Hydrodynamic Instabilities in Inertial Confinement Fusion: Physics, Numerical Methods, and Implementation

by

Samuel Carl Miller

Submitted in Partial Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

Supervised by Professor Valeri Goncharov

Department of Mechanical Engineering

Arts, Sciences and Engineering

School of Arts and Sciences

University of Rochester

Rochester, New York

2022

Table of Contents

Biographical Sketch

The author was born in Springville, NY in 1987. He received a Bachelor of Science in Aerospace Engineering from Embry-Riddle Aeronautical University in Daytona Beach, Florida (2010) with a focus on gas-turbine engine design. Upon completion of his undergraduate studies, he served as a commissioned officer and developmental engineer in the United States Air Force from 2010 - 2015, working for both the National Air and Space Intelligence Center and the Air Force Research Laboratory at Wright-Patterson Air Force Base in Dayton, Ohio. During his time in Dayton, the author completed a graduate program at the University of Dayton, receiving a Master of Science in Aerospace Engineering, advised by Professor Markus Rumpfkeil, in May 2015. His graduate thesis focused on fluid-structure interaction and computational fluid dynamics simulations. He separated from the Air Force as a Captain and entered the Department of Mechanical Engineering graduate program at the University of Rochester in 2015. The author began studies in hydrodynamic instabilities and inertial confinement fusion at the Laboratory for Laser Energetics as a Horton Fellow in 2016 under the guidance of Professor Valeri Goncharov and Dr. Radha Bahukutumbi.

First Author Publications

1. **Miller, S. C.**, and Goncharov, V. N., "Instability Seeding Mechanisms due to Internal Defects in Inertial Confinement Fusion Targets", Submitted to Physics of Plasmas March 2022.
2. **Miller, S. C.**, Knauer, J. P., Forrest, C. J., Glebov, V. Yu., Radha, P. B. and Goncharov, V. N., "Fuel-Shell Interface Instability Growth Effects on the Performance of Room Temperature Direct-Drive Implosions", Phys. Plasmas 26, 082701 (2019).

Co-Author Publications

1. V. Gopalaswamy, R. Betti, P. B. Radha, A. J. Crilly, K. M. Woo, A. Lees, C. Thomas, I. V. Igumenshchev, **S. C. Miller**, J. P. Knauer, C. Stoeckl, C. J. Forrest, O. M. Mannion, Z. L. Mohamed, H. G. Rinderknecht, and P. V. B. Heuer, "Analysis of Limited Coverage Effects on Areal Density Measurements in Inertial Confinement Fusion Implosions", Submitted to Physics of Plasmas January 2022.

2. Christopherson, A. R., Betti, R. , **Miller, S.** , Gopalaswamy, V. , Mannion, O. M. and Cao, D., "Theory of Ignition and Burn Propagation in Inertial Fusion Implosions", Physics of Plasmas 27, 052708 (2020).

3. V Gopalaswamy, R Betti, JP Knauer, N Luciani, D Patel, KM Woo, A Bose, IV Igumenshchev, EM Campbell, KS Anderson, KA Bauer, MJ Bonino, D Cao, AR Christopherson, GW Collins, TJB Collins, JR Davies, JA Delettrez, DH Edgell, R Epstein, CJ Forrest, DH Froula, V Yu Glebov, VN Goncharov, DR Harding, SX Hu, DW Jacobs-Perkins, RT Janezic, JH Kelly, OM Mannion, A Maximov, FJ Marshall, DT Michel, **S Miller**, SFB Morse, J Palastro, J Peebles, PB Radha, SP Regan, S Sampat, TC Sangster, AB Sefkow, W Seka, RC Shah, WT Shmyada, A Shvydky, C Stoeckl, AA Solodov, W Theobald, JD Zuegel, M Gatu Johnson, RD Petrasso, CK Li, JA Frenje "Tripled yield in direct-drive laser fusion through statistical modelling", Nature 565, 581-586 (2019).

First-Author Presentations

1. **Miller, S**, V. N. Goncharov, "Internal Perturbation Evolution and Amplification During the Early Phase of Inertial Confinement Fusion Implosions" (contributed), presented at the 63rd Annual Meeting of the American Physical Society Division of Plasma Physics, Virtual, November 8–12, 2021

2. **Miller, S**, V. N. Goncharov, "A Study of 2D Internal Perturbation Evolution in Inertial Confinement Fusion Implosions" (contributed), presented at the 62nd Annual Meeting of the American Physical Society Division of Plasma Physics, Virtual, November 9–13, 2020

3. **Miller, S**, V. N. Goncharov, and P. B. Radha, "A Study of Internal Perturbation Evolution in Inertial Confinement Fusion Implosions" (contributed), presented at the 61st Annual Meeting of the American Physical Society Division of Plasma Physics, Fort Lauderdale, FL, October 21–25, 2019

4. **Miller, S**, P.B. Radha, and V. N. Goncharov, "Deceleration-Phase Rayleigh-Taylor Growth Effects on Inferred Ion Temperatures in Room-Temperature, Direct-Drive Implosions" (contributed), presented at the 60th Annual Meeting of the American Physical Society Division of Plasma Physics, Portland, OR, November 5–9, 2018

5. **Miller, S**, Knauer, JP and Radha, PB and Goncharov, VN, "Finite Atwood Number Effects on Deceleration-Phase Instability in Room-Temperature Direct-Drive Implosions" (contributed), presented at the 59th Annual Meeting of the American Physical Society Division of Plasma Physics, Milwaukee, WI, 23-27 October 2017

Acknowledgments

First and foremost, this thesis would not have been possible without the support and encouragement that my wife, Bethany, has given me over these many years. Bethany, you made all of this possible and stuck with it in the midst of one of the most challenging seasons of our lives. Thank you for your patience and willingness to listen to me drone on about physics and coding late into the night, and for your enthusiasm as my talented editor-in-chief. To my children, Oliver, Clara, and Lawrence, thank you for grounding me, reminding me that there is life outside of school and that life is truly precious. And to my parents, siblings, and extended family, thank you for all of your support that went into making this thesis a reality.

I would like to thank my thesis advisors Drs. Valeri Goncharov and Radha Bahukutumbi for your patience and for helping me build my intuition about implosion physics. It's encouraging to see how much I've learned over these last few years and you have played a crucial role. Drs. Tim Collins and John Marozas, thank you for allowing me to plant myself in your office on countless occasions to vent and talk about coding. Tim, thank you for your continual support and mentorship both in physics and in my personal faith. Thank you to Drs. Ken Anderson, Alex Shvydky, and other members of the Integrated Modeling Group for helping me prepare this thesis and for providing insight into the physics of this work.

Thank you to Les Dodson, Michael Charissis, and Dr. Jonathan Carroll-Nellenback, for humoring and supporting me as I tried to use the latest-and-greatest in software and hardware. Fellow graduate students, Drs. Owen Mannion and Varchas Gopalaswamy, thank you for joining me in my journey to develop and use modern software tools to make life easier.

Abstract

Performance degradation in laser direct-drive (LDD) inertial confinement fusion (ICF) implosions is caused by several effects, including Rayleigh–Taylor (RT) and Richtmyer–Meshkov (RM) hydrodynamic instability growth. RT instability growth occurs in both the acceleration and deceleration phases of implosions. The first half of this thesis examines the evolution of internal perturbations that create seeds for instability growth during shock-transit (or early-time), while the second half describes the perturbation evolution during shell deceleration.

During shock-transit, perturbations from shell material density modulations and isolated defects plant seeds at various interfaces such as the ablation front and material interfaces. These seeds can become amplified due to secular feedout growth and shock-induced vorticity and will grow exponentially during the acceleration phase due to ablative RT. A comprehensive understanding of this evolution is essential to characterize the impact of internal defects on inflight shell integrity. Through detailed simulations and analysis, this thesis identifies several key physical processes that play a role in the evolution of perturbations created by these defects throughout the early stage of implosions. Simulations also predict that significant shell mass modulations develop during shell acceleration. The use of low density ablator materials (foam) is suggested as a potential mitigation strategy to reduce the effects created by these defects.

To perform this detailed study of internal defect evolution, two new high-fidelity physics codes were developed to track characteristic wave propagation in the ICF context using low-noise, low-dissipation, high-order spatial accuracy solution methods. Modern high performance computing (HPC) systems have becoming increasingly complex, and adapting existing or new software to

fully utilize them is a significant development challenge. Each code in this thesis examines the feasibility of different approaches: a modern design in a well-known HPC-centric language (Fortran), and a new language (Julia), which emphasizes developer productivity and shows the potential to be well-suited for HPC workloads.

Mass modulations at the ablation front, which grow during the acceleration phase, feed through to the inner surface of the shell and create seeds for deceleration phase RT growth at the inner surface. Deceleration instability growth was studied using laser direct drive implosions of room-temperature plastic targets. Perturbation growth in such implosions is enhanced by the density discontinuity and finite Atwood number at the fuel-shell interface. The magnitude of this density discontinuity can be controlled by changing the fuel composition (D:T, or ratio of deuterium to tritium). However, this thesis demonstrates that the stability of the interface is best characterized by the effective Atwood number, which is primarily determined by material densities at distances on the order of perturbation wavelength on either side of the interface, rather than the density ratio at the interface. Since the densities at these distances are defined not by fuel composition, but radiation heating of the shell, both simulation and experimental data show that target performance is insensitive to different D:T ratios.

Contributors and Funding Sources

This work was supervised by a dissertation committee consisting of Professor Valeri Goncharov (advisor) of the Department of Mechanical Engineering, Professor Adam Sefkow of the Department of Mechanical Engineering, Professor Dustin Froula of the Department of Physics and Astronomy, and Dr. Radha Bahukutumbi of the Laboratory for Laser Energetics at the University of Rochester. The defense committee was chaired by Petros Tzeferacos of the Department of Physics and Astronomy. Graduate study was supported by the Department of Mechanical Engineering at the University of Rochester dissertation research by the Horton Fellowship.

This material is based upon work supported by the Department of Energy National Nuclear Security Administration under Award Number DE-NA0001944 and DE-NA0003856, the University of Rochester, and the New York State Energy Research and Development Authority.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Inertial Confinement Fusion

Inertial confinement fusion [1] (ICF) seeks to initiate fusion by using multiple high-power, nanosecond laser beams with an overlap intensity of ~10^{15} W/cm^2 to compress and implode a millimeter-scale spherical target filled with deuterium-tritium (DT) fuel. Implosions are designed to increase the pressure and density of the fuel high enough such that DT fusion occurs and releases excess energy. Typical ICF target designs use an outer shell consisting of one or more high-density ablator layers (such as CH plastic or high-Z materials), a layer of cryogenically-frozen DT ice, and a central region of DT gas. There are two primary approaches to ICF: laser direct-drive (LDD), and laser indirect-drive (LID). In LDD, the laser beams directly illuminate the outer surface of the target and ablate the outer material away. This ablation process creates a rocket-like drive that pushes the target inward. In LID, the laser beams illuminate a hohlraum, or cylindrical-like can, that contains the target. As the laser irradiation ablates the hohlraum material (rather than the target material), the resulting x-ray radiation creates the drive that implodes the target. This thesis focuses on LDD implosions; however, many of the physics mechanisms apply to both ignition approaches. References 2, 3, 4, and 5 are excellent reviews of both LDD and LID research efforts.

There are four distinct phases in an ICF implosion: 1) shock propagation (early-time), 2) acceleration phase, 3) deceleration phase, and 4) hot-spot formation and peak compression. Early-time refers to the beginning of the implosion, when the laser ablates material away from the outer surface of the target, creating a rocket-like drive that launches a shock wave (or multiple shock waves) into the shell. These initial shocks must be properly timed to merge near the inner surface soon after they break out of the shell into the vapor region. After shock break-out, the shell accelerates inward (initiating the acceleration phase) while the shock converges and rebounds from the target center. Shell convergence causes the pressure and density of the central fuel region to increase, and, once the vapor pressure exceeds the shell pressure, the deceleration phase begins by launching an outgoing shock wave into the converging fuel. As the shell continues to decelerate, the decelerating shock passes through the shell and converts its kinetic energy into internal energy (in both the shell and vapor). This generates a hot spot in the core where the pressure, density, and temperature increase until peak compression. As the hot-spot pressure and temperature increase, fusion reactions start to occur in the hot spot. Maximum neutron yield, also referred to as "bang-time," occurs near shell stagnation time (when most of the shell mass ceases to move inward). In an igniting target, a hot spot is formed if hot-spot energy power gain (due to alpha particle heating and PdV work) exceeds power loss (due to radiation and thermal conduction), launching a burn wave into the main fuel.

1.1.1 Implosion Performance

Implosion performance is quantified through neutron yield, a function of hot-spot ion temperature (T_i) and pressure (p), and areal density (ρR) of the shell. Areal density is a measure of the amount of confinement that the shell achieves, which is necessary for hot-spot formation and efficient fuel burn. If the main fuel areal density, ρR, (a product of shell density to main fuel thickness at the onset of ignition) and T_i are high enough, DT fusion will create alpha particles which self-heat the hot spot, causing ignition, and launch a propagating burn wave through the fuel that amplifies yield [6]. Loss of compression or confinement (ρR) due to nonuniformities significantly degrades implosion performance. Examples of nonuniformities can include non-

spherical compression, shell punctures, mixing between the cold shell material and the hot spot, and non-radial bulk flows in the main fuel and hot spot [7, 8], all of which reduce performance.

Short-scale ($k\Delta > 1$, where k is the perturbation wavenumber and Δ is the inflight shell thickness) implosion stability can be improved by reducing the in-flight-aspect ratio (IFAR) of the shell (IFAR = shell radius / shell thickness) and increasing shell ablation velocity (ratio of shell mass ablation rate to the shell density). Both parameters depend on shell adiabat (α), which is defined as the ratio of shell pressure to Fermi-degenerate pressure at shell density (for fully-ionized DT plasma, $\alpha_{DT} = p/2.2\rho^{5/3}$, with p in Mbar, ρ in g/cm^3, see Ref. 9). Shell adiabat in LDD is primarily controlled by the timing of the first set of shocks that propagate through the shell. These shocks are typically launched by "pickets" in the laser pulse, which are short (~100 ps), Gaussian-like intensity pulses preceding the main laser pulse. Using pickets increases the shell stability by raising that adiabat at the ablation front, while keeping the inner part of the fuel at lower adiabat (adiabat shaping). A higher adiabat at the ablation front enhances the ablation velocity which reduces laser imprint effects and stabilizes RT growth [10, 11, 12]. Keeping the fuel entropy as low as possible ensures maximum compressibility and target performance. Implosion designs are typically classified as low-adiabat ($\alpha < 3$) or high-adiabat ($\alpha \geq 3$).

A canonical ignition design uses a laser pulse that consists of three main features: 1) one or more pickets that launch shocks into the shell which decay over time (for adiabat shaping and stability), 2) a low-intensity foot that sends a supported (or constant pressure) shock behind the picket shocks, and 3) a ramp to full power that adiabatically compresses the fuel. Shocks are timed such that they coalesce near the ice-vapor interface as the compression wave from the rise starts to move through the shell.

Low-adiabat implosions are designed to limit the amount of entropy buildup (due to the first shocks launched into the target) in the fuel before adiabatic compression by the rise. This is ideal considering 1-D fuel compressibility, but leaves such designs susceptible to hydrodynamic instabilities seeded by non-uniformities in the target or laser. High-adiabat implosions benefit from increased ablative stabilization. However, these result in reduced fuel compressibility. ICF

implosion design seeks to optimize the balance between 1-D performance and stability.

1.1.2 Sources of Non-Uniformity and Hydrodynamic Instability Growth

Non-uniformities that degrade implosion performance can originate from laser deposition patterns (determined by beam port locations) and beam speckles (short-scale nonuniformities), target positioning and manufacturing imperfections, or the various effects of laser-plasma instabilities (LPIs) in the coronal plasma that form around the target during the implosion. Target perturbations that grow as a result of these non-uniformities are typically decomposed into spherical harmonics (referred to as ℓ-modes), due to spherical implosion geometry.

Each non-uniformity source generates a seed for hydrodynamic instability growth which makes the shell a less-efficient piston. Beam imbalances (in timing or power) and target positioning offset (relative to center of beam pointing) introduce low-mode ($\ell \leq 10$), or long-wavelength, perturbations that reduce target performance by degrading the sphericity of the compression and introducing non-radial hot-spot bulk flows [8, 13, 14]. One particular form of LPI, cross-beam energy transfer (CBET) [15, 16, 17, 18], contributes to non-uniform irradiation patterns due to individual beams that interact and transfer energy between each other. Additionally, non-uniformity in individual laser beams (speckles) creates deposition hot spots that seed short-scale perturbations (laser imprint) [19, 20, 21]. Imperfections in target design and manufacturing can also create short-scale perturbation seeds. These include isolated "dome" features on the outer surface, gaps or separation between material layers, ice layer roughness, and internal defects such as voids and bubbles that occur from manufacturing processes. Tritium-decay is another source of imperfection, where the DT fuel deposits energy into the ablator and DT ice layers and creates $10^3 - 10^4$ Helium-3 bubbles per day, causing localized swelling in the plastic ablator material [22]. These short-scale, or high-mode ($\ell > 50$), perturbations reduce implosion performance by degrading compressed fuel ρR, cooling the hot spot through mixing, or causing the shell to break up prior to deceleration.

Hydrodynamic instability growth primarily occurs due to the Richtmyer-Meshkov (RM) [23, 24]

and Rayleigh-Taylor (RT) [25, 26] instabilities. References 27 and 28 provide excellent comprehensive reviews that include ICF-related physics. Traditionally, early-time perturbation evolution is attributed to the RM and RM-like instabilities at the ablation front and various material interfaces. The RM instability occurs when a shock passes through a perturbed interface between a heavy and light fluid and impulsively accelerates the material, creating bubble and spike perturbations that, for small perturbation amplitudes, grow linearly in time. RM perturbation evolution in both laser direct-drive (LDD) and laser indirect-drive (LID) designs exhibits oscillatory behavior that depends on thermal conduction, radiation transfer, and shock-induced vorticity [29]. The importance of RM evolution was emphasized in Refs. 30 and 31 in explaining stability properties of adiabat-shaped, as well as low- and high-foot, designs in LID implosions. The importance of the evolution of acoustic waves inside the shell in the early stages of implosions was recently studied in Ref. 32 concerning the analysis of imprint growth in adiabat-shaped LDD designs, and it was found that the material flow from inside the ablator to the ablation front reduced perturbation amplification during shock transit.

The Rayleigh-Taylor (RT) instability, which occurs when a light fluid pushes a heavy fluid with a perturbed interface, is the dominant instability during the acceleration and deceleration phases. Small perturbation amplitudes of unstable interfaces (η) grow exponentially in time, $\eta(t) = \eta_0\ e^{\gamma t}$, and develop classic bubble and spike structures. The growth rate for classical RT is $\gamma = \sqrt{A_T g k}$, where $A_T = (\rho_{\text{heavy}} - \rho_{\text{light}})/(\rho_{\text{heavy}} + \rho_{\text{light}})$ is the Atwood number, k is the wave number, g is the acceleration, and the heavy and light fluid densities are ρ_{heavy} and ρ_{light}, respectively.

In the acceleration phase, the ablation front is susceptible to ablative RT growth. Ablation stabilizes, or reduces the growth rate of, short-scale wavelengths compared to classical RT. Therefore, the classical ablative RT growth rate is modified to $\gamma = \sqrt{kg/(1+kL)} - \beta k v_{\text{abl}}$, where L is the density gradient scale length, v_{abl} is the ablation velocity, and $\beta \approx 1.5 - 3$ depending on the material used for the ablator [33]. Ablative RT stabilization effects have been studied extensively by Refs. 2, 21, 33, 34, 35, 36 and references therein. Acceleration phase RT growth degrades

implosion performance because it can break up the shell prior to peak compression and jet cold shell material into the hot spot, thereby reducing areal density.

Alternately, deceleration phase RT causes perturbations at the inner surface of the shell to grow and cool the hot spot, which also diminishes yield. Deceleration phase RT growth has been extensively examined for both cryogenic and room temperature implosions [37, 38, 39, 40, 41, 42, 43]. Deceleration phase RT evolves differently for cryogenic compared to room temperature targets due to the different inner layers of the target. Room temperature targets have a distinct material interface and finite Atwood number between the fuel and shell; cryogenic targets eliminate this by the use of solid DT ice as the inner fuel layer. Room temperature targets are uniquely influenced by the high-Z shell that absorbs radiation released by the high-temperature core, causing the stability of the inner shell to evolve differently compared to cryogenic implosions. Instability growth at the inner surface of the shell causes the neutron yield to end prematurely due to hot-spot cooling and mix.

1.2 Thesis Overview

This thesis studies the physics of hydrodynamic instability growth in two phases of the implosion and the practical application of the numerical methods required for simulation. Chapter 2 focuses on the early-time perturbation evolution during shock transit leading up to the start of the acceleration phase. During shock-transit, perturbations from shell material density modulations and isolated defects deposit seeds at various interfaces such as the ablation front and material interfaces. By using detailed simulations and analysis, several key physical processes are identified that play a role in the evolution of perturbations created by defects. This includes secular feedout growth created by rarefaction waves at the ablation front and characteristic wave reverberation between shocks and different interfaces. Mass modulations in the ablator material are also shown to deposit higher instability seeds (compared to modulations in the ice layer) due to increased levels of shock-induced vorticity. Simulations also predict that significant shell mass modulations develop during shell acceleration. The use of low density ablator materials (foam)

is suggested as a potential mitigation strategy to reduce the effects created by these defects.

Furthermore, perturbations at the ablation front feed through to the inner surface of the shell during the acceleration phase and deposit seeds for additional instability growth at the inner surface of the shell during deceleration. In the deceleration phase, the inner surface of the shell is susceptible to RT growth. Chapter 3 examines the stability of the fuel-shell interface in room temperature direct-drive implosions during the deceleration phase in both experiment and simulation. Classical Atwood number calculations, based on continuity in pressure and temperature across the interface, predict varying levels of stability at this material interface through the modification of the ion mass of the fuel, which is achieved by changing the ratio of deuterium to tritium (D:T) in the fuel. However, this chapter demonstrates that the stability of the interface is best characterized by the effective Atwood number, which is primarily determined by material densities at distances on the order of perturbation wavelength on either side of the interface, rather than the density ratio at the interface. Since the densities at these distances are defined not by fuel composition, but radiation heating of the shell, both simulation and experimental data show that target performance is insensitive to different D:T ratios.

Chapter 4 presents the numerical methods used to generate the simulation results in Chapter 2. This covers the finite-volume approach with a Riemann solver for hydrodynamics and an implicit solution method for thermal conduction. Chapter 5 discusses the practical implementation of the numerical methods introduced in Chapter 4. Two new 2D simulation codes were written focusing on low-noise, low-dissipation, high-order accuracy to track characteristic wave propagation in the ICF context. Modern Fortran (2008+) features were tested and applied to the high-order hydrodynamics methods using coarrays. A new multi-physics code was written in Julia using lessons learned from the first implementation and tested the viability of Julia as a language for high-performance computing (HPC). Finally, Chapter 6 concludes the thesis with an overview of lessons learned during the course of this research and opportunities for future work.

Chapter 2

Instability Seeding Mechanisms due to Internal Defects

Portions of this chapter are reproduced from [**Miller, S. C.** and Goncharov, V. N., "Instability Seeding Mechanisms due to Internal Defects in Inertial Confinement Fusion Targets", submitted to *Physics of Plasmas*.

A range of evidence from integrated ICF modeling and experiments supports the importance of accounting for voids and defects which arise from target imperfections and tritium-decay. Experimental data reported in Refs. 44 and 45 show that implosions with a fuel adiabat below a threshold value exhibit enhanced x-ray core emissions that can be explained by mixing CH ablator material into the core. Simulations, however, indicate that although short-scale laser imprint is capable of mixing carbon deep into the ablator, it cannot transport it all the way to the core. In addition, *DRACO* simulations of lower-adiabat OMEGA implosions show that imprint alone cannot explain ablator-fuel mixing observed in experiments [46]. Similar inconsistency with imprint-only simulations was observed in the measurement of the hot-spot self-emission evolution [47]. Surface defects ($5 - 20$ μm in diameter and $0.5 - 5$ μm high), however, have been shown to jet material into the hot-spot, including the ablator material, and degrade target performance [48]. The importance of defects was also emphasized in recent high-resolution

simulations of LID implosions, suggesting that seeding by defects and voids reduces fuel compressibility [49].

The aim of this chapter is to study the evolution mechanisms of these internal ("bulk") perturbations created by micron-scale features inside of the ablator and ice materials and to propose potential target designs reducing their degradation effects. First, Sec. 2.0.1 covers interface and characteristic wave dynamics in a typical ICF implosion as a foundation for explaining perturbation evolution mechanisms. Next, Sec. 2.1 presents the results of single-mode perturbations (short- and long-wavelength). Analyzing single-mode perturbations facilitates a better understanding of seeding mechanisms in the complex setting of a layered target with a low-adiabat laser pulse design. Section 2.2 investigates changing shell thickness and adiabat to determine which physical processes dictate seed amplitude evolution. Increasing the thickness alters internal wave dynamics by lengthening characteristic wave propagation times within the shell, and increasing the adiabat improves ablative stabilization for short-scale perturbations. Section 2.3 examines micron-scale isolated defect wave evolution and its similarities to single-mode perturbations. Perturbations from isolated defects also create complex wave interactions due to the multiple shocks, interfaces, and disparate sound speeds in the ice and ablator materials. Simulations predict that isolated voids create significant shell mass modulations during the acceleration phase. Section 2.4 concludes by presenting a potential strategy to reduce the impact of these internal perturbations though an alternate ablator material such as foam. A thicker ablator with a lower density material reduces secular feedout growth by increasing the rarefaction wave transit time through the shell, and experiences increased levels of ablative stabilization which is beneficial for reducing short-scale perturbation growth.

2.0.1 Wave and Interfacial Dynamics

Internal perturbation evolution is complex due to a multitude of waves (shocks, rarefaction, and compression waves), and their interaction with the inner- and outer-shell surfaces and material interfaces present in ICF target designs. Because spherical convergence effects are not important

during the early stages of an ICF implosion—and to illustrate the origin of different waves and their evolution—a planar target driven by a canonical low-adiabat pulse is used in this study as a surrogate for a typical OMEGA cryogenic implosion design. The target consists of three fluid regions: a 100-μm-thick, low-density $\rho = 0.001$ g/cm^3 layer representing the vapor region; a 40-μm-thick, $\rho = 0.25$ g/cm^3 layer representing DT ice; and an 8-μm-thick, heavier-density $\rho = 1$ g/cm^3 layer to mimic the plastic (CH) ablator. Simulations in this study were run from the beginning of the laser drive up to the early stages of shell acceleration.

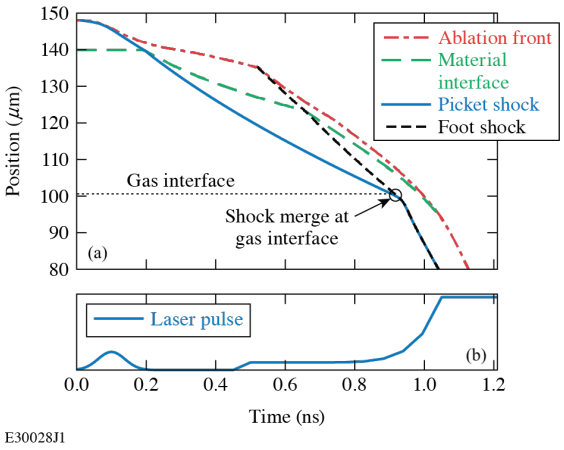

Figure 2.1: Interface and shock trajectories (position versus time) for the target and laser pulse.

Figure 2.1 shows the ablation front, material interface, and shock trajectories for this target driven by a low-adiabat laser pulse. The adiabat α is defined as the ratio of fuel pressure to Fermi-degenerate pressure at fuel density, and for fully ionized DT this can be written as $\alpha_{\mathrm{DT}} = p/2.2\rho^{5/3}$, with p in Mbar, and ρ in g/cm^3. This particular pulse sets the DT-ice layer minimum adiabat to $\sim$2 when the shock breaks out into the gas layer. The laser pulse consists of a single Gaussian-intensity pulse (picket) and a low-intensity foot followed by a rise to the main drive. The picket launches a shock at the beginning of the drive and when the picket intensity drops after $t = 100$ ps, a rarefaction wave (RW) travels from the ablation front to the shock front, reducing its strength (the shock begins to decay at this point). For the design shown in Fig. 2.1, this RW arrives at the shock front at $t = 200$ ps as the shock front passes through the CH–DT interface. Since the ablator material density is higher compared to the DT ice, a rarefaction wave (RW1) is launched back to the ablation front when the shock passes through the material

interface, and a transmitted shock of reduced strength is launched into the DT ice.

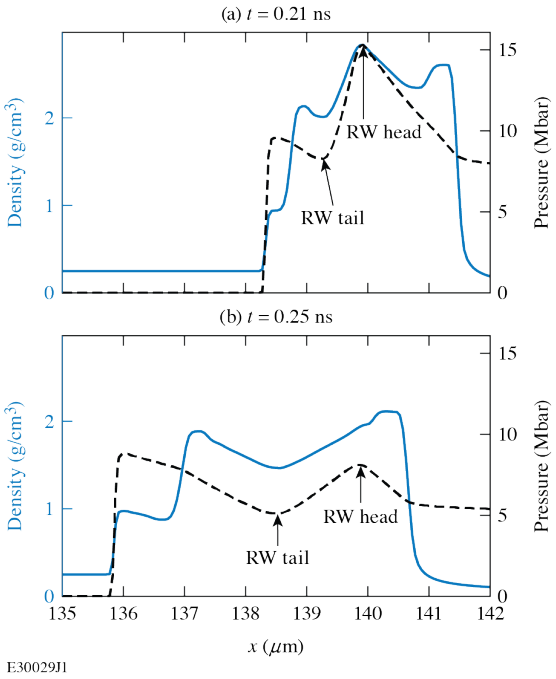

Figure 2.2: After the first shock passes through the material interface, a reflected rarefaction wave travels outward (to the right) to the ablation front. Density is shown in blue and pressure in black. (a) The transmitted shock and rarefaction begin to propagate to the left and right respectively. (b) The pressure and density gradients within the rarefaction wave flatten out as it moves toward the ablation front.

As RW1 propagates through the ablator, the shell has pressure and density profiles similar to the ones shown in Fig. 2.2. Figure 2.2(a) shows the profile immediately after the first shock has passed through the material interface. The rarefaction wave RW1, which has a distinct head and tail, moves from left to right in the figure—from the material interface toward the ablation front (near $x = 138.5$ μm and $x = 141.5$ μm, respectively, in the upper pane). This rarefaction process relaxes the local density and flattens out the pressure gradient later in time, which can be seen in Fig. 2.2(b). While the laser is off between the picket and foot, the shell thickness (distance between the material interface and ablation front) increases due to the reduction in ablation pressure. This rarefaction process is repeated again after the foot shock (launched by the foot of the pulse) passes through the material interface. The primary difference between the first and second shock is that the second shock is supported. The rarefaction after the second shock (RW2) takes less time to travel from the interface to the ablation front because of the

smaller ratio of ablator thickness to ablator sound speed. The second shock launches into the shell at 0.5 ns, reshocks the material interface near 0.65 ns, and merges with the first shock front close to the gas interface (near 0.9 ns, see Fig. 2.1).

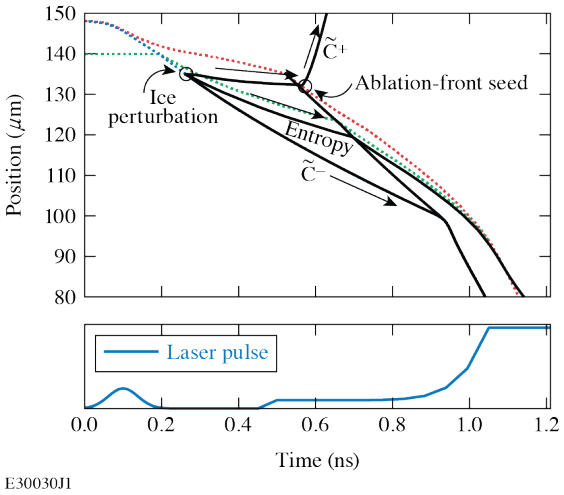

Figure 2.3: When the first shock hits the perturbation it creates three primary waves that carry information throughout the target ($\widetilde{\mathrm{C}}^{+}$, $\widetilde{\mathrm{C}}^{-}$, and entropy). The $\widetilde{\mathrm{C}}^{+}$ wave seeds the ablation front near 0.6 ns, as indicated by the black circle. The entropy wave travels with the local fluid velocity and will eventually be ablated. The $\widetilde{\mathrm{C}}^{-}$ wave travels along the leading shock trajectory.

When a shock wave passes through an internal perturbation, such as a defect, a void in the material, or an interfacial gap, it launches perturbation waves that travel along characteristic hypersurfaces. For small perturbations decomposed into Fourier harmonics, each wave harmonic travels along characteristics defined as: $(\mathrm{d}x/\mathrm{d}t)_{\mathrm{C}^+} = U + c_{\mathrm{s}}$ (the C^{+} characteristic), $(\mathrm{d}x/\mathrm{d}t)_{\mathrm{C}^-} = U - c_{\mathrm{s}}$ (the C^{-} characteristic), and $(\mathrm{d}x/\mathrm{d}t)_{\mathrm{e}} = U$ (the entropy wave that travels with the local fluid velocity), where U is local fluid velocity and c_{s} is local sound speed. References to C^{+} and C^{-} characteristics or trajectories indicate acoustic wave propagation associated with 1-D hydrodynamic flow (such as rarefaction and compression waves, see Fig. 2.3). References to $\widetilde{\mathrm{C}}^{+}$ and $\widetilde{\mathrm{C}}^{-}$ denote the acoustic waves that carry the modulation due to defects or perturbations. Figure 2.3 shows the three wave trajectories from a small defect in the ice located 5 μm away from the material interface. The $\widetilde{\mathrm{C}}^{+}$ characteristic carries perturbation information to the ablation front, the $\widetilde{\mathrm{C}}^{-}$ characteristic immediately catches up to the shock front, and the entropy wave is "frozen" into the fluid and carries the vorticity created by the shock–perturbation

interaction.

The first instance of a perturbation seed at the ablation front occurs when the $\widetilde{C}^+$ wave, which originated at the defect location, crosses over the ablation front. The time it takes for this $\widetilde{C}^+$ wave to traverse through the ablator and reach the ablation front is dictated by the position of the material defect, the thickness of the shell, and the velocity and sound speed of the material. Once the $\widetilde{C}^+$ wave moves out into the corona, the slope of its trajectory abruptly changes because of the larger sound speed and flow velocity in the blow-off plasma region, as shown in Fig. 2.3.

The entropy wave represents the trajectory of the fluid particles at the original position of the perturbation; vorticity created by the shock moving through the perturbation also travels with the entropy wave. Depending on the position of the material defect, the shock-induced vorticity can either be ablated (if a defect is close enough to the target's outside edge) or stay and evolve within the shell for the duration of the implosion. Figure 2.4 plots a series of entropy waves

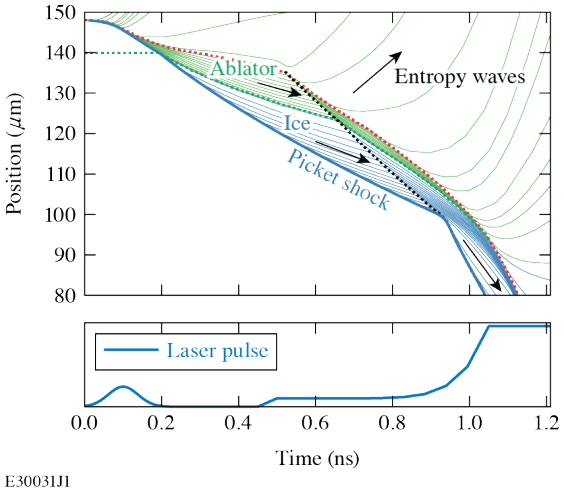

Figure 2.4: Entropy wave trajectories. These waves originate from the leading shock and travel with the local fluid velocity. Green lines represent trajectories that start in the plastic layer and blue represents those that start in the ice. The ablation front (dotted red), material interface (dotted green), and shock (dotted black) trajectories are included. The direction of propagation of each wave is indicated by the arrows.

that originate from the shock as it propagates through the shell. This illustrates the overall flow of shock-induced vorticity throughout the material leading up to the acceleration phase. The entropy waves are color-coded based on the material in which the trajectory begins. The figure

shows that vorticity created by the shock interacting with a defect in the ablator will affect the ablation-front distortion much earlier in time than those starting in the ice. In addition, vorticity from defects in the ice will not reach the ablation front until close to (or after) the acceleration phase begins, if at all.

Information at the ablation front is communicated into the target along C^- characteristics. Figure 2.5 illustrates this with a series of C^- trajectories starting at the ablation front at regular intervals. For example, when the second shock passes through the material interface, it creates a rarefaction wave (RW2) that reaches the ablation front. After the head of RW2 breaks out at the ablation front (and because the ablation-front pressure is supported by the laser foot drive), this wave is reflected back into the shell along C^- characteristics. Close examination of Fig. 2.5 reveals that a subset of C^- characteristics start to converge as they approach the inner surface of the target, which indicates the formation of a compression wave. The reflected RW2 and rise in the laser pulse (from foot intensity to full power) create compression waves that propagate inward to the inner surface. Following the propagation of acoustic waves along characteristics

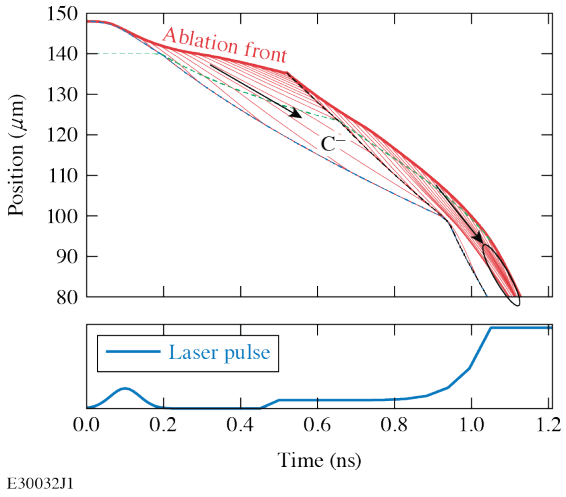

Figure 2.5: C^- characteristics originating from the ablation front propagate inward to the leading shock. The circle indicates the formation of a compression wave. The ablation front (dashed red), material interface (dashed green), and shock (dashed black) trajectories are included. Arrows indicate the direction of propagation of the waves.

allows us to track and understand how perturbation information moves throughout the shell. Target dimensions, material defect position, and shock-wave timing are the primary drivers that

determine how and when perturbations create seeds at the ablation front, as shown through the simplified implosion presented in this section. The following sections address wavelength dependency, phase interaction, and interfacial distortion growth.

2.1 Single-Mode Perturbations

This section studies ablation-front seeding by applying single-mode, sinusoidal density perturbations to the target material over a range of different depths x_{p} (relative to the material interface) within the plastic ablator and DT ice layers at $t = 0$ ns. The density modulation amplitude is chosen to ensure that the ablation-front and interface distortions stay linear ($k\eta < 1$, where k is the wave number and η is the distortion amplitude) for the duration of each simulation. Ablator perturbations are located at $x_{\mathrm{p}} \in [+1, +7]$ μm and ice perturbations are at $x_{\mathrm{p}} \in [-1, -35]$ μm. The interface is localized at $x = 140$ μm in this study. Perturbations are applied to the initial material density as

$$\eta_{(x,y)} = A_\rho e^{-k_x(x-x_0)^2} \left[1 - \cos(k_y y)\right], \tag{2.1}$$

where A_ρ is some fraction (1% to 10%) of the local density at x_0 and $k_{x,y} = 2\pi/\lambda_{x,y}$. The perturbation at x_0 is based on the grid location (see, for example, the ice perturbation at $x_0 = 135$ μm in Fig. 2.3), although throughout this paper this is reported as a depth relative to the material interface (x_{p}). The perturbation spatial decay rate in x (λ_x) is fixed at 2 μm and the wavelength in y (λ_y) is varied. Since λ_x is fixed, any references to λ refer to λ_y unless otherwise specified.

The position of the ablation front is tracked using the location of the steepest density gradient on the outside of the shell. The material interface is tracked by time integration of the local fluid velocity from the position of the interface at $t = 0$. This is the most robust method of tracking the interface since the code uses a fixed Eulerian mesh. The distortion of each interface is decomposed into Fourier harmonics and the fundamental mode is used to define the perturbation amplitude. The computational domain is a half-wavelength in y with reflective boundaries in

$y = 0$ and $y = \lambda/2$, a zero-gradient boundary at $-x$, and an outlet boundary at $+x$, where material is allowed to freely exit the domain.

The next section details perturbation evolution and ablation-front distortion seeding for long and short single-mode wavelengths where the importance of contribution of shock-induced vorticity is highlighted. For long-wavelength modes ($\lambda \geq 40$ μm), shock vorticity convection at the ablation front has a small effect, while short wavelengths ($\lambda < 40$ μm) experience oscillations caused by vorticity convection.

2.1.1 Long-Wavelength Perturbations

Long-wavelength ($\lambda =$ 40, 60, and 100 μm) single-mode perturbations are applied at multiple depths within the ablator and ice materials. Time history of the absolute value of the ablation-front distortion for a single-mode $\lambda = 100$-μm perturbation applied at four different depths (two in plastic, two in DT ice) is shown in Fig. 2.6. The laser pulse used for these results is shown in Fig. 2.1. The acceleration phase is shaded in light gray in Fig. 2.6 and starts at $t = 1.02$ ns. For each simulation, the seed amplitude η_0 is extrapolated from an $\eta(t) = \eta_0 e^{\gamma t}$ fit during the acceleration phase of the simulation, where γ is the growth rate. The seed for each case is annotated in the figure as the circle at the beginning of the fit.

The evolution of ablation-front distortion for the cases shown in the figure is determined by the initial position of the perturbation and characteristics wave ($\widetilde{\mathrm{C}}^+$, $\widetilde{\mathrm{C}}^-$, and entropy) propagation. These simulations show that the seed amplitude is largest for ablator material perturbations and reduces as the initial position moves further into the DT ice. As this depth increases (further away from the outer surface), the onset of distortion growth at the ablation front is delayed. Note the temporal separation of 200 ps for the two perturbations separated by only 2 μm on either side of the material interface ($x_{\mathrm{p}} = \pm 1$ μm, see Fig. 2.6). This is due to the relaxation of the shell material's density and the increase in the ablator thickness as RW1 travels toward the ablation front. The $\widetilde{\mathrm{C}}^+$ wave on the interior side of the material interface must propagate through a thicker shell compared to the perturbation in the ablator material.

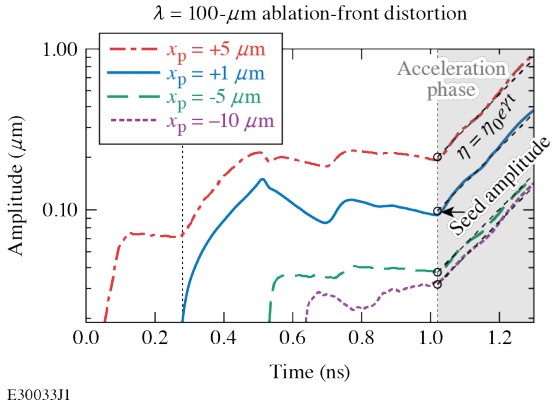

Figure 2.6: Ablation-front distortion history corresponding to perturbations at different locations in the ablator and DT ice. The acceleration phase is shaded in light gray. The dashed line is the exponential fit to $\eta = \eta_0 e^{\gamma t}$ and the seed amplitude is extracted from this curve. The initial perturbation depths are $x_p = +5, +1, -1$, and -10 μm relative to the material interface for the red, blue, green, and purple lines, respectively. The vertical black line near $t = 0.3$ ns indicates the arrival of the RW head at the ablation front.

The ablation-front distortion caused by the two ablator perturbations is shown in Fig. 2.6 in red and blue curves. The distortion grows early due to the rarefaction wave (RW1) that crosses over the ablation front between 0.3 and 0.5 ns. The arrival of the head of the rarefaction wave at the ablation front is indicated in the figure by the vertical black line. Prior to this event, the ablation-front distortion from the shallowest perturbation (red line) grows briefly (the $\widetilde{C}^+$ wave arrives at the ablation front at $t = 0.12$ ns), but the growth plateaus until the head of the rarefaction wave arrives. This early period of growth ($t = 0.3$ to 0.5 ns) for perturbations starting in the ablator can be explained by the following series of events, similar to the well-known feedout process [12, 50, 51] that is illustrated in Fig. 2.7. First, the shock front becomes distorted after interacting with the density modulation in the ablator (creating the distinctive peak and valley of a sinusoidal mode). Then, the peak of the distorted shock front arrives at the material interface slightly earlier than the valley, launching C^+ characteristics (that represent the head of the rarefaction wave) back to the ablation front (t_1 in Fig. 2.7). The head of the rarefaction wave is also out of phase with the ablation front at this point. Because of their earlier creation time, the C^+ characteristics originating from the peak arrive earlier at the ablation front and establish a pressure gradient (and acceleration) that influences the local velocity gradient of the ablation front at t_2. Acceleration is delayed at the valley, thus creating a velocity difference

along the ablation front that alters the phase. This velocity modulation leads to secular growth or linear-in-time amplitude amplification at t_3. Secular feedout growth continues until the second shock is launched into the ablator at 0.5 ns, that is seen by the abrupt change in growth history in Fig. 2.6.

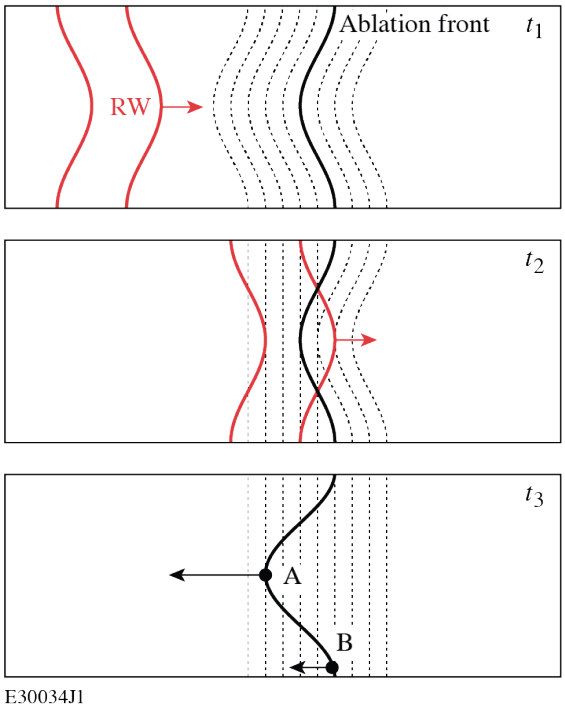

Figure 2.7: Secular growth at the ablation front is created by the out-of-phase rarefaction wave as it crosses the ablation front. The head and tail of the rarefaction wave are indicated as red lines moving from left to right. The ablation front (black line) starts out at t_1 with the velocity gradient (dashed lines) in phase with itself. As the rarefaction wave moves across the ablation front at t_2, the velocity gradient morphology goes out of phase and introduces localized acceleration that increases the amplitude of distortion at t_3. Points A and B experience different velocities ($v_A > v_B$), causing the distortion amplitude to stretch out and increase.

