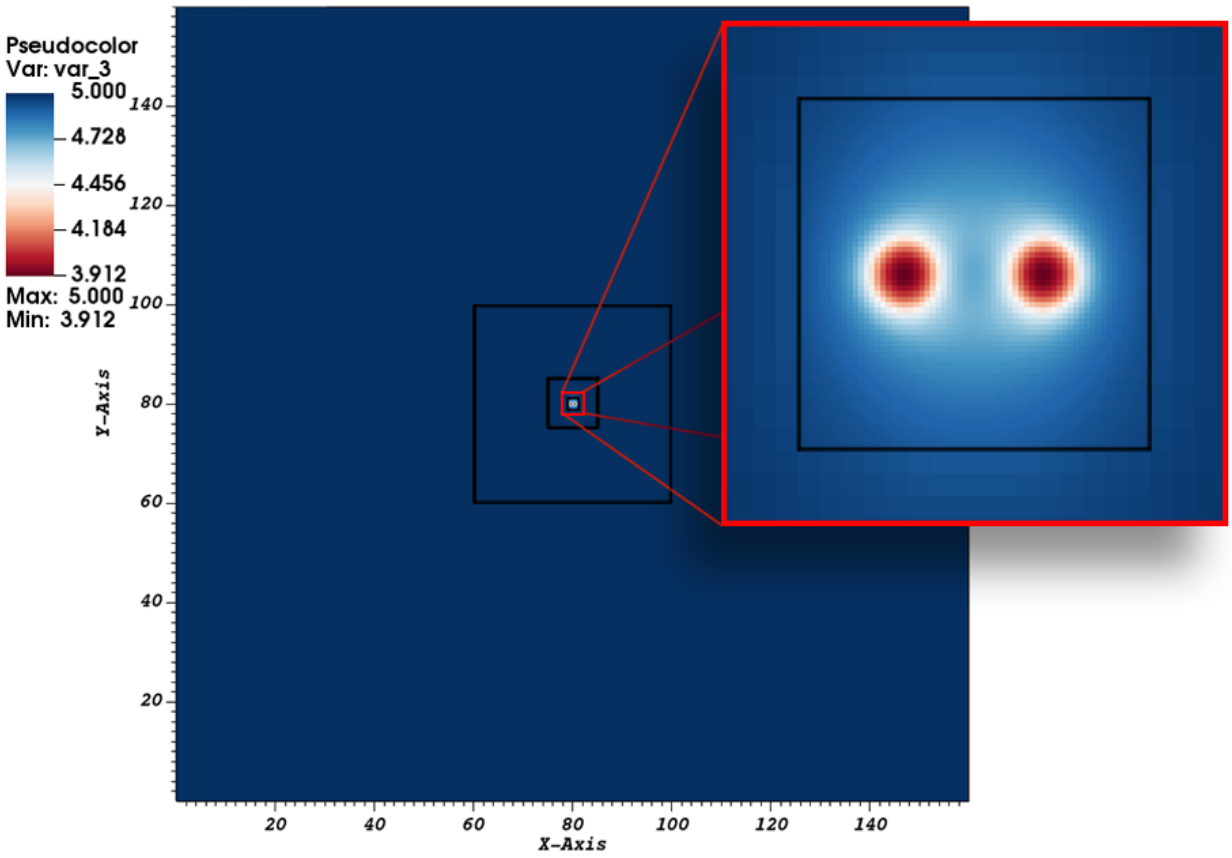


Figure 6.6: *Vortex pair pressure initial condition. Black outlines designate coarse-fine boundaries between the 4 levels of AMR refinement.*

is $\rho h = p_0\gamma/(\gamma - 1)$. The background pressure is $p_0 = 5$ corresponding to $M = 0.43$. We initialize the problem with two such vortices of radius $r_0 = 0.25$ with separation $2R = 1.0$. It has been shown in e.g. [32] that for $r_0/R < 0.59$, a pair of constant vorticity, co-rotating vortices will not merge but will rather orbit each other at a tangential velocity $v_r = v_0 r_0/2R$. The half period of the orbit is $\tau \equiv \pi R/v_r = 2\pi R^2/v_0 r_0$. The wavelength λ of emitted sound is $c\tau = \pi R/M_r$ where M_r is the Mach number based on the orbit speed $M_r \equiv v_r/c$. For our choice of parameters we have

$$v_r = 0.25 \quad \tau \approx 6 \quad M_r \approx 0.1 \quad \lambda \approx 16.$$

See Figure 6.5 for a graphical depiction of these parameters.

