REVEALING THE PHYSICS OF GALACTIC WINDS THROUGH MASSIVELY-PARALLEL HYDRODYNAMICS SIMULATIONS

by

Evan Elizabeth Schneider

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ASTRONOMY

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY
WITH A MAJOR IN ASTRONOMY AND ASTROPHYSICS

In the Graduate College

THE UNIVERSITY OF ARIZONA

2017

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Evan Elizabeth Schneider, titled Revealing the Physics of Galactic Winds through Massively-Parallel Hydrodynamics Simulations and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Date: 20 April 2017
Brant Robertson

Date: 20 April 2017
Gurtina Besla

Date: 20 April 2017
Daniel Marrone

Date: 20 April 2017
Philip Pinto

Date: 20 April 2017
Benjamin Weiner

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Date: 20 April 2017
Dissertation Director: Brant Robertson

Date: 20 April 2017
Dissertation Director: Gurtina Besla

STATEMENT BY AUTHOR

SIGNED: Evan Elizabeth Schneider

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help and support of many people, who I am delighted to have the opportunity to thank formally here. First and foremost, I thank my advisor, Brant Robertson, for your incredible investment in me and for being my constant champion for the last five years. When I first walked into your office five years ago, I had never thought I could be a theoretical astrophysicist or a computational scientist, and now I find I am both. I would also like to thank those people who have been members of my thesis committee over the years, including Gurtina Besla, Rachel Bezanson, Dan Marrone, Phil Pinto, and Ben Weiner, for guiding me through several mentoring meetings that all seemed to end with the primary takeaway, "Don't worry so much." I owe a special thank you to Gurtina, for giving me countless hours of excellent advice and for being my "advisor-in-residence" for the past two years. Next, I want to thank the many friends and colleagues who have made Steward such a wonderful place to work for the past six years. In particular, thank you to Cameron Hummels, who helped teach me how to code in the very beginning; to Steph Sallum, who is a great roommate and a great listener; to Justin Spilker, who is the only reason I come to work some days; and to Jordan Stone, who deserves thanks for so many things that I couldn't possibly list them all here, and I will content myself with simply his creation of the acronym, Cholla. Another special thank you goes to Rachel Bezanson, for being not only the best mentor I've ever had, but also a fantastic friend, and for teaching me all that I know about the whiskey with which I will celebrate this achievement. To my parents, William and Nancy, thank you for always believing in me, teaching me to value excellence, and challenging me not to settle for "good enough," but rather always try to do my best. To my sister, Michael, thank you for reminding me not to take life too seriously and always making me smile. And last, but certainly not least, to my wife, Kiera: for being with me this entire time, for supporting me when I said was going to quit and when I vowed to keep going, and for constantly reminding me what matters most, I thank you, from the bottom of my heart.

DEDICATION

For my wife, Kiera.

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

TABLE OF CONTENTS – *Continued*

LIST OF FIGURES

LIST OF FIGURES – *Continued*

LIST OF TABLES

ABSTRACT

This thesis documents the hydrodynamics code *Cholla* and a numerical study of multiphase galactic winds. *Cholla* is a massively-parallel, GPU-based code designed for astrophysical simulations that is freely available to the astrophysics community. A static-mesh Eulerian code, *Cholla* is ideally suited to carrying out massive simulations ($> 2048^3$ cells) that require very high resolution. The code incorporates state-of-the-art hydrodynamics algorithms including third-order spatial reconstruction, exact and linearized Riemann solvers, and unsplit integration algorithms that account for transverse fluxes on multidimensional grids. Operator-split radiative cooling and a dual-energy formalism for high mach number flows are also included. An extensive test suite demonstrates *Cholla*'s superior ability to model shocks and discontinuities, while the GPU-native design makes the code extremely computationally efficient - speeds of 5-10 million cell updates per GPU-second are typical on current hardware for 3D simulations with all of the aforementioned physics. The latter half of this work comprises a comprehensive study of the mixing between a hot, supernova-driven wind and cooler clouds representative of those observed in multiphase galactic winds. Both adiabatic and radiatively-cooling clouds are investigated. The analytic theory of cloud-crushing is applied to the problem, and adiabatic turbulent clouds are found to be mixed with the hot wind on similar timescales as the classic spherical case (4-5 t_{cc}) with an appropriate rescaling of the cloud-crushing time. Radiatively cooling clouds survive considerably longer, and the differences in evolution between turbulent and spherical clouds cannot be reconciled with a simple rescaling. The rapid incorporation of low-density material into the hot wind implies efficient mass-loading of hot phases of galactic winds. At the same time, the extreme compression of high-density cloud material leads to long-lived but slow-moving clumps that are unlikely to escape the galaxy.

CHAPTER 1

INTRODUCTION

This thesis consists of two parts: the development of the massively-parallel hydrodynamics code *Cholla*, and the use of that code to address a specific problem in the field of galaxy evolution, namely, the complicated dynamics of multiphase galactic winds. The motivation behind *Cholla*'s creation was not to build a code that would answer a particular question, but rather to make a code that could answer a host of questions in multiple fields. Similarly, galactic winds are an interesting subject in their own right, independent of the method used to study them. Therefore, rather than couch one topic in the context of the other, this introduction will cover the two topics separately. Section 1.1, numerical hydrodynamics in astrophysics, provides the background necessary to understand why I created *Cholla*, as well as the theoretical framework on which the code was developed. Section 1.2 motivates the study of multiphase galactic winds I have conducted using *Cholla*.