The ablation-front distortion caused by ice perturbations evolves less dynamically compared to the those that originate in the ablator. Since the $\widetilde{C}^+$ wave (originating from the ice perturbation) travels behind the tail of RW1, feedout amplification does not occur. Distortion growth also starts much later, as it takes the shock longer to reach the perturbation; the distortion amplitude is fairly constant once the $\widetilde{C}^+$ wave arrives at the ablation front. There are minor changes due to the second rarefaction wave (RW2), but these are much less influential on the seeding amplitude at the start of the acceleration phase.

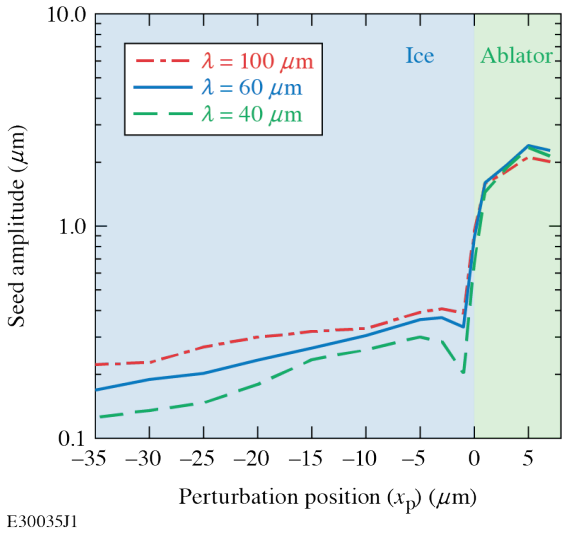

Figure 2.8: Scaled seed amplitude (η_s) as a function of perturbation position (x_p) for different wavelengths (λ).

Figure 2.8 summarizes the scaled seed amplitude extracted from three perturbation wavelengths ($\lambda = 40$, 60, and 100 μm) and differing initial depths. The scaled seed amplitude is defined as $\eta_s = \eta_0/A_\rho$, where η_0 is the distortion in microns (extracted from the fit) at the ablation front at the start of the acceleration phase, and A_ρ is the initial mass modulation amplitude (as fraction of 1-D value). To keep the perturbations at the ablation front in the linear regime ($k\eta < 1$) the initial modulation is 10% for $\lambda \geq 40$ μm. For short-wavelength perturbations (data for $\lambda < 20$ μm are presented in the next section) the initial modulation is decreased. Because normalization is based on the initial density-modulation amplitude, the seed amplitude is insensitive to this reduction. For wavelengths greater than 40 μm, the seed amplitude has weak wavelength dependence for ablator perturbations, whereas seed amplitudes from ice perturbations moderately increase with wavelength. The data show a clear trend that (1) ablator perturbations create the largest seeds and (2) seeds created by ice perturbations decrease in amplitude as the depth increases. This behavior is expected to also apply to wavelengths larger than 100 μm.

2.1.2 Short-Wavelength Perturbations

The perturbation seeding trend described in the previous subsection does not extend for wavelengths less than 20 μm, however, due to increased levels of shock-induced vorticity convection

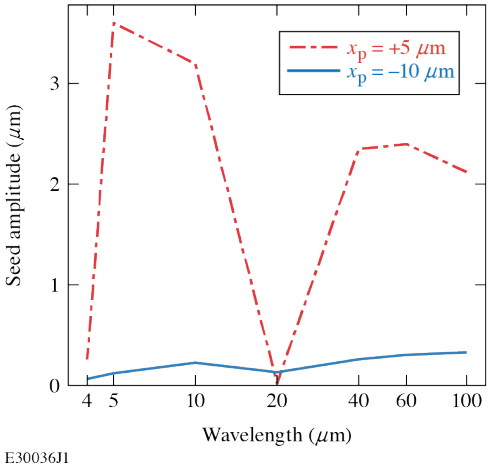

Figure 2.9: Scaled ablation-front seed amplitude (η_s) versus wavelength for an ice ($x_p = -10$ μm shown with the blue line) and ablator ($x_p = +5$ μm, shown with the red line) perturbation.

that causes the distortion at the ablation front to oscillate [29]. This can be seen by comparing ice and ablator perturbation seed amplitudes over a range of wavelengths from 4 μm to 100 μm, as shown in Fig. 2.9. For the shorter wavelengths, the initial density modulation was decreased to keep the ablation-front modulations in the linear regime: $A_\rho = 0.015$ for $\lambda < 10$ μm and $A_\rho = 0.05$ for $\lambda = 10$ to 20 μm. For ice perturbations, decreasing the wavelength leads to a decrease in the seed amplitude, whereas for ablator perturbations, decreasing the wavelength *increases* the seed amplitude (with the exception of $\lambda = 4$ μm and 20 μm). Seed amplitudes are drastically less for ablator perturbations of 4 μm and 20 μm due to phase oscillation of the distortion at the ablation front. Because phase oscillations (caused by shock amplitude oscillation and ablative stabilization mechanisms) play a larger role as wavelength decreases, instability seeding at the ablation front fluctuates significantly for short-scale perturbations. This explains why seed amplitudes have less obvious depth dependence trends for individual modes and that only an envelope trend must be considered (refer to Fig. 2.10). Although seed amplitudes from ice perturbations reduce as depth increases, the trend near the material interface is also affected by the classical RT instability that develops at the interface after the first shock transit. Here, the pressure decreases from the shock front toward the ablation front while density increases from the ice to the ablator, creating an RT-unstable hydrodynamic configuration at the interface. Classical RT instability growth (which rapidly increases as wavelength decreases due to the lack of

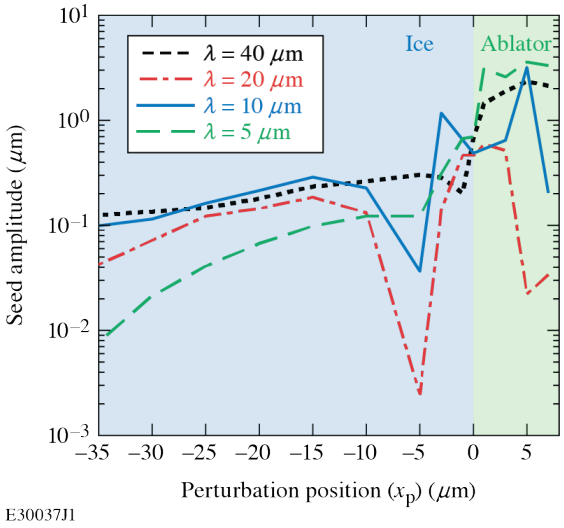

Figure 2.10: Scaled seed amplitude (η_s) as a function of perturbation position (x_p). The long-wavelength $\lambda = 40$ μm result shown by the dashed line is included for reference.

ablative stabilization) contributes to the complexity of the characteristic wave evolution within the shell, adding varying levels of constructive or destructive interference as waves reverberate.

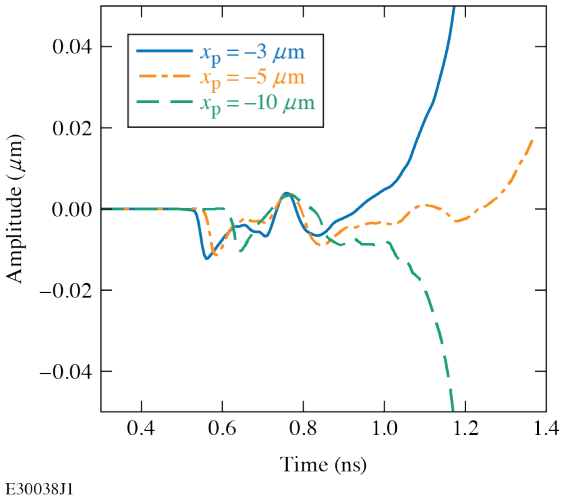

Figure 2.11: Ablation-front distortion growth for $\lambda = 20$ μm with ice perturbations at $x_p = -3, -5, -10$ μm. Destructive wave interference with the distorted material interface results in reduced growth amplification for $x_p = -5$ μm.

Wave interference with the distorted material interface causes the distortion at the ablation front to reverse phase and create a local minimum in Fig. 2.10 at $x_p = -5$ μm. Figure 2.11 compares the ablation-front distortion growth for a 20-μm perturbation in the ice at depths of $x_p = -3, -5,$ and -10 μm to demonstrate how this occurs. Here, exponential growth of the ablation front

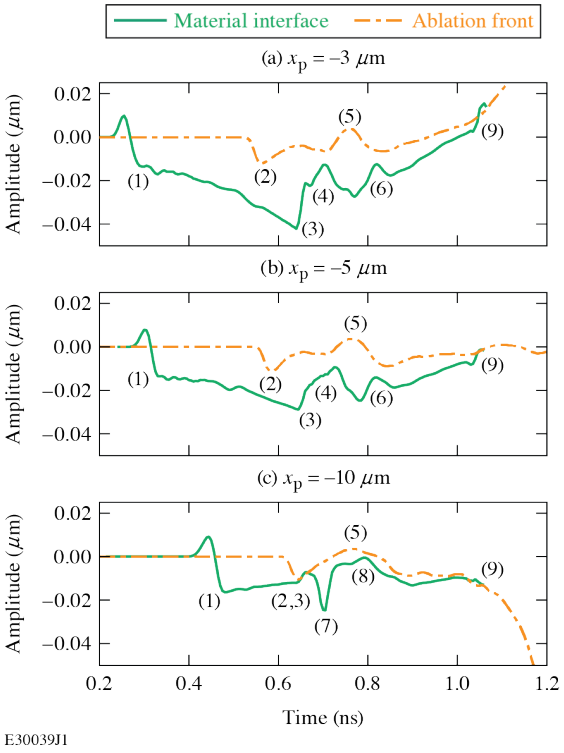

Figure 2.12: Ablation front (orange) and material interface (green) distortion growth for $\lambda = 20$ μm at depths of (a) -3 μm , (b) -5 μm, and (c) -10 μm relative to the material interface. Key events (1–9) are labeled in each plot; refer to the text for a full explanation of each event.

from the ice perturbation at $x_p = -5$ μm is delayed by 200 ps (from 1.0 to 1.2 ns) due to phase oscillation (indicated by the orange line in the figure). The distortion of the material interface contributes to this, as seen in Fig. 2.12. The evolution history in Fig. 2.12 can be explained by the sequence of events (and labels in the figure) described below. In each case, the distortion evolution starts when the first shock crosses over the ice perturbation and a $\widetilde{C}^+$ wave carries the distortion back to the material interface and ablation front.

1. The $\widetilde{C}^+$ wave, from the first shock interacting with the perturbation, crosses the material interface (moving outward) and seeds the perturbation there.

2. The $\widetilde{C}^+$ wave from (1) reaches the ablation front and initiates perturbation growth.

   - Note: The foot shock launches between (1) and (2) (near $t \approx 0.52$ ns) before the ablation front becomes perturbed, so no distortion shows up in the plot.

3. The second shock passes through the distorted material interface and reduces its pertur-

bation growth.

4. The $\widetilde{\mathrm{C}}^{+}$ wave, from the second shock interacting with the the original ice perturbation moving with the flow (entropy wave), crosses the material interface (moving outward).

5. RW2 (RW from the second shock crossing the material interface) reaches the ablation front.

6. The reflected C^{-} wave from (5) (beginning of a compression wave) passes through the material interface (moving inward).

7. The reflected C^{-} wave from (2) passes through the material interface. ($x_{\mathrm{p}} = -10$ μm only)

8. The C^{-} wave from (7) reflects off the shock front and now passes through the material interface (moving outward) as a new C^{+} wave. ($x_{\mathrm{p}} = -10$ μm only)

9. The material interface is ablated.

The deeper the ice perturbation, the longer it takes for this $\widetilde{\mathrm{C}}^{+}$ wave to seed the material interface and ablation front. Timing of the second shock is the same in each case, but the perturbation starting position determines how long these characteristic waves are allowed to evolve and reverberate inside the shell. The events (1–9) attempt to capture the main phenomena responsible for interfacial evolution, but the list is not exhaustive for the sake of simplicity. In essence, the $x_{\mathrm{p}} = -5$ μm position represents the inflection point where internal wave dynamics cause the phase of the ablation-front distortion to change signs. Subtle changes in perturbation depth in this shallow region of the ice lead to large changes in evolution due to these mechanisms. For example, the $x_{\mathrm{p}} = -10$ μm case evolves differently because it takes longer for the second shock to reach the perturbation than in the other cases shown. By the time waves start reverberating between the material interface and ablation front, the shell in this case is comparatively thinner. The local minimum in Fig. 2.10 due to perturbations at $x_{\mathrm{p}} = -5$ μm also occurs for $\lambda = 5$ μm and 10 μm.

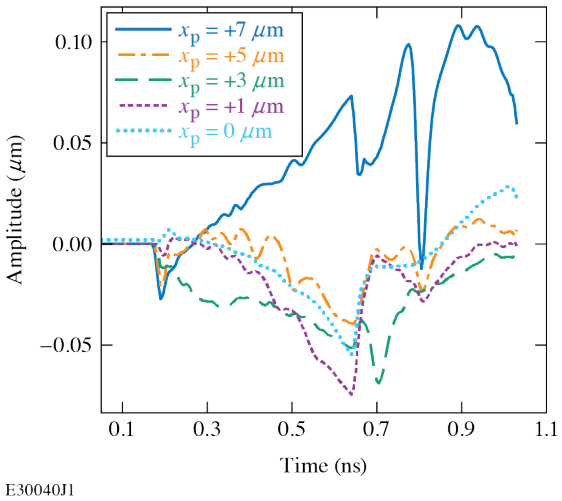

Figure 2.13: Material interface distortion growth for perturbations originating in the ablator material. The second shock passes through the material interface near $t = 0.65$ ns and briefly reduces growth.

Figure 2.13 shows the material interface growth for $\lambda = 20$ μm perturbations that start in the ablator material ($x_p = +7,\ +5,\ +3$, and $+1$ μm) and directly on the material interface (at $x_p = 0$ μm, where the perturbation straddles the interface and is applied to both materials). Each of these cases see material–interface distortion growth between $t = 0.20$ to 0.65 ns after the first shock from the RT instability established by the opposite directions of the pressure and density gradients. However, only the perturbation at $x_p = +7$ μm sees a positive phase value due to position and timing of the original perturbation. As described previously, timing of the $\widetilde{C}^+$ and $\widetilde{C}^-$ waves also has an impact on the evolution of the distortion, particularly near 0.8 ns, when a small compression wave (from RW2 reflected at the ablation front) crosses back over the material interface. This is event (6) in Fig. 2.12. Furthermore, the brief increase in amplitude starting at 0.7 ns is only experienced by the cases with perturbations at $x_p = +3$ μm and $+7$ μm (when a C^- wave crosses the interface), whereas remaining cases show minimal change from this event. All cases experience a form of RT growth at the material interface, but the initial position of the perturbation dictates how this information is communicated throughout the target. The perturbation growth at the ablation front and material interface is the result of complex interplay of characteristic waves reverberating within the shell and ablation stabilization mechanisms.

As the wavelength decreases, ablative stabilization increasingly limits the amount of ablation-

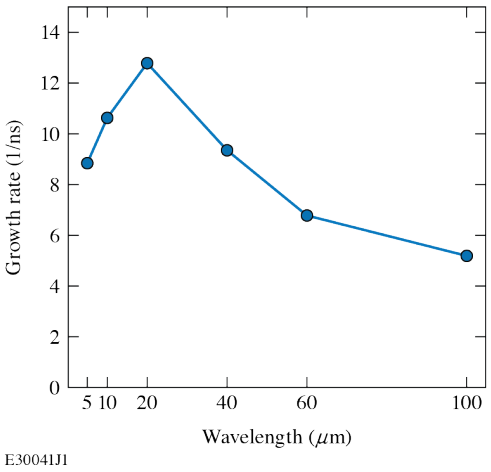

Figure 2.14: Average acceleration phase RT growth rate versus wavelength at the ablation front. Growth rates are extrapolated from an $\eta = \eta_0 e^{\gamma t}$ exponential fit.

front growth. Average RT growth rates during the acceleration phase are shown in Fig. 2.14 for $\lambda = 100,\ 60,\ 40,\ 20,\ 10,$ and 5 μm, and ablative stabilization can be seen in the reduction of growth rates for $\lambda < 20$ μm. For $1 < \lambda < 4$ μm, the perturbations still grow, but high-frequency phase oscillation makes growth-rate extrapolation inconsistent. Mass ablation totally stabilizes RT growth for $\lambda < 1$ μm. Therefore, these wavelengths are omitted from Fig. 2.14. Figure 2.15 shows the ablation-front distortion history for an ablator layer perturbation at $x_{\mathrm{p}} = +7$ μm (1 μm from the outer surface) for these wavelengths. The phase oscillation rates increase as λ decreases, which is a consequence of dynamic overpressure and vorticity convection effects [29]. This can be seen in the first phase transition between 0.1 and 0.4 ns, where $\lambda = 2$ μm has the earliest transition. Later in the implosion, the ablation-front modulation of the $\lambda = 4$ μm case is in the midst of a phase transition at the start of the acceleration phase ($t = 1.02$ ns). However, the $\lambda = 5$ μm case does not change phase at this point and the start of exponential growth is easily identifiable. Note that perturbation growth above the limit $\eta \approx 0.1\lambda$ is nonlinear, and wavelengths 2 to 5 μm quickly pass this threshold during the acceleration phase. For wavelengths of 2 to 4 μm, the ablation-front perturbation changes phase during and after the rise of the main pulse (starting at 0.8 ns), and the resulting oscillation delays exponential growth. The smallest wavelength, however, is completely stabilized by ablation throughout the simulation.

Even though ablative stabilization makes the RT growth rate become negative for wavelengths

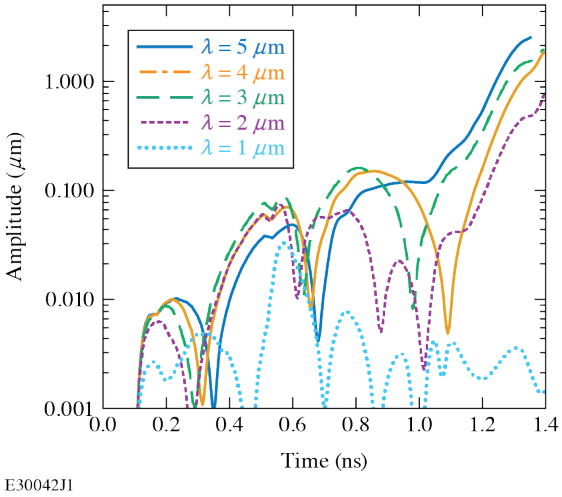

Figure 2.15: Ablation-front distortion for ablator perturbations at $x_p = +7$ μm (relative to the material interface) with wavelengths $\lambda = 1$ to 5 μm. Ablative stabilization occurs for $\lambda = 1$ μm and significantly reduces growth.

shorter than the cutoff, these wavelengths can still experience significant growth. This is shown in Fig. 2.16 where the evolution of 1-μm perturbations is plotted for several locations of the initial density modulation. This mode is completely stabilized by ablation, therefore, there is no RT amplification and modes that originate in the ablator and ice for $x_p > -5$ μm do not grow. However, for perturbations deeper than 10 μm in the ice, rapid amplification occurs as soon as the entropy wave (and shock-induced vorticity) originating at the location of initial perturbation reaches the ablation front. Although the acceleration phase starts at 1.02 ns, none of the deepest ice perturbations ($x_p = -10, -15$, and -20 μm) grow until later when the entropy wave meets the ablation front. The $\widetilde{C}^+$ wave has already seeded the ablation front prior to the start of the acceleration phase, although, as expected, ablative stabilization keeps the growth minimal. This phenomenon has been observed previously in Ref. 11 where it was alternatively described as the superposition of the internal convective mode (e.g., entropy wave) and the RT mode at the ablation front. The depth of the ice perturbation acts to shield and isolate the entropy wave from ablative stabilization until the ablation front has burned through enough material to reach it. By this time, the vorticity seeded by the shock-defect interaction has evolved due to gradients in hydrodynamic profiles. Figures 2.17 and 2.18 show density and y-velocity contours of the case where the perturbation is at $x_p = -10$ μm, before ($t = 1.03$ ns) and after ($t = 1.24$ ns) the

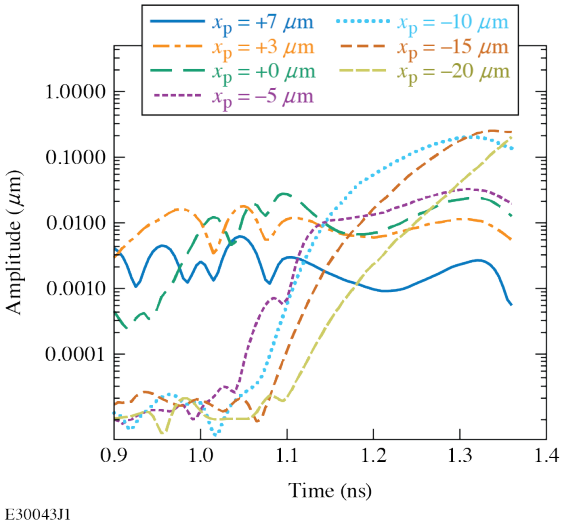

Figure 2.16: Ablation-front distortion growth for $\lambda = 1$ μm at multiple depths relative to the material interface [in ablator (+) and in ice (-)]. Ablative stabilization occurs for perturbations at $x_p \geq -5$ μm, but growth surprisingly still occurs for deeper ice perturbations.

entropy wave reaches the ablation front. The entropy wave is visible as the dipole-like feature in the y-velocity contour, and the ablation front is at the steepest density gradient at the outermost position of the shell (which accelerates from right to left).

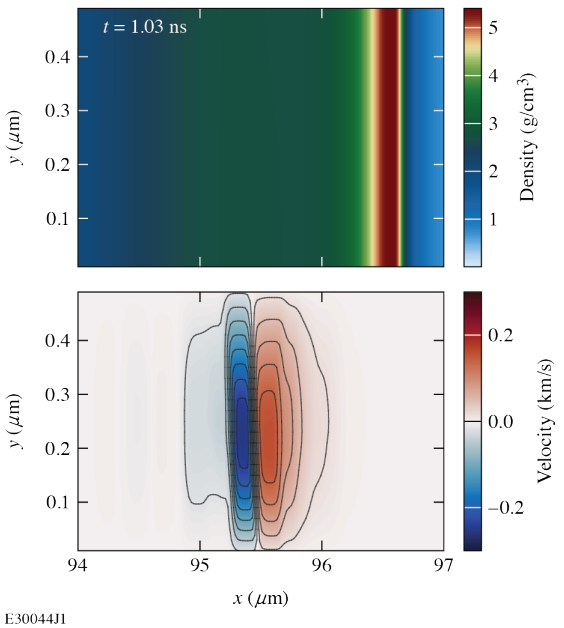

Figure 2.17: The 1-μm ice perturbation at $x_p = -10$ μm at $t = 1.03$ ns, just after the acceleration phase has started. The entropy wave (near $x = 95.5$ μm) has yet to reach the ablation front, located near $x = 96.7$ μm. The shell accelerates from right to left.

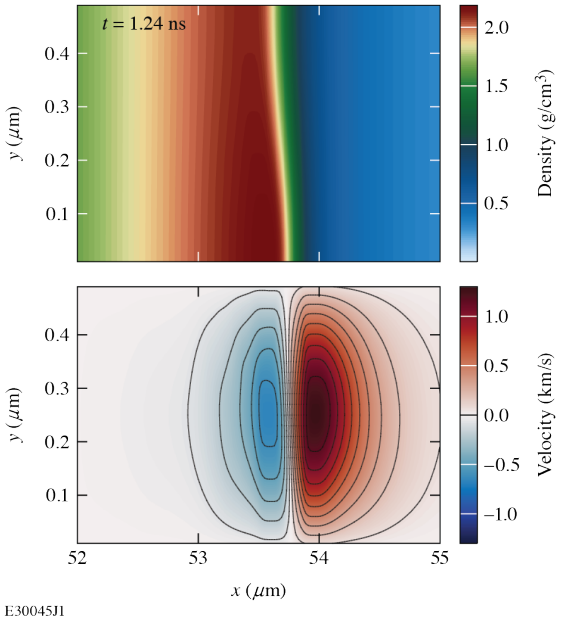

Figure 2.18: The 1-μm ice perturbation at $x_p = -10$ μm at $t = 1.24$ ns. The entropy wave is at the ablation front ($x \approx 53.5$ μm) and distortion is visible in the density contour. The shell accelerates from right to left.

In Fig. 2.17 ($t = 1.03$ ns), the perturbation has not yet reached the ablation front and can be seen near $x \approx 95.5$ μm in the y-velocity contour plot. By $t = 1.24$ ns (Fig. 2.18), the entropy wave is at the ablation front (the material between the perturbation and the original outer surface has been ablated) and the distortion can be seen in the density plot near $x \approx 53.5$ μm.

2.2 Altering Shell Adiabat and Ablator Thickness

Target hydrodynamic stability has been shown to improve for designs with thicker shells and higher fuel adiabats [9, 10, 11, 12]. Next, we examine the sensitivity of internal perturbation evolution to ablator thickness and shell adiabat. Increasing the adiabat increases the ablation velocity and ablative-stabilization effects. Modifying the strength of the picket shock (which changes the adiabat) and increasing the thickness of the ablator (from 8 to 10 μm) changes the dynamics of RW1, which impacts the level of secular feedout growth. The shell adiabat of the 8-μm ablator target is increased by modifying the original pulse (pulse A in Fig. 2.19) to incorporate a larger picket and earlier foot. This modified pulse (pulse B in Fig. 2.19) increases

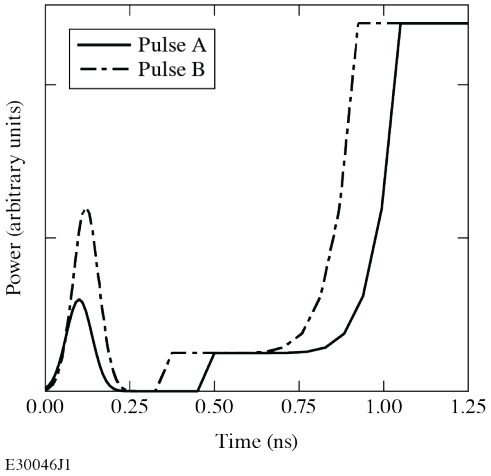

Figure 2.19: The different laser pulses used for comparison. Pulse A is a low-adiabat design used on 8-μm and 10-μm thick ablators. Pulse B increases the adiabat for the 8-μm ablator.

the minimum shell adiabat (when the foot shock breaks out into the vapor region) from $\alpha = 1.9$ to $\alpha = 3.1$. The larger picket increases the pressure behind the first shock (as it breaks out into the DT ice) from 22 Mbar to 38 Mbar. Pulse A is designed such that the picket and foot shocks merge at the ice–vapor interface; for pulse B this occurs in the vapor region. Pulse A is also applied to a thicker, 10-μm ablator design (with the same ice thickness), which sets the minimum shell adiabat to $\alpha = 1.9$.

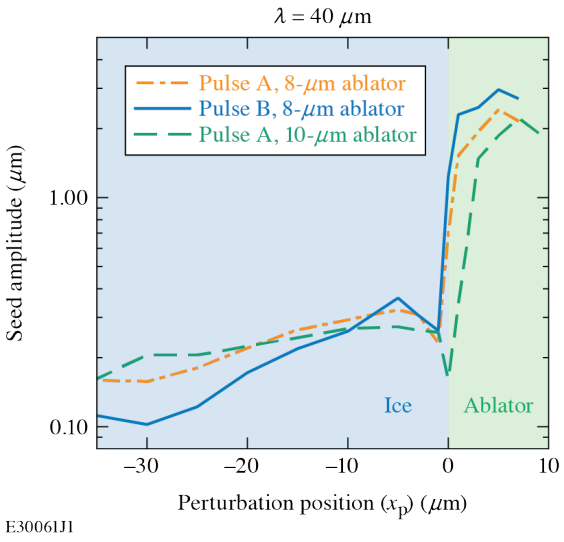

Figure 2.20: Seed amplitude (η_s) for each pulse and ablator configuration with 40-μm single-mode perturbations at different origin depths (x_p).

Figure 2.20 summarizes seed amplitude versus perturbation depth for a single-mode 40-μm

perturbation. The seed amplitudes show similar behavior regardless of ablator or pulse: ablator perturbations deposit larger seeds, ice perturbations decrease as the depth increases, and perturbations near the material interface result in a local minimum. The local minimum near the material interface is caused by the characteristic wave reverberation and the influence of the distorted interface. This local minimum disappears as the perturbation wavelength increases since the effect of the classical RT growth at the material interface decreases for increasingly larger wavelengths (cf. Fig. 2.8).

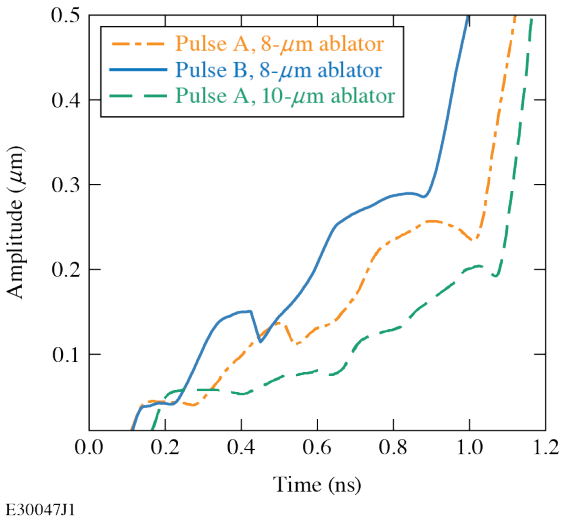

Figure 2.21: Ablation-front distortion history of a $\lambda = 40$ μm single mode ablator ($x_{\mathrm{p}} = +5$ μm) perturbation for laser pulses A and B with shell thicknesses of 8 and 10 μm.

Figure 2.21 plots the ablation-front distortion history due to an ablator perturbation at $x_{\mathrm{p}} = +5$ μm. Comparing perturbation evolution in different designs at longer wavelengths (the plot shows the results for $\lambda = 40$ μm) helps to separate the various effects that influence perturbation growth. It captures behavior like secular feedout growth without phase oscillation (of the ablation-front distortion) that introduces additional complexity at shorter wavelengths. In Fig. 2.21 the distortion of the ablation front for the 8-μm ablator with pulse B (shown as the blue line) evolves faster compared to pulse A due to the stronger picket and earlier foot. The secular feedout growth created by RW1 is shorter and occurs earlier in B versus A (for the 8-μm ablator) due to the stronger shock. The stronger shock also increases the sound speed comparatively, resulting in faster acoustic wave propagation. In addition, secular feedout growth is affected by

when the second shock launches. Between $t \approx 0.35$ and 0.42 ns, the secular feedout growth for B plateaus, as the tail of RW1 arrives at the ablation front and pressure gradients reduce before the second shock launches into the shell ($t = 0.42$ ns). For pulse A, there is no plateau because the tail of RW1 breaks out at the ablation front roughly when the second shock launches into the shell (at $t \approx 0.53$ ns). After the second shock, the distortion established by the secular feedout growth continues to increase due to RW2 and the hydrodynamic gradients established at the ablation front. Overall, pulse B compresses the evolution history in time compared to pulse A and slightly increases the seed amplitude (from 0.233 to 0.288 μm) for the 8-μm ablator.

For the 10-μm ablator, most of the ablation-front distortion growth occurs after the second shock wave, which is launched at $t = 0.53$ ns. There is negligible secular feedout growth because the thicker ablator delays the arrival of the rarefaction wave at the ablation front. The head of RW1 reaches the ablation front approximately 110 ps before the second shock, which gives RW1 less time to cross the ablation front compared to the 8-μm ablator cases. Although secular feedout growth is primarily due to the hydrodynamic gradients at the ablation front (which are steeper in the 8-μm ablator), the length of time the rarefaction wave passes through the ablation front also contributes (i.e., transit time or Δt_{RW}). When the head of RW1 reaches the ablation front, the rarefaction wave is 20% thicker (larger separation between head and tail in RW1) in the 10-μm ablator, which would suggest a larger Δt_{RW}, but in actuality, feedout growth is less. In the 8-μm ablator, RW1 has more than twice as much time to travel across the ablation front. Refer to Table 2.1 for the comparison between the pulse and ablator configurations regarding the shock and rarefaction timing. After the second shock launches from the ablation front, distortion continues to increase. RW2 briefly creates opposing pressure and density gradients at the ablation front and creates further growth but such growth is insignificant.

Next, Fig. 2.22 plots the ablation-front distortion history from ice perturbations at $x_{\mathrm{p}} = -5$ μm for the same target and pulse configurations. The different pulse and ablator thicknesses primarily shift the timing of the distortion history with minimal change in seed amplitude. The thick ablator shows a slightly reduced seed amplitude (0.0280 μm) compared to the 8-μm ab-

Table 2.1: Rarefaction wave timing for each pulse and ablator thickness. The head of the rarefaction wave arrives at the ablation front at t_{RW}, the second shock is launched at t_{SW}, and the RW1 transit time is Δt_{RW}.

Case	t_{RW} (ns)	t_{SW} (ns)	Δt_{RW} (ps)
A + 8 μm	0.285	0.530	245
B + 8 μm	0.235	0.435	200
A + 10 μm	0.420	0.530	110

lator (0.0375 μm for A, and 0.0350 μm for B) because the perturbation has less time to evolve before the acceleration phase starts.

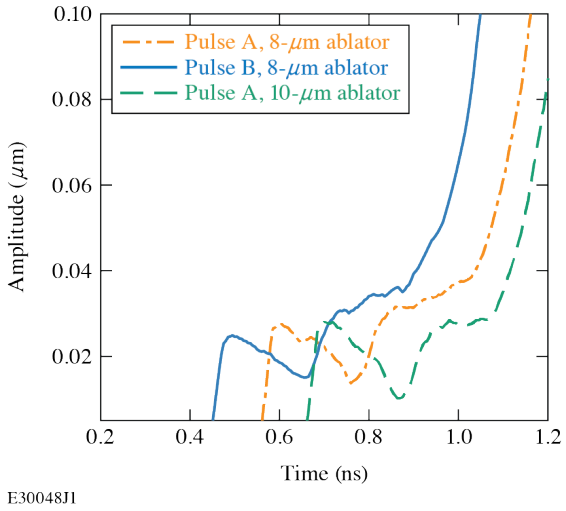

Figure 2.22: Ablation-front distortion history of a $\lambda = 40$ μm single-mode ice ($x_p = -5$ μm) perturbation for laser pulses A and B with shell thicknesses of 8 and 10 μm.

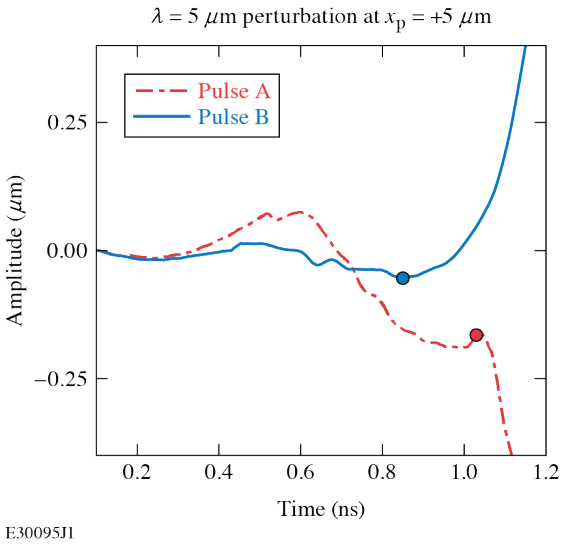

Figure 2.23: Ablation-front distortion history from a single-mode $\lambda = 5$ μm perturbation at $x_p = +5$ μm for the 8-μm ablator driven by pulses A and B. The start of the acceleration phase is indicated by the dot on each line at $t = 0.85$ ns and $t = 1.02$ ns for pulses A and B respectively. The ablator with pulse B experiences higher ablative stabilization due to the increased adiabat.

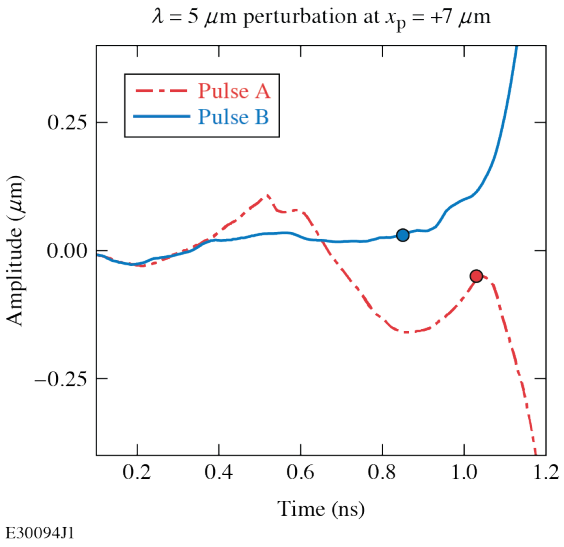

Figure 2.24: Ablation-front distortion history from a single-mode $\lambda = 5$ μm perturbation at $x_p = +7$ μm for the 8-μm ablator driven by pulses A and B. The start of the acceleration phase is indicated by the dot on each line at $t = 0.85$ ns and $t = 1.02$ ns for pulses A and B respectively. The ablator with pulse B experiences higher ablative stabilization due to the increased adiabat.

Figures 2.23 and 2.24 plot the ablation-front distortion history due to a short-scale $\lambda = 5$ μm perturbation at $x_p = +5$ μm and $x_p = +7$ μm in the 8-μm ablator. At this wavelength, phase oscillations, material-interface distortion, and ablative stabilization all influence the ablation-

front seed amplitudes. Pulse B increases ablative stabilization due to the higher adiabat; this reduces the amplitude of the ablation-front distortion oscillations. The start of the acceleration phase is indicated in each figure by the colored dots at $t = 0.85$ ns and $t = 1.02$ ns for pulses A and B respectively, and the seed amplitude (η_0) for the $x_{\rm p} = +5$ μm case with pulse B is 3$\times$ less compared to A.

To summarize the study of higher adiabat and thicker ablator designs, we conclude that density modulations localized in the ablator lead to larger seed amplitudes compared to modulations in the ice regardless of thickness or adiabat. For short-wavelength perturbations, increasing the adiabat (while fixing the ablator thickness) reduces both the seed amplitude and magnitude of the ablation-front distortion oscillations prior to the acceleration phase due to increased ablative stabilization. However, for long-wavelength perturbations, the stronger picket associated with the increased adiabat also increases secular feedout growth. Furthermore, by increasing the shell thickness, secular feedout growth is reduced by limiting the length of time the ablation-front experiences acceleration while RW1 crosses the ablation front.

2.3 Isolated Defects

While applying single-mode perturbations is useful in understanding fundamentals, a more-realistic representation of material defects, such as bubbles or voids, is necessary. These defects are represented in simulations as low-density Gaussian voids,

$$\eta(x, y) = A_\rho \exp\left[-\frac{(x - x_0)^2}{2\sigma^2} - \frac{(y - y_0)^2}{2\sigma^2}\right], \tag{2.2}$$

in the ablator and ice at similar locations to the single-mode cases previously discussed in Sec. 2.1. Here, σ is the variance that is related to the full width at half maximum as FWHM $= 2\sqrt{2\ln 2}\sigma$. The center of the void is at (x_0, y_0) and A_ρ is the amplitude of the void perturbation, which is a fraction of the density at (x_0, y_0). The defect FWHM's in simulations range from 1 to 4 μm with peak density reduction of 1% and 50% relative to the surrounding density. Each

simulation uses symmetry along $y = 0$ and a zero gradient boundary at $y = 60$ μm, and the isolated void is located at the $y = 0$ axis of symmetry. The boundary conditions at $\pm x$ are the same as in the single-mode simulations. Mesh cell spacing in x and y is 0.05 μm to provide sufficient resolution to capture the defect and resulting acoustic wave propagation. With this cell spacing, a 1-μm defect covers a region of approximately 720 cells.

2.3.1 Small-Amplitude Defects

Applying small-amplitude density reduction (1%) provides the opportunity to study the superposition of the many modes that contribute to the collective 2-D dynamics created by isolated voids, whereas large-scale nonlinear distortions from large-amplitude perturbations can mask wavelength-dependant behavior. These small-amplitude isolated defects show evolution similar to the results in Sec. 2.1, despite the differences between the broadband modal spectrum of a Gaussian and a single-sinusoidal mode.

Figures 2.25 and 2.26 show the outcome due to small-amplitude isolated defects at depths of $x_{\text{p}} = +5, -5$, and -15 μm. Part (a) in these figures plots the amplitude of each mode (via an FFT) of the ablation front, and (b) shows the shape of the ablation front (x versus y). Figure 2.25 shows this at the start of the acceleration phase ($t = 1.02$ ns) and Fig. 2.26 is after the shell has moved inward $\sim$90 μm ($t = 1.37$ ns). The spectrum for a Gaussian with a FWHM of 2 μm is included in both figures for reference.

At the beginning of the acceleration phase, the defects in the ablator layer have had time to expand their perturbations laterally ($y \approx \pm 12$ μm), as seen in Fig. 2.25 (b). The ablation front in the ice defect cases, however, shows minimal discernible distortion even though perturbations deposit seeds before the acceleration phase starts. In Fig. 2.26, perturbations from the ablator defect continue to expand, whereas perturbations from ice defects create localized growth extending to only $y \approx \pm 3$ μm. This difference is due to how the 2-D cylindrical sound waves (hypersurfaces originating from the defect) intersect with the ablation front, similar to $\widetilde{\text{C}}^{+}$ in 1-D. These intersections create modulations, or ripples, at the ablation front that will be amplified

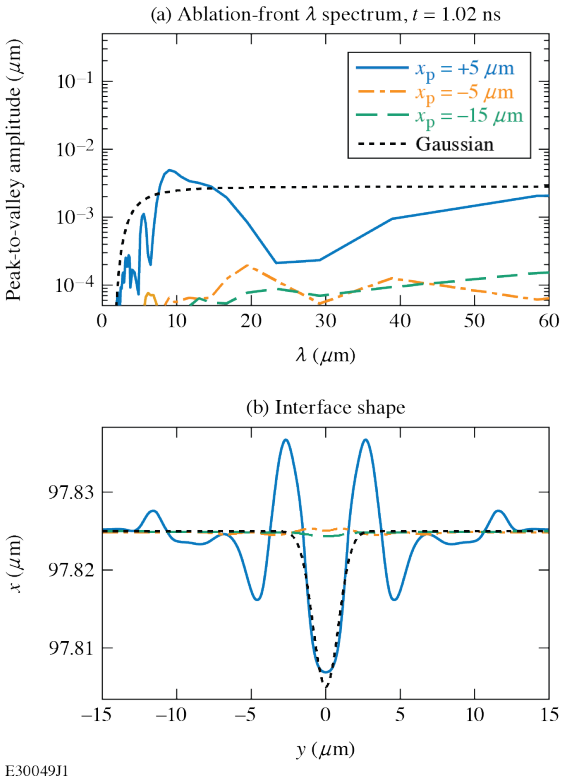

Figure 2.25: Ablation-front wavelength spectrum and shape for small-amplitude defects at the start of the acceleration phase ($t = 1.02$ ns). Defect starting depths (x_p) are relative to the material interface.

due to secular feedout growth from RW1 and RW2. The extent of the lateral expansion depends on both the evolution time and amount of characteristic wave reverberation between the various interfaces. Perturbations due to shallow defects (e.g., in the ablator) create more ripples at the ablation front compared to deep defects because the wavefronts intersect with the ablation front earlier (due to earlier shock interaction) and have longer time to evolve. Perturbations from the defect at $x_\mathrm{p} = +5$ μm have nearly 1 ns to propagate and reverberate before the acceleration phase starts. Although both ice defects have seeded the ablation before the acceleration phase, growth is minimal due to the lack of secular feedout amplification (as in single-mode simulations). Short-scale modes within the ripples are affected by ablative stabilization (see Figs. 2.14 and 2.15). Mid-scale modes ($10 < \lambda < 40$ μm) due to perturbations originating near the material interface (e.g., $x_\mathrm{p} = -5$ μm) are susceptible to limited growth from destructive wave interference (Fig. 2.10). Perturbations from ice defects experience localized growth from the interaction between the entropy wave (and shock-induced vorticity) and the ablation front

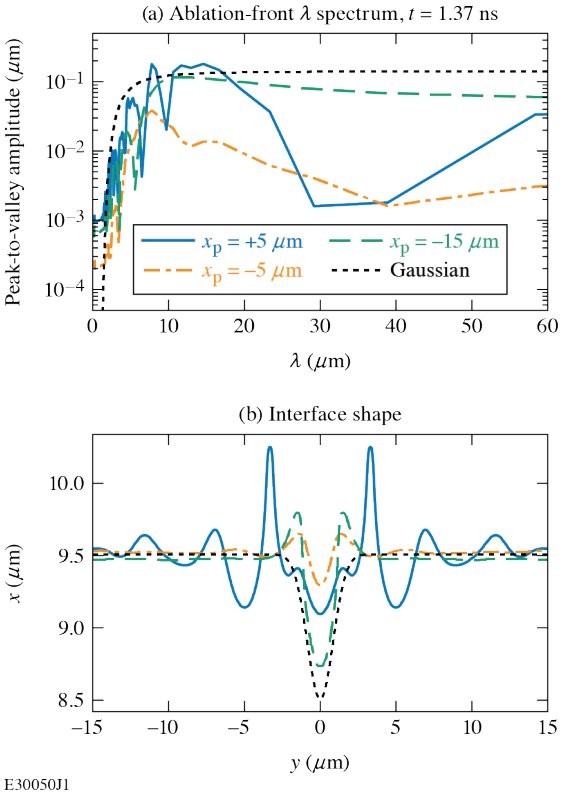

Figure 2.26: Ablation-front wavelength spectrum and shape for small-amplitude defects at the end of the simulation ($t = 1.37$ ns). Defect starting depths (x_p) are relative to the material interface.