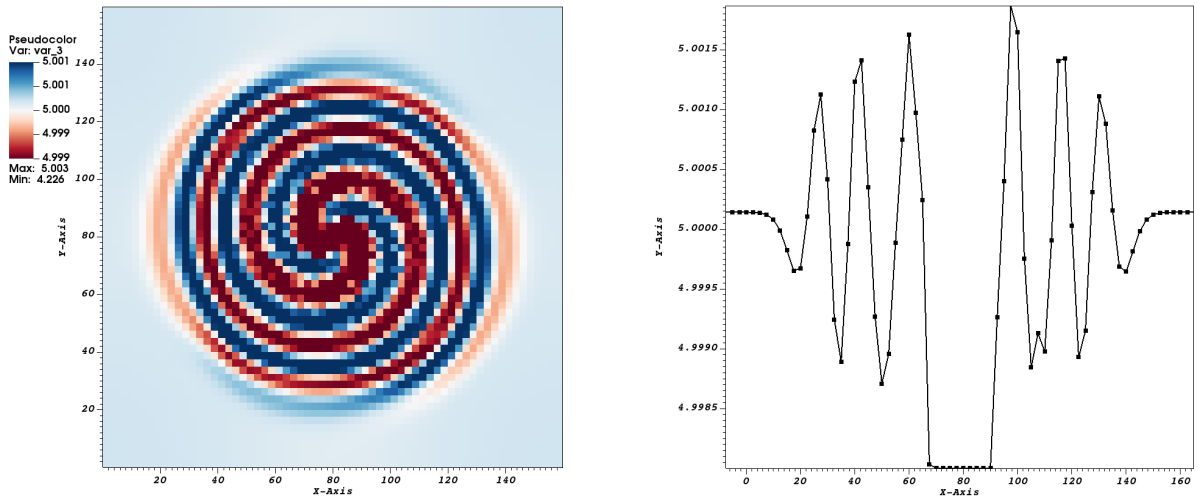


Figure 6.7: *Pressure distribution of the emitted sound wave at $t = 4\tau$. The left figure is the 2-dimensional distribution and the right figure is the cross section $y = 80$. Note that in the 1-dimensional cross section, the deep pressure well at the center has been thresholded for clarity.*

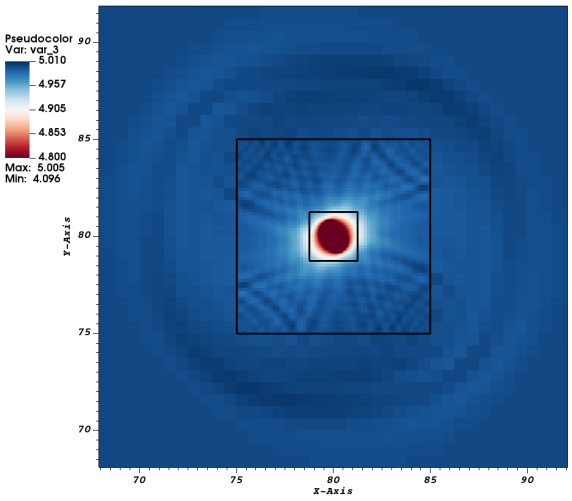


Figure 6.8: *Pressure distribution of the emitted sound wave at $t = 2\tau/3$. Waves can clearly be seen reflecting off the coarse-fine boundary.*

Sound Generation Results

The initial condition on pressure is shown in Figure 6.6. The domain $x, y \in [0, 160]$ has 4 levels of AMR refinement the finest of which covers only 1.56% of the total domain and contains the initial vortex pair. Figure 6.7 demonstrates how the resulting sound wave has propagated at $t = 4\tau$. The predicted pressure quadrupole is obviously present and the wavelength λ measured qualitatively as peak-to-peak distance is 15, a result in very good agreement with [34].

This problem does an excellent job of showcasing the power of AMR. to solve the equivalent problem on a single level would require 4096^D data points. The AMR implementation uses only $4 * 64^D$ data points, a factor of 16^D fewer.

Unfortunately we do observe some spurious behavior in the refined regions. Figure 6.8 serves to elucidate the situation. Clearly there is some amount of non-physical wave reflection occurring at the coarse-fine boundaries. This is similar to phenomena noted in [13] wherein electromagnetic waves are observed to reflect off of refinement boundaries in locally refined grids. In that study, a sponge layer was employed to fix the problem. We attempted a version of this solution but the results were not good, either resulting in unstable behavior or wave trapping near the coarse-fine boundaries.

We believe that the difficulties described above may be the result of either our staggered discretization or the fact that our current version of the algorithm is only second-order. Resolving these problems is a short term goal of future work on this algorithm.

Chapter 7

Conclusions and Future Work

In this study we have extended the all-speed projection algorithm discussed in [12] to the case of block-structured locally-refined grids. In the all-speed projection algorithm redundant equations are introduced to model the stiff acoustic dynamics. These redundant equations lead to an implicit solve for the stiff variables that takes the form of a well-behaved elliptic Helmholtz equation. The system is integrated semi-implicitly using a method of lines formulation and an implicit-explicit additive Runge-Kutta method. Treating the stiff variables implicitly eliminates the acoustic CFL as a constraint on the time step, allowing the advective time step to be used instead. The resulting method captures the $M \rightarrow 0$ limit and is valid for all Mach numbers in the absence of shocks. This method is based on a systematic finite-volume discretization that easily extends to high order.

Our contribution was to extend this algorithm to use AMR. This undertaking presented a number of challenges. Previous AMR multigrid methods took advantage of special discretizations at coarse-fine boundaries in order to use residual-correction multigrid solvers [33] [27]. It is unclear if such approaches are possible for the operators used in this algorithm, particularly at high order. Instead, we abandoned the residual-correction form and implemented our iterative solver using Full Approximation Scheme (FAS) multigrid. The resulting algorithm is considerably more flexible than the previous one. We have demonstrated that the AMR FAS implementation leads to a multigrid method that has the same convergence properties as the residual-correction form. The algorithm reduces the residual by about an order of magnitude with each v-cycle iteration and the number of iterations required to converge is independent of the grid spacing even for fourth-order discretizations of elliptic problems.

Our implementation also represents a new direction in the high-level software development of AMR frameworks. Previous AMR frameworks are based on two layers of abstraction. These frameworks contain one layer encoding distributed memory parallelism on a union of rectangular patches through a *C++* interface and another layer encoding patch-scope problem-specific physics through *Fortran* callbacks. This approach is unproductive because of the limitations of mixed-language programming and the limited expressiveness of *Fortran* in particular. This strategy also does not scale well with current trends in high performance

computing which promise a combinatorial increase in complexity when programming for performance at such a low level. In light of these observations, we have instead implemented problem specific physics using *Proto*. *Proto* has provided a substantial increase in productivity, and it is unlikely that a project of this scope would have been feasible by one person in the given time-frame without it.

We have demonstrated the efficacy of the AMR algorithm by applying it in two settings. We have shown that we recover the incompressible limit by simulating stratified vortical flow as well as a canonical shear layer. In both of these applications we observe stable behavior, and second-order accuracy in the divergence-free velocity and advected quantities. We have also successfully computed low-amplitude, long-wavelength acoustic waves generated by small scale vorticity. This problem would not have been economical in the previous, non-adaptive version of the all-speed algorithm, nor is it possible at all for a zero-Mach method which eliminates these effects.

There is still much work to be done with regards to the all-speed projection algorithm. In the short term are a number of relatively easy tasks: fourth-order accuracy, three-dimensional simulation, dynamic adaptivity, and complex physics. We are undeterred by both higher order and higher dimensionality because the framework is specifically designed to be extensible in these parameters. We also do not expect dynamic adaptivity to be a problem because the all-speed formulation lacks the difficulties associated with differential/algebraic systems that arise in zero-Mach formulations. Likewise, extending our algorithm to account for viscosity, reactions, and heat transfer is not expected to be to be prohibitively challenging due to the convenient abstractions furnished by *Proto*.

We are also interested in integrating more sophisticated AMR functionality into all-speed. Chiefly among these considerations is AMR refinement in time. This is another setting where zero-Mach methods have difficulty because of the need to provide boundary conditions in intermediate, fine time steps. Doing this is straightforward for Runge-Kutta methods using dense representation to compute coarse grid solutions at intermediate times. This was done in [31]; extending this idea to an ARK method is more complicated, but should be straightforward.