(which overcomes short-scale ablative stabilization—see Fig. 2.16). In Fig. 2.26, the larger amplitude of the localized growth for $x_\mathrm{p} = -15$ μm (compared to $x_\mathrm{p} = -5$ μm) is attributed to the destructive wave interference that was also observed in the mid-scale single-mode results for $x_\mathrm{p} = -5$ μm (Fig. 2.10).

2.3.2 Large-Amplitude Defects

Small-amplitude defects help to inform wavelength dependant evolution without nonlinear growth, but large-amplitude (50% density reduction) isolated defects represent a more probable density distribution. This amplitude could be considered a conservative estimate since a helium-3 bubble or void inside the ice, an air-gap between the ablator and ice layers, or a defect inside the ablator material are all likely to have at least an order of magnitude lower density than the surrounding material. However, even defects with a 50% density perturbation have a considerable

impact on the target material, as seen in Fig. 2.27.

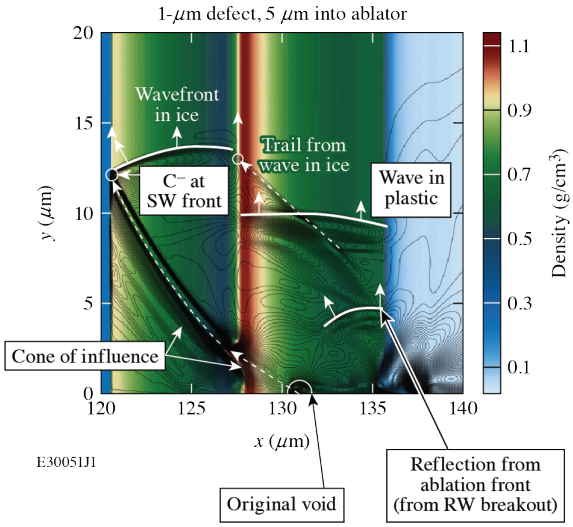

Figure 2.27: Wave evolution for an isolated ablator defect ($x_p = +5$ μm) at 0.5 ns after the first shock passage. Evolution in the y direction contributes to an extension of the maximum perturbation in y compared to the initial defect size. The contour colors show density and contour black lines show y velocity.

This figure shows the density contour of a target initiated with a 1-μm defect (FWHM) located in the ablator at $x_p = +5$ μm. The color corresponds to density and the black lines correspond to y velocity contours. The laser drive is applied from the right and the target accelerates from right to left. The axis of symmetry is at $y = 0$. This shows the target after the first shock has passed through the material interface and into the ice. The shock front is located near $x = 121$ μm and the material interface is near $x = 128$ μm. The 1-D entropy wave that originates with the void or defect is labeled in the figure just behind the material interface [near $(x, y) = (131, 0)$ μm]. In the figure, perturbation information can be seen propagating along various hypersurfaces, or ripples, after the shock has passed through the defect. When the shock interacts with the defect, the shock front becomes locally deformed. Since the perturbation wave has both x and y components, this deformation spreads laterally along the shock front, leaving a "trail" of vorticity in a cone-like manner, with its origin at the fluid trajectory of the original defect. The extent of this vorticity cone (labeled in the figure by the dashed white line) is determined by the material sound speed, shock strength, and defect placement. As the original perturbation cone from the first shock front expands into the target, new waves (such as rarefaction waves)

carry this updated information back to the ablation front and because of 2-D nature of the flow, laterally expand the distortion along the surface of the ablation front.

Lateral sound-wave propagation is shown in Fig. 2.27 by solid white lines. The layered target design complicates this wave propagation by introducing discontinuous jumps in the sound speed along material interfaces. For example, since the sound speed is higher in the ice, the lateral extension of the wave in ice outpaces the wavefront in the ablator and creates an additional trail behind it, similar to the one at the first shock front. As seen in Fig. 2.27, this trail crosses over the wavefront in the ablator material. This interface modulation will get carried back to the ablation front by subsequent rarefaction waves (e.g., after the second shock wave).

In single-mode simulations, the first rarefaction wave (RW1) moves through the ablation front and does not reflect back into the ablator since the ablation pressure is low (the pulse is off). Although the same pulse is used, the defect perturbations cause the ablation front to warp significantly before RW1 arrives (perturbation wavefronts arrive before the rarefaction wave because the defect is in the ablator). Because of this deformity, portions of the rarefaction wave reflect back into the target (primarily in the y direction), as shown by the curved bow wave labeled in the figure near the ablation front. However, reflections like this are small because the drive is off; any reflections occurring after the second shock will be much stronger since the drive pressure is higher.

The vorticity cone and lateral sound wave propagation (and reverberation) influence the level of ablation distortion (and shell degradation) that is amplified during the acceleration phase. This level of degradation is quantified using areal density (ρL) variation, which captures perturbation extent and amplitude and identifies shell weak spots or punctures. Areal density is defined as

$$\rho L = \int_0^L \rho \, \mathrm{d}x, \tag{2.3}$$

where ρ is the local density and L is the length of the grid in x. This is calculated along each row of cells in the simulation domain to obtain a $\rho L(y)$ distribution. The lateral extent (in microns) of

the perturbation is defined as the region where ρL deviates from its 1-D value [or the unperturbed $\rho L_{\text{1-D}} = \rho L(y = \text{max})$] by more than 1%. This is best seen in Fig. 2.28. The gray-shaded region in the figure shows the area flagged as having a larger than 1% variation from the 1-D ρL value. The lateral extent in this case is approximately 16 μm, and the areal density variation within this region is 18% of $\rho L_{\text{1-D}}$. The variance of the areal density is calculated inside the region where $|\rho L(y) - \rho L_{\text{1-D}}| > 0.01 \rho L_{\text{1-D}}$ as

$$\sigma_{\rho L} = \left[\frac{\sum_{j=1}^{N} (\rho L_j - \rho L_{\text{1-D}})^2}{N} \right]^{1/2}, \tag{2.4}$$

where N is the number of samples (i.e., N cells in the y direction), and ρL_j is the value along a single grid j line.

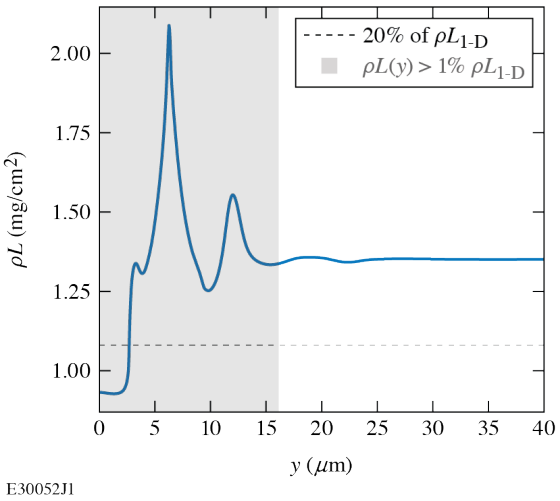

Figure 2.28: $\rho L(y)$ for a defect located at $x_{\text{p}} = -15$ μm (in the ice). The shaded region includes the values of ρL that are greater than 1% different than $\rho L_{\text{1-D}}$. The dashed line represents -20% $\rho L_{\text{1-D}}$.

Shell punctures or weak spots are defined as any region with ρL below 20% of $\rho L_{\text{1-D}}$. This is shown by the dashed horizontal line in Fig. 2.28 with a hole approximately 5 microns wide (accounting for the $y = 0$ axis of symmetry).

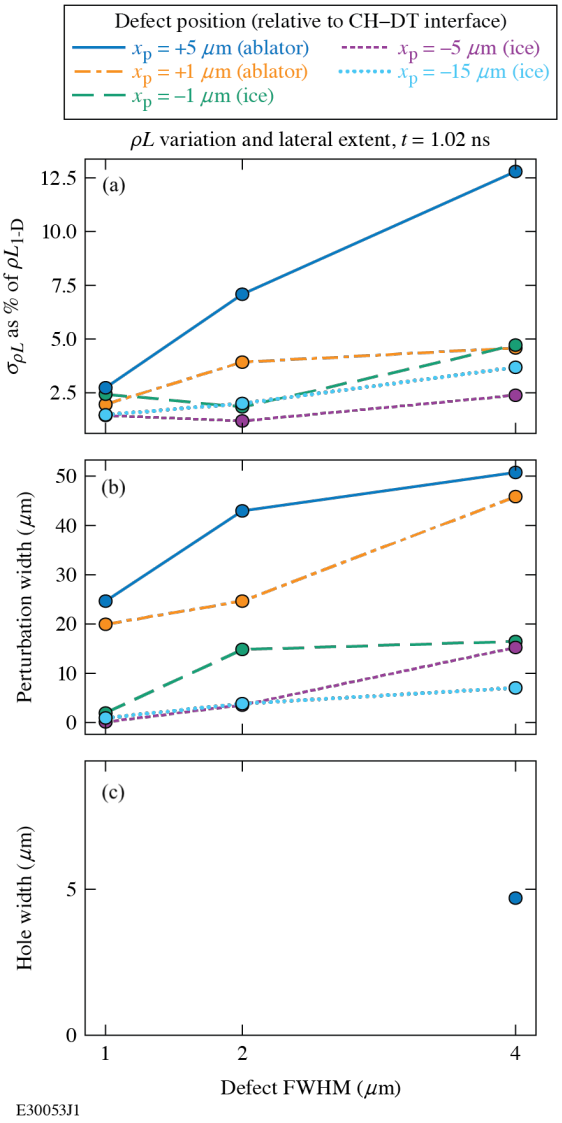

Figure 2.29: ρL variance and maximum lateral perturbation extent due to isolated defects of different sizes (FWHM) at the beginning of the acceleration phase ($t = 1.02$ ns).

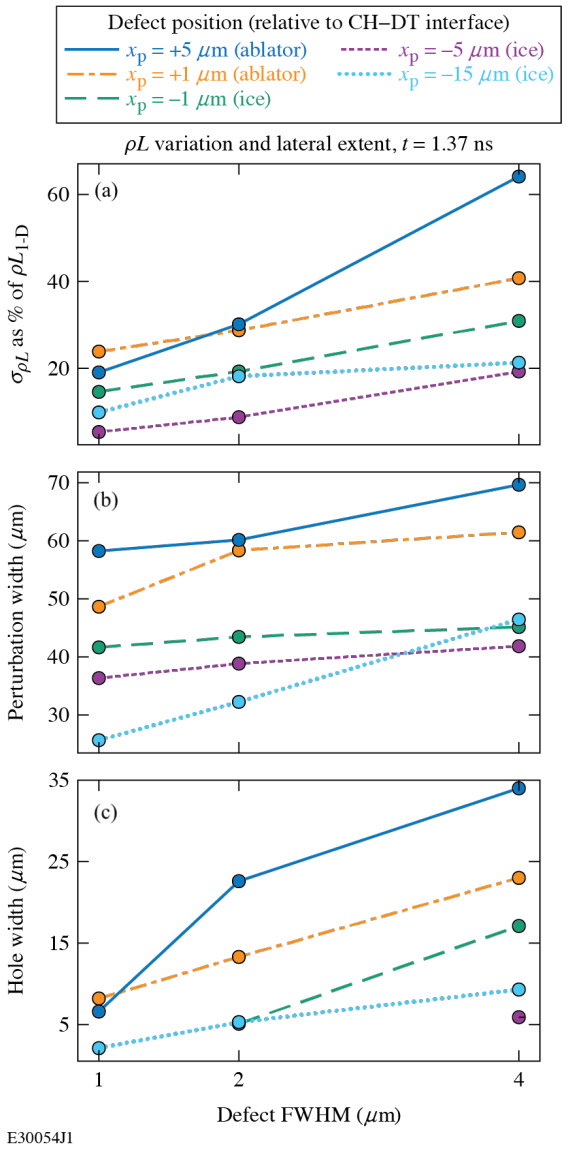

Figure 2.30: ρL variance and maximum lateral perturbation extent due to isolated defects of different sizes (FWHM) at end of the simulation ($t = 1.37$ ns).

Figures 2.29 and 2.30 summarize the impact of defects with the initial FWHMs of 1, 2, and 4 μm located at $x_p = +5, +1, -1, -5$, and -15 μm. The uppermost pane of both figures plots the variation of areal density within the perturbed shell region. The middle pane shows the maximum lateral extent or width of the areal density degradation created by the defect and the bottom pane is the size of the largest hole or weak spot in the shell. Figure 2.29 shows the areal density degradation at the start of the acceleration phase ($t = 1.02$ ns); most cases are less than 5% perturbed (with the exception of the outermost defect at $x_p = +5$ μm) even though the impact

of the defect is felt up to 50 μm away. Only the outermost 4-μm-wide defect at $x_{\mathrm{p}} = +5$ μm has weakened the shell enough to create a hole ($\sim$5 μm wide). The large lateral extent is due to the cone of influence expansion mechanism (described earlier) in conjunction with the reverberation inside the shell that moves the perturbation throughout the target. The outermost defects have the largest cones of influence and therefore the largest perturbation extents. Perturbations from ablator defects amplify due to feedout and expand the distortion region laterally before the second shock arrives. When the second shock launches, it encounters a much larger lateral extent of distortion at the ablation front and repeats the same process as the first shock.

Figure 2.30 shows the same defect cases 300 ps later in the simulation. At this point, every single defect has created significant areal density degradation. Nearly all of the defects have now weakened the shell enough to create holes at least as large as the original defect itself, and in some cases 5 to 10$\times$ larger. The extent, or width of perturbation at the ablation front, is roughly 25 μm (over 20$\times$ the size of the initial defect), and the areal density deviation has increased from an average of 2.5% up to 10 to 30% over a region 40 to 50 μm wide. Additionally, these results repeat two of the interesting findings from the single-mode and small-amplitude isolated defect simulations: (1) deep ice defects (e.g., $x_{\mathrm{p}} = -15$ μm) experience enhanced growth due to vorticity convection at the ablation front, and (2) perturbations near $x_{\mathrm{p}} = -5$ μm undergo destructive wave interference from the material interface, reducing perturbation growth. Enhanced growth for the defect at $x_{\mathrm{p}} = -15$ μm can be seen in Fig. 2.30 where it creates larger holes for all but one ice defect case (FWHM $= 4$ μm at $x_{\mathrm{p}} = +1$ μm). Destructive wave interference prevents the creation of holes in the case of the two smallest defects at $x_{\mathrm{p}} = -5$ μm, and although this defect depth still creates sizeable perturbation widths ($\sim$35 to 40 μm), the areal density variance is the lowest.

An example of the shell distortion created by a 2-μm-wide defect initially located at $x_{\mathrm{p}} = +1$ μm can be seen in Fig. 2.31. This defect creates a 13-μm-wide hole in the shell at $t = 1.37$ ns and expands its perturbations to a region $\sim$60 μm wide with an overall areal density-variation of up to 30% of 1-D. This is a significant source of implosion performance degradation that has the

potential to reduce compression.

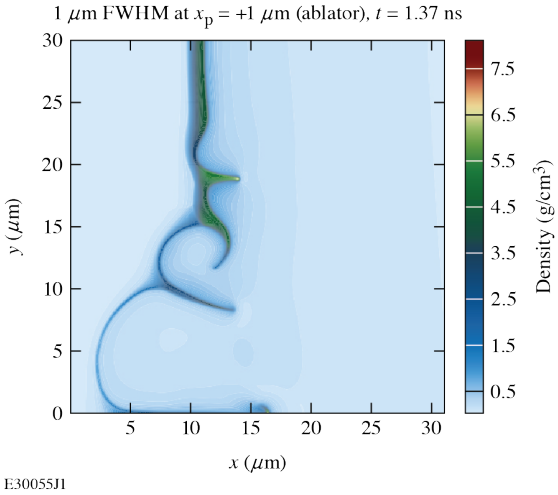

Figure 2.31: Density contours of the shell showing degradation effects due to a 2-μm isolated defect at $x_p = 1$ μm. A puncture of approximately 13 μm in size (2$\times$ what is shown due to symmetry about $y = 0$) has been created because of the defect.

2.4 A strategy to reduce defect degradation effects

Due to the potential for significant degradation created by isolated defects, strategies for reducing this effect are highly desirable. Both single-mode and isolated defect simulations show that ablator material defects create larger instability seeds and more areal density degradation compared to defects in the ice, so target designs should address this issue. Ablator defects are more dangerous, primarily due to their larger lateral expansion as described earlier. The other major contribution to perturbation amplification arises from perturbation feedout when the pressure gradients due to rarefaction waves cross the ablation front and amplify distortion.

A robust implosion design should seek to eliminate or reduce growth amplification from feedout. One potential strategy for growth mitigation is reducing the ablator density, thereby eliminating the rarefaction wave (RW1). To accomplish this, we present a surrogate for a wetted-foam-like ablator material, where the density is reduced from 1.0 g/cm^3 (for CH) to 0.3 g/cm^3 and the shell thickness is increased to 26.7 μm (up from 8 μm) to conserve total shell mass compared to the CH ablator design. The same low-adiabat laser (pulse A in Fig. 2.19) is used. This particular

target design is simplified to show that modifying the density of the ablator can help reduce the instability seeds at the ablation front by changing basic hydrodynamic wave evolution inside of the shell. Targets that utilize foam ablator materials have a long history [2]. The utilization of foam is complex for a variety of reasons including (but not limited to) manufacturing challenges, internal structures of the foam layer that could be a source of additional perturbation seeding, and modeling uncertainties, all of which are beyond the scope of this paper. This example does not seek to provide an optimal design with every detail included, but rather a starting point to demonstrate the effect of ablator density that could form the basis for future work.

In a manner similar to the CH ablator designs, 10% density single-mode perturbations are applied with a wavelength of $\lambda = 40$ μm at various depths in the target (in both the ice and ablator). Figure 2.32 compares the ablation-front distortion history for CH and foam ablators with this perturbation wavelength for a small subset of depths. Since the ablator thickness varies between the two designs, direct comparison between ablator materials and perturbations locations is nuanced. The $\widetilde{\mathrm{C}}^{+}$ wave that carries the first instance of perturbation to the ablation front arrives later in the foam ablator than in the CH ablator. Directly comparing the distortion history from a perturbation at $x_{\mathrm{p}} = +7$ μm in the foam versus the CH ablator reveals that the onset of distortion is delayed nearly 400 ps in the foam. The $\widetilde{\mathrm{C}}^{+}$ wave arrival timing is only matched when the perturbation originates at $x_{\mathrm{p}} = +14$ μm in foam and +1 μm in CH. Adjusting the perturbation origin, therefore, gives a better direct comparison of perturbation evolution than fixing the depth of the perturbation.

As expected, this direct comparison highlights the lack of secular feedout growth experienced by the foam ablator. The foremost reason for reducing the density of the ablator is to eliminate the feedout growth caused by the rarefaction wave interacting with the ablation front, a difference that is clearly seen in the CH and foam ablator between 0.3 and 0.5 ns. In fact, because of the thicker ablator, this rarefaction wave does not reach the ablation front before the second shock is launched. This can be seen in Fig. 2.33, which shows the pressure and density profiles for the two ablator types [CH is shown in (a) and foam in (b)] with labels for the material interface

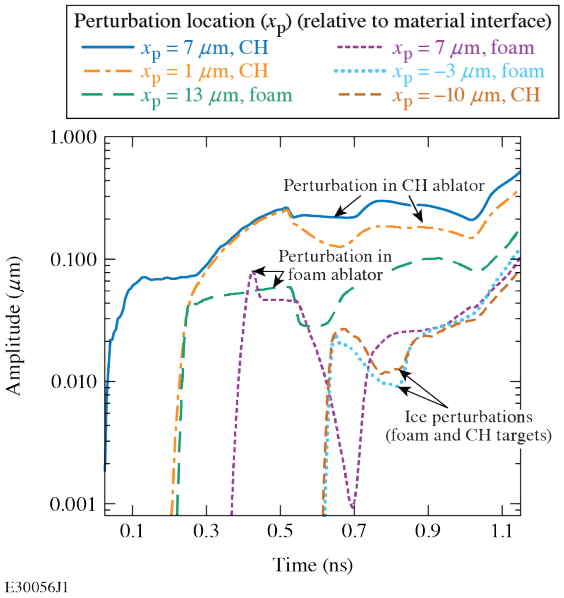

Figure 2.32: Time history of the amplitude of ablation-front distortion due to single-mode perturbations with $\lambda = 40$ μm in targets using CH and foam ablators. Perturbation origin depth (x_p) is reported as relative to the material interface ($+ \rightarrow$ in ablator, $- \rightarrow$ in ice).

and head and tail of the rarefaction wave. The profile snapshot for the CH design is shown at the time when RW1 starts to cross the ablation front. The foam design is shown just before the second shock launches into the ablator, and the rarefaction head has not reached the ablation front. However, even in the case when this rarefaction wave breaks out at the ablation front, the perturbation growth in the foam design is insignificant because of smaller pressure gradients. In test simulations where the foam rarefaction wave is allowed to cross the ablation front (no second shock wave), feedout growth amplification never occurs. Additionally, because the foam ablator is less dense and the ablation velocity is higher compared to the CH ablator, short-wavelength perturbations will benefit from the enhanced ablative stabilization during the shock propagation phase of the implosion.

For perturbations that originate in the ice, there is very little difference between the foam and CH ablator designs. Figure 2.32 shows a perturbation at $x_p = -3$ and -10 μm for the foam and CH targets respectively, which are chosen so that the seeding by the $\widetilde{C}^+$ wave is matched due to the change in ablator thickness and density. These two ice perturbation configurations show nearly identical distortion evolution due to the same ice layer dimensions and pulse selection.

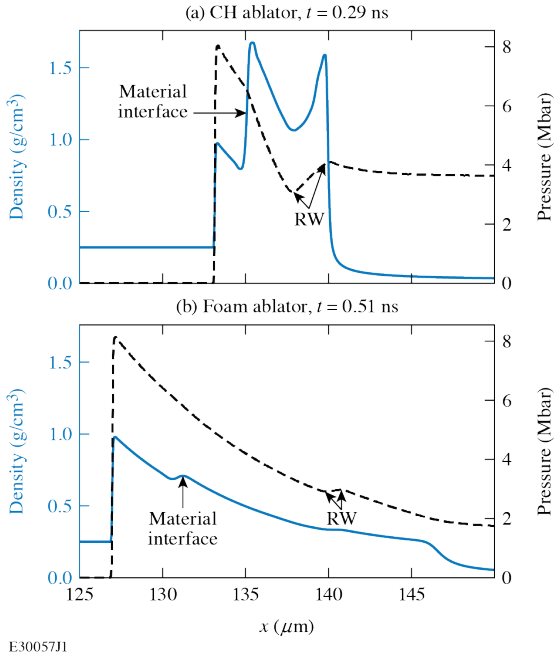

Figure 2.33: Pressure (black lines) and density (blue lines) profiles of the (a) CH and (b) foam ablators with the rarefaction wave (RW) near the ablation front. The RW in the foam ablator does not reach the ablation front or create secular growth before the second shock launches (at $t = 0.52$ ns).

In isolated defect simulations, using a low-density foam ablator also helps to reduce areal density modulation. A thicker ablator is more robust against punctures and increases the time it takes for the characteristic waves to reverberate inside the shell. By increasing the thickness by a factor of $\sim$3, the foam ablator allows the initial perturbation wave to expand outward (this process is the same in either ablator) without the influence of new reflected waves from either the material interface or ablation front. Figure 2.34 compares the areal density degradation in the CH and foam ablators due to an isolated defect $\sim$300 ps after the start of the acceleration phase. Because Fig. 2.30 showed that in general, larger defects increase the level of degradation overall, when comparing foam and CH, the defect size was fixed at 1 μm and only the starting depth was changed. In Fig. 2.34(a), areal density variation is fairly constant for the foam ablator, whereas the CH ablator ranges from 5 to 25% depending on the depth. The ablator defects in the foam target have smaller seed amplitudes due to the reduction in secular feedout growth, indicated by the lower areal density variation. The perturbation width trend is similar for both ablators since this is largely a function of defect placement [Fig. 2.34(b)], and, as shown in Fig. 2.34(c), there

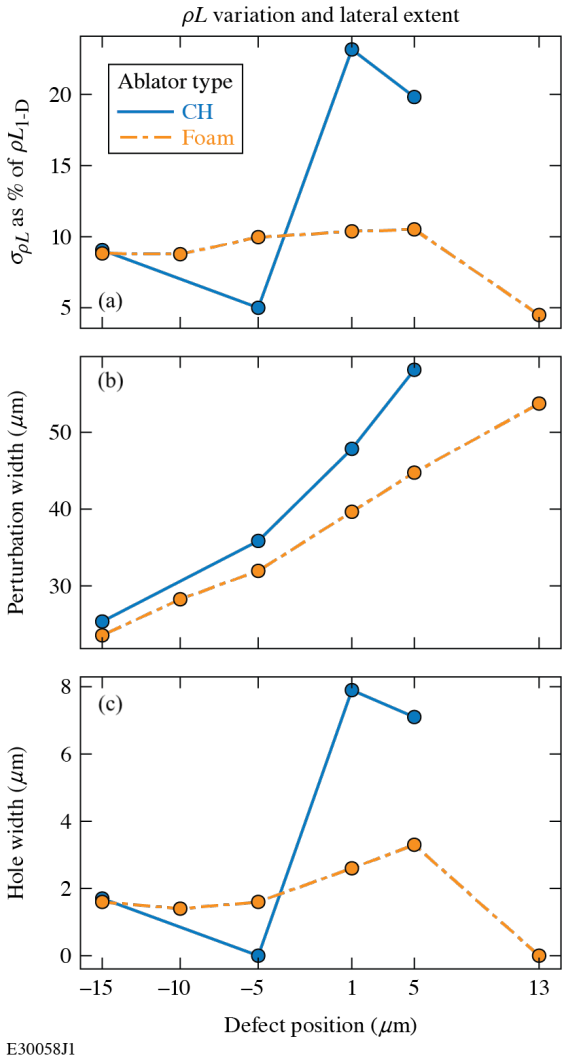

Figure 2.34: Areal density modulation due to 1-μm defects at various depths in foam and CH ablators.

is much less variation in shell puncture sizes in the foam versus CH. The outermost foam defect at $x_{\mathrm{p}} = +13$ μm does not create a hole, whereas all defects in CH ablators create sizeable holes. The defect at $x_{\mathrm{p}} = +13$ μm in the foam has the same $\widetilde{\mathrm{C}}^{+}$ timing as the $x_{\mathrm{p}} = +1$ μm defect in the CH ablator, so the ablation front is seeded at the same time in the implosion.

Figure 2.35 compares density contours for a CH and foam ablator target with a 1 μm defect at $x_{\mathrm{p}} = +1$ μm. The CH ablator has two primary puncture regions due to the defect and characteristic wave propagation. This can be explained as such (1) the larger of the two "holes" near $y = 0$ develops as a result of the picket shock and the starting location of the defect—the first shock encounters the defect at this lateral position, and (2) the second shock encounters the lat-

erally expanded distortion at the ablation front and creates the seed for the smaller hole to form near $y = -10$ μm. The lateral placement where the shock interacts with the defect determines the position of the primary seed for growth during the acceleration phase.

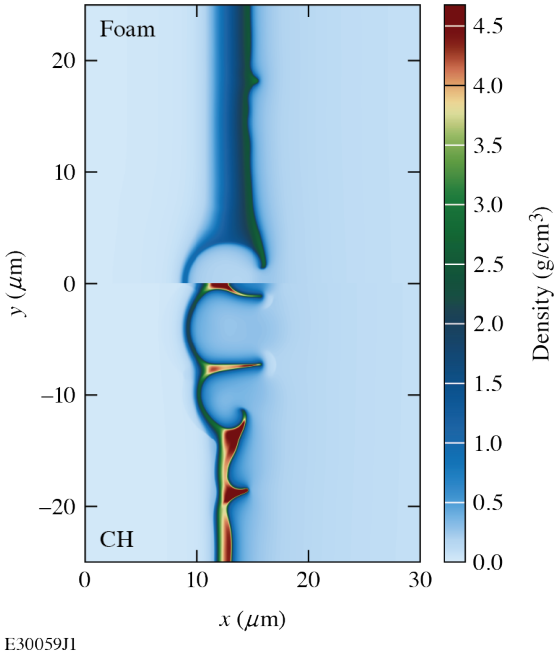

Figure 2.35: Shell density contours of the foam (top) and CH (bottom) ablator designs for a 1-μm defect at $x_p = +1$ μm. The foam ablator time is delayed by 30 ps (from $t = 1.350$ to 1.365 ns) to align with the CH ablator position.

In the foam ablator, the perturbation is localized near the position of the original defect along the $y = 0$ axis of symmetry. This defect is more isolated compared to the CH defect due to the thicker ablator. This exhibits behavior similar to the deep ice defects in the CH ablator designs, where growth is more localized and isolated by the late shock interaction compared to the ablator defects. Because of this, distortion only occurs near the position of the entropy wave. The bubble in the foam and largest bubble in the CH are comparable in size. The perturbation feature at the ablation front of the foam target near $y = 18$ μm is a result of the interaction between the second shock and the characteristic wavefront. Again, because the foam ablator is thicker, the defect characteristics can expand for a longer period without reverberation from rarefaction and other waves. In this case, when the second shock hits the ablation front, it only encounters significant distortion at the point of intersection of the characteristic wavefront and

ablation front since the rarefaction wave does not exit the ablator prior to the second shock's arrival. Additionally, the low-density foam has a higher ablation velocity compared to the CH ablator and benefits from increased stabilization, thereby reducing the the short-scale growth along the ablation front that is more readily seen in the CH ablator. Figure 2.36 compares the foam and CH ablators when the arrival timing of the $\widetilde{C}^{+}$ wave at the ablation front is matched. This corresponds to a defect location of $x_p = +13$ μm in foam and $x_p = +1$ μm in CH, as in the single-mode results shown in Fig. 2.32. This shows the foam target with considerably less deformation than the CH target.

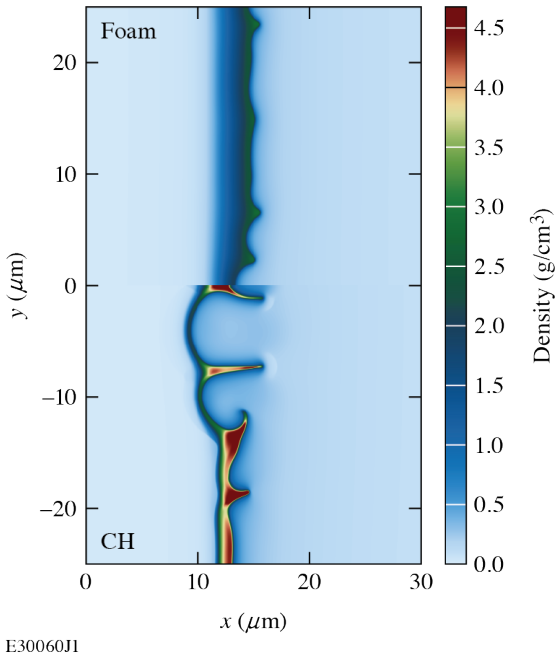

Figure 2.36: Shell density contours of the foam (top) and CH (bottom) ablator designs for a 1-μm defect at $x_p = +13$ μm and $x_p = +1$ μm, respectively, to align the C^{+} seed timing at the ablation front. The foam ablator time is delayed by 30 ps (from $t = 1.350$ to 1.365 ns) to align the with the CH ablator position.

2.5 Discussion and Conclusion

The inherent susceptibility of ICF implosions to hydrodynamic instability growth makes understanding seeding mechanisms a fundamental goal. This work seeks to understand the overall trends created by internal perturbations from sources such as target imperfections and fuel and ablator radiation damage.

Perturbations originating in the ablator material have been shown to create significantly higher seeds for instability growth due to a combination of position, timing, and 1-D hydrodynamic wave evolution. In general, the deeper the defect or perturbation is relative to the outer surface, the smaller the overall degradation effect becomes, although there are some outlier effects. Long-wavelength, single-mode simulations show that the rarefaction wave crossing over the ablation front after the first unsupported shock wave creates a large increase in the seed amplitude due to feedout created by hydrodynamic gradients. Short-wavelength simulations show that the interaction with the distorted material interface and phase oscillation of the ablation-front distortion create situations where shell thickness and perturbation placement can reduce or delay the onset of exponential growth. Results from short-scale perturbations buried deep in the ice reveal that vorticity convection to the ablation front can counteract the effects of ablative stabilization, thereby significantly enhancing the growth of perturbation wavelengths stabilized by ablation. Trends observed in single-mode simulations also apply to the evolution of isolated defects.

Isolated defect simulations of both small (1%) and large (50%) density perturbations reveal complex wave dynamics inside the target due to shock timing, defect placement, characteristic wave propagation at disparate sound speeds, and reverberation within the shell that expands the influence of these perturbations. These results support the findings from single-mode simulations that ablator layer perturbations create significantly larger seeds for instability growth. Modulations in areal density show that certain defect sizes and locations create distortions large enough to weaken, or even puncture, the shell shortly after the acceleration phase begins. Isolated defects buried deep within the ice also repeat the phenomenon where shock-induced vorticity interaction with the RT modes at the ablation front can overcome ablative stabilization effects. This surprising result shows that certain deep ice defects can be more detrimental than those close to the material interface.

By understanding the seeding mechanisms present for both single-mode perturbations and isolated defects, we propose a potential mitigation strategy. This strategy seeks to eliminate or reduce the secular growth (feedout) at the ablation front after the first shock and limit character-

istic wave reverberation within the shell with a thicker, low-density, wetted foam-like ablator. By conserving shell mass, a foam-like ablator with the same laser pulse is shown to reduce areal density degradation at the start of the acceleration phase compared to a CH ablator design.

Future work will examine the same internal defect evolution in 3-D as well as including additional effects like convergent geometry, radiation transport, and a more-detailed treatment of the materials (material-specific equations of state, radiation opacity, multiple materials, etc.). Additional work will seek to optimize shell thickness and continue to study alternative foam-like ablator designs.

Chapter 3

Fuel-Shell Interface Stability During the Deceleration Phase in Room Temperature Implosions

Portions of this chapter are reproduced from [**Miller, S. C.**, Knauer, J. P., Forrest, C. J., Glebov, V. Yu., Radha, P. B. and Goncharov, V. N., "Fuel-Shell Interface Instability Growth Effects on the Performance of Room Temperature Direct-Drive Implosions", *Physics of Plasmas* 26, 082701 (2019)], with the permission of AIP Publishing.

This chapter studies target performance sensitivity to different Atwood numbers (Eq. 3.1) at the fuel-shell interface in warm implosions on OMEGA.

$$A_T = \frac{\rho_{\text{heavy}} - \rho_{\text{light}}}{\rho_{\text{heavy}} + \rho_{\text{light}}} \tag{3.1}$$

Continuity in pressure and temperature across the interface suggests a density discontinuity that depends on the ratio of atomic weight to the average ion charge. Therefore, changing the ratio of deuterium to tritium (D:T) in the fuel changes ρ_{heavy} and ρ_{light}, and, as a result, the Atwood number. As described earlier, varying the Atwood number is expected to change perturbation

growth factors during shell deceleration.

Changes in perturbation growth can be investigated by selecting D:T ratios of 10:90, 25:75, and 50:50 to create *stable*, *neutrally stable*, and *unstable* conditions, respectively, during the deceleration phase. An experimental campaign at the OMEGA [52] laser system systematically varied this D:T ratio in the fuel, for the same target dimensions and laser pulse shape, to verify the effect of the density jump across the material interface on target performance.

This chapter first discusses methods for calculating the Atwood number at the material interface and its implications for classical RT instability growth in a purely hydrodynamic scenario in Sec. 3.1. Sec. 3.2 discusses the localized, perturbation-specific, effective Atwood number and RT growth factors within the context of ICF implosions — specifically the dynamics of the material interface during the deceleration phase. Finally, results from the experimental campaign and their comparison to simulations are presented in Sec. 3.3.

3.1 Stability of the Fuel-Shell Interface

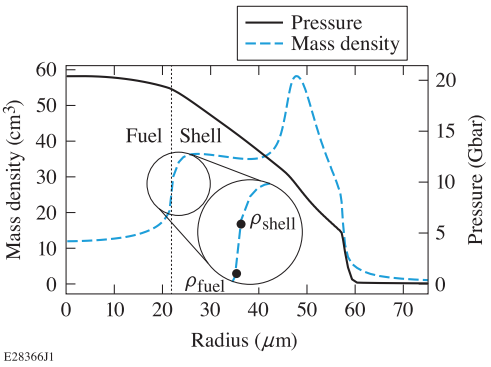

Figure 3.1: The fuel-shell interface of room temperature targets during the deceleration phase is classically unstable due to the jump in density. The simulated 1-D hydrodynamic profile during the deceleration phase at peak compression is shown above.

During the deceleration phase, opposing pressure and density gradients form at the material interface between the fuel vapor and shell (see Fig. 3.1). The figure shows mass density and pressure profiles of a room-temperature target at peak compression, and the inset highlights

the unstable material interface. Here the fuel-shell interface is unstable and susceptible to RT growth since the lower density fuel vapor is pushing on the higher density shell.

Any small interface perturbation amplitude (η) will grow exponentially in time as $\eta \sim \eta_0 e^{\gamma t}$, where $\gamma = \sqrt{A_T k g}$ is the growth rate, k is the perturbation wave number, and g is acceleration. While the Atwood number can be calculated directly from simulated hydrodynamic profiles, it is useful to estimate the value through material properties, particularly since the experimental density quantities are not readily available. Since the pressure is continuous across the interface, we can set the conditions on either side of the interface to equality and obtain a density ratio based solely on ion mass and charge. First, by assuming ideal gas, the total pressure of each material can be found as $P_{\text{total}} = n_{\text{i}} T_{\text{i}} + n_{\text{e}} T_{\text{e}}$, where n is the number density of ions and electrons and T is the temperature. Second, because of thermal conduction, temperature is continuous across the interface ($T_{\text{fuel}} = T_{\text{shell}}$). Note that these relations only hold directly on either side of the interface; as you move further away from the interface, $T_{\text{fuel}} \neq T_{\text{shell}}$. Within each material, we can assume that $T_{\text{i}} \approx T_{\text{e}}$, so $P_{\text{total}} = \frac{\rho}{m_{\text{i}}} T_{\text{i}} (1 + Z)$, since $n_{\text{i}} = Z n_{\text{e}}$ and $\rho = m_{\text{i}} n_{\text{i}}$. Then, only the mass density (ρ) and charge (Z) differ across the interface, so that the relation across the interface becomes:

$$\frac{\rho_{\text{fuel}}}{m_{\text{fuel}}}(1 + Z_{\text{fuel}}) = \frac{\rho_{\text{shell}}}{m_{\text{shell}}}(1 + Z_{\text{shell}}) \tag{3.2}$$

This is then simplified to obtain the $\rho_{\text{shell}}/\rho_{\text{fuel}}$ ratio needed to find the Atwood number:

$$\frac{\rho_{\text{shell}}}{\rho_{\text{fuel}}} = \frac{m_i^{\text{shell}}}{m_i^{\text{fuel}}} \frac{1 + Z_{\text{fuel}}}{1 + Z_{\text{shell}}} \tag{3.3}$$

Equation 3.3 tells us, as a baseline approximation, that the stability of the interface can be altered by changing the ion mass of the fuel. Through this method, the interface Atwood number $A_{T,\text{i}}$ is found to be stable for 10:90 ($A_{T,\text{i}} = -0.03$), neutrally stable for 25:75 ($A_{T,\text{i}} \approx 0.0$), and unstable for 50:50 ($A_{T,\text{i}} = 0.05$). This baseline approximation is valuable for stability analysis of the interface, but it still needs to be verified by measuring hydrodynamic profiles.

The perturbation growth factors of the interface for each D:T ratio can be analyzed with a basic

model that assumes a sharp interface [21]:

$$\frac{\ell}{\ell+1}\rho_{\text{shell}}\frac{d}{dt}\left(\frac{1}{\rho_{\text{shell}}r_{\text{i}}}\frac{d}{dt}\left(\rho_{\text{shell}}r_{\text{i}}^2\eta\right)\right)+ \rho_{\text{fuel}}\frac{d}{dt}\left(\frac{1}{\rho_{\text{fuel}}r_{\text{i}}}\frac{d}{dt}\left(\rho_{\text{fuel}}r_{\text{i}}^2\eta\right)\right) - l(\rho_{\text{shell}}-\rho_{\text{fuel}})\ddot{r}_{\text{i}}\eta = 0 \tag{3.4}$$

This model tracks the location of the interface (r_{i}) and the dynamics of the fuel and shell densities ($\rho_{\text{fuel}}, \rho_{\text{shell}}$) throughout the deceleration phase for a specific interfacial perturbation (η) and mode (ℓ). Time-dependant profiles from 1-D simulations of the deceleration phase provide the hydrodynamics of the interface needed by the model. The model is then solved for small-amplitude [$\dot{\eta}(0) \simeq 0.1,\ \eta(0) = 0$], incompressible velocity perturbations applied with different unstable ℓ modes. Figure 3.2 shows the instability growth of an unstable $\ell = 40$ perturbation for 10:90 versus 50:50 in simulations that have no radiation transport enabled (disabling radiation transport equates to a purely hydrodynamic scenario). As expected, 10:90 exhibits a stable interface, due to a negative $A_{T,\text{i}}$, whereas the perturbation of the 50:50 interface grows exponentially.

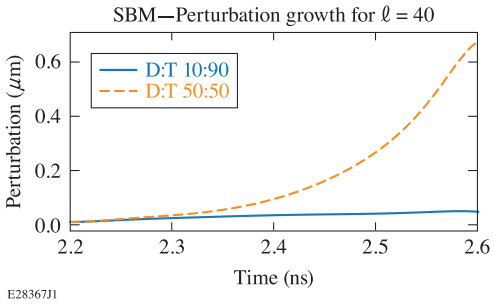

Figure 3.2: Interfacial perturbation growth for an unstable $\ell = 40$ mode, predicted by the sharp-boundary model (SBM) in Eq. 3.4, from 1-D hydrodynamic profiles during the deceleration phase.

The Atwood numbers obtained directly from the hydrodynamic simulations, $A_{T,\text{hydro}}$, are shown as a function of time in Fig. 3.3 as blue lines for 10:90 (solid line) and 50:50 (dashed line). The material interface Atwood numbers ($A_{T,\text{i}}$) are shown in orange. While the initial approximation worked to inform intuition on the difference in stability between the D:T ratios, it fails to recover the $A_{T,\text{hydro}}$ at the interface. This discrepancy is due to the initial assumption of ideal gas for the

equation of state (EOS) for each material.

Simulations confirm that the interface is in hydrodynamic equilibrium, e.g., pressure and temperature are continuous, but the total pressure in the shell under-predicts compared to the simulation, so an EOS correction factor is required. All the simulations in this work use a first-principles–based tabular EOS (FPEOS) [53], so a correction factor $f_{\text{EOS}} = (P_{\text{ideal}}/P_{\text{sim}})|_{\text{shell}}$ is used to recover the change in pressure in the shell. No correction factor was needed for the gas since the ideal gas approximation recovered the pressure. Now the interface relation takes the following form:

$$\frac{\rho_{\text{shell}}}{\rho_{\text{fuel}}} = \frac{m_{\text{i}}^{\text{shell}}}{m_{\text{i}}^{\text{fuel}}} \left(\frac{1 + Z_{\text{fuel}}}{1 + Z_{\text{shell}}} \right) f_{\text{EOS}} \tag{3.5}$$

Figure 3.3 shows that this new method $A_{T,\text{i}+f_{\text{EOS}}}$ (green) shifts upward and closer to the $A_{T,\text{hydro}}$ prediction by simulations that include all these effects self-consistently, compared to $A_{T,\text{i}}$ (orange). It must be emphasized that even though the ideal gas assumption does not directly recover $A_{T,\text{hydro}}$, it still captures the stability trend going from 10:90 to 50:50.

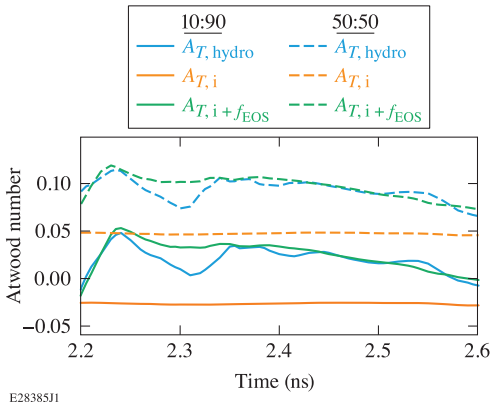

Figure 3.3: Different methods for calculating the Atwood number of the material interface. The different gas compositions are indicated by line styles (dashed = 50:50, solid = 10:90). The blue lines are the Atwood numbers calculated directly from mass density values across the interface. The orange lines are the Atwood numbers found purely from ion mass and charge. The green lines show the improvement in recovering the density-based Atwood number due to the addition of varying ion temperature within the material and equation-of-state correction factor.

3.1.1 Effective Atwood Number

Any perturbation modes present at the material interface will grow during the deceleration phase, as long as $A_{T,\mathrm{i}}$ is positive. Linear stability analysis of RT instability growth in semi-infinite density profiles has shown that these unstable modes are local to the interface, and that within the linear growth regime, the size of the unstable region is proportional to the perturbation wavelength [33]. The amplitude of the perturbation is highest on the interface itself and decays exponentially as the distance from the interface increases. In planar geometry, this decays according to wave number (k) and distance (y) as e^{-ky}. In spherical geometry the velocity perturbations decay as $(r_i/r)^{\ell+2}$ for $r > r_i$ and $(r/r_i)^{\ell-1}$ for $r < r_i$, for radial distance (r) from the material interface (r_i) and mode number (ℓ) . The *effective* Atwood number, defined as $A_T = (\rho^+ - \rho^-)/(\rho^+ + \rho^-)$, uses the mass density averaged over the perturbation region ($\pm r_i/\ell$). Figure 3.4 illustrates a mass density profile with unstable regions for an $\ell = 4$ and $\ell = 40$ perturbation. The Atwood number of the material interface only captures the density jump directly

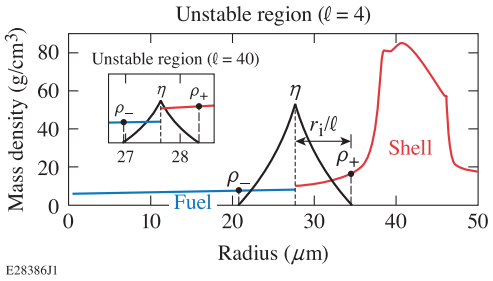

Figure 3.4: The effective Atwood number incorporates the averaged density profile within the mode-specific unstable regions. The unstable regions for both an $\ell = 4$ and $\ell = 40$ perturbation (η) are shown. The $(r_i/r)^{\ell}$ decay function is shown by the solid black line with the approximated $\rho^{\pm} \backsimeq \rho(r_i \pm r_i/\ell)$ values.

across the interface, whereas the effective Atwood number incorporates the average density over the perturbation region, which can be distinctly different than the interface. This is particularly important in room temperature ICF implosions, where radiation preheat significantly influences the inner shell region near the material interface.