The matter of the spurious reflections at coarse-fine boundaries is also an open question that needs further study. This problem has been observed before in [13] in the context of a fourth-order electromagnetic simulation with state variables collocated at cell centers. In that study, the reflected waves were mitigated through the use of a sponge layer or high-order linear damping. It is possible that extending these ideas to our staggered grid formulation may resolve the issue.

Implementing the all-speed algorithm in tandem with AMR FAS-multigrid mostly from scratch represented a relatively large software undertaking. *Proto*'s expressiveness went a long way to making this project tractable. However, there is room for improvement in *Proto* as well. Attempting to interface *Proto*'s single-patch scope with the existing distributed parallelism abstractions available in *Chombo* has been anything but productive. The chief difficulty here is the aliasing of different data containers with slightly different semantics. This is especially true in the case of data discretized on finite-volume faces. In light of

these observations, *Proto*'s scope should be extended to encapsulate distributed memory parallelism on unions of rectangles and better support face-centered discretizations. Doing so will allow AMR applications to use a single type of data container at all scopes, dramatically simplifying the implementation of the AMR framework and streamlining the API.

Physical systems which include fast (stiff) and slow (non-stiff) variables for which the fast dynamics can be expressed as a redundant system of equations are not unique to low-Mach fluid dynamics. This configuration is also observed in the case of fast magnetosonic waves in magnetohydrodynamics, gravity waves in atmospheres and oceans [18], and stiff dielectric relaxation effects in charged fluid models of plasma [15]. In all of these cases, stiff scales impose a large cost by restricting the stable time step, and methods for working around this impediment have been largely *ad-hoc*, especially for AMR. The all-speed projection formulation provides a systematic way of designing methods to efficiently represent stiff dynamics in these settings.

Bibliography

[1] Mark Adams et al. *Chombo Software Package for AMR Applications*. Tech. rep. 2015. URL: https://escholarship.org/content/qt5cs5d1sq/qt5cs5d1sq.pdf.

[2] Ann S. Almgren, John B. Bell, and William G. Szymczak. "A numerical method for the incompressible Navier-Stokes equations based on an approximate projection". In: *SIAM Journal of Scientific Computing* 17.2 (1996), pp. 358–369. ISSN: 10648275. DOI: 10.1137/S1064827593244213. URL: http://www.siam.org/journals/sinum/41-3/39628.html.

[3] Ann S. Almgren, Thomas Buttke, and Phillip Colella. "A Fast Adaptive Vortex Method in Three Dimensions". In: *Journal of Computational Physics* 113 (1994), pp. 177–200. DOI: 10.1006/jcph.1994.1129. URL: http://crd.lbl.gov/assets/pubs{_}presos/AMCS/ANAG/A124.pdf.

[4] Ann S. Almgren et al. "A Conservative Adaptive Projection Method for the Variable Density Incompressible Navier-Stokes Equations". In: *Journal of Computational Physics* 142.1 (1998), pp. 1–46. ISSN: 00219991. DOI: 10.1006/jcph.1998.5890.

[5] Ann S. Almgren et al. "Low Mach Number Modeling of Type Ia Supernovae. I. Hydrodynamics". In: *The Astrophysical Journal* 637.2 (2006), pp. 922–936. ISSN: 0004-637X. DOI: 10.1086/498426. URL: https://iopscience.iop.org/article/10.1086/498426.

[6] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Philadelphia: Society for Industrial and Applied Mathematics Publications, 1997. ISBN: 9781611971392. URL: https://books.google.com/books?id=2iXovtfcL74C.

[7] Michael Barad and Phillip Colella. "A fourth-order accurate local refinement method for Poisson's Equation". In: (2005). DOI: 10.1016/j.jcp.2005.02.027. URL: www.elsevier.com/locate/jcp.

[8] John B. Bell, Phillip Colella, and Harland M Glaz. "A second-order projection method for the incompressible navier-stokes equations". In: *Journal of Computational Physics* 85.2 (1989), pp. 257–283. ISSN: 00219991. DOI: 10.1016/0021-9991(89)90151-4. URL: http://www.sciencedirect.com/science/article/pii/0021999189901514.