ICF targets experience perturbations from a variety of different sources, each with their own unstable mode spectrum. These include low-mode ($\ell < 10$) asymmetries created by target offset

($\ell \approx 1$), laser geometry ($\ell \approx 10$ for OMEGA) and beam power imbalance ($\ell < 5$), and short-scale, $\ell > 10$ perturbations seeded by laser imprint and surface roughness [42, 54, 55, 56]. All of these unstable modes are present during the implosion, and each one grows at a different rate and scale-length, making the effective Atwood number particularly useful in identifying the level of instability that each mode experiences. Table 3.1 shows the Atwood number calculated for D:T 10:90 and 50:50 at the interface, which corresponds to $\ell = \infty$, as well as at $\ell = 4, 40, 200$. When the effective Atwood number is used in the sharp-boundary model mentioned previously, the 10:90 target now experiences half the growth of 50:50 for the same $\ell = 40$ perturbation on the interface (see Fig. 3.5). Compare this to the previous result which only used the material interface $A_{T,\mathrm{i}}$ in which the D:T 10:90 had a stable interface.

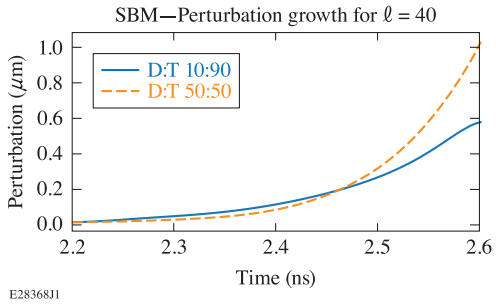

Figure 3.5: Sharp-boundary model results for an unstable $\ell = 40$ mode using the effective Atwood number. The model uses mass densities $\rho(r)$ taken at radial distances $r = \pm r_i/\ell$ from the material interface (r_i) for each given mode number (ℓ).

Table 3.1: $A_{T,\mathrm{i}}$ is the classical Atwood number of the interface found as a function of material properties. $A_{T,\mathrm{hydro}}$ is the Atwood number found as a function of mass density; both the interface ($\ell = \infty$) and effective $\ell = 4, 40, 200$ versions are shown. As ℓ increases, the Atwood number asymptotes to the interfacial value. All $A_{T,\mathrm{hydro}}$ values are taken from no-radiation (pure hydro) simulations at peak-compression.

		$A_{T,\mathrm{hydro}}$			
D:T	$A_{T,\mathrm{i}}$	$\ell = \infty$	$\ell = 4$	$\ell = 40$	$\ell = 200$
10:90	-0.03	0.023	0.15	0.025	0.023
25:75	0.00	0.044	0.17	0.047	0.044
50:50	0.05	0.087	0.21	0.090	0.087

3.2 Deceleration-Phase Rayleigh–Taylor Growth

The material interface stability discussed in the previous section was shown for a purely hydrodynamic scenario, but in ICF implosions, radiation transport is a significant effect that cannot be neglected. Early in time, radiation from the coronal plasma causes the shell to relax in density and thicken, compared to a simulation without radiation transport included. While radiation

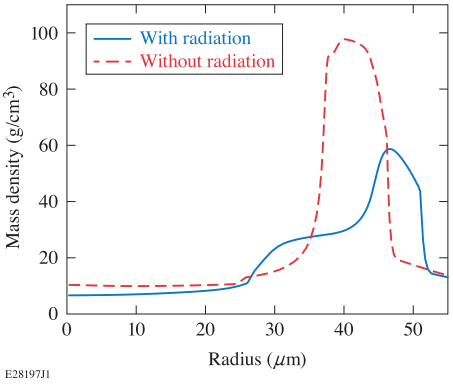

Figure 3.6: 1-D mass density profiles at peak neutron production with radiation transport turned on (solid) and off (dashed).

transport changes the shell dynamics, the material interface is relatively unchanged from coronal radiation early in time. However, late in time during the deceleration phase, the hotspot in the central fuel region is significantly heated up by compression and x-ray radiation is released in the DT fuel and subsequently absorbed into the colder CH shell. The absorption of x-ray energy into the shell causes the material to heat up and expand inward, resulting in a preheat region. This creates a thicker shell with increased density and $A_{T,\text{hydro}}$ near the material interface. This effect is shown in Fig. 3.6, which compares 1-D mass density profiles of two simulations with and without radiation transport at their respective peak neutron production times.

This density increase is the most significant for wavelengths with unstable regions that fall within the preheat portion of the shell. Figure 3.7 shows the effective A_T for unstable modes $\ell = 1 \rightarrow 200$ for D:T 10:90 and 50:50 both with (solid line) and without (dashed line) radiation transport enabled in 1-D simulations. The effective A_T is raised in both cases, with 10:90 experiencing a larger increase due to radiation ($\sim 5x$) compared to 50:50 ($\sim 3x$). In the limit

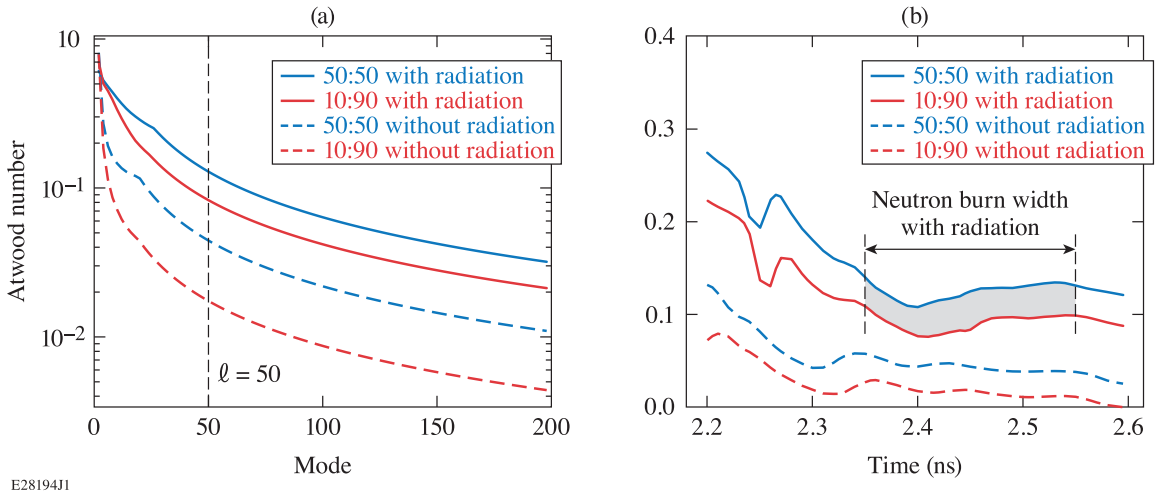

Figure 3.7: Radiation transport increases the effective Atwood number across all modes and D:T ratios. Compare the dashed and solid lines of each color to observe the increase caused by radiation. The $\ell = 50$ mode is indicated in (a) and plotted over time in (b) to demonstrate that this behavior persists during the entire deceleration phase

that the wavelength of the unstable mode goes to 0 (or $\ell \to \infty$), the effective A_T approaches the material interface $A_{T,\mathrm{i}}$.

For a sample $\ell = 50$ perturbation, Fig. 3.7(b) shows, that radiation raises the Atwood number over the entire deceleration phase, not just at peak neutron production time. For the cases with radiation enabled, neutron production occurs from approximately 2.35 to 2.55 ns, with peak neutron production occurring at 2.48 ns. Since radiation preheat is a dominant effect, radiation effects are included in all of the remaining simulation results in this chapter.

In addition to the sharp-boundary model presented in Sec. 3.1, RT growth factors can be estimated by various methods, the first of which uses a deceleration distance (ΔR) based on 1-D simulations, shown in Fig. 3.8. This distance describes the amount of deceleration of the material interface between the start of the deceleration phase and peak neutron production. The dashed line in the figure represents the trajectory of the material interface had it not undergone deceleration, i.e., free fall. This is used to find the linear RT growth rate scaling $\eta_0 \sim e^{\sqrt{2A_T k \Delta R}}$. Using the effective A_T for an $\ell = 40$ perturbation of the D:T 50:50 material interface, taken at the start of free fall, the free fall growth factor is GF ~ 5.8, where the growth factor is defined as GF $= \eta/\eta_0$. For 10:90, the growth factor is GF ~ 3.9, so 50:50 sees roughly a factor of 1.5

more growth.

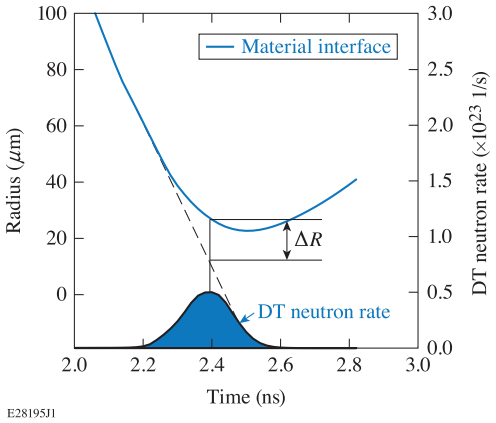

Figure 3.8: The trajectory of the material interface during the deceleration phase. The neutron burn history is shown at the bottom of the figure.

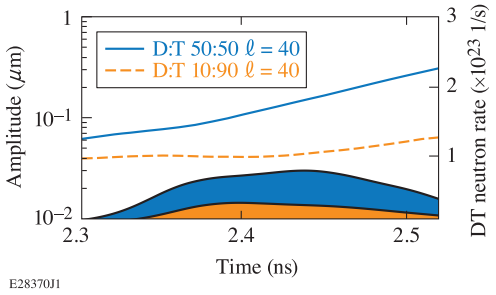

Figure 3.9: Single-mode, small-amplitude perturbation growth for D:T 10:90 (orange) and 50:50 (blue). Neutron production rates (right-most Y axis) are shown as filled curves at the bottom to convey where peak neutron production occurs.

Self-consistent, single-mode, small-amplitude ($\eta \leq$ 5e-3 μm) perturbations in 2-D *DRACO* [43] simulations, which include radiation transport, indicate that the material interface of D:T 10:90 experiences reduced growth compared to 50:50. Figure 3.9 compares 10:90 versus 50:50 for a small $\ell = 40$ perturbation at the material interface. This perturbation is applied as a surface imperfection at the beginning of the simulation, so that the growth remains self-consistent during the entire implosion. The perturbation growth found by the sharp-boundary model in Eq. 3.4 only applies the perturbation only at the start of the deceleration phase. Figure 3.9 shows the perturbation amplitude over time, measured as the peak-to-valley of the material interface. The single-mode growth behavior follows expected trends; D:T 50:50 experiences more growth compared to 10:90 due to a larger finite Atwood number.

Comparing the effective Atwood number for $\ell = 40$ against the Atwood number at the 10:90 interface reveals that the $\ell = 40$ perturbation sees a positive, finite Atwood number, and therefore a positive growth factor, as Fig. 3.9 demonstrates. This is also suggested by the free fall estimate, even though it lacks the self-consistent temporal dynamics of the interface that the perturbed simulation includes:

$$\frac{\mathrm{GF}^{2-D}_{50:50}}{\mathrm{GF}^{2-D}_{10:90}} \approx \frac{11}{2} = 5.5$$

When large-amplitude ($\eta \sim 0.2$ μm), high-mode ($\ell > 10$) perturbations are applied in 2-D simulations, the nonlinear growth rates are nearly indistinguishable between 10:90 and 50:50. Figure 3.10 shows the results from two 2-D simulations with identical surface perturbations applied at the material interface ($\ell = 120$ [0.1 μm], $\ell = 200$ [0.25 μm]) and outer edge ($\ell = 40$ [0.1 μm]) of the shell at $t = 0$. An 18° wedge was chosen to reduce the run-time of the simulation due to the high resolution required to resolve the short-scale modes throughout the implosion. The wedge is shown at early time (near shock convergence at the core) and at peak neutron production time, with the contour colors denoting the material fraction (blue = shell, red = fuel). The material interface is clearly seen at the boundary between the colors and is highly perturbed — even before the deceleration phase begins. These larger-scale perturbations were intentionally imposed to create increased growth during neutron production.

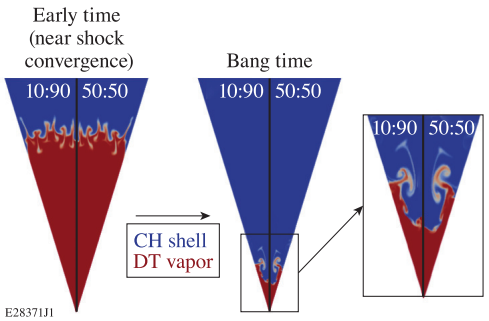

Figure 3.10: 2-D *DRACO* radiation-hydrodynamic simulations (18° wedge) with perturbations at the material interface ($\ell = 120$ [0.1 μm], $\ell = 200$ [0.25 μm]) and outer surface ($\ell = 40$ [0.1 μm]).

As shown in Fig. 3.7, the range of mode-specific effective Atwood numbers is raised due to radiation absorption, yielding a factor of approximately $2x$ difference between 10:90 and 50:50.

Both the free fall and single-mode, small-amplitude perturbation growth factors reflect this, but this trend does not carry over into the nonlinear growth regime. In Fig. 3.10, the difference in the size of the unstable region along the material interface is nearly indistinguishable between D:T ratios.

The lack of significant difference is due to the fact that in the nonlinear phase of RT growth, the interface perturbations (η_{NL}) (that are comprised of wide low-density bubbles and narrow high-density spikes) have bubble amplitudes that have been shown [57, 58] to scale linearly with time ($\eta_{\text{NL}} \sim V_b t$) under constant acceleration, with respect to bubble velocity (V_b). The bubbles then grow at the following rate:

$$V_b \sim \sqrt{\frac{2A_T}{1+A_T}\frac{g}{k}} \tag{3.6}$$

Here the Atwood number has less impact compared to linear RT growth, where the perturbation (η_{L}) grows exponentially in time ($\eta_{\text{L}} \sim \eta_0 e^{\sqrt{A_T g k}t}$). For a given $\ell = 40$ perturbation, this behavior is reflected in the GF ratio:

$$\frac{GF^{2-D}_{50:50}}{GF^{2-D}_{10:90}} \rightarrow \frac{V_b^{50:50}t}{V_b^{10:90}t} \sim 1.23$$

1-D and 2-D hydrodynamic simulations show nearly identical deceleration phase times, so the scaling based on bubble velocity and deceleration time can be compared to the free fall and single-mode, small-amplitude GF ratios. In free fall, the growth factor ratio does not capture the temporal evolution of the effective Atwood number, but it still indicates that 50:50 experiences increased growth compared to 10:90. Small-amplitude, single-mode perturbations, however, show that 10:90 experiences minimal growth, since it self-consistently includes the influence of the perturbation via the effective Atwood number. Large-amplitude, nonlinear perturbation simulations show that 10:90 and 50:50 look nearly identical. This is due to both the change in growth rate scaling within the nonlinear phase and effective Atwood numbers. In order for target performance to be affected by perturbations, the instability must grow to significant levels, i.e., transition to nonlinear, but in this regime, scaling laws show that the influence of the At-

wood number is only linear. During linear perturbation growth, the Atwood number influence is exponential. Radiation preheat causes the effective Atwood numbers to become less distinct between D:T ratios, and the growth of the perturbed material interface is only distinctly different in small-amplitude linear RT growth regimes where target performance is not significantly altered by the perturbations.

3.3 Experiment

An experimental campaign was undertaken to study the effect of varying room-temperature material interface A_T on target performance, and multiple targets were fabricated and shot on the OMEGA laser system. The targets were created to meet the design specification based on the classical material interface A_T [10:90 (stable), 25:75 (neutrally stable), 50:50 (unstable)]. Each target was designed to be 860 μm in diameter with 27 μm thick CH shells and a DT fuel fill pressure of 10 atm. The simulated and experimental pulse shape and baseline target dimension is shown in Fig. 3.11. Direct measurements of the deceleration-phase RT instability

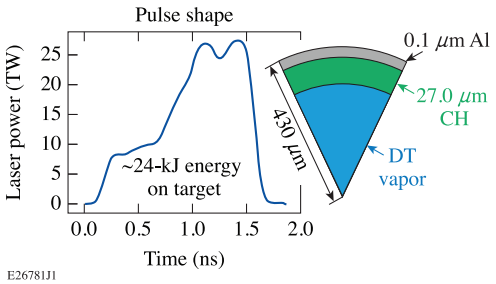

Figure 3.11: Pulse shape and target configuration used for both the experiment and simulations.

growth at the material interface are not currently possible, so performance is inferred through observable quantities such as primary DT yield and inferred DT ion temperatures ($T_{\rm i}$). These observables are collected through neutron time-of-flight (nTOF) detectors along six lines of sight within the OMEGA [52] target bay (Figure 3.12). Each nTOF detector is used to infer $T_{\rm i}$ from time-integrated neutron energy spectra of the hotspot from a particular direction. The neutron spectrum energy variance (Eq. 3.7) depends on the neutron ($m_{\rm n}$) and alpha particle (m_α) masses, plasma ion temperature ($T_{\rm i}$), mean energy of the primary neutron (E_0), and variance of the flow

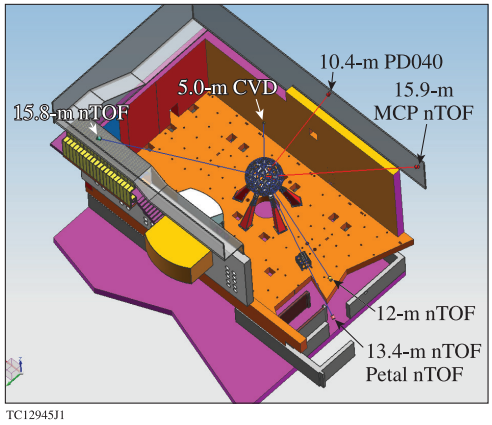

Figure 3.12: OMEGA target bay and neutron time-of-flight (nTOF) detector configuration.

velocity projected in the direction of the detector [Var($\vec{v} \cdot \vec{d}$)] [59, 60, 61]. The ion temperature is inferred through a procedure that numerically fits [62] the detector signal to the first term in Eq. 3.7 for a fusing plasma at a given temperature, also known as the Brysk temperature:

$$\sigma_{\mathrm{n}}^2 = \frac{2m_{\mathrm{n}}T_{\mathrm{i}}E_0}{m_{\mathrm{n}} + m_{\alpha}} + 2m_{\mathrm{n}}E_0\mathrm{Var}(\vec{v} \cdot \vec{d}) \tag{3.7}$$

Velocity variance along a particular line-of-sight causes the spectrum width to broaden (due to the second term in Eq. 3.7), resulting in an artificially high inferred T_{i}. Bulk fluid motion of the hotspot is also inferred from the shift in the peak of the primary neutron mean energy (E_0). A single line-of-sight measurement recorded the bulk fluid motion during each shot [7].

Additional targets were fabricated so that the actual fuel D:T could be measured by breaking open targets after they had been filled. Significant levels of protium (^{1}H) were found and were much higher than the amount in the initial fill. Protium is speculated to come from the CH shell due to the interaction with the 5.7 keV electrons born out of tritium beta decay [63]. Because the experiment was designed to set the average ion mass of the fuel through deuterium and tritium composition, the protium caused the actual material interface A_T to increase. Although A_T increased for all cases, the trend in stability from 10:90 to 50:50 remained intact (see Table 3.2).

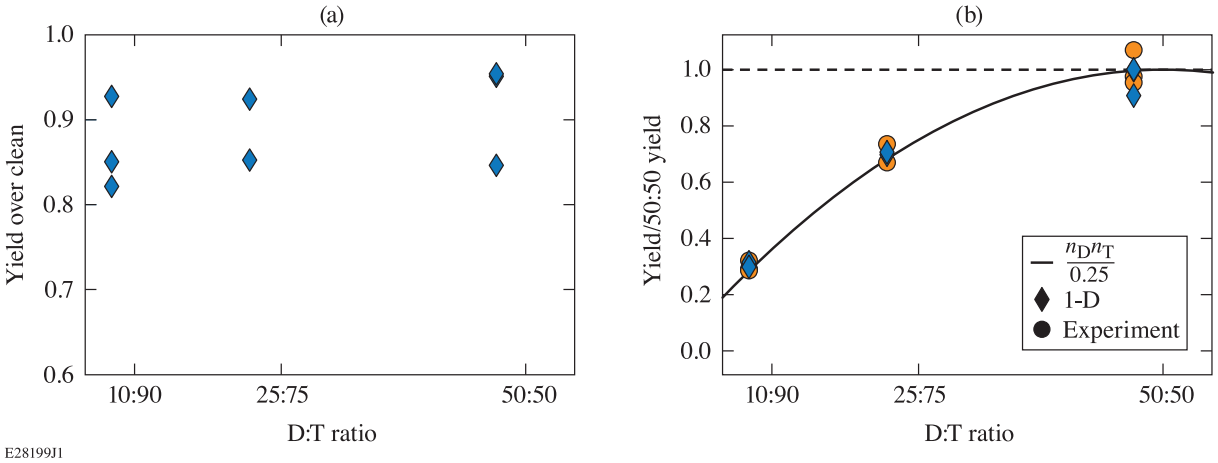

Figure 3.13: Yield over clean (Y_{exp}/Y_{1-D}) (a) and yield scaling (b). In (b), D:T 50:50 data points that lie above the dashed line at $y = 1.0$ indicate a higher than average yield.

Table 3.2: Protium effects on the Atwood number. Note that $A_{T,\mathrm{i}}$ represents the Atwood number calculated using the fully ionized material properties at the material interface.

Design		With ^{1}H	
D:T	$A_{T,\mathrm{i}}$	H:D:T	$A_{T,\mathrm{i}}$
10:90	-0.03	12:8:80	0.06
25:75	0.0	10:22:68	0.09
50:50	0.05	7:46:47	0.12

The yield over clean ratio (Y_{exp}/Y_{1-D}), which used post-shot simulations that included the measured H:D:T ratio, is consistent across all D:T ratios [see Fig. 3.13(a)]. This suggests that each shot experienced the same level of asymmetry and instability growth. Additionally, the yield of each target, for both measured and simulated, scaled according to the fraction of deuterium and tritium in the fuel. Figure 3.13(b) illustrates the DT yield of each shot when normalized to the respective (simulated or experimental) average 50:50 yield. The solid black curve represents a simple $Y_{\mathrm{DT}} \sim \frac{n_{\mathrm{D}} n_{\mathrm{T}}}{0.25}$ scaling relation based on the DT number densities in the fuel. Close clustering of the data points around this curve indicates that the yield scaled according to the fuel composition. Synthetic neutron spectra, generated with the Monte Carlo neutron-tracking code *IRIS3D* [64], are used to compare simulations to experimentally measured observables. Hydrodynamic profiles of implosions during neutron production are post-processed to generate

and track primary neutrons through the plasma to a set of detectors. Bulk fluid motion and T_i are inferred from these spectra using the same technique applied to the experimental spectra.

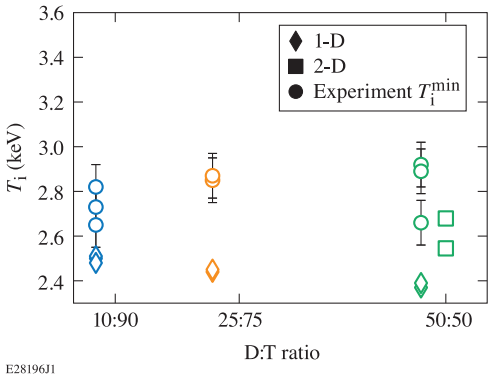

Figure 3.14: Minimum experimental inferred T_i (circles) versus 1-D *LILAC* thermal T_i and inferred 2-D *DRACO* T_i (diamonds). The 2-D results represent a minimal sample of perturbed 2-D runs that include target offset and low modes.

The minimum observed T_i is used as the surrogate for the thermal ion temperature of the hotspot. No true thermal T_i measurement is currently available, and the observed T_i is broadened by velocity effects (as shown in Eq. 3.7). The time-integrated neutron-averaged thermal T_i reported by 1-D *LILAC* and time-integrated inferred T_i from 2-D *DRACO* and *IRIS3D* simulations are compared to the minimum observed inferred T_i in Fig. 3.14. Experimental values, shown with their respective error bars, made any conclusive argument difficult, beyond the fact that the minimum T_i stayed fairly constant, signifying similar flow features within the hotspot across all D:T ratios. The 2-D data-points represent a subset of 2-D simulations that include target offset and $\ell = 3$ perturbations, and include the effects of velocity broadening. These data show the T_i inflation caused by hotspot flow effects in an asymmetric target, and bring the temperatures within range of the experiment.

Ion temperature asymmetry ($\Delta T_i = T_i^{max} - T_i^{min}$), currently taken from the set of six different nTOF measurements, is used to identify significant differences in T_i due to velocity broadening. Large ΔT_i indicates that there are significant nonradial components of velocity in the hotspot near peak neutron production time, most likely caused by instability growth and highly directional flow variance. The error bars (± 100 eV) arise from the noise level in the detector signal, uncertainty in the numerical fit analysis, and instrument response function (IRF) of each de-

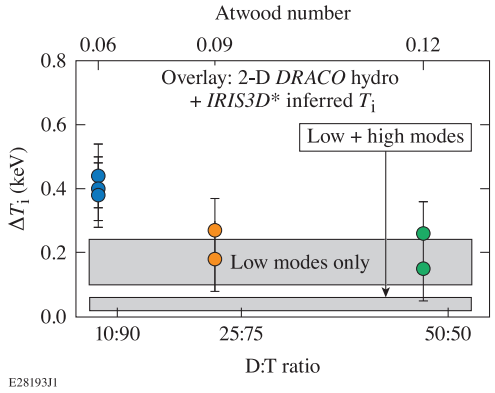

Figure 3.15: T_i asymmetry ($\Delta T_\mathrm{i} = T_\mathrm{i}^\mathrm{max} - T_\mathrm{i}^\mathrm{min}$) as a function of the D:T ratio and estimated Atwood number $A_{T,\mathrm{i}}$.

tector. The lower DT yield of the 10:90 targets created noisier signals compared to the 50:50 targets, which makes it difficult to interpret the data. Marginal trends may exist, but it is difficult to conclusively determine due to the level of error associated with each measurement. Ultimately, there is low sensitivity in ΔT_i due to varied D:T ratios and respective material interface Atwood numbers. Bulk fluid motion along a single line of sight [7] also reported velocities of less than 50 km/s. The trends from simulated results are superimposed on the experimental data in Fig. 3.15.

As mentioned earlier, low-mode asymmetry is seeded by offset, beam power imbalance, and shell distortions. Since not all sources are well known, a sensitivity study was performed to identify particular combinations that created similar target performance. These simulations show that ΔT_i varies only with respect to the particular perturbations applied to the target; altering the D:T ratio and material interface Atwood number had minimal impact. The highest amount of ΔT_i occurs when low modes are applied through the laser or target offset, but offset alone is not enough to generate significant ΔT_i. In fact, using offset values of up to 20 μm only generated $\Delta T_\mathrm{i} = 40$ to 50 eV. Adding an $\ell = 3$ mode increases the result to a value closer to the experiment $\Delta T_\mathrm{i} \approx 240$ eV, which is still a factor of 2 less than the largest experimental value. The $\ell = 3$ mode was chosen as a surrogate odd mode to introduce asymmetric flow patterns that develop into vortices co-located with neutron producing regions. These co-located vortices are necessary to create highly directional flow variance. Adding higher ℓ modes ($\ell > 40$) causes

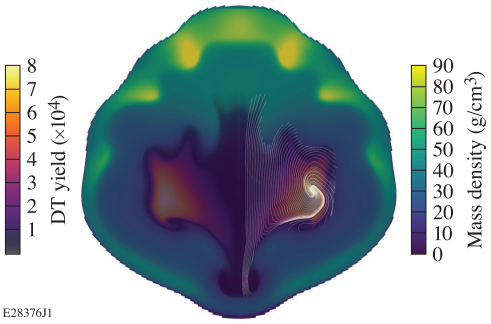

Figure 3.16: A 2-D mass density (purple/green/yellow) and DT yield (purple/red/yellow) contour of the shell and hotspot region near peak compression. The white streamlines represent the hotspot velocity flowfield. Velocity variation in fusing regions as viewed from a specific line of sight is essential for generating ΔT_{i}

ΔT_{i} to become indiscernible between all D:T ratios. High modes cause the hotspot to have a similar flow variance regardless of the direction from which the observation is made.

One of the simulations that generates the largest ΔT_{i} is shown in Fig. 3.16. The figure shows the shell region of the target at peak neutron production with contours of the mass density (purple/-green/yellow) and DT yield regions (purple/red/yellow) overlaid with fluid velocity streamlines. The streamlines give a visual indication of the hotspot flow features caused by the applied perturbations. This particular case was perturbed with a $20\ \mu$m target offset along the Z axis (up/down) and a 5% $\ell = 3$ laser asymmetry. The target offset induces a downward bulk flow of the target, while the $\ell = 3$ laser perturbation creates vorticity collocated within the highest yield regions. The flow vorticity in the high yield region is the source of the large ΔT_{i}. The detector along the Z axis infers a higher T_{i} than the X axis detector along the equator. Flow variation in colder, low yield regions does not significantly influence the neutron spectra due to the lack of neutrons.

3.4 Discussion

The classical material interface Atwood number, calculated based on continuity in pressure and temperature across the fuel-shell interface, does not adequately capture the stability during the deceleration phase of ICF implosions; the size and density of the unstable region must be included by means of the effective Atwood number. In the purely hydrodynamic context, altering

the ion mass of the DT fuel does change the growth factors, but radiation transport in ICF implosions alters this dynamic significantly. Simulations indicate that radiation preheat in both 10:90 and 50:50 cause the interface to have similar density profiles and therefore similar instability growth factors. Altering the overall shell stability through adiabat shaping would affect each D:T ratio in the same manner, since the shell is unchanged between D:T ratios. The initial instability seeds, however, would be higher in the low-adiabat implosions because of feedthrough from the ablation front.

Mass ablation rates are also analogous between the D:T ratios, so ablation stabilization caused by radiation and electron conduction will also be comparable. Mass ablation on the inner surface of the shell is responsible for the growth of the preheat shoulder, and simulations show similar effective Atwood numbers between all D:T ratios during the deceleration phase. This process is found to be the dominating influence on RT growth in the deceleration phase, since the stability is primarily determined by the preheat region rather than the density jump due to the D:T ratio and classical Atwood number at the material interface.

In both experiments and simulations, the yield of all the target configurations scales according to the composition of the fuel. Protium content increased the classical material interface A_T over the range of D:T ratios, but the difference between 10:90 and 50:50 remained consistent. Even with ideal D:T ratios, e.g., without protium included, simulations show similar growth factors for nonlinear growth. Both simulated and experimentally inferred ion temperatures indicate a comparable level of asymmetry across all D:T ratios. Significant ΔT_{i} (outside measurement uncertainty) requires highly-directional flow variance in order for detectors to observe differences from various lines of sight. Measurement uncertainty and noise levels make behavior trends inconclusive, and it is likely that the hotspot is relatively insensitive to changing the D:T ratio as simulations suggest.

3.5 Conclusion

Performance degradation in direct-drive inertial confinement fusion implosions can be caused by several effects, one of which is Rayleigh–Taylor (RT) instability growth during the deceleration phase. In room-temperature plastic target implosions, this deceleration-phase RT growth is enhanced by the density discontinuity and finite Atwood numbers at the fuel-shell interface. The influence of the Atwood number on deceleration-phase RT growth was studied through varied deuterium to tritium (D:T) ratios in the gas fill. An effective Atwood number calculation shows that radiation transport reduces the influence of D:T ratio on RT growth by raising the density near the material interface for all D:T ratios. Small amplitude perturbations showed a difference in linear growth factors between 10:90 and 50:50, but that difference was reduced when the growth was nonlinear. Because both the D:T 10:90 and 50:50 had similar effective Atwood numbers, the simulated deceleration RT instability growth was nearly identical for nonlinear RT growth, and there was little influence on the inferred T_{i} and ΔT_{i}. Both simulation and experimental data showed that yield performance scaled with the fraction of deuterium and tritium present in the fuel, and that ΔT_{i} has a small sensitivity to the different D:T ratios, due to similar levels of growth.

Chapter 4

Multi-physics: Numerical Methods

This chapter covers the numerical methods to obtain the solutions presented in Chapter 2. The methods provided here are in summary form such that their implementation is readily understood; refer to the included references for derivations and further detail.

4.1 Hydrodynamics

The 2D compressible Euler equations in conservative form (without source terms) are defined as

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial y} = 0 \tag{4.1}$$

where the vector $\vec{U}$ consists of the conserved variables, and $\vec{F}$ and $\vec{G}$ are the convective fluxes that are responsible for the transport of momentum and energy. These vectors are defined as

$$\vec{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix} \quad \vec{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(\rho E + p) \end{bmatrix} \quad \vec{G} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(\rho E + p) \end{bmatrix} \tag{4.2}$$

where ρ is the density, u and v are the x and y velocities respectively, p is total pressure, and E is the total energy. The Euler equations are combined with the ideal gas equation of state (EOS) to solve for the single-temperature, single-material hydrodynamics for the results presented in Chapter 2. In the ideal gas EOS, $\gamma = c_p/c_v$ where c_p is heat capacity at constant pressure, and c_v is heat capacity at constant volume. The ratio of specific heats for ideal gases can be found as $\gamma(f) = 1 + (2/f)$ where f is the number of degrees of freedom. Typical values are $\gamma = 1.4$ for dry air at standard atmospheric conditions and $\gamma = 5/3$ for a monatomic gas with three degrees of freedom. The ICF-related simulations in the thesis use $\gamma = 5/3$. In Eq. 4.2, total energy is defined as $E = e + \frac{|\vec{v}|^2}{2}$ where e is the internal energy of the gas, and $\vec{v}$ is the velocity. Total enthalpy of the gas is defined as

$$H = h + \frac{|\vec{v}|^2}{2} = E + \frac{p}{\rho} \tag{4.3}$$

In addition to the conserved vector $\vec{U}$, various numerical methods use the vector of primitive variables, which is defined as $\vec{V} = (\rho, u, v, p)^T$, where pressure p can be found from total energy by the following equation.

$$p = \rho(\gamma - 1)\Big(E - \frac{\vec{v}^2}{2}\Big) \tag{4.4}$$

The solution of Equation 4.1 is obtained by the method of lines, which separates the spatial and temporal discretization into separate steps [65]. This provides the option to choose different discretization algorithms for space and time.

4.1.1 Temporal Discretization

The temporal discretization, or integration, step is handled by multi-stage Runge-Kutta explicit methods. The 2-stage, 2$^{\text{nd}}$ order (Eq. 4.5) nonlinear Strong-Stability-Preserving (SSP) Runge-Kutta method is provided below [66]. The timestep Δt is determined by the CFL condition [67].

$$\begin{aligned}\vec{U}^{(1)} &= \vec{U}^n + \frac{\partial \vec{U}^n}{\partial t}\Delta t \\ \vec{U}^{n+1} &= \frac{1}{2}\vec{U}^n + \frac{1}{2}\vec{U}^{(1)} + \frac{1}{2}\frac{\partial \vec{U}^{(1)}}{\partial t}\Delta t\end{aligned} \tag{4.5}$$

Equation 4.6 increases the maximum stable CFL number and resulting timestep with the optimal 3-stage 3rd order method.

$$\begin{aligned}\vec{U}^{(1)} &= \vec{U}^n + \Delta t\frac{\partial \vec{U}^n}{\partial t} \\ \vec{U}^{(2)} &= \frac{3}{4}\vec{U}^n + \frac{1}{4}\vec{U}^{(1)} + \frac{1}{4}\Delta t\frac{\partial \vec{U}^{(1)}}{\partial t} \\ \vec{U}^{n+1} &= \frac{1}{3}\vec{U}^{(n)} + \frac{2}{3}\vec{U}^{(1)} + \frac{2}{3}\Delta t\frac{\partial \vec{U}^{(2)}}{\partial t}\end{aligned} \tag{4.6}$$

In both methods, the $\partial\vec{U}^{(1)}/\partial t$ term is provided by the Riemann solver. The $\vec{U}^{(1)}$ and $\vec{U}^{(2)}$ are inter-stage conserved variable states, and $\vec{U}^{(n+1)}$ is the solution at the next iteration with timestep Δt.

4.1.2 Spatial Discretization

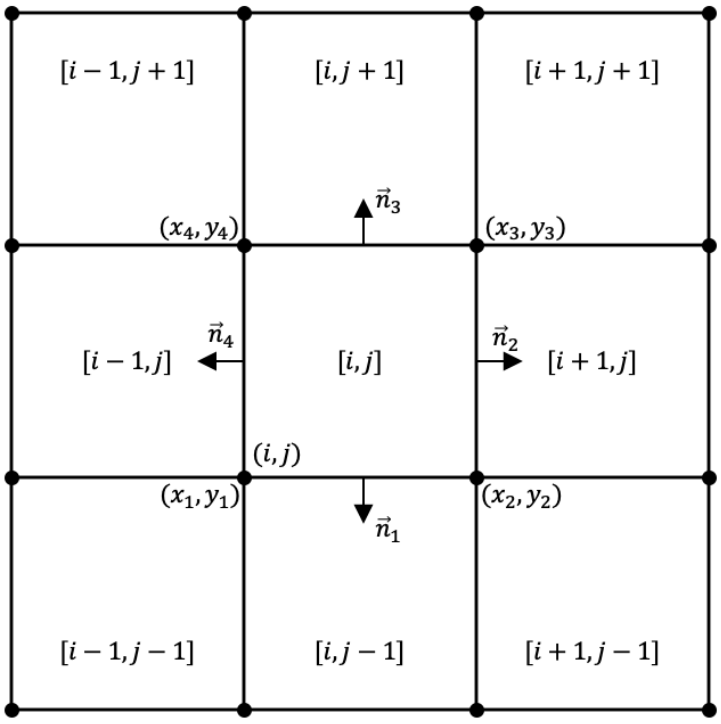

Figure 4.1: The 2D finite volume layout used to solve for hydrodynamics.

Spatial discretization is accomplished using the cell-centered finite volume method on a fixed mesh. The 2D domain configuration is shown in Figure 4.1. The all state variables (e.g. conserved state $\vec{U}$ or primitive state $\vec{V}$) are stored at the centroid of each cell. Each cell edge has a normal vector $\vec{n}$ and corner vertices (or nodes) at (x_n, y_n). The current cell is logically addressed by the indices (i, j) with neighbors at $(i \pm 1, j \pm 1)$. Any cell-centered variable is referred to by $\bar{\Phi}$. Because the grid is fixed, the temporal derivative of the control volume, or cell can be expressed in the form

$$\frac{\partial \vec{U}}{\partial t} = -\frac{1}{\Omega} \oint_{\partial\Omega} \vec{F}_c \, dS \tag{4.7}$$

where Ω is the volume, and the $\vec{F}_c$ is the vector of the convective fluxes, or

$$\vec{F}_c = \vec{F} n_x + \vec{G} n_y \tag{4.8}$$

Equation 4.8 is discretized over the grid into the following form which is solved numerically.

$$\frac{\partial \vec{U}^n}{\partial t} = -\frac{1}{\Omega_{ij}} \sum_{n=1}^{N} (\vec{F}_c)_n \, \delta S_n \tag{4.9}$$

Here Ω_{ij} is the cell volume (or area in 2D), N is the number of cell edges, $(\vec{F}_c)_n$ is the convective flux vector of each edge, and δS_n is the length of each edge. In the domain layout shown in Figure 4.1, $N = 4$ for a quadrilateral cell.

The convective flux vector $\vec{F}_c$ is determined by a flux solver, which solves the discontinuous Riemann problem (via Godonov's method [68]) at each cell edge. An *upwind* solver separately interpolates the state on either side (based on the characteristics of the Euler eqs.) of the cell edge based on the left and right neighboring average values. This interpolation (the upwind MUSCL method of Van Leer [69]) provides a higher-order approximation of the state at the edge ($\Phi_{L,R}$) to pass to the Riemann solver of choice.

The MUSCL reconstruction scheme is illustrated in Figure 4.2 in 1-D. This process is extended to 2-D or 3-D by dimensional splitting. The reconstructed flow state variable ($\Phi_{L,R}$) at the edge

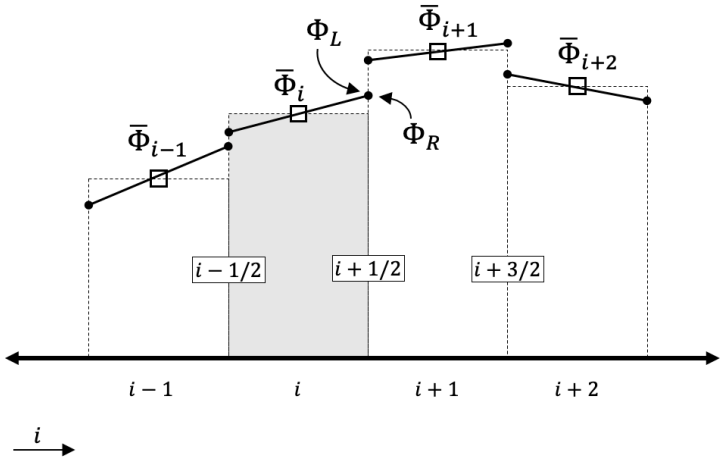

Figure 4.2: MUSCL reconstruction scheme

$i+1/2$ uses the cell-average state of $\bar{\Phi}_i$ and $\bar{\Phi}_{i\pm1}$. A second order reconstruction is found using

$$\begin{aligned} \Phi_L &= \Phi_i + .5\phi(r_{L,i})\Delta\Phi_{i-1/2} \\ \Phi_R &= \Phi_{i+1} - .5\phi(r_{R,i+1})\Delta\Phi_{i+3/2} \end{aligned} \tag{4.10}$$

where the cell-average jump values $\Delta\Phi$ are defined by the following.

$$\begin{aligned} \Delta\Phi_{i-1/2} &= \bar{\Phi}_i - \bar{\Phi}_{i-1} \\ \Delta\Phi_{i+1/2} &= \bar{\Phi}_{i+1} - \bar{\Phi}_i \\ \Delta\Phi_{i+3/2} &= \bar{\Phi}_{i+2} - \bar{\Phi}_{i+1} \end{aligned} \tag{4.11}$$

The $\phi(r_{L,i})$ and $\phi(r_{R,i+1})$ terms are *limiter* functions. The $r_{L,i}$ and $r_{R,i+1}$ terms are defined by the following.

$$\begin{aligned} r_{L,i} &= \Delta\Phi_{i+1/2}/\Delta\Phi_{i+1/2} \\ r_{R,i+1} &= \Delta\Phi_{i+1/2}/\Delta\Phi_{i+3/2} \end{aligned} \tag{4.12}$$

Limiters are used to prevent unphysical oscillations resulting from the higher-order interpolations, particularly near discontinuous jumps such as shocks. Total variation diminishing (TVD) schemes, introduced in Reference 70, are designed to limit extrema in the reconstruction step, and perform well at capturing shocks. Some examples of well-known, robust limiter functions

are defined below [71].

$$\begin{aligned} \phi_{\text{MinMod}}(r) &= \max(0, \min(r, 1)) \\ \phi_{\text{VanLeer}}(r) &= (r + |r|)/(1 + |r|) \\ \phi_{\text{SuperBee}}(r) &= \max(0, \min(2r, 1), \min(r, 2)) \end{aligned} \tag{4.13}$$

The MinMod limiter is the most robust (but overly diffusive), while SuperBee is the most compressive (and generates sharper gradients).

In multiple dimensions, TVD schemes were found to be limited to first order accuracy [72]. By focusing on oscillation control rather than strict monotonic solutions, Kim et. al introduced the Multidimensional Limiting Process in Reference 73, which can recover higher order solutions with multidimensional limiters. This builds on the idea of TVD and combines nearest-neighbor information with high-order interpolation functions to achieve 3rd and 5th order spatial reconstruction. The interpolation functions to obtain $\Phi_{L/R}$ at each cell interface is shown below.

$$\begin{aligned} \Phi_L &= \bar{\Phi}_{i,j} + .5 \max(0, \min(\alpha_L r_{L,i}, \alpha_L, \beta_L)) \Delta\Phi_{i-1/2} \\ \Phi_R &= \bar{\Phi}_{i+1,j} - .5 \max(0, \min(\alpha_R r_{R,i+1}, \alpha_R, \beta_R)) \Delta\Phi_{i+3/2} \end{aligned} \tag{4.14}$$

The high-order 3rd and 5th order polynomial reconstruction β functions are provided by

$$\begin{aligned} \beta_L^{(3)} &= \frac{1 + r_{L,i}}{3} \\ \beta_R^{(3)} &= \frac{1 + r_{R,i+1}}{3} \end{aligned} \tag{4.15}$$

$$\begin{aligned} \beta_L^{(5)} &= \frac{-2/r_{L,i-1} + 11 + 24 r_L - 3 r_L r_{L,i+1}}{30} \\ \beta_R^{(5)} &= \frac{-2/r_{R,i+2} + 11 + 24 r_{R,i+1} - 3 r_{R,i+1} r_R}{30} \end{aligned} \tag{4.16}$$

MLP combined with 3rd or 5th order reconstruction is referred to as MLP3 and MLP5 respectively. The additional neighbor r terms (due to higher order) are defined by Equations 4.17 and

4.18.

$$\begin{aligned} r_{L,i-1} &= \Delta\Phi_{i-1/2}/\Delta\Phi_{i-3/2} \\ r_{L,i+1} &= \Delta\Phi_{i+3/2}/\Delta\Phi_{i+1/2} \\ r_{R,i+1} &= \Delta\Phi_{i+1/2}/\Delta\Phi_{i+3/2} \\ r_{R,i+2} &= \Delta\Phi_{i+3/2}/\Delta\Phi_{i+5/2} \end{aligned} \tag{4.17}$$

$$\begin{aligned} \Delta\Phi_{i+3/2} &= \bar{\Phi}_{i+2,j} - \bar{\Phi}_{i+1,j} \\ \Delta\Phi_{i-3/2} &= \bar{\Phi}_{i-1,j} - \bar{\Phi}_{i-2,j} \\ \Delta\Phi_{i+5/2} &= \bar{\Phi}_{i+3,j} - \bar{\Phi}_{i+2,j} \end{aligned} \tag{4.18}$$

The α and $\tan\theta$ terms account for the multi-dimensionality of the problem and neighbors within a nine-point stencil ((i,j) and $(i \pm 1, j \pm 1)$). The α term is dependent on the direction of the flux, namely at the $i+1/2$ or $j+1/2$ interface. For reconstruction along the $i+1/2$ edge, α is the following.

$$\begin{aligned} \alpha_L &= g\left[\frac{2\max(1, r_{L,i})(1+\max(0, \frac{\tan\theta_{i+1}}{r_{R,i+1}}))}{1+\tan\theta_i}\right] \\ \alpha_R &= g\left[\frac{2\max(1, r_{R,i+1})(1+\max(0, \frac{\tan\theta_i}{r_{L,i}}))}{1+\tan\theta_{i+1}}\right] \end{aligned} \tag{4.19}$$

where $g(x) = \max(1, \min(2, x))$. For reconstructing along the $j+1/2$ edge, α uses the following form.

$$\begin{aligned} \alpha_L &= g\left[\frac{2\max(1, r_{L,j})(1+\max(0, \frac{\tan\theta_{j+1}}{r_{R,j+1}}))}{1+\tan\theta_j}\right] \\ \alpha_R &= g\left[\frac{2\max(1, r_{R,j+1})(1+\max(0, \frac{\tan\theta_j}{r_{L,j}}))}{1+\tan\theta_{j+1}}\right] \end{aligned} \tag{4.20}$$

The $\tan\theta$ terms are defined in Equation 4.21. These are synonymous with the $\Delta\Phi$ terms in 1D, but account for discontinuities that cross the mesh at oblique angles (not aligned with the mesh).

$$\begin{aligned} \tan\theta_i &= \left|\frac{\Phi_{i,j+1} - \Phi_{i,j-1}}{\Phi_{i+1,j} - \Phi_{i-1,j}}\right|, \quad \tan\theta_{i+1} = \left|\frac{\Phi_{i+1,j+1} - \Phi_{i+1,j-1}}{\Phi_{i+2,j} - \Phi_{i,j}}\right| \\ \tan\theta_j &= \left|\frac{\Phi_{i+1,j} - \Phi_{i-1,j}}{\Phi_{i,j+1} - \Phi_{i,j-1}}\right|, \quad \tan\theta_{j+1} = \left|\frac{\Phi_{i+1,j+1} - \Phi_{i-1,j+1}}{\Phi_{i,j+2} - \Phi_{i,j}}\right| \end{aligned} \tag{4.21}$$

For derivation of MLP and a complete description of each term, see Reference 73.

4.1.3 Boundary Conditions

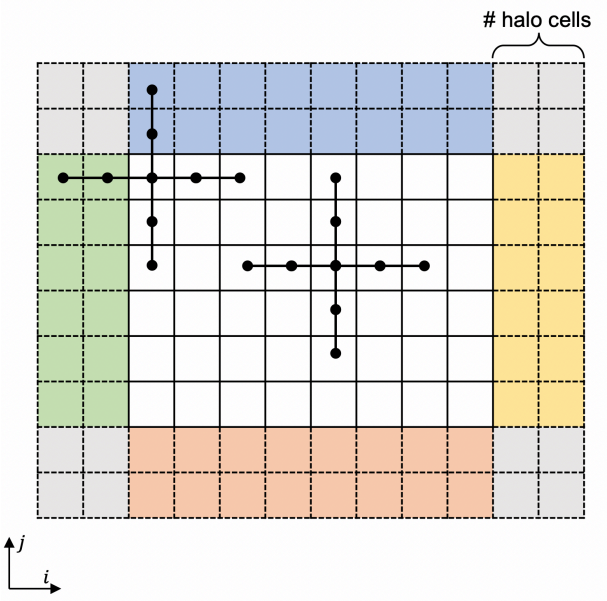

Figure 4.3: Boundary conditions are facilitated by halo cells. The real domain consists of the white cells, and the halo edge regions are light green, blue, yellow, and orange. If the reconstruction method uses diagonal neighbor cells, e.g. $(i+1, j+1)$, corner cells (in gray) are also required. The example 9-point stencils shown used by spatial reconstruction require 2 halo cells (Refer to Equations 4.17 and 4.18).

Boundary conditions are applied using halo cells, and the number of halo cells is determined by the spatial order of the reconstruction method (see Figure 4.3). Two halo cells are used in 2^{nd} and 3^{rd} order and 3 halo cells are used for 5^{th} order. Various boundary conditions such as inflow, outflow, reflection, periodic, and zero-gradient. Periodic conditions simply copy data from the opposing edge and apply it to the halo region. Zero-gradient conditions set the halo region to equal the edge of the domain so the change (or gradient) is zero. Reflection conditions apply mirror symmetry in the halo region respective of the edge of the domain. Inflow and outflow conditions extrapolate the conserved values of the edge and halo region depending on the direction of the characteristics, i.e. super or subsonic flow. These require setting conditions at the boundary, such as pressure [74].

4.1.4 The (M-)AUSMPW+ Riemann Solver

The hydrodynamics of internal defects (presented in Chapter 2) requires a low dissipation scheme with high-order accuracy to capture characteristic wave propagation in an extreme environment. ICF implosions involve strong shocks, material interfaces, a wide range of Mach number (both subsonic and supersonic), and extreme pressures (across a wide range of densities). Multiple inviscid flux schemes were tested, including the following:

1. Finite-Volume Evolution Galerkin (FVLEG) [75]
2. Roe [76] and Roe-M [77]
3. Advection Upwind Splitting Method (AUSM$^+$) [78]
4. AUSM$^+$-up [79]
5. Simple Low-dissipation AUSM (SLAU) [80]
6. SLAU2 and SD-SLAU [81]
7. HLLC [82] and HLLC-M [83]
8. AUSMPW+ [84], and M-AUSMPW+ [85]

AUSMPW+ and M-AUSMPW+ were chosen over the other Riemann solvers due to their simplicity and robustness against shock instabilities (such as even-odd decoupling [86] and the carbuncle phenomenon [87]) and numerical noise. They have been shown to be highly successful in the hypersonic aerodynamic community due to these characteristics [81] and have also been extended to MHD problems [88]. M-AUSMPW+ is a modified form of AUSMPW+ with additional discontinuity sensors a better upwind characteristics.

Because they are upwind schemes, they maintain correct characteristic flow in supersonic regions. Both are low-dissipation schemes that limit numerical smearing effects on acoustic wave propagation, and they were found to be the most robust against numerical noise in the high pressure, low density coronal region in ICF implosion problems. Test problems that imposed small

amplitude density perturbations in planar layered targets (like those used in Chapter 2) revealed significant noise due to the disparity between the extreme pressures and low densities. While numerical noise issues can be controlled through software development techniques, the fluxing algorithms of both AUSMPW+ and M-AUSMPW+ were the most robust for this issue.

The AUSMPW+ scheme can be summarized by the following. At its core it is an advection upstream splitting method (AUSM) (which itself is a flux-vector splitting scheme [89]) which decomposes the flux (Equation 4.22) at cell interfaces into a convective part (terms between brackets) and pressure part (terms between parentheses). The sign of the convective flux depends on the result of specific interpolation functions and characteristic values. In the all of following equations, interfacial quantities have the ($_{1/2}$) subscript and the ($_{L/R}$ and $^{\pm}$) terms denote the left and ride side of the interface respectively.

$$\vec{F}_{1/2} = \left[\bar{M}_L^{+} c_{\frac{1}{2}} \vec{\Psi}_{L,\frac{1}{2}} + \bar{M}_R^{-} c_{\frac{1}{2}} \vec{\Psi}_{R,\frac{1}{2}}\right] + \left(P_L^{+}\vec{P}_L + P_R^{-}\vec{P}_R\right) \tag{4.22}$$

The convective flux $\vec{\Psi}_{(L,R)_{1/2}}$ and pressure flux $\vec{P}_{(L,R)}$ are defined as:

$$\vec{\Psi}_{(L,R)_{1/2}} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho H \end{bmatrix}_{(L,R)_{1/2}} \qquad \vec{P}_{(L,R)} = \begin{bmatrix} 0 \\ n_x p \\ n_y p \\ 0 \end{bmatrix}_{(L,R)} \tag{4.23}$$

Here, the quantities n_x and n_y are the normal vector components of the cell interface and interface enthalpy $H_{(L,R)_{1/2}}$ is defined as:

$$H_{(L,R)_{1/2}} = \frac{\gamma}{\gamma - 1}\frac{p_{(L,R)_{1/2}}}{\rho_{(L,R)_{1/2}}} + \frac{1}{2}(u^2_{(L,R)_{1/2}} + v^2_{(L,R)_{1/2}}) \tag{4.24}$$

In Equation 4.22, the Mach splitting functions ($M^{\pm}_{(L/R)}$) and pressure splitting functions ($P^{\pm}_{L/R}$)

to help determine the direction of flux. These are defined as

$$M^{\pm}_{L,R} = \begin{cases} \pm\frac{1}{4}(M_{L,R} \pm 1)^2, & |M_{L,R}| \leq 1. \\ \frac{1}{2}(M_{L,R} \pm |M_{L,R}|), & |M_{L,R}| > 1. \end{cases} \tag{4.25}$$

$$P^{\pm}_{L,R} = \begin{cases} \pm\frac{1}{4}(M_{L,R} \pm 1)^2(2 \mp M_{L,R}), & |M_{L,R}| \leq 1. \\ \frac{1}{2}(1 \pm \text{sign}(M_{L,R})), & |M_{L,R}| > 1. \end{cases} \tag{4.26}$$

which depend on the Mach number for the left and right state is defined as $M_{L/R} = U_{L/R}/c_{1/2}$. The interfacial sound speed is obtained by

$$c_{1/2} = \begin{cases} c_s^2/\max(|U_L|, c_s), & \text{if } \frac{1}{2}(U_L + U_R) \geq 1 \\ c_s^2/\max(|U_R|, c_s), & \text{if } \frac{1}{2}(U_L + U_R) < 0 \end{cases} \tag{4.27}$$

where the sound speed is defined as $c_s = \sqrt{2((\gamma-1)/(\gamma+1))H_{\text{normal}}}$ and the interface-normal enthalpy is $H_{\text{normal}} = \min(H_L - 0.5 \cdot V_L^2, H_R - 0.5 \cdot V_R^2)$. Note that the U and V velocity components correspond to the normal and transverse components respective to the cell interface. The pressure-based and Mach number-based weighting functions (w and f respectively) are used to suppress oscillations near discontinuities.

$$w = 1 - \min\left(\frac{p_L}{p_R}, \frac{p_R}{p_L}\right)^3 \tag{4.28}$$

$$f = \begin{cases} (\frac{p_{L,R}}{p_s} - 1)\min(1, \frac{\min(p_{1,L}, p_{1,R}, p_{2,L}, p_{2,R})}{\min(p_L, p_R)}), & \text{if } p_s \neq 0 \\ 0, & \text{elsewhere} \end{cases} \tag{4.29}$$

$$p_s = P_L^+ p_L + P_R^- p_R \tag{4.30}$$

The terms $p_{1,L},\ p_{1,R},\ p_{2,L},$ and $p_{2,R}$ are the pressures of the cells adjacent to the cell interface. See Ref. 84 for more details. Finally, the Mach interpolation function $\bar{M}^{\pm}_{L/R}$ is obtained based

on the previous pressure and Mach splitting and weighting functions.

$$\bar{M}_L^+ = \begin{cases} M_L^+ + \\ M_R^- \cdot [(1-w)\cdot(1+f_R) - f_L], & \text{if } m_{1/2} >= 1 \\ M_L^+ \cdot w \cdot (1+f_L), & \text{if } m_{1/2} < 0 \end{cases} \tag{4.31}$$

$$\bar{M}_R^- = \begin{cases} M_R^- \cdot w \cdot (1+f_R), & \text{if } m_{1/2} >= 1 \\ M_R^+ + \\ M_L^- \cdot [(1-w)\cdot(1+f_L) - f_R], & \text{if } m_{1/2} < 0 \end{cases} \tag{4.32}$$

where $m_{1/2} = M_L^+ + M_R^-$. Equations 4.31 and 4.32 are then combined with 4.23 and 4.26 to obtain the flux at the interface defined in Equation 4.22. The M-AUSMPW+ method modifies the AUSMPW+ pressure and Mach number weighting functions (w and f respectively) to improve oscillation suppression near discontinuities.

$$w = \max(w_1, w_2) \tag{4.33}$$

$$w_1 = 1 - \min\left(\frac{p_L}{p_R}, \frac{p_R}{p_L}\right)^3 \tag{4.34}$$

The w_1 detects a shock in the direction normal to the cell interface. The p_L, p_R terms are the reconstructed pressure values at the cell interface. Shock instability detection is improved with the w_2 function, which uses the neighboring cell-averaged pressure values $\bar{p}$ in the following equations.

$$w_{2,i} = \left[1 - \min\left(1, \frac{\bar{p}_{i+1,j} - \bar{p}_{i,j}}{.25(\bar{p}_{i+1,j+1} + \bar{p}_{i+1,j} - \bar{p}_{i+1,j-1} + \bar{p}_{i,j-1})}\right)\right]^2 \times \left[1 - \min\left(\frac{\bar{p}_{i,j}}{\bar{p}_{i+1,j}}\frac{\bar{p}_{i+1,j}}{\bar{p}_{i,j}}\right)\right]^2 \tag{4.35}$$

$$w_{2,j} = \left[1 - \min\left(1, \frac{\bar{p}_{i,j+1} - \bar{p}_{i,j}}{.25(\bar{p}_{i+1,j+1} + \bar{p}_{i+1,j} - \bar{p}_{i-1,j+1} + \bar{p}_{i-1,j})}\right)\right]^2 \times \left[1 - \min\left(\frac{\bar{p}_{i,j}}{\bar{p}_{i,j+1}}\frac{\bar{p}_{i,j+1}}{\bar{p}_{i,j}}\right)\right]^2 \tag{4.36}$$

Each function is called depending on the direction of the cell interface, i.e. $w_{2,i}$ along the $i+1/2$ edge, and $w_{2,j}$ along the $j + 1/2$ edge. M-AUSMPW+ also improves the convective behavior of the AUSMPW+ method re-evaluate the convective quantity $\Phi = (\rho, \rho u, \rho v, \rho E)$ after spatial reconstruction. This is done by imposing the following requirements: 1) distinguish between a discontinuity or continuous region, 2) be monotonic, 3) remain upwind in supersonic flow. For 1 and 2, the convective quantities is evaluated with both the SuperBee and user-defined limiter (typically MLP-based) to properly determine the correct amount of variation at the cell interface. For 3, re-evaluation determines the sonic state of the cell interface. M-AUSMPW+ achieves this using Equations 4.37 and 4.38.

$$\Phi_{L,1/2} = \Phi_L + \frac{\max[0, (\Phi_R - \Phi_L)(\Phi_{L,SB} - \Phi_L)]}{(\Phi_R - \Phi_L)|\Phi_{L,SB} - \Phi_L|} \min\left[a\frac{|\Phi_R - \Phi_L|}{2}, |\Phi_{L,SB} - \Phi_L|\right] \tag{4.37}$$

$$\Phi_{R,1/2} = \Phi_R + \frac{\max[0, (\Phi_L - \Phi_R)(\Phi_{R,SB} - \Phi_R)]}{(\Phi_L - \Phi_R)|\Phi_{R,SB} - \Phi_R|} \min\left[a\frac{|\Phi_L - \Phi_R|}{2}, |\Phi_{R,SB} - \Phi_R|\right] \tag{4.38}$$

The terms with $\Phi_{L/R,SB}$ are computed by MUSCL interpolation with the SuperBee limiter, and the $\Phi_{L/R}$ use the spatial reconstruction chosen by the user. The function $a = 1 - \min(1, \max(|M_L|, |M_R|))^2$ determines the sonic condition of the interface. The derivation and additional details can be found in Reference 85.

4.2 Thermal Conduction

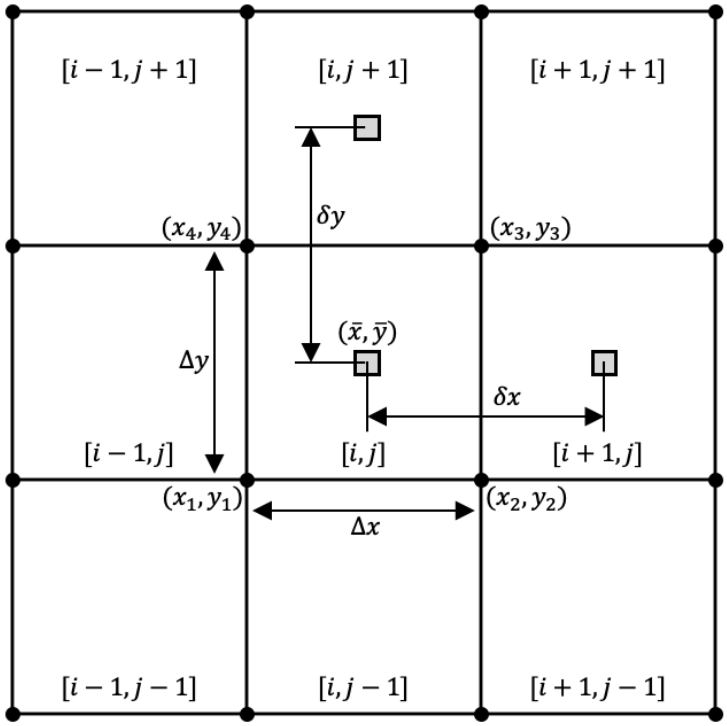

Figure 4.4: The 2D finite control volume layout used to solve thermal conduction.

The governing equation for time-dependant thermal conduction is given by the following.

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(\kappa \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(\kappa \frac{\partial T}{\partial y}\right) + \dot{q} \tag{4.39}$$

Here the temperature T, is based on density ρ, heat capacity at constant pressure c_p, thermal conductivity κ, and a source term $\dot{q}$. In the context of this dissertation, the thermal conductivity follows that of Spitzer [90] for a fully ionized plasma where $\kappa \sim T^{5/2}$. Section 4.2.3 goes into more detail regarding the calculation of κ. Equation 4.39 is discretized in space and time following the method provided by Reference 91, with the domain schematic shown in Figure 4.4. Equation 4.40 is the discretized form which can be solved implicitly using a variety of numerical methods, including the alternating direction implicit method (covered in Section 4.2.1).

$$a_{i,j}T_{i,j} = a_{i+1,j}T_{i+1,j} + a_{i-1,j}T_{i-1,j} + a_{i,j+1}T_{i,j+1} + a_{i,j-1}T_{i,j-1} + b \tag{4.40}$$

In the discretized equation, $T_{i,j}$ represents the cell average temperature at the centroid. The indices (i, j) logically represent the current cell or and neighboring cells are at $(i \pm 1, j \pm 1)$. The $a_{i\pm1,j\pm1}$ terms represent the thermal conductance of the neighboring control volumes. These are defined by Equation 4.41.

$$\begin{aligned} a_{i+1,j} &= \frac{\kappa_{i+1/2}\Delta y_{i+1/2}}{\delta x_{i+1,j}} \\ a_{i-1,j} &= \frac{\kappa_{i-1/2}\Delta y_{i-1/2}}{\delta x_{i-1,j}} \\ a_{i,j+1} &= \frac{\kappa_{j+1/2}\Delta x_{j+1/2}}{\delta y_{i,j+1}} \\ a_{i,j-1} &= \frac{\kappa_{j-1/2}\Delta x_{j-1/2}}{\delta y_{i,j-1}} \end{aligned} \tag{4.41}$$

The $\kappa_{i\pm1/2,j\pm1/2}$ terms represent the thermal conductivity of the edges between the current and neighboring cells. These are computed using the harmonic mean of the cell averaged conductivity of the control volumes ($\bar{\kappa}$), as shown below:

$$\begin{aligned} \kappa_{i+1/2} &= \frac{2\bar{\kappa}_{i,j}\bar{\kappa}_{i+1,j}}{(\bar{\kappa}_{i,j} + \bar{\kappa}_{i+1,j})} \\ \kappa_{i-1/2} &= \frac{2\bar{\kappa}_{i,j}\bar{\kappa}_{i-1,j}}{(\bar{\kappa}_{i,j} + \bar{\kappa}_{i-1,j})} \\ \kappa_{j+1/2} &= \frac{2\bar{\kappa}_{i,j}\bar{\kappa}_{i,j+1}}{(\bar{\kappa}_{i,j} + \bar{\kappa}_{i,j+1})} \\ \kappa_{j-1/2} &= \frac{2\bar{\kappa}_{i,j}\bar{\kappa}_{i,j-1}}{(\bar{\kappa}_{i,j} + \bar{\kappa}_{i,j-1})} \end{aligned} \tag{4.42}$$

In Equation 4.41, the $(\Delta x, \Delta y)$ terms are the lengths of the cell edges, and the $(\delta x, \delta y)$ terms are the distances between neighboring cell-centers (or centroids). The centroids are located at $(\bar{x}, \bar{y})$, and the spacing is defined by the following.

$$\begin{aligned} \delta x_{i+1,j} &= |\bar{x}_{i+1,j} - \bar{x}_{i,j}| \\ \delta x_{i-1,j} &= |\bar{x}_{i-1,j} - \bar{x}_{i,j}| \\ \delta y_{i,j+1} &= |\bar{y}_{i,j+1} - \bar{y}_{i,j}| \\ \delta y_{i,j-1} &= |\bar{y}_{i,j-1} - \bar{y}_{i,j}| \end{aligned} \tag{4.43}$$

The control volume edge lengths are defined below (subscripts ($_{1-4}$) correspond to the vertex values in the figure).

$$\begin{aligned}
\Delta x_{i+1/2} &= [(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2} \\
\Delta x_{i-1/2} &= [(x_3 - x_2)^2 + (y_3 - y_2)^2]^{1/2} \\
\Delta y_{j+1/2} &= [(x_4 - x_3)^2 + (y_4 - y_3)^2]^{1/2} \\
\Delta y_{j-1/2} &= [(x_1 - x_4)^2 + (y_1 - y_4)^2]^{1/2}
\end{aligned} \tag{4.44}$$

The remaining terms in Equation 4.40 are as follows:

$$a_{i,j} = a_{i,j+1} + a_{i,j-1} + a_{i,j+1} + a_{i,j-1} + a^0_{i,j} \tag{4.45}$$

$$b = \Omega_{i,j}\dot{q}_{i,j} + a^0_{i,j}T^0_{i,j} \tag{4.46}$$

$$a^0_{i,j} = \frac{\rho_{i,j}\ c_p \Omega_{i,j}}{\Delta t} \tag{4.47}$$

where Ω_{ij} is the cell volume (or area in 2D), and $T^0_{i,j}$ is the temperature of the control volume at the current time-step. The source term $\dot{q}_{ij}$ is injected as a power density term, with units like erg/(s·cm^3) or W/cm^3. This source term is how the laser energy is deposited into the system for the multi-physics simulations in Chapter 2. The coefficient $a^0_{i,j}$ represents the internal energy of the current control volume divided by the time-step.

4.2.1 The Alternating Direction Implicit Method

Equation 4.40 is solved using the 2$^{\text{nd}}$ order Alternating Direction Implicit (ADI) method [92]. This method solves the system of equations in two iterations or directionally-biased sweeps. First, Eq. 4.40 is rearranged so that the coefficients $a_{i,j}$ fit well into a matrix as shown below.

$$-a_{i+1,j}T_{i+1,j} - a_{i-1,j}T_{i-1,j} + a_{i,j}T_{i,j} - a_{i,j+1}T_{i,j+1} - a_{i,j-1}T_{i,j-1} = b$$

This coefficient matrix forms the A in the classic $Ax = b$ problem, where the solution vector x represents the temperature in each control volume. While ADI is the method presented here to obtain the solution, a fully implicit solution algorithm, such as an iterative Krylov will also work well, albeit at a higher cost. This implicit formulation creates a sparse banded matrix which is numerically expensive to solve. By using the ADI method, the problem is reduced to a series of directionally-biased iterations that only need to solve a tri-diagonal matrix, which is much cheaper and is solved directly with the Thomas algorithm. Each ADI iteration sweeps over a specific dimension while the remaining dimensions are held fixed at some intermediate time-step, e.g. $\Delta t_{1/2}$. The problem is also simplified by linearizing thermal conductivity, by fixing the value of κ over the entire time-step, rather than non-linearly iterating to compute $\kappa \sim T^{5/2}$.

In 2D, the ADI method solves Equation 4.40 in two sweeps: 1) solve for along x-direction and hold y fixed in time, 2) solve for along y-direction and hold x fixed from the previous iteration. The first iteration in x is defined by:

$$\begin{aligned} -a_{i-1,j}T^{n+1/2}_{i-1,j} + (a_{i+1,j} + a_{i-1,j} + a^0_{i,j})T^{n+1/2}_{i,j} - a_{i+1,j}T^{n+1/2}_{i+1,j} = \\ (a_{i,j+1}(T^n_{i,j+1} - T^n_{i,j}) + a_{i,j-1}(T^n_{i,j-1} - T^n_{i,j}) + a^0_{i,j}T^n_{i,j}) + \dot{q}_{i,j}\Omega_{i,j} \end{aligned} \tag{4.48}$$

In matrix form this results in the following:

$$\begin{bmatrix} a_{i,j} & -a_{i+1,j} & & 0 \\ -a_{i-1,j} & \ddots & \ddots & \\ & \ddots & \ddots & -a_{M-1,N} \\ 0 & & -a_{M,N} & a_{M,N} \end{bmatrix} \begin{bmatrix} T_{i,j} \\ T_{i+1,j} \\ \dots \\ T_{M,N} \end{bmatrix} = \begin{bmatrix} b_{i,j} \\ \dots \\ \dots \\ b_{M,N} \end{bmatrix} \tag{4.49}$$

where b_{ij} is the right-hand-side term of Eq. 4.48. The source term $\dot{q}_{i,j}$ is held constant for the time-step Δt, and T^n is the control volume temperature at the current time-step. This system of equations provides the temperature solution vector of $\vec{T}^{n+1/2}$ at the next half time-step, which will be used in the y-sweep. The y-sweep is defined in Equation 4.48. Note the use of the

temperature solution from the previous half time-step ($T^{n+1/2}$).

$$\begin{aligned} -a_{i,j-1}T_{i,j-1}^{n+1} + (a_{i,j+1} + a_{i,j-1} + a_{i,j}^{0})T_{i,j}^{n+1} - a_{i,j+1}T_{i,j+1}^{n+1} = \\ (a_{i+1,j}(T_{i+1,j}^{n+1/2} - T_{i,j}^{n+1/2}) + a_{i-1,j}(T_{i-1,j}^{n+1/2} - T_{i,j}^{n+1/2}) + a_{i,j}^{0}T_{i,j}^{n+1/2}) + \dot{q}_{i,j}\Omega_{i,j} \end{aligned} \tag{4.50}$$

The matrix form is as follows.

$$\begin{bmatrix} a_{i,j} & -a_{i,j+1} & & 0 \\ -a_{i,j-1} & \ddots & \ddots & \\ & \ddots & \ddots & -a_{M-1,N} \\ 0 & & -a_{M,N} & a_{M,N} \end{bmatrix} \begin{bmatrix} T_{i,j} \\ T_{i,j+1} \\ \dots \\ T_{M,N} \end{bmatrix} = \begin{bmatrix} b_{i,j} \\ \dots \\ \dots \\ b_{M,N} \end{bmatrix} \tag{4.51}$$

Now $b_{i,j}$ is the right-hand-side term of Eq. 4.50, which uses the temperature solution vector from the previous ADI direction sweep. Note that the matrices for the two sweeps are the same size, but the change in direction requires special care, particularly due to boundary conditions. For example, if the y-sweep follows the x-sweep, the solution vector must be transposed to maintain the same domain shape.

4.2.2 Boundary Conditions

Boundary conditions are handled through the use of ghost (halo) layers (similar to hydrodynamics). These are control volumes that extend past the physical domain of the problem, typically by one volume, that are used to impose boundary conditions. For example, zero-gradient boundary conditions ($\partial T/\partial x = 0$) are imposed by enforcing that the ghost layer have the same temperature as the edge of the physical domain.

4.2.3 Spitzer Thermal Conductivity

Spitzer [90] thermal conductivity for a fully ionized plasma is used for all of the ICF-related simulations in this dissertation. This is defined as

$$\kappa_e = \left(\frac{8}{\pi}^{3/2}\right)\left(\frac{k_B^{7/2}}{e^4\sqrt{m_e}}\right)\left(\frac{1}{1+(3.3/\bar{z})}\right)\left(\frac{T_e^{5/2}}{\bar{z}+\lambda_{ei}}\right) \tag{4.52}$$

where m_e is the electron mass, e is the fundamental charge, k_B is the Boltzmann constant, $\bar{z}$ is the average atomic z value, T_e is the electron temperature, and λ_{ei} is the electron-ion coulomb logarithm. The electron-ion coulomb logarithm is defined as:

$$\lambda_{ei} = \max[1,\ \ln(b_{\max}/b_{\min})] \tag{4.53}$$

For high-energy-density plasma, $b_{\max} \approx b_{\min}$, so λ_{ei} can become small or negative, so setting a floor value of 1 avoids this, as in *FLASH* [93]. The minimum and maximum impact factors $b_{\max}$ and $b_{\min}$ are defined by the following equations.

$$b_{\max} = \sqrt{\frac{k_B T}{4\pi e^2 n_e}} \tag{4.54}$$

$$b_{\min} = \max\left[\frac{ze^2}{3k_B T_e}, \frac{\overline{h}}{2\sqrt{3k_B T_e m_e)}}\right] \tag{4.55}$$

All values are defined in the centimeter-gram-second (CGS) system with eV temperature units. The physics models assume a single temperature and material, so $\kappa = \kappa_e$ and $T = T_e$.

Chapter 5

Multi-physics: Code Development

High-performance computing systems are becoming increasingly complex in the exascale era (systems capable of 10^{18} floating-point operations per second). As of 2020, the newest clusters coming online at the leading DOE laboratories are heterogeneous, and utilize a combination of both CPUs and GPUs from various vendors such as Intel, AMD, and NVIDIA. This presents a major challenge to scientific developers wishing to update existing software to work in a heterogeneous environment. Porting legacy code (primarily written in Fortran or C/C++) is a significant undertaking, and its success is highly dependent on developer productivity, vendor support (both software and hardware), and libraries available to enable the transition. Addressing this problem, by creating tools that aid scientific software developers who need to adapt pre-existing code to run on heterogeneous systems at extreme scales, is one of the tasks of the Department of Energy's Exascale Compute Project (ECP). These libraries are primarily focused on Fortran, C/C++, Python, and GPU languages such as CUDA or OpenCL.

The interdisciplinary study *Multiphysics Simulations: Challenges and Opportunities*, by D. Keyes et. al [94], identifies several needs and challenges for successful future multi-physics software. Future performance portable HPC software must meet the following needs:

1. Interfaces with appropriate levels of encapsulation and granularity while permitting adequate performance

2. Common infrastructure and ability to exploit commonality among different physics operators
3. Strong focus on meeting practical user expectations (ease of building, documentation, support, and transparent open-source licensing)
4. Extensibility to the largest current and anticipated future problems and to future extreme-scale systems

Some of the challenges to meeting these needs include:

1. Enabling introduction of new models, algorithms, and data structures
2. Creating interfaces that are independent of physical process, algorithms, and data structures
3. Designing intuitive high-level abstractions
4. Dealing with changes in architecture (shared vs distributed memory, CPUs vs GPU, etc.)
5. Developing and analyzing multi-physics coupling operators

These set a high standard that requires a series of breakthroughs in both code and algorithm designs, and science-oriented development often lags in adopting best practices learned in the software engineering industry [95]. The study also establishes a series of *insertion mechanisms* through which these breakthroughs could occur. Two of these mechanisms are: 1) small side-efforts with minimal side effects (undertaken by a single developer who can devote significant time) and 2) full rewrites, which presents a significant risk and investment. While this dissertation makes no claims on achieving ideal design and performance, it does however seek to test the viability or proofs-of-principle for applying different hydrodynamic methods and software design. This classifies as a "small" side-effort that has limited side effects outside this current work.

Multi-physics codes are traditionally written in Fortran or C/C++ for performance, and are sometimes combined with a scripting language interface (such as Python) for user interactivity. One

of the goals of this dissertation is to investigate the viability of different approaches to multi-physics software development that improves developer productivity, scales well to large systems, and accommodates algorithm experimentation through a flexible design. This includes applying parallelism using Fortran coarrays in an object-oriented design. Fortran excels at array-based numerical computation, yet, even as a high-level language, it struggles with other general tasks outside of this paradigm. Object-oriented language features were added in the 2003 standard, which help alleviate this issue to some extent, and native parallelism was also added to the 2008 standard, in the form of *coarrays* [96]. Traditionally, parallelism is added with compiler directives (specialized syntax) via OpenMP [97] (shared memory) or MPI [98] (distributed memory) which can create verbose code, but coarrays offer a performance portable solution with simple native Fortran syntax. While MPI is well-supported in the high performance computing community and is the de-facto method of distributed parallelization, coarrays offer an compelling alternative with performance on par with MPI [99].

Julia [100], a relatively new language comparatively, offers an attractive alternative due to its unique combination of high performance and dynamic syntax. Julia is designed specifically for numerical and scientific computing with an emphasis on performance and usability. The unique combination of clear syntax and performance provide a desirable environment for increased developer productivity in designing modern multi-physics software that is performance-portable across a variety of heterogeneous systems.

The first of the following sections (Sec. 5.1) details the development and design of *Cato*, a single-physics finite-volume hydrodynamic code written in object-oriented, coarray Fortran. This code was developed to test various hydrodynamic solver methods (i.e. Riemann solvers) with high spatial order and low dissipation to simulate the evolution of internal defects in ICF targets (which is presented in Chapter 2). However, while the high-order hydrodynamic methods performed well at capturing characteristic wave evolution, extensive testing revealed that thermal conduction needed to be added to properly account for laser energy deposition. Using a pressure-drive via boundary conditions or source terms (without a thermal conduction zone

in the corona) created spurious waves that limited its usefulness in simulating implosion-like problems. While adding thermal conduction did not require a complete re-write of *Cato* (due to its modular design), it presented a unique opportunity to be at the forefront of applying Julia to multi-physics problems. As of 2020, Julia is not as extensively used in the HPC community compared to existing languages such as C/C++, Fortran, and Python, but its adoption rates have been steadily increasing [101]. Section 5.2 presents the design and development of *Cygnus*, a parallel, multi-physics code written exclusively in Julia. Much of the hydrodynamic capability was carried over from *Cato* into *Cygnus*, and a comparison of performance and design is presented in Sections 5.3-5.4.1.

5.1 Hydrodynamics Implementation in Modern Fortran

Cato is a modern Fortran (2008+) code written to solve the 2D Euler equations for a single material on a fixed Cartesian grid to investigate the propagation of characteristic waves created by internal defects in ICF targets. Following the recommendations of the aforementioned interdisciplinary study of multi-physics software, the software-oriented goals of *Cato* are threefold: 1) A flexible and extensible design to facilitate algorithm experimentation, 2) Performant parallelism for large problems, 3) Clear and readable implementation for maintainability. Object-oriented design and abstract data type (ADT) calculus provides flexibility and readability. Flexibility is needed to enable algorithm experimentation to select the best performing Riemann solver and integration methods. Both fine- and coarse-grained parallelism are combined to performantly scale for large problems.

5.1.1 High-level Design

5.1.1.1 Object-Oriented Interfaces

Software is typically written in three primary styles: procedural, functional, and object-oriented [102, 103]. Procedural style is based on the concept that software consists of a list of functions, subroutines, or "procedures" that are the list of steps the computer follows to solve a problem.

Functional programming, as the name suggests, emphasizes solving the problem through the composition of many functions. This involves chaining functions together to achieve the desired output. *Pure* functional programming emphasizes the creation of functions that have no side-effects (no changes to its inputs), which is advantageous when concurrency and parallelism are required. Object-oriented design seeks to solve problems through the interaction of high-level objects that best fit in the context of the problem.

Fortran is an imperative programming language designed to translate formulas into code, and traditionally follows the procedural style. Imperative languages focus on how the program behaves. It is not a true functional language like Clojure, Haskell, or others, but certain aspects of this style are possible, including the use of the `pure` keyword for functions or subroutines. Object-oriented features were not added to the language standard until 2003, but Fortran can now incorporate aspects of each paradigm [96].

Object-oriented programming revolves around the idea of a class, or object, and builds on four main concepts of *abstraction*, *encapsulation*, *inheritance*, and *polymorphism*. *Abstraction* hides the details of computation underneath an object that is simpler to understand at the interface level. *Encapsulation* limits information access, which protects (or hides) data from other portions of the code that don't need to know about, or have the ability to change, the data inside a particular module. *Inheritance* establishes a hierarchy that allows for code-reuse and relationships between different objects or types that are relevant to a specific problem. Inheritance applies to both behavior and data. If done well, inheritance can vastly simplify the problem and aid the developer by limiting the amount of code that needs to be maintained. Finally, *polymorphism* allows functions, or methods, to be used on different classes, allowing for re-use and improved maintainability.

Object-oriented design was used within *Cato* for maximum flexibility and maintainability. Many of the design patterns such as Puppeteer, Factory, and Strategy were incorporated into *Cato* from the well-known software development book *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma, Helm, Johnson, and Vlissides [104]. These patterns facilitate

creational, structural, and behavioral patterns that aid in designing the code at the interface level rather than the implementation. This allows implementations to be optimized and modified without interfering with the high-level design. At the beginning of code development, it was clear that there would be a need to test multiple Riemann solvers, integration methods, and data structures to find the best fit for the physics problem. Using a class structure vastly simplified the work required to experiment with different algorithms. For example, when a different Riemann solver needed to be tested, it only required creating a new class that inherited from the high-level solver class (that every Riemann solver inherited from). This meant that nothing else in the code needed to be updated or modified since the high-level behavior was already predetermined. The implementation of the solver was left to the solver class, of which there were many different versions (i.e. HLLC, AUSM, SLAU, etc.) to choose from. This benefit extended to other modules of the code and also made ADT calculus possible.

5.1.1.2 Abstract Data Type Calculus

Rouson et. al. define *Abstract Data Type calculus* as the "valuation of expressions involving the application of mathematical operators to ADT instances" [105]. In other words, high level objects interacting through mathematical operators, similar to what one would see in a math or physics textbook. ADT calculus combines elements of object-oriented and function programming styles to achieve performance and readability.

In modern Fortran, a user-defined type such as `type(field_t)::T` could represent a scalar field of some physical significance. If the temporal derivative was defined mathematically as $\partial T/\partial t = \alpha\nabla^2 T$, then the corresponding code representation would look something like `dT_dt=alpha*laplacian(T)`. Both `dT_dT` and `T` are `field_t` types, and `alpha` is a `real`. Here the implementation of the Laplacian on a field is left to the function `laplacian`, but the syntax is immediately recognizable in the specific context, and the code documents itself. Polymorphism of the function would dictate the implementation based on the dimensionality of the field (1D, 2D, or 3D) or (`field_1d_t`, `field_2d_t`, or `field_3d_t`), but again, the implementation (and performance-critical code) is underneath.

In another example, the first stage in a Runge-Kutta temporal integration scheme is represented by the mathematical expression $U^{n+1} = U^n + \partial_t U \Delta t$. The physical state of U^{n+1} depends on the current state U^n, time-step Δt, and temporal derivative $\partial_t U$. In code, this can be similarly stated as `U = U + U%t()*dt`. In this example, `U%t()` represents calling the `t()` function defined in the `U` class that implements the temporal derivative calculation of `U`. The mathematical operators `+` and `*` must be re-defined, or overloaded, in order to inform the compiler how to perform the operations.

These two simple examples demonstrate how ADT calculus can encapsulate physical constructs that make sense to domain scientists and engineers. The code should be easy to follow, look as close to the math as possible, and still maintain the high performance features needed to run large multi-physics simulations.

Listing 5.1 defines a simple type, similar to the one used in *Cato*, that encapsulates each of the physical fields needed to represent a `fluid` state. This type consists of the primitive variables of density, velocity, and pressure. Because it is a user-defined type, it must re-define, or overload, operations such as `=+-*/` in order to be used in a mathematical expression. While Fortran defines the behavior of `a=b*c` where `a,b,c` are base types such as `real` or an array of `real`, it does not apply to user-defined types. The option to define objects and overload their methods was introduced in the Fortran 2003 standard, which makes ADT calculus possible. In Listing 5.1, the assignment (`=`) and addition operators are defined by the `assign_fluid` and `fluid_add_[fluid,real,int]` procedures. Any number of overloaded procedures are possible, depending on the level of need for the type.

Listing 5.2 shows an example of how this type is used with both the assignment and addition operators. The implementation of how this is accomplished is hidden underneath the operators, leaving the high level interface clean and understandable. Line 3 will call `fluid_add_fluid()`, `fluid_add_real()`, and then `assign_fluid()` behind the scenes. One criticism of ADT calculus revolves around memory allocation and re-use. In the example, due to the pure functional-like operations in `+` and `=`, data is copied rather than updated in-place, which can be limited

```
type fluid_t
  real(real64), dimension(:, :), allocatable :: rho !< (i, j); density
  real(real64), dimension(:, :), allocatable :: u !< (i, j); x-velocity
  real(real64), dimension(:, :), allocatable :: v !< (i, j); y-velocity
  real(real64), dimension(:, :), allocatable :: p !< (i, j); pressure
contains
 generic, public :: assignment(=) => assign_fluid
 generic, public :: operator(+) => fluid_add_fluid, fluid_add_real,
     fluid_add_int
end type
```

Listing 5.1: Defining a fluid type with overloaded operators for `+` and `=`

```
   type(fluid_t) :: fluid_A, fluid_B, fluid_B
   real(real64) :: x = 1.0_real64
   fluid_A = fluid_B + fluid_C + x
```

Listing 5.2: Using the `fluid_t` type

due to memory-bandwidth capabilities. However, proper optimization and careful data layout design can alleviate this issue. One of the trade-offs in this situation is the balance between asynchronous side-effect free pure functions versus in-place subroutines. Pure functions eliminate unintended side-effects, but memory copies can be expensive if not dealt with properly. Table 5.1 summarizes a few examples of ADTs used in *Cato*, both abstract and concrete. Abstract types in Fortran can hold data, but the implementations of the methods or functions attached to them are deferred to the concrete types that inherit them. In the spirit of ADT calculus, each data type in the table is designed to fit well within the context of the problem, i.e. hydrodynamics, and hide the implementation underneath the interface. Examples include `fluid_t` to handle the fluid state and methods, `grid_block_t` to hold the geometry information, `muscl_interpolation_t` to manage the spatial reconstruction and determine the spatial accuracy of the problem, and `flux_solver_t` to handle the Riemann solver to compute the fluxes at the edges of each cell.

An example of a concrete type inheriting from an abstract type can be seen with `boundary_condition_t` and `periodic_bc_t`. The abstract `boundary_condition_t` type does

not define any particular behavior, other than the `apply()` method, which is deferred. It only contains data required to apply a boundary condition, such as location or current time (for time-dependent cases). This means that any concrete type that inherits from it must define the behavior of `apply()`. So the `periodic_bc_t` class, which is an ancestor of `boundary_condition_t`, defines how to apply a periodic boundary condition given the particular input. This relationship makes it possible for any classes that interact with boundary conditions to only need to use the abstract type. Polymorphism allows the code to work with different types without having to make complicated logic when the object is used. This makes it incredibly simple to apply boundary conditions in the spatial reconstruction phase, for example, since the calling function only needs to call `apply()` to the list of boundary condition types held in the `fluid_t` class.

Table 5.1: *Cato* ADT Examples

Abstract Type	Concrete Type Example	Description
N/A	`master_pupeeteer_t`	Driver class to manage the simulation
`grid_block_t`	`grid_block_1d_t`, `grid_block_2d_t`	Store mesh variables, compute volumes, load initial conditions
N/A	`fluid_t`	Hold fluid relevant data: fluid state, boundary conditions, flux solver, etc.
`muscl_interpolation_t`	`muscl_tvd2_t`, `mlp5_t`	Spatial reconstruction methods (TVD, MLP)
`flux_solver_t`	`hllc_solver_t`, `ausmpw_plus_solver_t`	Riemann solvers like HLLC, M-AUSMPW$^+$, and others
`boundary_condition_t`	`periodic_bc_t`, `outflow_bc_t`, `symmetry_bc_t`	Manage and apply boundary conditions like periodic, symmetry, outflow, etc.
N/A	`contour_writer_t`	Manage I/O of XDMF and HDF5 files
N/A	`eos_t`	Manage equation of state calculations

```
subroutine ssp_rk_2_2(U, grid, error_code)
  !< Strong-stability preserving 2nd order Runge-Kutta
  class(fluid_t), intent(inout) :: U ! fluid
  class(grid_block_t), intent(in) :: grid ! geometry
  integer(ik), intent(out) :: error_code ! error handling
  type(fluid_t), allocatable :: U_1 ! first stage
  real(rk) :: dt, time

  dt = U%dt
  time = U%time

  allocate(U_1, source=U)

  ! Stage 1
  U_1 = U + U%t(grid) * dt

  ! Stage 2
  U = 0.5_rk * U + 0.5_rk * U_1 + (0.5_rk * dt) * U_1%t(grid)

  call U%sanity_check(error_code) ! Check for negatives, NaNs
  call U%calculate_derived_quantities()

  ! Convergence history
  call write_residual_history(first_stage=U_1, last_stage=U)
  deallocate(U_1)
end subroutine ssp_rk_2_2
```

Listing 5.3: Implementation of the Strong-Stability Preserving, 2nd Order Runge-Kutta time integration

Listing 5.3 shows a code segment from *Cato* that does 2nd order Runge-Kutta temporal integration using ADT calculus. The mathematical construct for the 2 stage integration process is the following:

$$U^{(1)} = U^n + \partial_t U^n \Delta t \tag{5.1}$$

$$U^{n+1} = \frac{1}{2} U^{(1)} + \frac{1}{2} \partial_t U^{(1)} \Delta t \tag{5.2}$$

where U represents the conserved state vector of the fluid, $^{(1)}$ is the stage number, ∂_t is the derivative with respect to time (which is provided by the Riemann solver), and Δt is the time-step.

The implementation in the code listing closely follows the mathematical notation and is simple to follow due to ADT calculus and operator overloading. The actual SSP-RK2 stage calculation is achieved in only two lines of code (15 and 18), while the rest handles argument input/output, error checking, and memory management. The subroutine also computes the convergence history and derived quantities such as sound speed, Mach number, and the primitive state vector (ρ, u, v, p) needed by other portions of the code, since the integration scheme uses the conserved state vector $(\rho, \rho u, \rho v, \rho E)$.