[9] Marsha J Berger and Phillip Colella. *Local Adaptive Mesh Refinement for Shock Hydrodynamics*. Tech. rep. 1989, pp. 64–84. DOI: 10.1016/0021-9991(89)90035-1.

[10] Kathryn E. Brenan, Stephen L. Campbell, and Linda R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Philadelphia: Society for Industrial and Applied Mathematics Publications, 1996. DOI: 10.1137/1.9781611971224.

[11] William Briggs, Henson Van Emden, and Steve McCormick. *A Multigrid Tutorial, 2nd Edition*. Society for Industrial and Applied Mathematics Publications, 2000. ISBN: 978-0-89871-462-3. URL: https://www.doi.org/10.1137/1.9780898719505.

[12] Christopher M Chaplin. "An improved all-speed projection algorithm for low Mach number flows". PhD thesis. University of California: Berkeley, 2018. URL: https://escholarship.org/content/qt0bf8314r/qt0bf8314r{_}noSplash{_}336930618952346fc7846pdf?t=phm4bu.

[13] Sven Chilton. "A Fourth-Order Adaptive Mesh Refinement Solver for Maxwell's Equations". PhD thesis. University of California, Berkeley, 2013. ISBN: 9788578110796. arXiv: arXiv:1011.1669v3. URL: https://escholarship.org/content/qt1vp1238g/qt1vp1238g.pdf.

[14] Alexandre Joel Chorin. "Numerical Solution of the Navier-Stokes Equations". PhD thesis. New York University, 1968. DOI: 10.1090/S0025-5718-1968-0242392-2.

[15] Phillip Colella, Milo R. Dorr, and Daniel D Wake. *Numerical Solution of Plasma Fluid Equations Using Locally Refined Grids*. Tech. rep. 1999, pp. 550–583. DOI: 10.1006/jcph.1999.6245. URL: http://www.idealibrary.comon.

[16] Phillip Colella and Karen Pao. "A Projection Method for Low Speed Flows". In: *Journal of Computational Physics* 149 (1999), pp. 245–269. DOI: 10.1006/jcph.1998.6152.

[17] Marc S. Day and John B. Bell. "Numerical simulation of laminar reacting flows with complex chemistry". In: *Combustion Theory and Modelling* 4.4 (2000), pp. 535–556. ISSN: 13647830. DOI: 10.1088/1364-7830/4/4/309. URL: https://www.tandfonline.com/action/journalInformation?journalCode=tctm20.

[18] Caroline Gatti-Bono and Phillip Colella. "An anelastic allspeed projection method for gravitationally stratified flows". In: *Journal of Computational Physics* 216.2 (2006), pp. 589–615. ISSN: 00219991. DOI: 10.1016/j.jcp.2005.12.017. URL: http://www.sciencedirect.com/science/article/pii/S0021999106000027.

[19] David J. Griffiths. *Introduction to Electrodynamics*. 4th ed. Illinois: Pearson Education Inc., 2013. ISBN: 9781108357142. DOI: 10.1119/1.4766311.

[20] Christopher A Kennedy and Mark H Carpenter. *Additive Runge-Kutta Schemes for Convection-Diffusion-Reaction Equations*. Tech. rep. 2001. DOI: 10.1016/S0168-9274(02)00138-1. URL: http://www.sti.nasa.gov.

[21] Marcin Krotkiewski and Marcin Dabrowski. "Efficient 3D stencil computations using CUDA". In: *Parallel Computing* 39.10 (2013), pp. 533–548. ISSN: 01678191. DOI: 10.1016/j.parco.2013.08.002.

[22] Mindy Lai, John B. Bell, and Phillip Colella. "A projection method for combustion in the zero mach number limit". In: *11th Computational Fluid Dynamics Conference, 1993*. American Institute of Aeronautics and Astronautics Inc, AIAA, 1993, pp. 776–783. DOI: 10.2514/6.1993-3369. URL: http://arc.aiaa.org.

[23] Mindy F. Lai. "A Projection Method for Reacting Flow in the Zero Mach Number Limit". PhD thesis. University of California, Berkeley, 1993. URL: https://www.proquest.com/openview/fad5819ac2975cab7b45f092fd5a81cd/1?pq-origsite=gscholar{\&}cbl=18750{\&}diss=y.