Note the use of the types `fluid_t` and `grid_block_t`. The `class` keyword tells the compiler that the type will be polymorphic, e.g. `grid_block_t` could be 1D or 2D, but it does not make a difference regarding the temporal integration. The temporal derivative is handled by the `U%t(grid)` call where the geometry information in the `grid` is passed to the fluid type `U`, and ultimately calls the Riemann flux solver to determine the new state.

5.1.1.3 Solving the Hydrodynamic Equations

Cato uses the ADT paradigm to facilitate the high-level interaction between the grid, fluid state, flux solver, I/O, and more. The following list provides the basic step-by-step process that *Cato* follows to solve the Euler equations.

1. Initialize the grid, state variables, and boundary conditions - this generates types to manage operations, e.g. `grid_t`, `fluid_t`, `boundary_condition_t`, and `edge_split_flux_solver_t`.

2. Find the time-step – The `master_pupeeteer_t` type delegates this to `fluid_t` , which determines Δt via the CFL stability condition.

3. Integrate in time – again, this is initiated by the `master_puppeteer_t` type. Each physics module handles it's own time integration method.

4. The `fluid_t` object calls the time-integration subroutine (i.e. a multi-stage Runge-Kutta explicit scheme).

5. The time-integration scheme calls the particular flux solver to provide $\partial\vec{U}/\partial t$ at each stage.

6. The flux solver calls the `boundary_condition_t` methods to apply boundary conditions.

7. The flux solver initiates the spatial reconstruction scheme (e.g. `muscl_interpolation_t`), which provides the edge interface values of each cell using a 2^{nd} order (or higher) function.

8. The cell interface values $(\bar{\Phi}_L,\ \bar{\Phi}_R)$ are passed to the Riemann solver flux function, e.g. HLLC, to provide the flux values for $\partial\vec{U}/\partial t$.

9. This process is repeated for each stage, depending on the R-K scheme in use - calculate convergence history after the final stage.

10. Derived quantities such as sound speed and Mach number are calculated after the time-integration in finished.

11. Write contour files to disk at user-specified intervals.

12. Loop steps 2-11 until the user-specified end criteria, such as maximum iterations or simulation time.

5.1.2 Parallelization

Cato achieves parallelization through two paradigms: single-program, multiple-data (SPMD), and single-instruction, multiple-data (SIMD). These represent the two extremes of parallel *granularity*, or the amount of work that is performed by a particular task. SPMD is *coarse-grained* parallelism and splits the problem into large tasks via domain decomposition. Vectorization, or SIMD, is *fine-grained* parallelism that breaks the problem into many small tasks at the loop level. Each parallelization style is intended to maximize the efficiency of the algorithm and hardware utilization.

5.1.2.1 Coarse-grained Parallelism: Domain Decomposition

Cato utilizes domain decomposition to achieve the SPMD paradigm. This approach splits up the logic, or "program", and divides the workload across multiple processors (in shared or distributed memory). This is traditionally done with message passing, or MPI, but *Cato* uses coarrays, a relatively new addition to the Fortran standard.

Coarrays were added to the Fortran 2008 standard to provide native parallelism based on the Partitioned Global Address Space (PGAS) programming model [96]. PGAS makes the most sense when compared to existing memory models, such as shared memory and distributed memory. In a shared memory environment, e.g. a multi-threaded CPU within a single node, each thread can access any portion of the memory, since it is shared by all cores. Work can be divided between threads at the loop level without any extra work to properly access portions of the memory. In a distributed memory environment, as in an HPC cluster with multiple nodes connected via a network (e.g. Infiniband or Ethernet), each process can only access the local memory on the node it lives in. Data must be passed via messages, e.g. MPI, across the network for any global operations. In the PGAS model, global memory can be accessed by any remote process with syntax similar to a shared memory system. In the coarray model, global memory operations are implemented via one-sided operations that do not necessarily require two-way syncing, as is often used in MPI. MPI does include one-sided communication, however, and most coarray compiler implementations use MPI underneath.

In coarray terminology, the P, or program in SPMD, is known as an *image*. Typically the number of images corresponds to the number of processors used to split up the work. An example of coarray syntax is shown below:

```
1    real(real64), dimension(:,:), codimension[*] :: A
```

Listing 5.4: Declaring a 2D 64-bit real coarray in Fortran 2008+

This declares that `A` is a 2D array that will have multiple images of dimension `codimension[*]`. In order to access the data of `A` on the current image or process, the syntax is unchanged, e.g.

`A(i,j)`. This implies `A(i,j)[current_image]`. Accessing the same data on a different image requires syntax such as `A(i,j)[image_id]`. Transferring data from image 1 to image 2 can be accomplished by `A(:,:)[2] = A(:,:)[1]`. Accessing the current id of the processor is accomplished via `this_image()` and the number of total images as `num_images()`. Syncing over the global domain is done with `sync all`. Localized syncing can be achieved with a unique set of image ids, such as nearest neighbors, with syntax such as `sync images(list_of_neighbors)`. Specific restrictions as to how coarrays are used can be found in the standard or in compiler manuals.

Figure 4.3 shows a diagram illustrating how *Cato* utilizes domain decomposition. In the figure, each block represents a portion of the domain. In this example, the global domain is split between 4 processors (note that the IDs start with 1 rather than 0 as in MPI to follow Fortran convention). Global domain synchronization is done by "halo", or "ghost" cells. These cells are extensions of the domain to facilitate communication with neighbors or boundary conditions. The halo cells are shown in the figure with the dashed boundaries, and the real domain uses solid boundary lines. The size of the halo region can vary depending on the algorithm required for the problem. Higher-order methods use more than one neighboring cell in stencil calculation, and in *Cato* the halo region is typically 2 or 3 halo cells.

In *Cato*, the boundaries of the domain are referred to by the `ilo/jlo` or `ihi/jhi` identification, which correspond to the minimum and maximum indices of the array in `i` or `j` dimensions. In the figure, image 3 syncs the i boundary with image 4. Here the `ihi` edge of image 3 is passed to the `ilo` halo edge on image 4. The `ilo` edge of image 4 is passed to the `ihi` halo edge of image 3. This edge synchronization is done with all neighbors, including corners (if required by the algorithm). The domain in the figure has periodic boundaries on all edges.

In *Cato*, the type `field_2d_t` encapsulates a generic scalar physical quantity or field, such as density, on an isolated decomposed domain based on the strategy provided in Ref. 106. It manages the halo exchange with nearby domains and overloads the arithmetic operators so that mathematical operations with this type are simple to read. It also handles operations like gather,

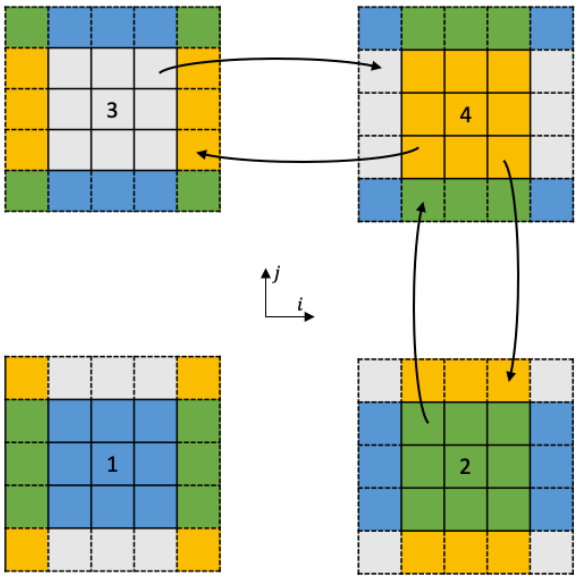

Figure 5.1: The halo cell exchange between neighboring processes (1-4). Halo cells have dashed borders and domain cells have solid borders. Cells are colored by the originated domain process. Data movement and direction is indicated by the arrows.

which is useful in moving all of the data onto a single process for I/O if required.

Overloading the arithmetic operators `+ - * /` as well `minval()`, `maxival()`, `minloc()`, `maxloc()` makes it simple to use each `field_2d_t` in other ares of the code. For example, the formula to calculate the total energy in an ideal gas is:

$$E = \left(\frac{p}{\rho(\gamma - 1)} \right) + \frac{1}{2}(u^2 + v^2)$$

and the corresponding code implementation using `field_2d_t` types is:

```
E = (p / (rho * (gamma - 1))) + (0.5 * (u * u + v * v))
```

In the code, each variable (`E, p, rho, u, v`) is a `field_2d_t` type and the syntax is identical to the mathematical formula. This single line will run this operation across all parallel domains, whether 2 or 2000. The details of implementation are abstracted away underneath the types, and can be further optimized later, if the need arises. The `field_2d_t` type has the option to sync automatically on any assignment (`=`), but in the total energy example above no neighbor information is required, so synchronization can be skipped to eliminate communication costs.

```
1  subroutine sync_edges(self)
2    class(field_2d_t), intent(inout) :: self
3    real(real64), dimension(:, :), allocatable, save :: ilo_edge[:]
4    ...
5    ! synchronize with the set of unique neighboring coarray images
6    sync images(self%unique_neighbors, stat=sync_stat, errmsg=sync_err_msg)
7    ...
8    ! send ihi edge data to the ilo edge of the neighbor on ihi side
9    ilo_edge(:, :)[self%neighbors(ihi_neighbor)] = &
10              self%data(ihi - nh + 1:ihi, jlo:jhi)
11   ...
12   ! synchronize with the set of unique neighboring coarray images
13   sync images(self%unique_neighbors, stat=sync_stat, errmsg=sync_err_msg)
14   ...
15   ! Now copy the edge data to the halo cells of the current image
16   if (.not. self%on_ilo_bc) then
17     self%data(ilo_halo:ilo - 1, jlo:jhi) = ilo_edge
18   end if
19   ...
20 end
```

Listing 5.5: Edge synchronization using the `field_2d_t` type

Communication, or edge synchronization, can be the most bug-prone portion in the code due to simple logic or accounting errors, as shown in Listing 5.5. This subroutine is taken from *Cato* with the most pertinent lines included. The lines removed from the snippet only check for errors and manage memory allocation.

In the listing, the subroutine is a method attached to the `field_2d_t` class. The `self` term in the subroutine refers to the object or type that owns the method. This `sync_edges` subroutine is called explicitly, or whenever an assignment is made, e.g. `=`. The `ilo_edge` acts as a buffer object to hold the halo cell data on the `ilo` side. At line 9, the `ihi` edge region (from `self%data`) is copied to an `ilo_edge` coarray living on the adjacent neighbor image `self%neighbors(ihi_neighbor)`. Here, the `ihi - nh + 1:ihi` indexing accounts for the thickness of the halo region. Lines 6 and 13 sync adjacent neighbor processes before the halo exchange, which is required to ensure data correctness. After the sync, the coarray data is assigned to the adjacent neighbor's halo region on the `ilo` edge at line 13. This is the same data commu-

nication as the uppermost arrow in Figure 5.1 between processes 3 and 4 (gray with solid border to gray with dashed borders). The listing excludes the remaining `ihi`, `jlo`, and `jhi` edges since the logic follows the same process as `ilo`.

In the explicit SSP Runge-Kutta time integration schemes (Eqs. 4.5 and 4.6), edge synchronization must occur at each stage. This has implications for parallel scaling performance, when communication costs can dominate runtimes. This can occur when the cost of integrating each decomposed domain is less than the cost of communicating halo cells. Implementing alternate time integration algorithms with reduced communication frequencies or careful selection of domain size vs number of processes can alleviate this issue.

5.1.2.2 Fine-Grained Parallelism: Threading and Vectorization

Cato applies fine-grained parallelism through the extensive use of vectorization and threading. Vectorization groups arithmetic instructions so that they can be applied to more than one element of data at a time. A high level of vectorization is absolutely necessary to maximize hardware utilization, specifically with CPUs. Fine-grained threading divides the work into small tasks within a shared-memory context, typically at the loop level. Multi-threading is achieved with OpenMP directives that surround loops in *Cato*. An example in Listing 5.6 highlights the use of these directives to parallelize a trivial loop.

The `!$omp parallel` directive tags a particular region as parallel. Within this block, the `!$omp do` tags specify that a loop is to be split among threads, and the `default`, `private` and `shared` keywords indicate how memory is to be protected (or not) between threads. OpenMP threading can be enabled or disabled at compile-time based on the desire of the user. Threading at this level typically only scales well when there is sufficient work to be done by each thread. For *Cato*, threading is not enabled by default, since domain decomposition is the primary method of parallelization.

However, SIMD vectorization is enabled by default in *Cato*. OpenMP provides an industry-standard, portable method of enabling vectorized loops, since compiler vendor implementations

```
module subroutine field_add_field_cpu(lhs, f, res)
  !< Implementation of the field_2d_t + field_2d_t operation
  class(field_2d_t), intent(in) :: lhs !< left-hand-side of the operation
  class(field_2d_t), intent(in) :: f
  type(field_2d_t), intent(inout) :: res

  integer(ik) :: i, j

  !$omp parallel default(none), &
  !$omp private(i, j) shared(lhs, res, f)
  !$omp do
  do j = lhs%jlo, lhs%jhi
    !$omp simd
    do i = lhs%ilo, lhs%ihi
      res%data(i, j) = lhs%data(i, j) + f%data(i, j)
    end do
    !$omp end simd
  end do
  !$omp end do
  !$omp end parallel
end subroutine field_add_field_cpu
```

Listing 5.6: Loop-level parallelism in an ADT arithmetic operation

and syntax vary. Inner loops like the one in Listing 5.6 enable vectorization with the `!$omp simd` directive.

Vectorization applies a single instruction to a group of elements, so data must be localized in memory and fit inside the particular vector lengths that the hardware and compiler supports. Recent compilers must be used to enable the specific instruction sets for the latest hardware. For example, a modern CPU with AVX-512 vectorization can use a vector 512-bit wide. This will apply a single instruction, e.g. add, multiply, etc., to 8 64-bit (double precision) numbers in a single CPU cycle. Without vectorization, this would take 8 separate cycles, or *scalar* instructions. With the proper data layout, this provides a distinct advantage to code developers.

Branch-free loops and data locality are two requirements for vectorization. Branch-free loops are loops that do not contain if-then statements that create different instruction sets for portions of the loop, which cannot be vectorized. Data locality or contiguous memory access is ensured by keeping the data within array structures. This aids vectorization by making it simple for the

hardware to place the data into a vector to apply the instruction. *Cato* achieves this through contiguous arrays within types, such as the `fluid_t` in Listing 5.1. These Structure of Array (SoA) types are the most amenable to vectorization on both CPUs and GPUs due to data locality. Details regarding trade-offs between data structures and best-practices for vectorization can be found in *"Parallel and Higher Performance Computing"* by Robey and Zamora [103].

5.1.3 Input and Output

Cato is controlled by a simple text input file in the `.ini` format. Initial conditions and the mesh are provided by an HDF5 file. These are usually generated by a Python script before running *Cato*.

Industry-standard file formats are a key component of *Cato*'s design. Using formats that are well-supported by external libraries or software packages make the task of both the user and developers simpler. *Cato* uses the `.ini` file format because it is simple to read, many text editors understand its syntax for highlighting (which makes writing them less error-prone), and third-party libraries are available to read and write `.ini` files.

Parsing the input file inside *Cato* was achieved through the use of a freely-available library (`cgfio`) made for modern object-oriented Fortran. This parser made it simple to apply defaults, check for errors, and provide meaningful error messages in the event of a user error. While Fortran namelists (`.nml`) files work well for less complex problems, `.ini` files are better supported elsewhere and make the user's job easier. Listing 5.7 shows a portion of an `input.ini` file used to run a *Cato* simulation.

Input and output of "heavy" data, or field & geometry data, is handled by HDF5 files. Because of widespread adoption throughout the industry, HDF5 files are straightforward to read and write in different languages. *Cato* itself handles HDF5 files with the third-party library `h5fortran`. This is a modern object-oriented Fortran library freely available online that makes HDF5 I/O incredibly simple.

```
[general]
title = "2D Sedov Blast Wave"

[time]
; time is in seconds
max_time = 2.0e-9
integration_strategy = "ssp_rk3"
cfl = 0.6

[scheme]
flux_solver = "M-AUSMPW+"
spatial_reconstruction = "MUSCL"
limiter = "MLP5"

[initial_conditions]
read_from_file = true
initial_condition_file = "sedov.h5"
```

Listing 5.7: An example of an input file used to run *Cato*

The heavy data is accompanied by an XML file using the XDMF format. These are written at user-specified intervals and contain the primary physical state of the problem, namely: density, velocity, pressure, and sound speed. The XDMF file format allows the developer/user to save the connectivity and location of the data, e.g. light data, in a human-readable XML format, and the heavy-data or arrays, in a separate HDF5 file. Since XDMF is also an industry-standard file format, post-processors such as ParaView or VisIt already know how to read the data. The HDF5 can also be post-processed separately in a user script written in Python or Matlab, which gives the user flexibility and the option to use their preferred method of data analysis without sacrificing time.

5.1.4 Continuous Integration and Testing

Cato was developed using modern software development practices such as Test-Driven-Design (TDD), Continuous Integration (CI), and and version control. TDD instructs the developer to write unit tests *while* developing the code rather than after. While it seems counter-intuitive, functions or code rarely perform perfect the first time. This helps the developer avoid writing

massive functions that do too much (and are prone to error propagation) and instead write simple clean functions that do one thing well (that can be tested and verified). It also provides a level of confidence in the code that is otherwise difficult to achieve. Version control was managed using GitLab, and CI was accomplished using the "runners" available on GitLab. These runners complete a series of commands or scripts whenever a new change is submitted to the git repository on GitLab. Failures at any level are sent to the developer directly via email. This automates testing and can aid in finding bugs or tracking performance. Runners in the *Cato* repository were configured to include both unit tests and integration tests.

5.1.4.1 Unit tests

Unit tests are applied to individual functions and subroutines, and need to run fast so that the developer can rapidly evaluate test correctness. Unit tests in *Cato* include tasks such as validation of parallel neighbor communication and various boundary conditions (such as periodic, zero-gradient, or open). Unit tests for boundary conditions worked on small, predetermined arrays where the output is known, even if not physically relevant. Functions that deal with boundaries or neighbor information are particularly prone to errors from indexing or accounting mistakes, and unit tests excel and catching these types of bugs. Unit testing for *Cato* is achieved using the pFUnit [107] framework.

5.1.4.2 Integration Tests

Integration tests examine how well different sections of the code work together. This was done for *Cato* through the use of a series of well-known physics benchmark problems. These tested the correctness of the physical problem and performance of particular algorithms such as reconstruction schemes or Riemann solvers, as well as tracking error through convergence history reports. Convergence history is a useful tool to track how well the solution behaves during the simulation. For steady-state problems, the error tolerance should "converge" to a final steady value, typically $\epsilon < 10^{-8}$, depending on the problem. For unsteady problems, this type of convergence can be too restrictive for simulations that need to finish in a reasonable amount of time,

and often the tolerance can be relaxed to a maximum of 10^{-3} or 10^{-4}. Spatial reconstruction accuracy, grid resolution, and time-step contribute to the level of convergence.

The time integrators used in *Cato* are all explicit Runge-Kutta multistage schemes (2-3 stages). After the final stage completes, the convergence is calculated by the normalized L2-norm of each state variable in the conserved vector $\vec{U} = (\rho, \rho u, \rho v, \rho E)$ with respect to the initial stage. This is shown in Equation 5.3

$$R_U = \frac{\|U^{(m)} - U^{(0)}\|_2}{\|U^{(0)}\|_2} \tag{5.3}$$

This history is saved to file over the entire simulation history for post-processing. Examples of convergence history can be seen in the following integration test problems.

5.1.4.3 Sod Shock Tube

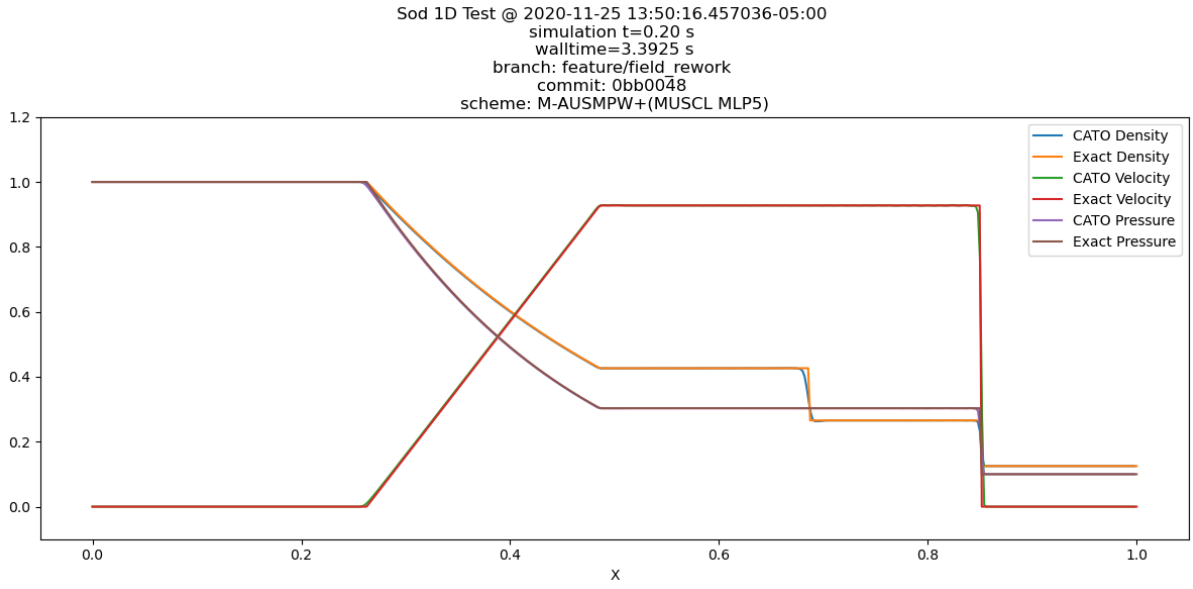

Figure 5.2: The 1D Sod shock tube generated using the M-AUSMPW+ Riemann solver combined with the MLP5 slope limiting scheme.

The Sod shock tube [108] is an excellent test used to evaluate how well a scheme can capture shocks, rarefaction waves, and material discontinuities. The initial conditions for the Sod test

are as follows:

$$(\rho,\ u,\ v,\ p) = \begin{cases} (1.0,\ 0,\ 0,\ 1.0), & \text{if } x < 0.5 \\ (.125,\ 0,\ 0,\ 0.1), & \text{if } x \geq 0.5 \end{cases} \tag{5.4}$$

The M-AUSMPW+ scheme combined with 5$^{\text{th}}$ order MLP generates extremely clean results for the 1D Sod shock tube as seen in Figure 5.2, which is compared to the exact solution. The resolution for this case is 250 cells in x. This figure is the result of an integration test run during CI. Details such as timestamp, git details of the specific commit, and scheme uses are all displayed to track historical changes in test results. Results like these are saved for posterity's sake.

5.1.4.4 Kelvin-Helmholtz

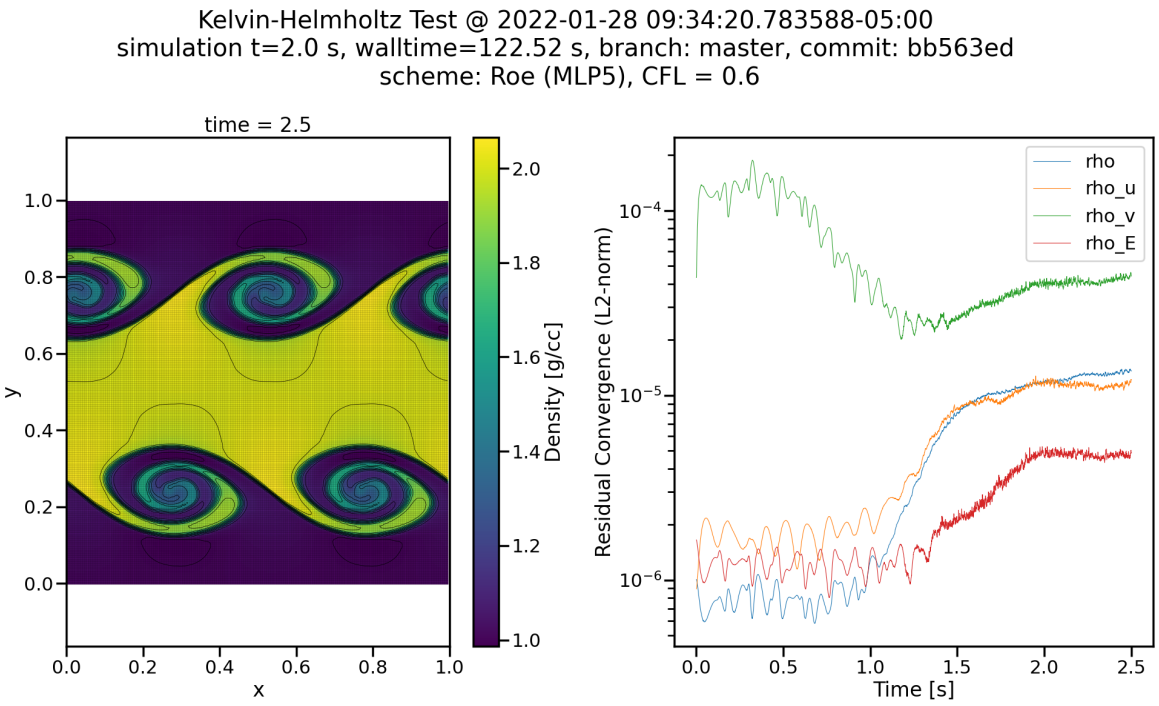

Figure 5.3: The 2D Kelvin-Helmholtz instability test. Density contours are on the left and residual convergence history is on the right.

The Kelvin-Helmholtz Instability test problem, as proposed in Reference 109, provides a metric for the performance of a particular solver applied to shear flow along a sharp interface. The Kelvin-Helmholtz Instability is a well known phenomenon that occurs when velocity shear is

present along an interface. The problem definition and fluid state for (ρ, u, v, p) is given below. Density is defined as:

$$\rho = \begin{cases} \rho_1 - \rho_m e^{\frac{y-1/4}{L}} & \text{if } 1/4 > y \geq 0 \\ \rho_2 + \rho_m e^{\frac{-y+1/4}{L}} & \text{if } 1/2 > y \geq 1/4 \\ \rho_2 + \rho_m e^{\frac{-(3/4-y)}{L}} & \text{if } 3/4 > y \geq 1/2 \\ \rho_1 - \rho_m e^{\frac{-(y-3/4)}{L}} & \text{if } 1 > y \geq 3/4 \end{cases} \tag{5.5}$$

Where $\rho_m = (\rho_1 - \rho_2)/2$, $\rho_1 = 1$, and $\rho_2 = 2$, with the smoothing parameter $L = 0.025$. The x-velocity (u) is given as:

$$u = \begin{cases} U_1 - U_m e^{\frac{y-1/4}{L}} & \text{if } 1/4 > y \geq 0 \\ U_2 + U_m e^{\frac{-y+1/4}{L}} & \text{if } 1/2 > y \geq 1/4 \\ U_2 + U_m e^{\frac{-(3/4-y)}{L}} & \text{if } 3/4 > y \geq 1/2 \\ U_1 - U_m e^{\frac{-(y-3/4)}{L}} & \text{if } 1 > y \geq 3/4 \end{cases} \tag{5.6}$$

With $U_1 = 0.5, U_2 = -0.5$ and $U_m = (U1 - U2)/2$, using the same smoothing parameter L. Y-velocity is defined as $v = 0.01 \sin(4\pi x)$, and pressure is defined as $p = 2.5$. The ideal gas polytropic index is $\gamma = 5/3$.

The results in Figure 5.3 show the density contour and convergence history. The approximate Riemann solver of Roe [76, 83] is used with the 5$^{\text{th}}$ order MLP reconstruction method. The history shows that residual error stays below 10^{-4} for the majority of the run, indicating a well-converged result.

5.1.4.5 Sedov Blast Wave

The Sedov blast wave problem is a test used to examine shock propagation and grid-imprinting effects. The initial conditions for the Sedov test in the figure are $(\rho, u, v, p) = (0.001, 0, 0, p_0)$, where $p_0 = 0.001$ everywhere except at the center, where $p = p_{\text{max}} e^{-((x^2+y^2)/\sigma)} + p_0$, with

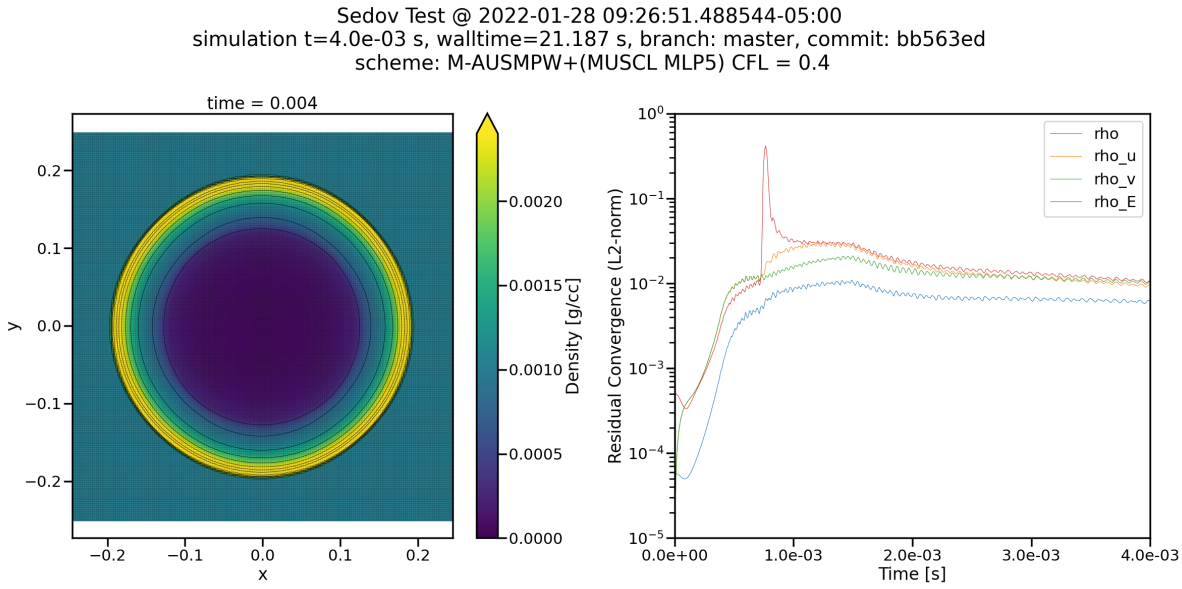

Figure 5.4: The 2D Sedov blast wave test problem. Density contours are on the left and residual convergence history is on the right.

$\sigma = 0.001$ and $p_{\max} = 10$. This deposits a large amount of energy at the center of the domain which expands outward as a blast wave.

Density contours and convergence history are shown in Figure 5.4. The results shown use the M-AUSMPW+ solver combined with the 5$^{\text{th}}$ order MLP reconstruction method. The Sedov test excels at catching symmetry bugs. Low-order methods struggle to maintain the sphericity of the blast wave, which is readily observed when present. The residual convergence history shows error that is relatively large ($\epsilon > 10^{-3}$), which indicates a finer grid resolution is needed, but the error tolerance is relaxed for the sake of speed in this test.

5.1.4.6 2D Implosion Test

The 2D implosion test of Liska and Wendroff [110] is useful in catching noise created by numerical round-off in an environment with strong shocks. This problem features a low density core surrounded by a high pressure region that launches a strong shock inward. Only the upper right quadrant $(0 < (x, y) < 0.3)$ of the problem was simulated. The initial conditions are as follows:

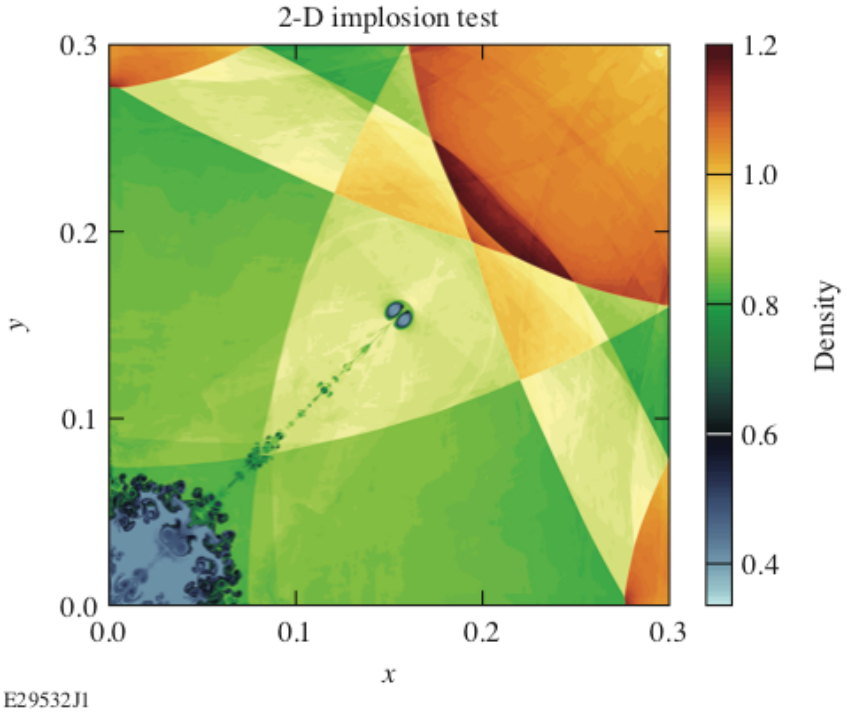

Figure 5.5: Density contours of a 2D implosion test. Numerical error build-up causes the jet along the $y = x$ axis to deflect, which is non-existent in this result.

$$(\rho,\ u,\ v,\ p) = \begin{cases} (0.125, 0, 0, 0.14), & \text{if } x + y \leq 0.15 \\ (1.0, 0, 0, 1.0), & \text{if } x + y > 0.15 \end{cases} \tag{5.7}$$

The results of the test are shown in Figure 5.5, which plots the density contour at $t = 1$ second. Violent shock waves create a small jet feature that forms along the $y = x$ line that is sensitive to asymmetries. Error build-up due to floating point noise will show up early in the simulation and deflect the jet [111]. The M-AUSMPW+ scheme combined with 5^{th} order MLP shows no asymmetry along the jet feature. This case had 16M cells and took approximately 26 hrs to run with 120 cores. This particular test result was not included in the integration test suite due to the large resources required.

5.1.5 Performance

Multi-physics and single-physics codes, such as *Cato*, need to scale well on large HPC systems to tackle complex problems. Scaling can be classified as either *weak* or *strong*. *Strong scaling* is defined by the total run-time of a fixed problem size for a variable number of processors. In other words, for a fixed problem, the run-time should decrease as the number of processors increase. *Weak scaling* is defined by the run-time for a problem that is fixed per processor, for a variable number of processors. This reflects the idea that larger problems can be completed in the same amount of time by increasing the number of processors.

For strong scaling, Amdahl's law [112] states that the maximum amount of speedup of a parallelized code has a ceiling set by the amount of run-time that is serial.

$$\text{speedup} = \frac{1}{s + p/N} \tag{5.8}$$

This is based on the fraction of the code that is serial (s), parallelized (p), and number of processors (N). Weak scaling is based on Gustafson's law [113] which states that maximum speedup is achieved by:

$$\text{speedup} = N - s \times (N - 1) \tag{5.9}$$

where s and N are the same as in Amdahl's law. Weak scaling is primarily focused on the size of the problem, whereas strong scaling determines the level of speedup possible for a given code.

In *Cato*, file input/output is the largest serial portion of the code. However, neither scaling law takes into account the characteristics of the algorithms used, namely whether they are limited by memory bandwidth (e.g. memory-bound) or compute capability (compute-bound). *Cato* uses the finite-volume method, which is dominated by memory-bound algorithms. In particular, spatial reconstruction functions need access to nearest-neighbors via stencils in order to reconstruct interfacial cell values. For example, the 5$^{\text{th}}$ order MLP function needs information from `(i-2, i-1, i, i+1, i+2)` as well as `(j-2, j-1, j, j+1, j+2)`. For the 2D arrays used throughout

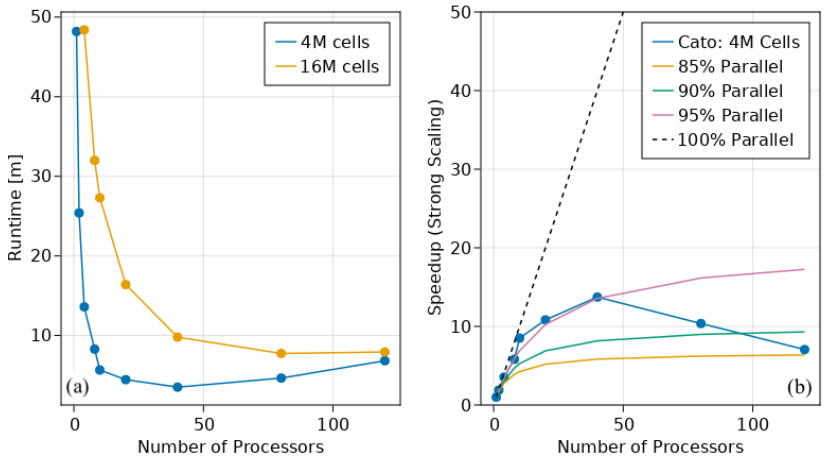

Figure 5.6: Scaling results from a Sedov blast wave problem with 4 million and 16 million cells. Run-times in minutes are shown in (a) and strong scaling speedup is shown in (b).

Cato that contain the fluid state information, data at indices `(j-2, j-1, j+1, j+2)` is no longer local, due to memory layout. In order to recover from limits imposed by memory bandwidth, hybrid data structure layouts such as Array of Structure of Arrays (AoSoA) [103] or stencil-oriented layouts (or brick-like mini domains) [114] are required, which add complexity to the code.

Figure 5.6 shows the results of a 2D Sedov blast wave problem run with 4 million and 16 million cells for different core counts compiled with the Intel 2020.4 compiler. Figure 5.6a a shows the total run-time for each problem. Run-time reduction for core counts higher than 40 and 80 for the 4M and 16M cases quickly approaches diminishing returns. Increasing the core-count only slows the problem down more, due to the increased level of communication. Gustafson's law shows that increasing the work load per processor helps to improve scaling. In parallel decomposed problems such as these, the amount of time spent on communication starts to overwhelm the time spent actually solving the hydrodynamic equations. Figure 5.6b shows the strong scaling result of the 4M cell case, including the results from Amdahl's law indicating the total percentage of the code that is parallel. For runs that include up to 40 cores, 95% of *Cato* is running in parallel, but as the core count increases, efficiency drops. This is due to increased communication time and I/O time.

In the future, *Cato*'s parallel scaling could be improved by making I/O parallel (e.g incorporating parallel HDF5), optimizing on-node performance to reduce memory-bandwidth penalties, and improving algorithms such as spatial reconstruction and time integration. The two most expensive functions within *Cato* relate to spatial reconstruction and the Riemann solver. Switching to alternate time integration schemes such as ADER [115] that only require one communication update per time-step (compared to 2-3+ for Runge-Kutta schemes) would also improve scaling.

5.2 Multi-physics Implementation in Julia

Cygnus is the multi-physics code written in Julia that couples hydrodynamics with thermal conduction on a fixed grid. The hydrodynamics module solves the 2D compressible Euler equations for a single material with the ideal gas equation of state, and the thermal dynamics module solves the unsteady heat conduction equation with source terms assuming a temperature- and density-dependent thermal conductivity. The hydrodynamics methods implemented in *Cygnus* were translated over from *Cato* with minimal changes. The design of *Cygnus* is very similar to Cato albeit with different implementations due to the differences between Fortran and Julia.

5.2.1 Julia

Julia is a dynamic, high-performance, high-level programming language designed for scientific and numerical computing created in 2009 by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman [100]. It emphasizes type-driven design and multiple-dispatch with just-in-time (JIT) compiling (built on the LLVM infrastructure [116]) to achieve high performance. Julia also includes meta-programming via macros and incorporates multiple levels of parallelism, including vectorization, threading, and distributed computing. The high-level and dynamic syntax is similar to Matlab, yet performance is comparable to Fortran and C. This allows for rapid algorithm prototyping without the performance hit associated with typical high-level dynamic languages.

A study in 2020 ran HPC benchmarks comparing Julia vs C in MPI memory-bound applica-

tions and found that Julia was on-par with C for the majority of cases [117]. The authors report that Julia is a serious competitor for HPC codes given the significantly smaller amount of code and increased readability to achieve similar performance to C. In 2021, Julia was applied to a series of benchmark mini-apps, for both memory bandwidth-bound and compute-bound problems, and was found to be as performant as the competition implementations [118]. These benchmarks were run across a variety of hardware vendors (Intel, AMD, NVIDIA) and compared to existing parallel frameworks such as OpenMP [97], Kokkos [119], OpenCL [120], NVIDIA CUDA [121], and Intel oneAPI [122]. The authors report that porting mini-apps to Julia and applying them to multiple accelerator types (e.g. GPUs) was straightforward due to the high-quality libraries available in Julia. Julia, therefore, is beneficial to both non-experts (who can rapidly prototype algorithms) and experts (who can fully optimize to the underlying hardware) and runs on a variety of hardware types such as CPUs, GPUs, or other specialized accelerators [123].

The most compelling aspect of Julia is that it solves the two-language problem; namely that developers must decide between a language for speed or for ease-of-use. Fortran excels at array-based computation; however, though recent versions have made large strides toward including modern features (see Section 5.1), many simple common tasks outside this paradigm are difficult to accomplish. Dynamic languages such as Matlab or Python can be extremely productive for prototyping, data science, or post-processing simulation results, but performance-critical code must often be delegated to functions written in C or Fortran underneath. While Julia can also call existing C or Fortran code directly, it achieves its performance through type-based design and the LLVM compiler infrastructure. When functions are called for the first time, they are compiled to machine code, and subsequent function calls reuse the machine-compiled version thereafter. Properly written functions allow Julia to create optimized code based on argument data types. This practice, known as templating, is possible in other languages, such as C++ (or even Fortran to a limited extent). However, Julia accomplishes this at a high level, which allows user code to be significantly more readable comparatively.

5.2.1.1 Parametric Types and Multiple Dispatch

Multiple dispatch is the ability of the compiler to call (or dispatch) a function based on the uniqueness or attributes of the function's arguments. Within Julia, this leads to considerable code re-use and sharing, both within modules and across the larger community [124]. The impact of multiple dispatch on design and performance can be demonstrated by the following example.

```
abstract type AbstractShape end

struct Polygon{T <: Real} <: AbstractShape
  nodes::AbstractArray{T,2} # list of nodes in x,y
end

struct Cube{T <: Real} <: AbstractShape
  x::T
end

struct Sphere{T <: Real} <: AbstractShape
  r::T # radius
end

volume(s::AbstractShape) = @error("Undefined volume for $(typeof(s))")
volume(s::Sphere) = (4/3)π * s.r^3
volume(c::Cube) = c.x^3
```

Listing 5.8: Example: Parametric types and multiple dispatch in Julia

Listing 5.8 defines 3 different parametric types that inherit from a common `AbstractShape` ancestor. The `Sphere` and `Cube` types inherit from the `AbstractShape`, and each define their own internal data. The `T <: Real` syntax is a form of generic programming that indicates that behavior (or creation) of the type (i.e. `struct`) will parametrically behave based on how the object is created. This syntax requires the data for each type to be `Real`, which includes floating point and integer data. For example, the `Cube` type can be created as `cube = Cube(2.0)` or `cube = Cube(2.0f0)`, which means that the internal data will consist of a 64 bit or 32 bit floating point number respectively. When code is generated at run-time for the `volume()` function, this will generate specific code that is internally optimized for the 64 bit or 32 bit floating point numbers.

Julia differs from many other object-oriented languages in that sub-types cannot inherit data. The

hierarchy established by `AbstractShape` only determines the behavior; concrete types define their own data. The multiple definitions of the `volume()` function set the implementation on a type basis, since multiple dispatch will correctly call the type-specific behavior.

```
julia> poly = Polygon(rand(Float32,4,2)); # uses Float32
julia> cube = Cube(2); # uses Integer
julia> sphere = Sphere(2.0); # uses Float64
julia> volume(sphere)
33.510321638291124
julia> volume(cube)
8
julia> volume(poly)
Error: Undefined volume for Polygon{Float32}
@ Main REPL[5]:1
```

Listing 5.9: Example: Multiple dispatch in Julia

Listing 5.9 creates each shape type and calculates its volume. The `julia>` syntax denotes the use of the REPL, or "read-evaluate-print loop" functionality, which allows users to develop and run code inside an interactive console. This simplistic example demonstrates the extent to which the clean syntax of Julia makes it straightforward to write generic functions that optimize based on data types. Multiple dispatch makes it simple to write a function that specializes the implementation. The `volume()` function can be called elsewhere without needing to understand the implementation or write complicated logic that determines how to calculate the volume of a specific shape.

In single-dispatch languages, this typically involves multiple branching if-statements that check the type and dispatch behavior accordingly. Single dispatch languages do not allow the re-definition of the same function. Instead, this is handled in an object-oriented manner where the `volume()` function is held inside the type or class, and called using syntax like `shape.get_volume()`), which can create problems for code re-use.

For example, in an object-oriented single-dispatch language, if a developer wishes to extend the capability of a class by adding new methods they must: 1) ask the original developers to add functionality, 2) if the class's code is open-source, they can push the edits upstream to the

developers and hope the change gets accepted, 3) make a new class and inherit from the one they wish to extend, 4) write external functions that operate on the class that needs to be extended, or 5) re-invent the wheel. Options 1 and 2 depend on the library developers, and are dependent on their priorities. Option 3 can be problematic because a new class must be defined with new functions, and these new methods will not work on the super class (class being inherited from) without complicated wrapper code. Option 4 is not ideal since it makes the code less generic and limits flexibility. Unfortunately, option 5 is often chosen because it gives the developer the most control, despite the extra work required to re-implement the capability. This severely limits code re-use in the broader developer community.

Community-wide code re-use in Julia is surprisingly effective and extensive, as demonstrated in Reference 124. In the example, two separate libraries are combined to solve ordinary differential equations with uncertainty propagation. The ODE is solved using the library `DifferentialEquations.jl`, which provides several high-quality ODE solvers. Uncertainty error propagation is handled by the `Measurements.jl` library. These two libraries are not dependent on one another, nor are they aware of each other's capabilities. However, because of multiple-dispatch and Julia's type design, ODE uncertainty propagation can be achieved by simply adjusting the initial conditions to include a measurement uncertainty. In the example, one of the inputs is gravitational acceleration g, which is changed from $g = 9.8$ to $g = 9.79 \pm 0.02$, thereby updating the ODE solution to include uncertainty with minimal performance reduction. This required no modification to either library, and could even be extended by adding the library `Unitful.jl`, to add physical units to the problem. This type of code re-use is incredibly beneficial, and only possible due to multiple dispatch and Julia's type-driven design.

5.2.2 High-Level Design

Cygnus takes full advantage of multiple dispatch and parametric types. Similar to *Cato*, modules within *Cygnus* are responsible for single-physics (hydrodynamics and thermodynamics), equations of state, numerical solver methods, mesh geometry, multi-physics coupling and more.

Abstract types determine the behavior at the interface level and concrete types determine the implementation and data layout for high-level generic code. Table 5.2 summarizes many of the concrete and abstract types (similar to Table 5.1 for *Cato*). In the table, abstract types are inherited by one or more concrete types, and a general description is provided for each.

The type relationships are best explained with an example. In *Cygnus*, as in *Cato*, there are different methods available for spatial reconstruction. Spatial reconstruction occurs each time the Riemann solver is called to compute the fluxes at each cell interface (see Section 4.1 for details). Each Riemann solver type implements the interface `solvehydro!(riemann_solver, W, mesh, EOS, recon_scheme)` that updates the flux at each of the cell edges (the `!` in `solvehydro!` is Julian syntax convention that indicates the argument(s) of the function are updated in-place). Inside the `solvehydro!` function call, there is a `reconstruct!` function call that will provide the spatial reconstruction of the primitive variables at each interface.

The `solvehydro!` function does not need to know anything about the implementation of the `reconstruct!` function, as long as it uses the correct interface (i.e. provide the right arguments). Julia dispatches the `reconstruct!` function based on the arguments. Here the `recon_scheme` provided to the `solvehydro!` function determines the type of reconstruction to run, such as 2nd order with MinMod limiting or 5th order with MLP limiting (c.f. Section 4.1). Listing 5.10 shows example segments of the function definitions for 4 different types of reconstruction, whether in 1D or 2D and using 2nd, 3rd, or 5th order. In an object-oriented language, this could also be handled by something similar to `recon_scheme.reconstruct()` as long as each `recon_scheme` inherits from the same ancestor. If a new scheme is added in the future that does not fit well into this particular structure, re-designing the code could present difficulties. Having written both Julia and object-oriented Fortran implementations, the multiple dispatch version results in cleaner code with better maintainability.

```
"""1D MUSCL reconstruction"""
function reconstruct!(RS::MUSCLReconstruction, U::AbstractArray{T,2},
                      i_edge) where {T<:AbstractFloat}
    ... # Implementation details
end
```

Table 5.2: Examples of abstract and concrete types in *Cygnus*

Abstract Type	Concrete Type Example	Description
`AbstractMesh`	`Mesh1DSpherical`, `Mesh2DCartesian`	Store mesh and geometry information (volume,edge lengths, normal vectors, etc.)
`AbstractEOS`	`IdealEOS`	Handle equation of state computation,thermal conductivity, sound speed
`AbstractBC`	`PeriodicBC`, `ZeroGradientBC`	Applies boundary conditions (thermal, hydro, etc.)
`AbstractRiemannSolver`	`HLLC1D`, `HLLC2D`,`M_AUSMPWPlus2D`	Riemann solver used to compute hydro fluxes at cell boundaries
`AbstractMUSCLRecon`	`MLP5Reconstruction`, `MUSCLReconstruction`	Perform spatial reconstruction
`AbstractFluid`	`Fluid1DIdeal`, `Fluid2DThermal`	Structure to hold hydro state variables and manage integration of EOS, BCs, source terms, and temporal integration scheme, such as Runge-Kutta
`AbstractLinearSolver`	`ThermalADISolver`	Solve thermal conduction using ADI
`AbstractOperatorSplit`	`StrangOperatorSplit`, `LinearOperatorSplit`	Manage multi-physics coupling between hydro, thermal, and other future physics packages
`AbstractEvent`	`ContourEvent`, `InSituEvent`, `HydroSolverUpdateEvent`	Events are used to trigger behavior like a save event to file, a restart, changing discretization details, or an in-situ event to plot a variable or run a user-defined function
`AbstractContourWriter`	`XDMFContour`, `HDF5Contour`, `VTKContour`	Write contour data to file in various formats

6

```
"""2D MUSCL reconstruction"""
function reconstruct!(RS::MUSCLReconstruction, U::AbstractArray{T,3},
                      i_edge, j_edge) where {T<:AbstractFloat}
    ... # Implementation details
end

"""2D 3rd order MLP reconstruction"""
function reconstruct!(RS::MLP3Reconstruction, U::AbstractArray{T,3},
                      i_edge, j_edge) where {T<:AbstractFloat}
    ... # Implementation details
end

"""2D 5th order MLP reconstruction"""
function reconstruct!(RS::MLP5Reconstruction, U::AbstractArray{T,3},
                      i_edge, j_edge) where {T<:AbstractFloat}
    ... # Implementation details
end
```

Listing 5.10: Spatial reconstruction in *Cygnus*

In Listing 5.10, syntax such as `RS::MUSCLReconstruction` tells Julia the type of the argument `RS`. The `U::AbstractArray{T,3}` annotation indicates that `U` is a 3D array comprised of `AbstractFloat`. The data layout of `U` is $((\rho, \rho u, \rho v, \rho E), i, j)$, where i and j are logical domain indices. The type of the number contained in the array can be `Float64`, `Float32`, or any other ancestor of `AbstractFloat`. Further specialization could be achieved by specifying `U` with concrete type, such as `U::CuArray{T,3}`, which indicates that the array resides on a CUDA GPU. This would dispatch the function to the GPU instead of the CPU. The current implementation is CPU only, but the framework can be further extended to include GPU capability with minimal impact on the design. Arrays `i_edge` and `j_edge` are not type-annotated, as it does not change the dispatch mechanism in this situation.

This abstract vs concrete type relationship is implemented throughout *Cygnus*. The types listed in Table 5.2 only show a sampling of the more significant types and implementations. Types can be used to implement algorithms or coupling between different aspects of the code. The following sections describe the multi-physics coupling and single-physics implementations.

5.2.3 Multi-Physics Coupling

Multi-physics coupling is achieved through the 2nd order Strang splitting approach [125]. This occurs in three steps: 1) hydrodynamic time integration is run for a half time-step, 2) thermal conduction is run for a full time-step using the physical state from step 1, and finally, 3) solve for hydrodynamics for another half time-step using the updated fluid state from step 2. Linear coupling is also an option, where each operation runs for a full time-step using the state from the previous single-physics operation, but accuracy is reduced to 1st order. Coupling behavior is selected at runtime by the user via the `StrangOperatorSplit` or `LinearOperatorSplit` types, both of which inherit from `AbstractOperatorSplit`.

5.2.4 Solving the Hydrodynamic Equations

The solution of the hydrodynamic equations closely follows those given in Section 5.1.1.3 for *Cato*. The primary difference is in the implementation details following the type-oriented design detailed above. This is summarized by the following:

1. Initiate the `hydro_step` function (which is called by the `AbstractOperatorSplit` type) to handles the hydrodynamic time integration based on the provided `AbstractFluid`, `AbstractMesh`, time-step, `AbstractRiemannSolver`, and `AbstractMUSCLReconstruction`.
2. Call the time integration algorithm, i.e. `SSPRK3!` to start integration process.
3. Within `SSPRK3!`, loop through the three stages in the explicit RK method.
4. Compute $\partial\vec{U}/\partial t$ in each stage, which is delegated to the `AbstractRiemannSolver`.
5. Apply the boundary conditions via `applyBC!(AbstractBC, field, EOS)`.
6. Apply spatial reconstruction methods (e.g. `MLP5Reconstruction`) - this provides the cell interface values of each cell $(\bar{\Phi}_L, \bar{\Phi}_R)$ using a 2nd order (or higher) function.
7. Pass cell interface values to the Riemann flux function, e.g. `solvehydro!`, with the provided `AbstractRiemannSolver`

8. Sum fluxes along each cell interface to provide $\partial\vec{U}/\partial t$.

9. Move to the next timestep after all stages finish if convergence is adequate.

10. If convergence fails or there are negative densities or pressures, reduce the time-step and redo steps 3-9.

11. Calculate derived quantities such as sound speed and Mach number after the time-integration in finished (in `AbstractFluid`).

12. Write contour files to disk at user-specified intervals, via the `AbstractContourWriter`.

13. Loop steps 2-11 until the user-specified end criteria, such as maximum iterations or simulation time.

5.2.5 Solving for Thermal Conduction

Thermal conduction is dispatched by the multi-physics coupling type, and the solution is found with the following steps:

1. The `thermal_conduction!` function manages all solver-related behavior, based on the provided solver methods, source terms, equations of state, and boundary conditions. This is called by the `AbstractOperatorSplit` type, which determines when to run (in relation to the hydro step) and what time-step to use.

2. Using the fluid state, solve for temperature using $T = (E - \rho(u^2 + v^2)/2)/\rho c_v$

3. Compute thermal conductivity κ of each cell. This is currently linearized by assuming κ is fixed for the timestep, rather than $\kappa = \kappa(\rho, T)$

4. Apply thermal boundary conditions, which was typically $\partial T/\partial x = 0$

5. Apply the energy deposition source term (if any). Energy deposition placement and amplitude is handled in a separate type, e.g. `TimeDependentEnergyDeposition2D`.

6. Perform implicit solve using the ADI method (see Section 4.2.1). This is iterated over 2-3

times for stability using the corresponding reduced time-step, e.g. $\Delta t_{\text{new}} = \Delta t / n_{\text{iter}}$

7. Total energy is updated with the new temperature using $E = \rho(c_v T + (u^2 + v^2)/2)$

5.2.6 Parallelization

Parallelization is done at the loop level via multi-threading and SIMD vectorization. Multi-threading and SIMD vectorization functionality is included in the base Julia package. Because of the data layout for field data in *Cygnus*, fine-grained (loop level) parallelism was easily implemented using macros. The primary data layout within *Cygnus* is Structure of Arrays (SoA), which is better for vectorization and memory locality [103]. The example below (Listing 5.11) shows the style of loop-level parallelization in *Cygnus*.

```
@threads for j in jlo:jhi
  for i in ilo:ihi
    @simd for phi in 1:4
      fluid.U[phi,i,j] = ...
  end
end
```

Listing 5.11: Multi-threaded and vectorized loops via `@macros`

The `@threads` macro divides the loop between all available threads, and the `@simd` macro applies vectorization to the inner-most loop. Macros are used in Julia for meta-programming, which generates additional code at runtime without performance penalties. The example in Listing 5.12 uses a third-party library, `LoopVectorization.jl` to generate a multi-threaded, vectorized loop with additional optimizations compared to the standard library options (`@threads,@simd`). Here the `@tturbo` macro combines the threaded and SIMD directives in one statement, as well as additional optimization based on the loop and types involved. Macro usage is situation dependent, since certain loops do not work well due to branching statements or poor data layout.

```
"""Conserved variables to primitive variables conversion"""
function cons2prim!(EOS::Union{IdealThermalSpitzerEOS,IdealThermalEOS},
                    U::AbstractArray{NumType,3}, # conserved vector
                    W::AbstractArray{NumType,3}, # primitive vector
                    T::AbstractArray{NumType,2}) where (NumType <: AbstractFloat)
```

```
6
7     R = EOS.R
8     @tturbo for j in 1:size(U, 3)
9        for i in 1:size(U, 2)
10          W[1, i, j] = U[1, i, j] # density
11          W[2, i, j] = U[2, i, j] / U[1, i, j] # x-velocity
12          W[3, i, j] = U[3, i, j] / U[1, i, j] # y-velocity
13          W[4, i, j] = U[1, i, j] * R * T[i, j] # pressure
14       end
15    end
16 end
```

Listing 5.12: Converting conserved variables to primitive variables with a multi-threaded, vectorized loop

Fine-grained parallelization does not scale as well as coarse-grained for the type of workloads that *Cygnus* and *Cato* solve [103]. Scaling results are given later in Section 5.2.9. Future iterations of *Cygnus* will incorporate MPI-based parallelism (using `MPI.jl` library) for domain decomposition similar to *Cato*. GPU-based parallelization is also possible using high-level GPU kernels via `KernelAbstractions.jl` and more vendor-specific capability via `CUDA.jl`, `AMDGPU.jl`, or `oneAPI.jl`, similar to the study in Reference 118. Because of the advantages that Julia offers, this parallelization can be added incrementally.

5.2.7 Input/Output

Contour file output in *Cygnus* is similar to *Cato*, with XDMF files for "light" data and HDF5 files for the "heavy" data. The VTK format can also be used via the `WriteVTK.jl` library. The `AbstractContourWriter` type hierarchy makes simplifies the process of adding new formats.

Physics codes, such as Cato and Cygnus are traditionally run in two ways: 1) via an input "deck" which is a text file, or 2) with scripts or source code that links to the library directly. Each approach has its merits and pitfalls. Text input files are problematic in that users can easily make syntax mistakes, especially if the file format is not standardized. Using a standardized format like a Fortran namelist, `.toml`, `.yaml`, `.xml`, or `.json` is advantageous since there are well-written existing libraries available online for developers to use to read the input options into the code. Most text editor programs also offer convenient syntax highlighting or parsing

to catch errors before submitting the job to run on a remote HPC cluster, thereby eliminating the frustration caused when a job, that has been sitting and waiting in a queue for hours after submission, immediately crashes due to a syntax error.

Syntax issues aside, maintainability also suffers when input options change, or the type of simulation a user wants to run does not fit well into the input file structure. Restart or checkpoint capability also requires the developer to manage variables that can be changed mid-way through the run, which adds complexity to both input file and code structure.

Running a code as a library can alleviate some of these issues, but this can also be problematic in other ways. First, running as a library requires the user posses some knowledge of code structure, which steepens the learning curve considerably. (This can be alleviated by including different interface levels, i.e. a high-level interface, to make running common problems simple, and a low-level interface for experts). Second, the user could unknowingly run the code improperly. Third, requiring the user to link to, or compile, the code at each run adds a burden on the user. Configuration and compile scripts can be notoriously complicated and not portable between systems, especially if dependency issues arise with the use of multiple third-party libraries. Much of this can be helped by good package managers or coupling the multi-physics code with a scripting language interface, such as Python.

Cygnus is written to be controlled as a library, and the design of Julia addresses many of the issues associated with this approach. As Julia is both dynamic *and* high-performance, there is no need to use two languages to delegate between an interface and the code itself. A high-level interface is provided to aid users, but the managing user interaction is much simpler with the absence of the two-language problem. An example from the high-level interface is the `IdealFluidProblem` type, which is created by combining the mesh, initial fluid state, Riemann solver, reconstruction scheme, and contour format preference. The REPL functionality of Julia includes in-line help that users can interactively query for documentation on how to build up the `IdealFluidProblem`. Inline help pulls from code documentation in the library with descriptions and examples. Users can still make mistakes at runtime, but these can be alleviated through help-

ful error messages (which is true of any code, regardless of programming language). Because users can control *Cygnus* interactively, jobs can be easily tested on-the-fly prior to submission to a cluster. Compilation and configuration is simplified because Julia is interactive and dynamic, and installation is handled via the package manager. User run scripts simply need to include `using Cygnus` like any other library.

An example of a run script is shown in Listing 5.13 below. This represents the building-block like structure to create a simulation.

```
using Cygnus, Unitful
x, y = ... # define mesh node positions

# nhalo = halo region size, e.g. 2
mesh = Mesh2DCartesian(ncells_x, ncells_y, nhalo)
initialize!(mesh, x, y)

eos = IdealEOS(5/3)
hydro_bcs = Dict("ilo" => PeriodicBC2D(nhalo, :ilo),
                 "ihi" => PeriodicBC2D(nhalo, :ihi),
                 "jlo" => PeriodicBC2D(nhalo, :jlo),
                 "jhi" => PeriodicBC2D(nhalo, :jhi))

# Initialize fluid with appropriate physical units
rho, u, v, p = ...

# User-defined function to run insitu
function insitu_function(prob::AbstractProblem)
  # ... i.e. plot at a specific location, do inline analysis
end

t_end = 1.0*u"s" # solution end time. u"s" applies units of seconds
interval_t = 0.1*u"s" # contour I/O save interval

events = [ContourEvent(interval_t, XDMFContour(CGSUnitSet)),
          InSituEvent((0:.1:1)u"s", insitu_function)]

prob = IdealFluidProblem("Sedov", mesh, rho, u, v, p,
                         eos, HLLC2D, MLP5Reconstruction, hydro_bcs)

CFL = 0.5
solve!(prob, tend_s, CFL, events)
```

Listing 5.13: An example of a Cygnus run script (details of grid and field initialization are left out for simplicity)

Line 1 imports `Cygnus` as well as `Unitful` (for physical units). Lines 2-6 create a `Mesh2DCartesian` type to contain all mesh-related data. Lines 8-12 create the equation of state and boundary conditions. Line 15 is where the user would define initial conditions for the fluid state. Line 18 defines an *in situ* function that will run user-defined code. The `prob` variable contains the problem state (mesh, fluid, etc.) at the current time-step. This function can include plotting a specific variable to an image file, running in-line analysis, and more. Lines 22-26 define the solution end time, an interval time for contour output, and the list of events to pass to the `IdealFluidProblem`. Contour files are written in XDMF format and `CGSUnitSet` asserts that the units must be in the CGS system. Lines 28-32 define the `IdealFluidProblem` from the building-blocks, set the stability criteria with the CFL number, and finally solve the problem.

The example shows how a high level interface can provide a simple yet powerful means to create a simulation input. The events passed to the solution can be in many forms, including, but not limited to, a restart event, change of solvers, switching temporal integration or spatial reconstruction methods. All of this is accomplished without having to manually track internal restart variables or complicated logic for changing solver types. Each building block (type) manages its own functionality without needing to know the higher level interaction.

5.2.8 Continuous Integration and Testing

The same practices of test-driven development and continuous integration (CI) from *Cato* (in Section 5.1.4) are applied to *Cygnus*. This includes both unit and integration tests. All of the fluid-only integration tests were carried over from *Cato* to *Cygnus*. The unit testing framework is included with Julia, so this was vastly simplified compared to Fortran + pFUnit. An example of a unit test is shown in Listing 5.14.

```
1 global eos = IdealEOS(5/3) # γ=5/3
2 @testset "Periodic1D" begin
3    nhalo = 2 # num halo cells
4    scalar_field = collect(1.0:10.0)
5    # scalar field contents
6    # [1.0 2.0 | 3.0 4.0 5.0 6.0 7.0 8.0 | 9.0 10.0]
7    iloBC = PeriodicBC1D(nhalo, :ilo) # create periodic BC at ilo
```

```
    ihiBC = PeriodicBC1D(nhalo, :ihi) # create periodic BC at ihi
    Cygnus.BoundaryConditionTypes.applyBC!(iloBC, scalar_field, eos)

    @test scalar_field[1:2] == scalar_field[7:8]
    @test scalar_field[9:10] == scalar_field[3:4]
end;
```

Listing 5.14: An example of a boundary condition unit test in *Cygnus*

This tests the `PeriodicBC1D` type and its correctness. While the test data `scalar_data` is physically meaningless, the periodic boundary can still be tested for a 1D field. The `@test` macro ensures the truth of the logical statement to its right. Each unit test reports a pass/fail report for the developer to read.

5.2.9 Performance

Cygnus performance timing is tracked across each module courtesy of benchmarking and timing macros within Julia. This provides a breakdown of run-time and memory allocation for each flagged function (with no performance penalty). Below, an example timing summary is provided from a large isolated defect simulation (from Section 2.3) that took approximately 53 hrs to run on 20 cores.

Timing Summary:

		Time			Allocations		
Tot / % measured:		189328s / 100%			89.0GiB / 98.9%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
integrate!	56.8k	170069s	89.8%	2.99s	1.20GiB	1.37%	22.2KiB
SSPRK3!	113k	129031s	68.2%	1.14s	463MiB	0.51%	4.18KiB
ThermalRHS2D::...	340k	120064s	63.4%	353ms	309MiB	0.34%	954B
solvehydro!	340k	109837s	58.0%	323ms	177MiB	0.20%	544B
reconstruct!	340k	73861s	39.0%	217ms	129KiB	0.00%	0.39B
AUSMPW+ i-...	340k	17988s	9.50%	52.9ms	88.3MiB	0.10%	272B
AUSMPW+ j-...	340k	17974s	9.49%	52.8ms	88.3MiB	0.10%	272B
flux_edges	340k	9076s	4.79%	26.7ms	77.9MiB	0.09%	240B
cons2prim!	340k	1045s	0.55%	3.07ms	5.19MiB	0.01%	16.0B
applyBC!	340k	87.4s	0.05%	257µs	49.7MiB	0.06%	153B
NaN check	340k	683ms	0.00%	2.01µs	0.00B	0.00%	0.00B
hydro_residuals	113k	7125s	3.76%	62.8ms	10.4MiB	0.01%	96.0B
thermal_conduction!	56.8k	29500s	15.6%	519ms	341MiB	0.38%	6.14KiB
adi_y_sweep! m...	170k	7568s	4.00%	44.4ms	69.8MiB	0.08%	430B

```
     adi_x_sweep! t...   170k    4974s   2.63%  29.2ms   15.6MiB  0.02%    96.0B
     adi_y_sweep! t...   170k    4575s   2.42%  26.8ms   10.8MiB  0.01%    66.3B
     get_cell_condu...  56.8k    4565s   2.41%  80.3ms   9.81KiB  0.00%    0.18B
     adi_x_sweep! m...   170k    3171s   1.68%  18.6ms   65.0MiB  0.07%     400B
     hydro_BCs          56.8k    13.3s   0.01%   234µs   6.94MiB  0.01%     128B
     thermal_BCs        56.8k    4.16s   0.00%  73.1µs   2.94KiB  0.00%    0.05B
   conserved_to_pri...  56.6k     183s   0.10%  3.22ms    884KiB  0.00%    16.0B
   calculate_derive...   113k     150s   0.08%  1.32ms   1.73MiB  0.00%    16.0B
get_Δt                 56.6k   19121s   10.1%   338ms    114MiB  0.13%  2.06KiB
save_contour             280     126s   0.07%   450ms   86.7GiB  98.5%   317MiB
```

Timing breakdowns such as this highlight code hotspots and large memory allocations and their impact on total runtime. Here, we can see that the hydrodynamics portion accounts for 68% of the total runtime, whereas thermal conduction is only 16%. The actual thermal conduction solver is more expensive than the hydrodynamic solver (519 ms vs 353 ms), but the hydrodynamic portion is called twice for each time-step due to operator splitting. The next most expensive portion is the time-step calculation. Within the hydrodynamics, the spatial reconstruction phase is the most expensive at 39%. This is also one of the most memory-bound portions of the code due to the stencil calculations (see Reference 103 and 114 for stencil-related performance issues). Section 5.1.5 also discusses the performance implications of the spatial reconstruction phase.

Figures 5.7 and 5.8 shows the runtime performance of *Cygnus* on the Sedov blast wave test problem in 2D with a 200x200 mesh (with and without thermal conduction). This test case only includes the single-physics hydrodynamics version. In both figures, part (a) shows the runtime reduction for increasing core counts and part (b) shows the strong scaling similar to Section 5.1.5. The multi-physics strong scaling performance suffers since the thermal conduction ADI direct solver is serial. The matrix construction for the ADI method is multi-threaded, but a parallel domain decomposed solver is needed for better scaling.

The 5^{th} order version has better strong scaling in both tests, but the increasing core count starts to approach diminishing returns after 6 cores. Multi-threaded loops perform better with larger workloads, due to the overhead of spawning threads at each loop, so increasing the mesh resolution will improve the scaling (as shown in Figure 5.6). The 2^{nd} order version has poorer scalability compared to the 5^{th} order because it is less arithmetically intense. The 5^{th} order method

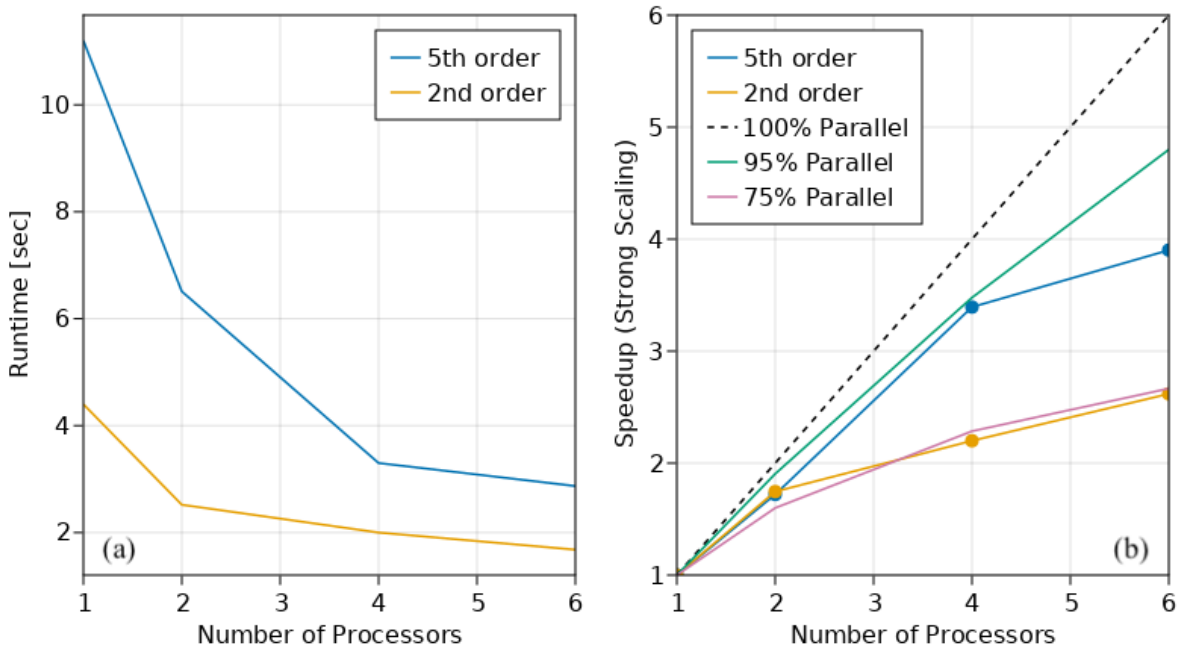

Figure 5.7: *Cygnus* scaling results from the Sedov blast wave test without thermal conduction. The 2nd and 5th results represent the use of different spatial reconstruction orders. The strong scaling results in (b) represent the scaling based on the percentage of the code that runs in parallel.

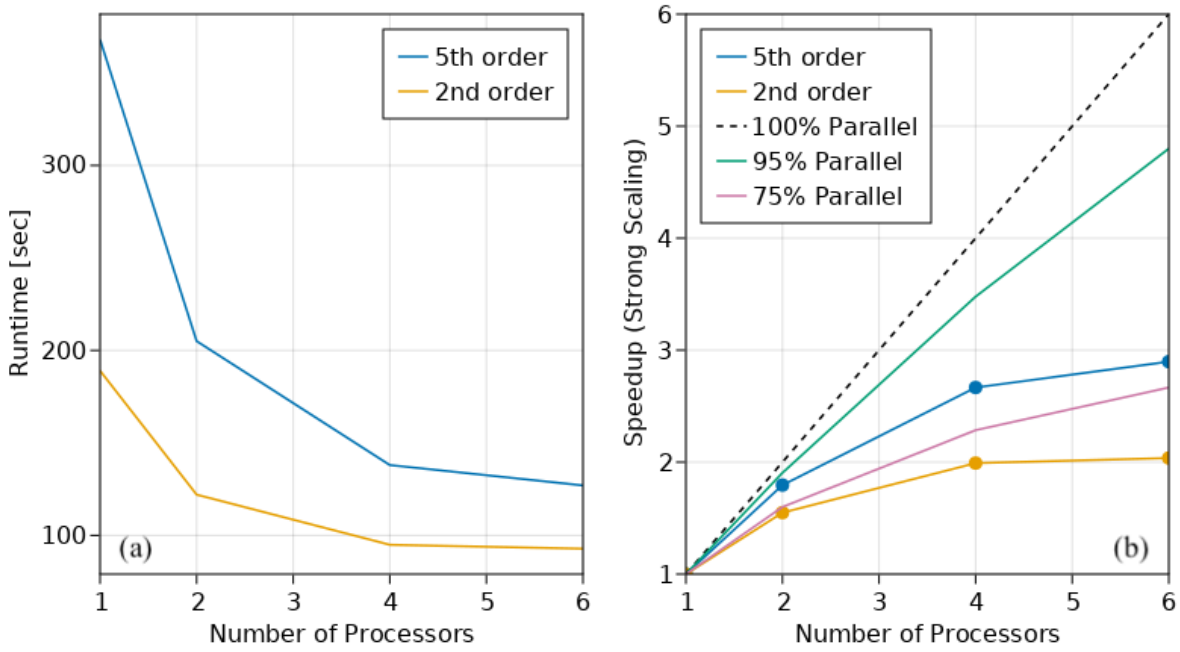

Figure 5.8: *Cygnus* scaling results from the Sedov blast wave test using thermal conduction. The 2nd and 5th results represent the use different spatial reconstruction orders. The strong scaling results in (b) represent the scaling based on the percentage of the code that runs in parallel.

has a higher number of floating point operations and but also suffers from memory bandwidth issues (due to a larger stencil size). Loop level multi-threading will only result in modest parallel performance gains, and memory bandwidth is the limiting factor in multi-threaded scientific applications [103, 126]. Better scalability will be incrementally added in future versions of *Cygnus* via domain decomposition similar to *Cato*.

5.3 Implementation Similarities

Cato and *Cygnus* use nearly identical functions for hydrodynamic spatial reconstruction, Riemann solver implementation, and temporal integration. Much of the hydrodynamic-oriented testing and functionality was accomplished first in *Cato* and then translated to *Cygnus* for the multi-physics capability. Fortran and Julia array indexing and loop syntax are similar, so translating the specific low-level functions was straightforward.

One of the common strategies applied in both codes relates to numerical noise. Simulating the internal perturbation physics detailed in Chapter 2 requires a noise-free implementation, since the perturbation waves, or disturbances, can be overshadowed by unphysical noise in the system. This was accomplished through the use of various techniques. There are two particular regions in the code that are susceptible to round-off error propagation: 1) in the spatial reconstruction functions, and 2) in the function that sums up the flux at each edge.

In spatial reconstruction, edge interpolation requires the computation of cell quantity jumps, or $\Delta\Phi_{i-1/2} = \bar{\Phi}_{i,j} - \bar{\Phi}_{i,j}$ (see Section 4.1). This is susceptible to catastrophic cancellation, where precision is lost when two nearly identical floating point numbers are subtracted from each other [103]. Comparing differences to floating point machine epsilon ($\epsilon_{\text{FP}} \approx 2.22 \times 10^{-16}$ for 64 bit) can eliminate extraneous difference values that otherwise would be zero. See Listing 5.15 for an example within a loop. (Note that doing this in loops can cause unnecessary branch conditions (if-statements), that do not vectorize well).

```
eps_FP = eps(Float64) # machine epsilon
@batch for j in jlo:jhi
```

```
3   for i in ilo:ihi
4     for phi in 1:4 # no simd due to if
5       dphi_minus_half = U[phi, i, j] - U[phi, i, j - 1]
6       if abs(dphi_minus_half) < eps_FP dphi_minus_half = 0 end
7     end
8   end
9 end
```

Listing 5.15: Checking for machine epsilon with if statements in a loop

Listing 5.16 improves vectorization by applying a logical mask rather than an if-statement. Line 6 in Listing 5.15 is changed to a logical mask operation (`abs(dphi_minus_half) >= eps_FP`), which is 1 or 0 if the result is true or false. In *Cygnus*, this makes it possible to use the `@tturbo` macro, with SIMD vectorization optimizations (and threading), and in *Cato*, this is accomplished via SIMD. In Fortran, the compiler will not optimize the loop if it does not think the loop can be vectorized. This needs to be checked through diagnostics, since the compiler will typically not warn the developer by default. In Julia, the `@tturbo` macro will fail at runtime, immediately signalling that vectorization did not occur.

```
1 eps_FP = eps(Float64) # machine epsilon
2 @tturbo for j in jlo:jhi # multi-threaded+simd since no if statements
3   for i in ilo:ihi
4     for phi in 1:4
5       dphi_minus_half = U[phi, i, j] - U[phi, i, j - 1]
6       dphi_minus_half = dphi_minus_half * (abs(dphi_minus_half) >= eps_FP)
7     end
8   end
9 end
```

Listing 5.16: Checking for machine epsilon using logical masks

Benchmark runs using a `Float64` array with size `[4,512,512]` (which is a typical conserved/primitive vector size) shows that the vectorized version (Listing 5.16) runs 10x faster, e.g. $t_1 = 598$ ns vs $t_2 = 6.2$ µs (using 4 threads and Julia 1.7 on a 2.6 GHz Intel Core i7). While the code listings here are from *Cygnus*, the same structure is also applied in *Cato*.

Relative error tolerance and scaling tricks are also used. Scaling is used in situations where there are large disparities in amplitudes that can be improved by scaling everything as close

to unity as possible. The pressure sensing functions in M-AUSMPW+ and AUSMPW+ (see Section 4.1.4) benefit from scaling the neighbor pressure values all relative to the current cell. The extreme conditions in ICF implosions (high pressures with low densities) amplifies the large disparities in values in many algorithms, since amplitudes span many orders of magnitude (e.g. $p = 10^{15}$ barye and $\rho = 10^{-2}$ g/cm^3, and $\dot{q}_{\text{laser}} = 10^{25}$ erg/s/cm^3). In a hydrodynamics-only implementation, the Euler equations can be scaled and non-dimensionalized to alleviate this issue; however, this is less straightforward in a multi-physics situation.

Relative error tolerance checks are used during the flux integration step. Listing 5.17 shows how a relative error check of 6 orders of magnitude is applied when computing the flux. Line 22 implements Equation 4.9, which determines $\partial\vec{U}/\partial t$ at each timestep. This scales by the maximum flux provided by the Riemann solver across the dimension-split i and j edges. This was needed to dampen noise build-up in the high-pressure, low-density coronal region in ICF implosions. Minor differences in pressure at adjacent cells (partially due to spatial reconstruction/interpolation) would initiate unphysical small-scale flows. For example, this would occur when neighbor cell-averaged pressure values were on the order of $p = 10^{13}$ and the difference between neighbors was $\Delta p = 10^2$. This created physically insignificant flux values well under the tolerance level. Checking for a relative error tolerance to within 5-6 orders of magnitude successfully damped out numerical noise without limiting physically relevant perturbation propagation.

```
1  eps_FP = eps() # floating point epsilon
2  eps_rel = 1e-6 # 6 orders of magnitude for relative error
3
4  for j in jlo:jhi
5   for i in ilo:ihi
6     for phi in 1:4
7       ds1, ds2, ds3, ds4 = @views Mesh.edge_len[:, i, j]
8
9       iflux = solver.F[phi, i, j] - solver.F[phi, i - 1, j]
10      jflux = solver.G[phi, i, j] - solver.G[phi, i, j - 1]
11      maxF = max(solver.F[phi, i, j], solver.F[phi, i - 1, j])
12      maxG = max(solver.G[phi, i, j], solver.G[phi, i, j - 1])
13
14      # Checking for catastropic cancellation
15      iflux = iflux * (abs(iflux) >= eps)
16      jflux = jflux * (abs(jflux) >= eps)
17
```

```
18      # Checking for relative error tolerance
19      iflux = iflux * (abs(iflux) >= eps_rel * maxF)
20      jflux = jflux * (abs(jflux) >= eps_rel * maxG)
21
22      dUdt[phi, i, j] = -(iflux * ds2 + jflux * ds3) / Mesh.volume[i,j]
23    end # phi
24  end # i
25 end # j
```

Listing 5.17: Checking for relative error and catastrophic cancellation in the flux integration function

5.4 Performance Comparison

One of the main questions at the start of *Cygnus*'s development was whether Julia could obtain performance on par with Fortran. Claims that one language is faster or better than another are often misleading because it is so heavily situation dependent. Since *Cato* and *Cygnus* were written by the same author, solve the same type of problem in parallel, and have similar data layouts and loop structure, this provided a unique opportunity to do a reasonable benchmark comparison of language capabilities.

The hydrodynamics-only 2D Sedov explosion problem is used to compare single and multi-threaded performance. The problem uses a square domain with 200x200 cells and was run with identical fixed-timesteps, initial conditions, temporal integration methods (SSP-RK3), Riemann solvers (AUSMPW+), and spatial reconstruction (MLP5). Neither case includes I/O time. *Cato* was compiled with GFortran 9.2 and *Cygnus* used Julia 1.6, which uses the LLVM 11.0 underneath. Both were run at the highest level of optimization (`-O3`) on the same system and runtimes are provided in Table 5.3.

In this scenario, *Cygnus* is just as performant as *Cato*. This means that the low-level machine code generated by the Julian compiler does as well as the Fortran compiler. The key result is the single-threaded time, rather than the parallel results, because the codes use slightly different approaches to parallelization, and the problem is not significantly large. Sections 5.1.5 and 5.2.9 show that the domain decomposition method scales better for large problems. Adapting *Cygnus*

Table 5.3: *Cato* vs *Cygnus* for the 2D Sedov Blast Problem

Code	**Threads**	**Average [s]**	**Speedup**	**Speedup vs Cato (%)**
Cygnus	1	6.270	1.00	5.8
	2	3.320	1.89	22.3
	4	1.893	3.31	36.2
Cato	1	6.658	1.00	-
	2	4.275	1.56	-
	4	2.965	2.25	-

to use domain-decomposition would alleviate this disparity.

5.4.1 Discussion

Object-oriented coarray Fortran and ADT calculus provide a means for writing clean, readable, performant code with a flexible design. Developing for parallel performance was fairly straightforward, since coarray syntax is incredibly simple and results in cleaner, less error-prone code. However, as a user, the experience was much less straightforward. Even as of 2022, compiler support for coarrays (which were introduced in the 2008 standard), is not ideal. Only Cray, Intel and GNU support enough of the 2008 standard for coarrays to be useful, and support for the 2018 standard is even more vendor dependent [127]. Using coarrays with the GNU compiler requires the OpenCoarrays [128] library which adds another, albeit simple, step in compilation. The `CMake` configuration and build system helped simplify this in a cross-platform manner. *Cato* was run with the latest GNU (v9.1) and Intel (2019 - 2020) compilers on the `typhoon` and `blizzard` clusters at the University of Rochester's Laser Laboratory for Energetics, and locally on a 2019 MacBook Pro.

Running with OpenCoarrays requires a selecting the parallel back-end implementation (either OpenMPI or MPICH). While this is straightforward to do, selecting the most performant MPI configuration took considerable trial and error. Scaling experiments revealed the OpenMPI ver-

sion had poor performance because the job was running on a network filesystem. This was fixed by moving the job temporarily to the local hard drive on the node so that the shared-memory parallelization could use the memory-mapping protocol within Linux and recover the performance. However, this would impact any codes that use MPI underneath. The GNU compiler also struggled with a particular memory leak in the original ADT calculus design that required a significant re-work of *Cato*. The Intel compiler did not have this particular memory leak and had slightly better single-core performance, but it suffered from other MPI-related issues.

Getting *Cato* to run consistently using the 2019 & 2020 Intel compilers was particular challenging. The Intel MPI library that handles the coarray parallelization was significantly changed between the 2019 and 2020 versions. Finding the correct environment variables and settings to get *Cato*, or even simple test programs, to run with coarrays was fraught with errors. The same exact code could be run with different compiler versions and have varying levels of success. Certain versions only worked with shared memory or, in some cases, would not run at all.

Codes similar to *Cato* can benefit greatly from object-oriented and ADT design paradigms. Future versions of *Cato* would benefit from memory-based optimizations that could further improve scaling. As compilers and libraries mature, the user experience will improve with proper support of coarrays from each vendor.

Adapting the hydrodynamic implementation to create a new multi-physics capability in Julia was straightforward, even when simultaneously *learning* Julia. Though *Cygnus* can benefit from better parallelization (both in the explicit hydrodynamics solver and implicit thermal solver), the language is not the limiting factor. The language is simple to learn and is well suited to the future HPC landscape. The Climate Modelling Alliance (CliMA) of Caltech has recently developed weather prediction capability from the ground-up in Julia for both CPUs and GPUs that includes high-resolution turbulence modelling, data analysis, and machine learning [129]. Julia has also been applied to a solve high-order hyperbolic PDEs (including the compressible Euler and ideal magneto-hydrodynamic equations) with high-performance parallel adaptive mesh refinement capability [130]. Benchmark studies have also shown that Julia performs well in HPC workloads

across a variety of hardware [117, 118]. Julia provides for increased developer productivity through its language design and the availability of community-wide packages, which limit the need to re-invent the wheel for each new code. It also has the capability to call existing Python, Fortran, and C/C++ code that makes incremental changes possible without necessarily requiring a complete re-write of existing codes. In this situation, the high level interface and design would be handled by Julia, but existing low-level functionality could be incrementally included and translated as needed.

Chapter 6

Conclusion

This thesis studied the physics of hydrodynamic instability growth in two phases of ICF implosions and the numerical methods employed in the simulation of these instabilities. Chapter 2 focused on early-time perturbation evolution during shock transit leading up to the start of the acceleration phase. It was shown that perturbations created by internal defects such as voids or bubbles have the potential to create significant hydrodynamic instability seeds. Ablator material perturbations were shown to create significantly higher seeds due to a combination of position, timing, and 1D hydrodynamic wave evolution. Additionally, isolated defect simulations revealed complex wave dynamics inside the target due to characteristic wave propagation at disparate sound speeds, and reverberation within the shell that expanded the influence of these perturbations. Modulations in areal density showed that certain defect sizes and locations created distortions large enough to weaken or even puncture the shell shortly after the start of the acceleration phase. A strategy was proposed that reduced the secular feedout growth at the ablation front after the first shock and limited characteristic wave reverberation within the shell through the use of a thicker, low-density, wetted foam-like ablator.

Chapter 3 examined the stability of the fuel-shell interface, in both experiment and simulation, for room temperature direct-drive implosions during the deceleration phase. RT growth is enhanced in room temperature targets during the deceleration phase by the density discontinuity

and finite Atwood numbers at the fuel-shell interface. Small amplitude perturbations showed a difference in linear growth factors between 10:90 and 50:50, a difference that was reduced when the growth was nonlinear. Because both D:T 10:90 and 50:50 had similar effective Atwood numbers, the simulated deceleration RT instability growth was nearly identical for nonlinear RT growth, and there was little influence on the inferred T_{i} and ΔT_{i}.

Chapter 4 presented the numerical methods used to generate the results in Chapter 2. This covered both the solution of the hydrodynamic equations using a robust, noise-free Riemann solver coupled with high order spatial reconstruction, and the solution of the thermal conduction equation with the efficient direction-split ADI method.

Chapter 5 discussed the practical implementation of the numerical methods in Chapter 4. Two new 2D simulation codes were written that prioritized accurate simulation of characteristic wave propagation in the ICF context. Each code tested the feasibility of prioritizing flexible and clean design practices in different language paradigms. *Cato*, a parallel object-oriented code written in coarray Fortran, was applied to single-physics hydrodynamics. A new multi-physics code (*Cygnus*), written in Julia, applied lessons learned from the first implementation and tested the viability of Julia in high-performance computing. Performance and design comparisons showed that Julia is well-suited for HPC workloads.

6.1 Future Work

Future work on simulating ICF target internal defects could involve a transition to 3D geometry and the inclusion of additional physics effects. Internal dynamics due to defects depends heavily on shock-induced vorticity, which is limited in 2D, and hydrodynamic instability is also modified in 3D for both RT and RM growth [27, 28, 131]. Additional physics mechanisms such as laser ray-tracing (at least in 1D), realistic equations-of-state, multiple materials, two-temperature plasma treatment, radiation, and ionization could be added to *Cygnus* to more closely replicate an ICF implosion. A different mesh geometry that accounts for convergent geometry effects that play a role later in the acceleration phase would also be necessary.

To accommodate the additional physics mechanisms, *Cygnus* would need to be adapted to run in 3D, which is a significant undertaking. Parallelization via domain-decomposition is a must for larger 3D problems. The thermal conduction solver would also need to be updated to handle domain-decomposition. The MLP spatial reconstruction method is currently 2D-only, but 3D modifications have been presented in Ref. 132. Alternate solution methods, such as multi-dimensional Riemann solvers [133], WENO reconstruction [134], and ADER time integration could improve multi-dimensional performance [135, 136]. Incremental parallelization improvements could be added via GPUs, which have been widely proven to significantly improve performance, although adapting algorithms within the multi-physics paradigm is challenging. Rapid prototyping enabled by Julia, however, significantly improves this process [123], particularly compared to Fortran (which is only possible through vendor-specific OpenMP directives or compilers).

Experimental validation of perturbation evolution and growth from to micron-scale voids is highly desirable. Having the capability of observe features of this scale is a significant engineering. Recent developments in x-ray radiography and Fresnel zone plate imaging have been able to obtain a resolution of approximately $1.5\ \mu$m [137]. Target characterization at this scale is also a challenge, particularly regarding engineering repeatable features such as a void within the ablator material. Additionally, while direct observation of a defect may not yet be possible, observation of the secondary effects created by the defect perturbation may be possible in the near future.

Bibliography

[1] John Nuckolls, Lowell Wood, Albert Thiessen, and George Zimmerman. Laser Compression of Matter to Super-High Densities: Thermonuclear (CTR) Applications. *Nature*, 239 (5368):139–142, September 1972. ISSN 0028-0836, 1476-4687. doi:10.1038/239139a0.

[2] R. S. Craxton, K. S. Anderson, T. R. Boehly, V. N. Goncharov, D. R. Harding, J. P. Knauer, R. L. McCrory, P. W. McKenty, D. D. Meyerhofer, J. F. Myatt, A. J. Schmitt, J. D. Sethian, R. W. Short, S. Skupsky, W. Theobald, W. L. Kruer, K. Tanaka, R. Betti, T. J. B. Collins, J. A. Delettrez, S. X. Hu, J. A. Marozas, A. V. Maximov, D. T. Michel, P. B. Radha, S. P. Regan, T. C. Sangster, W. Seka, A. A. Solodov, J. M. Soures, C. Stoeckl, and J. D. Zuegel. Direct-drive inertial confinement fusion: A review. *Physics of Plasmas*, 22(11):110501, November 2015. ISSN 1070-664X, 1089-7674. doi:10.1063/1.4934714.