[24] Hans Petter Langtangen and Xing Cai. "On the Efficiency of Python for High-Performance Computing: A Case Study Involving Stencil Updates for Partial Differential Equations". In: *Modeling, Simulation and Optimization of Complex Processes*. Springer Berlin Heidelberg, 2008, pp. 337–357. DOI: 10.1007/978-3-540-79409-7_23. URL: https://link.springer.com/chapter/10.1007/978-3-540-79409-7{_}23.

[25] Bryan N. Lawrence et al. "Crossing the Chasm: How to develop weather and climate models for next generation computers?" In: *Geoscientific Model Development Discussions* September (2017), pp. 1–36. ISSN: 1991-962X. DOI: 10.5194/gmd-2017-186. URL: https://www.geosci-model-dev-discuss.net/gmd-2017-186/gmd-2017-186.pdf{\%}0Ahttps://www.geosci-model-dev-discuss.net/gmd-2017-186/.

[26] Andrew Majda and James Sethian. "The derivation and numerical solution of the equations for zero mach number combustion". In: *Combustion Science and Technology* 42.3-4 (1985), pp. 185–205. ISSN: 1563521X. DOI: 10.1080/00102208508960376. URL: https://www.tandfonline.com/action/journalInformation?journalCode=gcst20.

[27] Daniel F. Martin and Keith L. Cartwright. *Solving Poisson's Equations Using Adaptive Mesh Refinement*. Tech. rep. 1996. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.3056{\&}rep=rep1{\&}type=pdf.

[28] Daniel F. Martin and Phillip Colella. "A Cell-Centered Adaptive Projection Method for the Incompressible Euler Equations". In: *Journal of Computational Physics* 163.2 (2000), pp. 271–312. ISSN: 00219991. DOI: 10.1006/jcph.2000.6575. URL: http://www.sciencedirect.com/science/article/pii/S0021999100965756.

[29] Daniel F. Martin, Phillip Colella, and Daniel T. Graves. "A cell-centered adaptive projection method for the incompressible NavierStokes equations in three dimensions". In: *Journal of Computational Physics* 227.3 (2008), pp. 1863–1886. ISSN: 00219991. DOI: 10.1016/j.jcp.2007.09.032. URL: http://www.sciencedirect.com/science/article/pii/S0021999107004366.

[30] Morgan McCorkle. *ORNL Launches Summit Supercomputer*. 2018. URL: https://www.ornl.gov/news/ornl-launches-summit-supercomputer.

[31] Peter W. McCorquodale and Phillip Colella. "A High-Order Finite-Volume Method For Conservation Laws on Locally Refined Grids". In: *Communications in Applied Mathematics and Computational Science* 6.1 (2011). DOI: 10.2140/camcos.2011.6.1.

[32] Mogens V. Melander, Norman J. Zabusky, and James C. McWilliams. "Symmetric vortex merger in two dimensions: Causes and conditions". In: *Journal of Fluid Mechanics* 195 (1988), pp. 303–340. ISSN: 14697645. DOI: 10.1017/S0022112088002435. URL: https://doi.org/10.1017/S0022112088002435.

[33] Michael L Minion. *On the Stability of Godunov-Projection Methods for Incompressible Flow*. Tech. rep. 1996, pp. 435–449. DOI: 10.1006/jcph.1996.0035.

[34] Brian E. Mitchell, Sanjiva K. Lele, and Parviz Moin. "Direct computation of the sound from a compressible co-rotating vortex pair". In: *Journal of Fluid Mechanics* 285 (1995), pp. 181–202. ISSN: 14697645. DOI: 10.1017/S0022112095000504.

[35] Yoshio Oyanagi. "Future of supercomputing". In: *Journal of Computational and Applied Mathematics* 149.1 (2002), pp. 147–153. ISSN: 03770427. DOI: 10.1016/S0377-0427(02)00526-5.

[36] Richard B. Pember et al. "An Adaptive Projection Method for Unsteady, Low-Mach Number Combustion". In: *Combustion Science and Technology* 140.1-6 (1998), pp. 123–168. ISSN: 00102202. DOI: 10.1080/00102209808915770. URL: https://www.tandfonline.com/action/journalInformation?journalCode=gcst20.