[3] E.M. Campbell, V.N. Goncharov, T.C. Sangster, S.P. Regan, P.B. Radha, R. Betti, J.F. Myatt, D.H. Froula, M.J. Rosenberg, I.V. Igumenshchev, W. Seka, A.A. Solodov, A.V. Maximov, J.A. Marozas, T.J.B. Collins, D. Turnbull, F.J. Marshall, A. Shvydky, J.P. Knauer, R.L. McCrory, A.B. Sefkow, M. Hohenberger, P.A. Michel, T. Chapman, L. Masse, C. Goyon, S. Ross, J.W. Bates, M. Karasik, J. Oh, J. Weaver, A.J. Schmitt, K. Obenschain, S.P. Obenschain, S. Reyes, and B. Van Wonterghem. Laser-direct-drive program: Promise, challenge, and path forward. *Matter and Radiation at Extremes*, 2(2): 37–54, March 2017. ISSN 2468080X. doi:10.1016/j.mre.2017.03.001.

[4] Stephen E. Bodner, Denis G. Colombant, John H. Gardner, Robert H. Lehmberg, Stephen P. Obenschain, Lee Phillips, Andrew J. Schmitt, John D. Sethian, Robert L. Mc-

Crory, Wolf Seka, Charles P. Verdon, James P. Knauer, Bedros B. Afeyan, and Howard T. Powell. Direct-drive laser fusion: Status and prospects. *Physics of Plasmas*, 5(5):1901–1918, May 1998. ISSN 1070-664X, 1089-7674. doi:10.1063/1.872861.

[5] N B Meezan, M J Edwards, O A Hurricane, P K Patel, D A Callahan, W W Hsing, R P J Town, F Albert, P A Amendt, L F Berzak Hopkins, D K Bradley, D T Casey, D S Clark, E L Dewald, T R Dittrich, L Divol, T Döppner, J E Field, S W Haan, G N Hall, B A Hammel, D E Hinkel, D D Ho, M Hohenberger, N Izumi, O S Jones, S F Khan, J L Kline, A L Kritcher, O L Landen, S LePape, T Ma, A J MacKinnon, A G MacPhee, L Masse, J L Milovich, A Nikroo, A Pak, H-S Park, J L Peterson, H F Robey, J S Ross, J D Salmonson, V A Smalyuk, B K Spears, M Stadermann, L J Suter, C A Thomas, R Tommasini, D P Turnbull, and C R Weber. Indirect drive ignition at the National Ignition Facility. *Plasma Physics and Controlled Fusion*, 59(1):014021, January 2017. ISSN 0741-3335, 1361-6587. doi:10.1088/0741-3335/59/1/014021.

[6] A. R. Christopherson, R. Betti, S. Miller, V. Gopalaswamy, O. M. Mannion, and D. Cao. Theory of ignition and burn propagation in inertial fusion implosions. *Physics of Plasmas*, 27(5):052708, May 2020. ISSN 1070-664X, 1089-7674. doi:10.1063/1.5143889.

[7] O. M. Mannion, V. Yu. Glebov, C. J. Forrest, J. P. Knauer, V. N. Goncharov, S. P. Regan, T. C. Sangster, C. Stoeckl, and M. Gatu Johnson. Calibration of a neutron time-of-flight detector with a rapid instrument response function for measurements of bulk fluid motion on OMEGA. *Review of Scientific Instruments*, 89(10):10I131, October 2018. ISSN 0034-6748, 1089-7623. doi:10.1063/1.5037324.

[8] O. M. Mannion, I. V. Igumenshchev, K. S. Anderson, R. Betti, E. M. Campbell, D. Cao, C. J. Forrest, M. Gatu Johnson, V. Yu. Glebov, V. N. Goncharov, V. Gopalaswamy, S. T. Ivancic, D. W. Jacobs-Perkins, A. Kalb, J. P. Knauer, J. Kwiatkowski, A. Lees, F. J. Marshall, M. Michalko, Z. L. Mohamed, D. Patel, H. G. Rinderknecht, R. C. Shah, C. Stoeckl, W. Theobald, K. M. Woo, and S. P. Regan. Mitigation of mode-one asymmetry in laser-

direct-drive inertial confinement fusion implosions. *Physics of Plasmas*, 28(4):042701, April 2021. ISSN 1070-664X, 1089-7674. doi:10.1063/5.0041554.

[9] V. N. Goncharov, T. C. Sangster, R. Betti, T. R. Boehly, M. J. Bonino, T. J. B. Collins, R. S. Craxton, J. A. Delettrez, D. H. Edgell, R. Epstein, R. K. Follett, C. J. Forrest, D. H. Froula, V. Yu. Glebov, D. R. Harding, R. J. Henchen, S. X. Hu, I. V. Igumenshchev, R. Janezic, J. H. Kelly, T. J. Kessler, T. Z. Kosc, S. J. Loucks, J. A. Marozas, F. J. Marshall, A. V. Maximov, R. L. McCrory, P. W. McKenty, D. D. Meyerhofer, D. T. Michel, J. F. Myatt, R. Nora, P. B. Radha, S. P. Regan, W. Seka, W. T. Shmayda, R. W. Short, A. Shvydky, S. Skupsky, C. Stoeckl, B. Yaakobi, J. A. Frenje, M. Gatu-Johnson, R. D. Petrasso, and D. T. Casey. Improving the hot-spot pressure and demonstrating ignition hydrodynamic equivalence in cryogenic deuterium–tritium implosions on OMEGA. *Physics of Plasmas*, 21(5):056315, May 2014. ISSN 1070-664X, 1089-7674. doi:10.1063/1.4876618.

[10] V. N. Goncharov, J. P. Knauer, P. W. McKenty, P. B. Radha, T. C. Sangster, S. Skupsky, R. Betti, R. L. McCrory, and D. D. Meyerhofer. Improved performance of direct-drive inertial confinement fusion target designs with adiabat shaping using an intensity picket. *Physics of Plasmas*, 10(5):1906–1918, May 2003. ISSN 1070-664X, 1089-7674. doi:10.1063/1.1562166.

[11] Kenneth Scott Anderson. *Adiabat Shaping in Direct-Drive Inertial Confinement Fusion Implosions*. PhD thesis, University of Rochester, Rochester, NY, 2006.

[12] T. J. B. Collins and S. Skupsky. Imprint reduction using an intensity spike in OMEGA cryogenic targets. *Physics of Plasmas*, 9(1):275–281, January 2002. ISSN 1070-664X, 1089-7674. doi:10.1063/1.1425840.

[13] K. S. Anderson, C. J. Forrest, O. M. Mannion, F. J. Marshall, R. C. Shah, D. T. Michel, J. A. Marozas, P. B. Radha, D. H. Edgell, R. Epstein, V. N. Goncharov, J. P. Knauer, M. Gatu Johnson, and S. Laffite. Effect of cross-beam energy transfer on target-offset

asymmetry in direct-drive inertial confinement fusion implosions. *Physics of Plasmas*, 27(11):112713, November 2020. ISSN 1070-664X, 1089-7674. doi:10.1063/5.0015781.

[14] O. M. Mannion, K. M. Woo, A. J. Crilly, C. J. Forrest, J. A. Frenje, M. Gatu Johnson, V. Yu. Glebov, J. P. Knauer, Z. L. Mohamed, M. H. Romanofsky, C. Stoeckl, W. Theobald, and S. P. Regan. Reconstructing 3D asymmetries in laser-direct-drive implosions on OMEGA. *Review of Scientific Instruments*, 92(3):033529, March 2021. ISSN 0034-6748, 1089-7623. doi:10.1063/5.0043514.

[15] C. J. Randall. Theory and simulation of stimulated Brillouin scatter excited by nonabsorbed light in laser fusion systems. *Physics of Fluids*, 24(8):1474, 1981. ISSN 00319171. doi:10.1063/1.863551.

[16] C. J. McKinstrie, J. S. Li, R. E. Giacone, and H. X. Vu. Two-dimensional analysis of the power transfer between crossed laser beams. *Physics of Plasmas*, 3(7):2686–2692, July 1996. ISSN 1070-664X, 1089-7674. doi:10.1063/1.871721.

[17] C. J. McKinstrie, A. V. Kanaev, V. T. Tikhonchuk, R. E. Giacone, and H. X. Vu. Three-dimensional analysis of the power transfer between crossed laser beams. *Physics of Plasmas*, 5(4):1142–1147, April 1998. ISSN 1070-664X, 1089-7674. doi:10.1063/1.872645.

[18] I. V. Igumenshchev, D. H. Edgell, V. N. Goncharov, J. A. Delettrez, A. V. Maximov, J. F. Myatt, W. Seka, A. Shvydky, S. Skupsky, and C. Stoeckl. Crossed-beam energy transfer in implosion experiments on OMEGA. *Physics of Plasmas*, 17(12):122708, December 2010. ISSN 1070-664X, 1089-7674. doi:10.1063/1.3532817.

[19] M. H. Emery, J. H. Gardner, R. H. Lehmberg, and S. P. Obenschain. Hydrodynamic target response to an induced spatial incoherence-smoothed laser beam. *Physics of Fluids B: Plasma Physics*, 3(9):2640–2651, September 1991. ISSN 0899-8221. doi:10.1063/1.859976.

[20] R. J. Taylor, J. P. Dahlburg, A. Iwase, J. H. Gardner, D. E. Fyfe, and O. Willi. Measurement and Simulation of Laser Imprinting and Consequent Rayleigh-Taylor Growth.

Physical Review Letters, 76(10):1643–1646, March 1996. ISSN 0031-9007, 1079-7114. doi:10.1103/PhysRevLett.76.1643.

[21] V. N. Goncharov, P. McKenty, S. Skupsky, R. Betti, R. L. McCrory, and C. Cherfils-Clérouin. Modeling hydrodynamic instabilities in inertial confinement fusion targets. *Physics of Plasmas*, 7(12):5118–5139, December 2000. ISSN 1070-664X, 1089-7674. doi:10.1063/1.1321016.

[22] D. R. Harding and W. T. Shmayda. Stress- and Radiation-Induced Swelling in Plastic Capsules. *Fusion Science and Technology*, 63(2):125–131, April 2013. ISSN 1536-1055, 1943-7641. doi:10.13182/FST13-A16329.

[23] Robert D. Richtmyer. Taylor instability in shock acceleration of compressible fluids. *Communications on Pure and Applied Mathematics*, 13(2):297–319, May 1960. ISSN 00103640, 10970312. doi:10.1002/cpa.3160130207.

[24] E. E. Meshkov. Instability of the interface of two gases accelerated by a shock wave. *Fluid Dynamics*, 4(5):101–104, September 1969. ISSN 1573-8507. doi:10.1007/BF01015969.

[25] Lord Rayleigh. Investigation of the character of the equilibrium of an incompressible heavy fluid of variable density. *Proceedings of the London Mathematical Society*, 14: 170–177, 1883.

[26] Geoffrey Taylor. The instability of liquid surfaces when accelerated in a direction perpendicular to their planes. I. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 201(1065):192–196, March 1950. ISSN 0080-4630, 2053-9169. doi:10.1098/rspa.1950.0052.

[27] Ye Zhou. Rayleigh–Taylor and Richtmyer–Meshkov instability induced flow, turbulence, and mixing. I. *Physics Reports*, 720–722:1–136, December 2017. ISSN 03701573. doi:10.1016/j.physrep.2017.07.005.

[28] Ye Zhou. Rayleigh–Taylor and Richtmyer–Meshkov instability induced flow, turbulence, and mixing. II. *Physics Reports*, 723–725:1–160, December 2017. ISSN 03701573. doi:10.1016/j.physrep.2017.07.008.

[29] V. N. Goncharov, O. V. Gotchev, E. Vianello, T. R. Boehly, J. P. Knauer, P. W. McKenty, P. B. Radha, S. P. Regan, T. C. Sangster, S. Skupsky, V. A. Smalyuk, R. Betti, R. L. McCrory, D. D. Meyerhofer, and C. Cherfils-Clérouin. Early stage of implosion in inertial confinement fusion: Shock timing and perturbation evolution. *Physics of Plasmas*, 13 (1):012702, January 2006. ISSN 1070-664X, 1089-7674. doi:10.1063/1.2162803.

[30] D. S. Clark, J. L. Milovich, D. E. Hinkel, J. D. Salmonson, J. L. Peterson, L. F. Berzak Hopkins, D. C. Eder, S. W. Haan, O. S. Jones, M. M. Marinak, H. F. Robey, V. A. Smalyuk, and C. R. Weber. A survey of pulse shape options for a revised plastic ablator ignition design. *Physics of Plasmas*, 21(11):112705, November 2014. ISSN 1070-664X, 1089-7674. doi:10.1063/1.4901572.

[31] D. T. Casey, J. L. Milovich, V. A. Smalyuk, D. S. Clark, H. F. Robey, A. Pak, A. G. MacPhee, K. L. Baker, C. R. Weber, T. Ma, H.-S. Park, T. Döppner, D. A. Callahan, S. W. Haan, P. K. Patel, J. L. Peterson, D. Hoover, A. Nikroo, C. B. Yeamans, F. E. Merrill, P. L. Volegov, D. N. Fittinghoff, G. P. Grim, M. J. Edwards, O. L. Landen, K. N. Lafortune, B. J. MacGowan, C. C. Widmayer, D. B. Sayre, R. Hatarik, E. J. Bond, S. R. Nagel, L. R. Benedetti, N. Izumi, S. Khan, B. Bachmann, B. K. Spears, C. J. Cerjan, M. Gatu Johnson, and J. A. Frenje. Improved Performance of High Areal Density Indirect Drive Implosions at the National Ignition Facility using a Four-Shock Adiabat Shaped Drive. *Physical Review Letters*, 115(10):105001, September 2015. ISSN 0031-9007, 1079-7114. doi:10.1103/PhysRevLett.115.105001.

[32] I V Igumenshchev, A L Velikovich, V N Goncharov, R Betti, E M Campbell, J P Knauer, S P Regan, A J Schmitt, R C Shah, and A Shvydky. Rarefaction Flows and Mitigation of Imprint in Direct-Drive Implosions. *PHYSICAL REVIEW LETTERS*, page 5, 2019.

[33] V N Goncharov. *Self-Consistent Stability Analysis of Ablations Fronts in Inertial Confinment Fusion*. PhD thesis, University of Rochester, Rochester, NY, 1998.

[34] H. Takabe, K. Mima, L. Montierth, and R. L. Morse. Self-consistent growth rate of the Rayleigh–Taylor instability in an ablatively accelerating plasma. *Physics of Fluids*, 28 (12):3676, 1985. ISSN 00319171. doi:10.1063/1.865099.

[35] J. D. Kilkenny, S. G. Glendinning, S. W. Haan, B. A. Hammel, J. D. Lindl, D. Munro, B. A. Remington, S. V. Weber, J. P. Knauer, and C. P. Verdon. A review of the ablative stabilization of the Rayleigh–Taylor instability in regimes relevant to inertial confinement fusion. *Physics of Plasmas*, 1(5):1379–1389, May 1994. ISSN 1070-664X, 1089-7674. doi:10.1063/1.870688.

[36] H. Takabe. Self-consistent eigenvalue analysis of Rayleigh–Taylor instability in an ablating plasma. *Physics of Fluids*, 26(8):2299, 1983. ISSN 00319171. doi:10.1063/1.864388.

[37] Stefano Atzeni and Mauro Temporal. Mechanism of growth reduction of the deceleration-phase ablative Rayleigh-Taylor instability. *Physical Review E*, 67(5):057401, May 2003. ISSN 1063-651X, 1095-3787. doi:10.1103/PhysRevE.67.057401.

[38] R. Betti, K. Anderson, V. N. Goncharov, R. L. McCrory, D. D. Meyerhofer, S. Skupsky, and R. P. J. Town. Deceleration phase of inertial confinement fusion implosions. *Physics of Plasmas*, 9(5):2277–2286, May 2002. ISSN 1070-664X, 1089-7674. doi:10.1063/1.1459458.

[39] Josselin Garnier and Catherine Cherfils. A multiscale analysis of the hotspot dynamics during the deceleration phase of inertial confinement capsules. *Physics of Plasmas*, 12 (1):012704, January 2005. ISSN 1070-664X, 1089-7674. doi:10.1063/1.1825389.

[40] F. Hattori, H. Takabe, and K. Mima. Rayleigh–Taylor instability in a spherically stagnating system. *Physics of Fluids*, 29(5):1719, 1986. ISSN 00319171. doi:10.1063/1.865637.

[41] H. Sakagami and K. Nishihara. Rayleigh–Taylor instability on the pusher–fuel contact surface of stagnating targets. *Physics of Fluids B: Plasma Physics*, 2(11):2715–2730, November 1990. ISSN 0899-8221. doi:10.1063/1.859395.

[42] P. B. Radha, V. N. Goncharov, T. J. B. Collins, J. A. Delettrez, Y. Elbaz, V. Yu. Glebov, R. L. Keck, D. E. Keller, J. P. Knauer, J. A. Marozas, F. J. Marshall, P. W. McKenty, D. D. Meyerhofer, S. P. Regan, T. C. Sangster, D. Shvarts, S. Skupsky, Y. Srebro, R. P. J. Town, and C. Stoeckl. Two-dimensional simulations of plastic-shell, direct-drive implosions on OMEGA. *Physics of Plasmas*, 12(3):032702, March 2005. ISSN 1070-664X, 1089-7674. doi:10.1063/1.1857530.

[43] P. B. Radha, T. J. B. Collins, J. A. Delettrez, Y. Elbaz, R. Epstein, V. Yu. Glebov, V. N. Goncharov, R. L. Keck, J. P. Knauer, J. A. Marozas, F. J. Marshall, R. L. McCrory, P. W. McKenty, D. D. Meyerhofer, S. P. Regan, T. C. Sangster, W. Seka, D. Shvarts, S. Skupsky, Y. Srebro, and C. Stoeckl. Multidimensional analysis of direct-drive, plastic-shell implosions on OMEGA. *Physics of Plasmas*, 12(5):056307, May 2005. ISSN 1070-664X, 1089-7674. doi:10.1063/1.1882333.

[44] T. C. Sangster, V. N. Goncharov, R. Betti, P. B. Radha, T. R. Boehly, D. T. Casey, T. J. B. Collins, R. S. Craxton, J. A. Delettrez, D. H. Edgell, R. Epstein, C. J. Forrest, J. A. Frenje, D. H. Froula, M. Gatu-Johnson, Y. Yu. Glebov, D. R. Harding, M. Hohenberger, S. X. Hu, I. V. Igumenshchev, R. Janezic, J. H. Kelly, T. J. Kessler, C. Kingsley, T. Z. Kosc, J. P. Knauer, S. J. Loucks, J. A. Marozas, F. J. Marshall, A. V. Maximov, R. L. McCrory, P. W. McKenty, D. D. Meyerhofer, D. T. Michel, J. F. Myatt, R. D. Petrasso, S. P. Regan, W. Seka, W. T. Shmayda, R. W. Short, A. Shvydky, S. Skupsky, J. M. Soures, C. Stoeckl, W. Theobald, V. Versteeg, B. Yaakobi, and J. D. Zuegel. Improving cryogenic deuterium–tritium implosion performance on OMEGA. *Physics of Plasmas*, 20(5):056317, May 2013. ISSN 1070-664X, 1089-7674. doi:10.1063/1.4805088.

[45] R. Epstein, R. C. Mancini, D. T. Cliche, R. C. Shah, T. J. B. Collins, C. Stoeckl, P. W. McKenty, P. B. Radha, S. P. Regan, and V. N. Goncharov. Self-radiography of imploded

shells on OMEGA based on additive-free multi-monochromatic continuum spectral analysis. *Physics of Plasmas*, 27(12):122709, December 2020. ISSN 1070-664X, 1089-7674. doi:10.1063/5.0021489.

[46] T. J. B. Collins, C. Stoeckl, R. Epstein, W. A. Bittle, C. J. Forrest, V. Yu. Glebov, V. N. Goncharov, D. R. Harding, S. X. Hu, D. W. Jacobs-Perkins, T. Z. Kosc, J. A. Marozas, C. Mileham, F. J. Marshall, S. F. B. Morse, P. B. Radha, S. P. Regan, B. Rice, T. C. Sangster, M. J. Shoup, W. T. Shmayda, C. Sorce, W. Theobald, and M. D. Wittman. Causes of fuel–ablator mix inferred from modeling of monochromatic time-gated radiography of OMEGA cryogenic implosions. *Physics of Plasmas*, 29(1):012702, January 2022. ISSN 1070-664X, 1089-7674. doi:10.1063/5.0060477.

[47] R. C. Shah, S. X. Hu, I. V. Igumenshchev, J. Baltazar, D. Cao, C. J. Forrest, V. N. Goncharov, V. Gopalaswamy, D. Patel, F. Philippe, W. Theobald, and S. P. Regan. Observations of anomalous x-ray emission at early stages of hot-spot formation in deuterium-tritium cryogenic implosions. *Physical Review E*, 103(2):023201, February 2021. doi:10.1103/PhysRevE.103.023201.

[48] I. V. Igumenshchev, V. N. Goncharov, W. T. Shmayda, D. R. Harding, T. C. Sangster, and D. D. Meyerhofer. Effects of local defect growth in direct-drive cryogenic implosions on OMEGA. *Physics of Plasmas*, 20(8):082703, August 2013. ISSN 1070-664X, 1089-7674. doi:10.1063/1.4818280.

[49] Brian M. Haines, R. E. Olson, W. Sweet, S. A. Yi, A. B. Zylstra, P. A. Bradley, F. Elsner, H. Huang, R. Jimenez, J. L. Kline, C. Kong, G. A. Kyrala, R. J. Leeper, R. Paguio, S. Pajoom, R. R. Peterson, M. Ratledge, and N. Rice. Robustness to hydrodynamic instabilities in indirectly driven layered capsule implosions. *Physics of Plasmas*, 26(1): 012707, January 2019. ISSN 1070-664X, 1089-7674. doi:10.1063/1.5080262.

[50] R. Betti, V. Lobatchev, and R. L. McCrory. Feedout and Rayleigh-Taylor Seeding Induced by Long Wavelength Perturbations in Accelerated Planar Foils. *Physi-*

cal Review Letters, 81(25):5560–5563, December 1998. ISSN 0031-9007, 1079-7114. doi:10.1103/PhysRevLett.81.5560.

[51] Alexander L. Velikovich, Andrew J. Schmitt, John H. Gardner, and Nathan Metzler. Feedout and Richtmyer–Meshkov instability at large density difference. *Physics of Plasmas*, 8(2):592–605, February 2001. ISSN 1070-664X, 1089-7674. doi:10.1063/1.1335829.

[52] T.R Boehly, D.L Brown, R.S Craxton, R.L Keck, J.P Knauer, J.H Kelly, T.J Kessler, S.A Kumpan, S.J Loucks, S.A Letzring, F.J Marshall, R.L McCrory, S.F.B Morse, W Seka, J.M Soures, and C.P Verdon. Initial performance results of the OMEGA laser system. *Optics Communications*, 133(1-6):495–506, January 1997. ISSN 00304018. doi:10.1016/S0030-4018(96)00325-2.

[53] S. X. Hu, V. N. Goncharov, P. B. Radha, J. A. Marozas, S. Skupsky, T. R. Boehly, T. C. Sangster, D. D. Meyerhofer, and R. L. McCrory. Two-dimensional simulations of the neutron yield in cryogenic deuterium-tritium implosions on OMEGA. *Physics of Plasmas*, 17(10):102706, October 2010. ISSN 1070-664X, 1089-7674. doi:10.1063/1.3491467.

[54] S. X. Hu, D. T. Michel, A. K. Davis, R. Betti, P. B. Radha, E. M. Campbell, D. H. Froula, and C. Stoeckl. Understanding the effects of laser imprint on plastic-target implosions on OMEGA. *Physics of Plasmas*, 23(10):102701, 2016. doi:10.1063/1.4962993.

[55] Brian M. Haines, Gary P. Grim, James R. Fincke, Rahul C. Shah, Chad J. Forrest, Kevin Silverstein, Frederic J. Marshall, Melissa Boswell, Malcolm M. Fowler, Robert A. Gore, Anna C. Hayes-Sterbenz, Gerard Jungman, Andreas Klein, Robert S. Rundberg, Michael J. Steinkamp, and Jerry B. Wilhelmy. Detailed high-resolution three-dimensional simulations of OMEGA separated reactants inertial confinement fusion experiments. *Physics of Plasmas*, 23(7):072709, July 2016. ISSN 1070-664X, 1089-7674. doi:10.1063/1.4959117.

[56] I. V. Igumenshchev, V. N. Goncharov, F. J. Marshall, J. P. Knauer, E. M. Campbell, C. J. Forrest, D. H. Froula, V. Yu. Glebov, R. L. McCrory, S. P. Regan, T. C. Sangster, S. Skupsky, and C. Stoeckl. Three-dimensional modeling of direct-drive cryogenic implosions on OMEGA. *Physics of Plasmas*, 23(5):052702, May 2016. ISSN 1070-664X, 1089-7674. doi:10.1063/1.4948418.

[57] V. N. Goncharov. Analytical Model of Nonlinear, Single-Mode, Classical Rayleigh-Taylor Instability at Arbitrary Atwood Numbers. *Physical Review Letters*, 88(13):134502, March 2002. ISSN 0031-9007, 1079-7114. doi:10.1103/PhysRevLett.88.134502.

[58] U. Alon, J. Hecht, D. Ofer, and D. Shvarts. Power Laws and Similarity of Rayleigh-Taylor and Richtmyer-Meshkov Mixing Fronts at All Density Ratios. *Physical Review Letters*, 74(4):534–537, January 1995. ISSN 0031-9007, 1079-7114. doi:10.1103/PhysRevLett.74.534.

[59] B Appelbe and J Chittenden. The production spectrum in fusion plasmas. *Plasma Physics and Controlled Fusion*, 53(4):045002, April 2011. ISSN 0741-3335, 1361-6587. doi:10.1088/0741-3335/53/4/045002.

[60] T. J. Murphy. The effect of turbulent kinetic energy on inferred ion temperature from neutron spectra. *Physics of Plasmas*, 21(7):072701, July 2014. ISSN 1070-664X, 1089-7674. doi:10.1063/1.4885342.

[61] H Brysk. Fusion neutron energies and spectra. *Plasma Physics*, 15(7):611–617, July 1973. ISSN 0032-1028. doi:10.1088/0032-1028/15/7/001.

[62] R. Hatarik, D. B. Sayre, J. A. Caggiano, T. Phillips, M. J. Eckart, E. J. Bond, C. Cerjan, G. P. Grim, E. P. Hartouni, J. P. Knauer, J. M. Mcnaney, and D. H. Munro. Analysis of the neutron time-of-flight spectra from inertial confinement fusion experiments. *Journal of Applied Physics*, 118(18):184502, November 2015. ISSN 0021-8979, 1089-7550. doi:10.1063/1.4935455.

[63] M. D. Wittman, M. J. Bonino, D. H. Edgell, C. Fella, D. R. Harding, and J. Sanchez. Effect of Tritium-Induced Damage on Plastic Targets from High-Density DT Permeation. *Fusion Science and Technology*, 73(3):315–323, April 2018. ISSN 1536-1055, 1943-7641. doi:10.1080/15361055.2017.1380496.

[64] F. Weilacher, P. B. Radha, and C. Forrest. Three-dimensional modeling of the neutron spectrum to infer plasma conditions in cryogenic inertial confinement fusion implosions. *Physics of Plasmas*, 25(4):042704, April 2018. ISSN 1070-664X, 1089-7674. doi:10.1063/1.5016856.

[65] Robert D. Richtmyer, Keith W. Morton, and Robert Davis Richtmyer. *Difference Methods for Initial-Value Problems*. Number 4 in Interscience Tracts in Pure and Applied Mathematics. Interscience Publ, New York, NY, 2. ed edition, 1967. ISBN 978-0-470-72040-0.

[66] Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. Strong Stability-Preserving High-Order Time Discretization Methods. *SIAM Review*, 43(1):89–112, January 2001. ISSN 0036-1445, 1095-7200. doi:10.1137/S003614450036757X.

[67] R. Courant, K. Friedrichs, and H. Lewy. On the Partial Difference Equations of Mathematical Physics. *IBM Journal of Research and Development*, 11(2):215–234, March 1967. ISSN 0018-8646, 0018-8646. doi:10.1147/rd.112.0215.

[68] Sergei K. Godunov and I. Bohachevsky. Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. *Matematičeskij sbornik*, 47(89)(3):271–306, 1959.

[69] Bram van Leer. Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. *Journal of Computational Physics*, 32(1):101–136, July 1979. ISSN 00219991. doi:10.1016/0021-9991(79)90145-1.

[70] Ami Harten. High resolution schemes for hyperbolic conservation laws. *Journal of Com-*

putational Physics, 49(3):357–393, March 1983. ISSN 00219991. doi:10.1016/0021-9991(83)90136-5.

[71] P. K. Sweby. High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws. *SIAM Journal on Numerical Analysis*, 21(5):995–1011, October 1984. ISSN 0036-1429, 1095-7170. doi:10.1137/0721062.

[72] Jonathan B Goodman and Randall J LeVeque. On the Accuracy of Stable Schemes for 2D Scalar Conservation Laws. *Mathematics of Computation*, 45(171):15–21, July 1985.

[73] Kyu Hong Kim and Chongam Kim. Accurate, efficient and monotonic numerical methods for multi-dimensional compressible flows, Part II: Multi-dimensional limiting process. *Journal of Computational Physics*, 208(2):570–615, September 2005. ISSN 00219991. doi:10.1016/j.jcp.2005.02.022.

[74] Jiri Blazek. *Computational Fluid Dynamics: Principles and Applications*. Butterworth-Heinemann, Amsterdam, third edition edition, 2015. ISBN 978-0-08-099995-1.

[75] Yutao Sun and Yu-Xin Ren. The finite volume local evolution Galerkin method for solving the hyperbolic conservation laws. *Journal of Computational Physics*, 228(13):4945–4960, July 2009. ISSN 00219991. doi:10.1016/j.jcp.2009.04.001.

[76] P.L Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357–372, October 1981. ISSN 00219991. doi:10.1016/0021-9991(81)90128-5.

[77] Nico Fleischmann. A shock-stable modification of the HLLC Riemann solver with reduced numerical dissipation. *Journal of Computational Physics*, 401:21, January 2020. ISSN 00219991. doi:10.1016/j.jcp.2019.109004.

[78] Meng-Sing Liou. A Sequel to AUSM: AUSM+. *Journal of Computational Physics*, 129 (2):364–382, December 1996. ISSN 00219991. doi:10.1006/jcph.1996.0256.

[79] Meng-Sing Liou. A sequel to AUSM, Part II: AUSM+-up for all speeds. *Journal of Computational Physics*, 214(1):137–170, May 2006. ISSN 00219991. doi:10.1016/j.jcp.2005.09.020.

[80] Eiji Shima and Keiichi Kitamura. Parameter-Free Simple Low-Dissipation AUSM-Family Scheme for All Speeds. *AIAA Journal*, 49(8):1693–1709, August 2011. ISSN 0001-1452, 1533-385X. doi:10.2514/1.J050905.

[81] Keiichi Kitamura and Eiji Shima. Towards shock-stable and accurate hypersonic heating computations: A new pressure flux for AUSM-family schemes. *Journal of Computational Physics*, 245:62–83, July 2013. ISSN 00219991. doi:10.1016/j.jcp.2013.02.046.

[82] E. F. Toro, M. Spruce, and W. Speares. Restoration of the contact surface in the HLL-Riemann solver. *Shock Waves*, 4(1):25–34, July 1994. ISSN 0938-1287, 1432-2153. doi:10.1007/BF01414629.

[83] Nico Fleischmann. A shock-stable modification of the HLLC Riemann solver with reduced numerical dissipation. *Journal of Computational Physics*, page 21, 2020.

[84] Kyu Hong Kim, Chongam Kim, and Oh-Hyun Rho. Methods for the Accurate Computations of Hypersonic Flows, Part I AUSMPW+ Scheme. *Journal of Computational Physics*, 174(1):38–80, November 2001. ISSN 00219991. doi:10.1006/jcph.2001.6873.

[85] Kyu Hong Kim and Chongam Kim. Accurate, efficient and monotonic numerical methods for multi-dimensional compressible flows, Part I: Spatial Discretization. *Journal of Computational Physics*, 208(2):527–569, September 2005. ISSN 00219991. doi:10.1016/j.jcp.2005.02.021.

[86] James J. Quirk. A contribution to the great Riemann solver debate. *International Journal for Numerical Methods in Fluids*, 18(6):555–574, March 1994. ISSN 0271-2091, 1097-0363. doi:10.1002/fld.1650180603.

[87] Maurizio Pandolfi and Domenic D'Ambrosio. Numerical Instabilities in Upwind Methods: Analysis and Cures for the "Carbuncle" Phenomenon. *Journal of Computational Physics*, 166(2):271–301, January 2001. ISSN 00219991. doi:10.1006/jcph.2000.6652.

[88] Hyung-Min Kang, Kyu Hong Kim, and Dong-Ho Lee. A new approach of a limiting process for multi-dimensional flows. *Journal of Computational Physics*, 229(19):7102–7128, September 2010. ISSN 00219991. doi:10.1016/j.jcp.2010.06.001.

[89] Bram van Leer. Flux-vector splitting for the Euler equations. In H. Araki, J. Ehlers, K. Hepp, R. Kippenhahn, H. A. Weidenmüller, J. Zittartz, and E. Krause, editors, *Eighth International Conference on Numerical Methods in Fluid Dynamics*, volume 170, pages 507–512. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982. ISBN 978-3-540-11948-7 978-3-540-39532-4. doi:10.1007/3-540-11948-5_66.

[90] Lyman Spitzer and Raymond J. Seeger. Physics of Fully Ionized Gases. *American Journal of Physics*, 31(11):890–891, November 1963. ISSN 0002-9505, 1943-2909. doi:10.1119/1.1969155.

[91] Suhas V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Series in Computational Methods in Mechanics and Thermal Sciences. Hemisphere Pub. Corp. ; McGraw-Hill, Washington : New York, 1980. ISBN 978-0-07-048740-6.

[92] D. W. Peaceman and H. H. Rachford, Jr. The Numerical Solution of Parabolic and Elliptic Differential Equations. *Journal of the Society for Industrial and Applied Mathematics*, 3 (1):28–41, March 1955. ISSN 0368-4245, 2168-3484. doi:10.1137/0103003.

[93] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *The Astrophysical Journal Supplement Series*, 131(1):273–334, November 2000. ISSN 0067-0049, 1538-4365. doi:10.1086/317361.

[94] David E Keyes, Lois C McInnes, Carol Woodward, William Gropp, Eric Myra, Michael Pernice, John Bell, Jed Brown, Alain Clo, Jeffrey Connors, Emil Constantinescu, Don Estep, Kate Evans, Charbel Farhat, Ammar Hakim, Glenn Hammond, Glen Hansen, Judith Hill, Tobin Isaac, Xiangmin Jiao, Kirk Jordan, Dinesh Kaushik, Efthimios Kaxiras, Alice Koniges, Kihwan Lee, Aaron Lott, Qiming Lu, John Magerlein, Reed Maxwell, Michael McCourt, Miriam Mehl, Roger Pawlowski, Amanda P Randles, Daniel Reynolds, Beatrice Rivière, Ulrich Rüde, Tim Scheibe, John Shadid, Brendan Sheehan, Mark Shephard, Andrew Siegel, Barry Smith, Xianzhu Tang, Cian Wilson, and Barbara Wohlmuth. Multiphysics simulations: Challenges and opportunities. *The International Journal of High Performance Computing Applications*, 27(1):4–83, February 2013. ISSN 1094-3420, 1741-2846. doi:10.1177/1094342012468181.

[95] Victor R. Basili, Jeffrey C. Carver, Daniela Cruzes, Lorin M. Hochstein, Jeffrey K. Hollingsworth, Forrest Shull, and Marvin V. Zelkowitz. Understanding the High-Performance-Computing Community: A Software Engineer's Perspective. *IEEE Software*, 25(4):29–36, July 2008. ISSN 0740-7459, 1937-4194. doi:10.1109/MS.2008.103.

[96] Michael Metcalf, John Ker Reid, and Malcolm Cohen. *Modern Fortran Explained: Incorporating Fortran 2018*. Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, new edition edition, 2018. ISBN 978-0-19-881188-6 978-0-19-881189-3.

[97] L. Dagum and R. Menon. OpenMP: An industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, Jan.-March/1998. ISSN 10709924. doi:10.1109/99.660313.

[98] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The MPI Message Passing Interface Standard. In Karsten M. Decker and René M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 213–218. Birkhäuser Basel, Basel, 1994. ISBN 978-3-0348-9668-9 978-3-0348-8534-8. doi:10.1007/978-3-0348-8534-8_21.

[99] Soren Rasmussen. Development and performance comparison of MPI and Fortran Coarrays within an atmospheric research model. In *PAW-ATM 18: Parallel Applications Workshop, Alternatives to MPI*, page 4, Dallas, TX, USA, 2018.

[100] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, January 2017. ISSN 0036-1445, 1095-7200. doi:10.1137/141000671.

[101] HPCwire: Julia Update: Adoption Keeps Climbing; Is It a Python Challenger?, January 2021. URL https://www.hpcwire.com/2021/01/13/julia-update-adoption-keeps-climbing-is-it-a-python-challenger/.

[102] Damian Rouson, Jim Xia, and Xiaofeng Xu. *Scientific Software Design the Object-Oriented Way*. Cambridge University Press, Cambridge, 2014. ISBN 978-0-521-88813-4 978-0-511-97738-1.

[103] Robert Robey and Yuliana Zamora. *Parallel and High Performance Computing*. Manning, Shelter Island, 2021. ISBN 978-1-61729-646-8.

[104] Erich Gamma, editor. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, Mass, 1995. ISBN 978-0-201-63361-0.

[105] Damian W.I. Rouson. Towards Analysis-Driven Scientific Software Architecture: The Case for Abstract Data Type Calculus. *Scientific Programming*, 16(4):329–339, 2008. ISSN 1058-9244, 1875-919X. doi:10.1155/2008/393918.

[106] Milan Curcic. *Modern Fortran: Building Efficient Parallel Applications*. O'REILLY MEDIA, Place of publication not identified, 2020. ISBN 978-1-61729-528-7.

[107] Tom Clune. pfunit: Parallel Fortran Unit Testing Framework, 2019. URL https://github.com/Goddard-Fortran-Ecosystem/pFUnit.

[108] Gary A Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1–31, April 1978. ISSN 00219991. doi:10.1016/0021-9991(78)90023-2.

[109] Colin P. McNally, Wladimir Lyra, and Jean-Claude Passy. A WELL-POSED KELVIN-HELMHOLTZ INSTABILITY TEST AND COMPARISON. *The Astrophysical Journal Supplement Series*, 201(2):18, August 2012. ISSN 0067-0049, 1538-4365. doi:10.1088/0067-0049/201/2/18.

[110] Richard Liska and Burton Wendroff. Comparison of Several Difference Schemes on 1D and 2D Test Problems for the Euler Equations. *SIAM Journal on Scientific Computing*, 25(3):995–1017, January 2003. ISSN 1064-8275, 1095-7197. doi:10.1137/S1064827502402120.

[111] Nico Fleischmann, Stefan Adami, and Nikolaus A. Adams. Numerical symmetry-preserving techniques for low-dissipation shock-capturing schemes. *Computers & Fluids*, 189:94–107, July 2019. ISSN 00457930. doi:10.1016/j.compfluid.2019.04.004.

[112] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference on - AFIPS '67 (Spring)*, page 483, Atlantic City, New Jersey, 1967. ACM Press. doi:10.1145/1465482.1465560.

[113] John L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5): 532–533, May 1988. ISSN 0001-0782, 1557-7317. doi:10.1145/42411.42415.

[114] Tuowen Zhao, Samuel Williams, Mary Hall, and Hans Johansen. Delivering Performance-Portable Stencil Computations on CPUs and GPUs Using Bricks. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 59–70, Dallas, TX, USA, November 2018. IEEE. ISBN 978-1-72810-220-7. doi:10.1109/P3HPC.2018.00009.

[115] E. F. Toro, R. C. Millington, and L. A. M. Nejad. Towards Very High Order Godunov Schemes. In E. F. Toro, editor, *Godunov Methods*, pages 907–940. Springer US, New York, NY, 2001. ISBN 978-1-4613-5183-2 978-1-4615-0663-8. doi:10.1007/978-1-4615-0663-8_87.

[116] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86, San Jose, CA, USA, 2004. IEEE. ISBN 978-0-7695-2102-2. doi:10.1109/CGO.2004.1281665.

[117] Sascha Hunold and Sebastian Steiner. Benchmarking Julia's Communication Performance: Is Julia HPC ready or Full HPC? In *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 20–25, GA, USA, November 2020. IEEE. ISBN 978-1-66542-265-9. doi:10.1109/PMBS51919.2020.00008.

[118] Wei-Chen Lin and Simon McIntosh-Smith. Comparing Julia to Performance Portable Parallel Programming Models for HPC. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, page 12, St. Louis, MO, 2021. doi:10.1109/PMBS54543.2021.00016.

[119] Christian R. Trott, Damien Lebrun-Grandie, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah Wilke. Kokkos 3: Programming Model Extensions for the Exascale Era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):805–817, April 2022. ISSN 1045-9219, 1558-2183, 2161-9883. doi:10.1109/TPDS.2021.3097283.

[120] John E. Stone, David Gohara, and Guochun Shi. OpenCL: A Parallel Programming Stan-

dard for Heterogeneous Computing Systems. *Computing in Science & Engineering*, 12 (3):66–73, May 2010. ISSN 1521-9615. doi:10.1109/MCSE.2010.69.

[121] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable Parallel Programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, March 2008. ISSN 1542-7730, 1542-7749. doi:10.1145/1365490.1365500.

[122] Intel. oneapi: A New Era of Heterogeneous Computing, 2019. URL https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html.

[123] Tim Besard, Valentin Churavy, Alan Edelman, and Bjorn De Sutter. Rapid software prototyping for heterogeneous and distributed platforms. *Advances in Engineering Software*, 132:29–46, June 2019. ISSN 09659978. doi:10.1016/j.advengsoft.2019.02.002.

[124] Stefan Karpinski. JuliaCon: The Unreasonable Effectiveness of Multiple Dispatch, 2019. URL https://www.youtube.com/watch?v=kc9HwsxE1OY.

[125] Gilbert Strang. On the Construction and Comparison of Difference Schemes. *SIAM Journal on Numerical Analysis*, 5(3):506–517, September 1968. ISSN 0036-1429, 1095-7170. doi:10.1137/0705041.

[126] Sheheeda Manakkadu and Sourav Dutta. Bandwidth Based Performance Optimization of Multi-threaded Applications. In *2014 Sixth International Symposium on Parallel Architectures, Algorithms and Programming*, pages 118–122, Beijing, China, July 2014. IEEE. ISBN 978-1-4799-3845-2 978-1-4799-3844-5. doi:10.1109/PAAP.2014.51.

[127] John Reid, Bill Long, and Jon Steidel. History of coarrays and SPMD parallelism in Fortran. *Proceedings of the ACM on Programming Languages*, 4(HOPL):1–30, June 2020. ISSN 2475-1421. doi:10.1145/3386322.

[128] Alessandro Fanfarillo, Tobias Burnus, Valeria Cardellini, Salvatore Filippone, Dan Nagle, and Damian Rouson. OpenCoarrays: Open-source Transport Layers Supporting

Coarray Fortran Compilers. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models - PGAS '14*, pages 1–11, Eugene, OR, USA, 2014. ACM Press. ISBN 978-1-4503-3247-7. doi:10.1145/2676870.2676876.

[129] Ali Ramadhan, Gregory Wagner, Chris Hill, Jean-Michel Campin, Valentin Churavy, Tim Besard, Andre Souza, Alan Edelman, Raffaele Ferrari, and John Marshall. Oceananigans.jl: Fast and friendly geophysical fluid dynamics on GPUs. *Journal of Open Source Software*, 5(53):2018, September 2020. ISSN 2475-9066. doi:10.21105/joss.02018.

[130] Hendrik Ranocha, Michael Schlottke-Lakemper, Andrew R. Winters, Erik Faulhaber, Jesse Chan, and Gregor J. Gassner. Adaptive numerical simulations with Trixi.jl: A case study of Julia for scientific computing. *JuliaCon Proceedings*, 1(1):77, January 2022. ISSN 2642-4029. doi:10.21105/jcon.00077.

[131] P. A. Kuchugov, V. B. Rozanov, and N. V. Zmitrenko. The differences in the development of Rayleigh-Taylor instability in 2D and 3D geometries. *Plasma Physics Reports*, 40(6): 451–458, June 2014. ISSN 1063-780X, 1562-6938. doi:10.1134/S1063780X14060038.

[132] Sung-Hwan Yoon, Chongam Kim, and Kyu-Hong Kim. Multi-dimensional Limiting Process for Two- and Three-dimensional Flow Physics Analyses. In Herman Deconinck and E. Dick, editors, *Computational Fluid Dynamics 2006*, pages 185–190. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-540-92778-5 978-3-540-92779-2. doi:10.1007/978-3-540-92779-2_27.

[133] Dinshaw S. Balsara. Multidimensional Riemann problem with self-similar internal structure. Part I – Application to hyperbolic conservation laws on structured meshes. *Journal of Computational Physics*, 277:163–200, November 2014. ISSN 00219991. doi:10.1016/j.jcp.2014.07.053.

[134] Gioele Janett, Oskar Steiner, Ernest Alsina Ballester, Luca Belluzzi, and Siddhartha Mishra. A novel fourth-order WENO interpolation technique: A possible new tool de-

signed for radiative transfer. *Astronomy & Astrophysics*, 624:A104, April 2019. ISSN 0004-6361, 1432-0746. doi:10.1051/0004-6361/201834761.

[135] Dinshaw S. Balsara. Higher-order accurate space-time schemes for computational astrophysics—Part I: Finite volume methods. *Living Reviews in Computational Astrophysics*, 3(1):2, December 2017. ISSN 2367-3621, 2365-0524. doi:10.1007/s41115-017-0002-8.

[136] Saray Busto, Simone Chiocchetti, Michael Dumbser, Elena Gaburro, and Ilya Peshkov. High Order ADER Schemes for Continuum Mechanics. *Frontiers in Physics*, 8:32, March 2020. ISSN 2296-424X. doi:10.3389/fphy.2020.00032.

[137] F. J. Marshall, S. T. Ivancic, C. Mileham, P. M. Nilson, J. J. Ruby, C. Stoeckl, B. S. Scheiner, and M. J. Schmitt. High-resolution x-ray radiography with Fresnel zone plates on the University of Rochester's OMEGA Laser Systems. *Review of Scientific Instruments*, 92(3):033701, March 2021. ISSN 0034-6748. doi:10.1063/5.0034903.