[37] Linda Petzold. "Differential/Algebraic Equations are not ODEs". In: *SIAM Journal on Scientific and Statistical Computing* 3.3 (1982), pp. 367–384. ISSN: 0196-5204. DOI: 10.1137/0903023.

[38] William H. Press et al. *Numerical Recipes in C*. 2nd ed. Cambridge University Press, 1988. ISBN: 0521431085. URL: http://www.nr.com.

[39] Jonathan Ragan-Kelley et al. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. Tech. rep. 2013. DOI: 10.1145/2499370.2462176.

[40] Ronald Rehm and Howard Baum. "The equations of motion for thermally driven, buoyant flows". In: *Journal of Research of the National Bureau of Standards* 83.3 (1978), pp. 297–308. URL: https://nvlpubs.nist.gov/nistpubs/jres/83/jresv83n3p297{_}A1b.pdf.

[41] Erich Strohmaier et al. *Top 500 List*. 2021. URL: https://www.top500.org/lists/top500/.

[42] Mark C. Thompson and Joel H. Ferziger. *An Adaptive Multigrid Technique for the Incompressible Navier-Stokes Equations*. Tech. rep. 1989, p. 94121. DOI: 10.1016/0021-9991(89)90037-5.

Appendix A

Definitions

Variable Name	Variable Definition
D	Number of dimensions
d	Coordinate direction (e.g. x, y, z)
n	Normal direction
h	Uniform grid spacing
ℓ	AMR or Multigrid level
Ω	Region of space in Z^D
Γ	Grid associated with an AMR or Multigrid level
$\boldsymbol{i}$	A point in $\mathbb{Z}^D$
$\mathbf{e}_d$	Unit vector in the coordinate direction d
$\mathbf{e}$	The vector [1,1,1,...,1]
χ^k	Partial sum of stage-k Runge-Kutta method
a, b, c	Runge-Kutta butcher tableau coefficients

Table A.1: *Symbols related to discretization and local refinement.*

Variable Name	Variable Definition
$\mathcal{I}_{face}$	Cell average to cell face interpolation
$\mathcal{I}_{cell}$	Cell face to cell average interpolation
$\mathcal{I}_u$	Cell average to cell face interpolation with upwinding
$\mathcal{I}_r^0$	Piecewise constant interpolation between levels
$\mathcal{I}_r^{ho}$	High order boundary value interpolation between levels
$\mathbb{C}$	Cell centered convolution operator
$\mathbb{C}_{\mathbf{f}}$	Face centered convolution operator
$\mathcal{A}_r$	Average down between levels refined by r
$\mathbb{D}_d$	Cell-to-face derivative operator in direction d
$\mathcal{G}$	Gradient operator
$\mathcal{D}$	Divergence operator
$\mathcal{L}$	A general level operator used in AMR or multigrid
$\mathcal{J}$	A level agnostic operator used in the definition of $\mathcal{L}$
$\mathcal{F}$	The flux component of $\mathcal{J}$
$\mathcal{S}$	The source component of $\mathcal{J}$
$\mathcal{I}$	The diagonal component of $\mathcal{J}$
β	Scaling coefficients of $\mathcal{J}$
$\mathbb{Q} \equiv \nabla(\Delta^{-1})\nabla\cdot$	Potential projection
$\mathbb{P} \equiv \mathbb{I} - \mathbb{Q}$	Solenoidal / divergence-free projection

Table A.2: *Operators*

Variable Name	Variable Definition
$\mathbf{x}$	Position vector in $\mathbb{R}^{\mathrm{DIM}}$
ρ	Density
$\mathbf{v}$	Total velocity
$\mathbf{v}_p \equiv \mathbb{Q}(\mathbf{v})$	Potential / acoustic velocity
$\mathbf{v}_d \equiv \mathbb{P}(\mathbf{v})$	Solenoidal / advective / divergence-free velocity
p	Total pressure
h	Specific enthalpy
$\gamma \equiv C_p/C_v$	Heat capacity ratio
$\mathbf{U}$	Vector of evolved state variables
$\mathbf{G}$	Right-hand side forcing
$\mathbf{R}$	Residual vector of $\mathbf{U}$
$\mathbf{E}$	Error vector in $\mathbf{U}$
$\phi, \theta, \mathbf{u}$	General scalar or vector quantities

Table A.3: *Physical quantities*