1.1 Numerical Hydrodynamics in Astrophysics

Numerical simulations have become a critical tool for modern theoretical astrophysics research. Simulations can model systems too complex to investigate analytically, and provide an avenue for experimentation in a field where much of the physics behind observed phenomena proves impossible to recreate in a laboratory. In subjects ranging from the evolution of protoplanetary disks to cosmological structure formation, hydrodynamics codes - those that evolve the fluid equations in one form or another - have been used to study problems that are analytically intractable.

Owing to their broad applicability, a wide variety of astrophysical hydrodynamics codes have been developed. The smoothed particle hydrodynamics (SPH) (Lucy, 1977; Gingold and Monaghan, 1977) and adaptive mesh refinement (AMR) (Berger and Oliger, 1984; Berger and Colella, 1989) techniques led to the creation of codes that can simulate astrophysical systems over many orders of magnitude in dynamic range. As the first generation of codes matured, physics beyond hydrodynamics was added, including the effects of conduction, radiation, and magnetic fields. Current state-of-the-art codes used to model galaxy evolution now typically include a particle-based gravity solver, a hydrodynamics (or magnetohydrodynamics) integration algorithm, a method to calculate self-gravity of the gas, a treatment of radiative cooling, and prescriptions for star formation and stellar feedback (e.g. Kravtsov, 1999; Fryxell et al., 2000; Springel, 2005, 2010a; Bryan et al., 2014; Hopkins, 2015).

1.1.1 The Importance of Supercomputing

The codes described above are necessarily complex, and the hydrodynamics integration is particularly computationally demanding. Therefore, achieving sufficient resolution to follow galaxy evolution over cosmic timescales relies heavily on the increasingly powerful computers available to researchers today. As the number and power of central processing units (CPUs) in high performance computing clusters has increased, so too has the size of the astrophysical simulations being run (see Figure 1.1). Modern hydrodynamics simulations often run on thousands to hundreds of thousands of CPU cores, meaning even those codes originally designed for serial CPU architectures must be updated to appropriately utilize parallel systems.

For many years, increasing the power of supercomputers meant increasing the number of CPU cores available, and increasing the clock speed of those cores. In the last decade, however, supercomputing has undergone a revolution. The fastest modern supercomputers gain the bulk of their speed from specialized hardware, or *accelerators*, such as the Intel Xeon Phi coprocessor or NVIDIA graphics processing units (GPUs)[1]. For example, Titan, the largest open science supercomputer in the United States, contains over 18,000 NVIDIA GPUs. With careful construction, codes that take advantage of these accelerators can speed up simulations by orders of magnitude (see Figure 1.2).

[1]`http://www.top500.org`

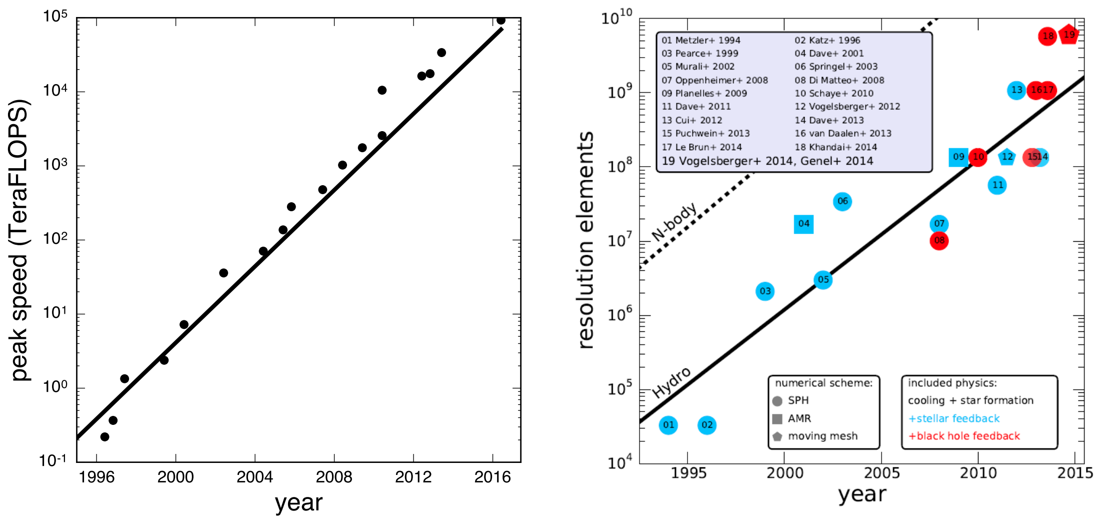

Figure 1.1: As computational resources have grown and algorithms have improved, the size of astrophysical simulations has increased. The plot on the left shows the peak speed of the fastest supercomputer in the world as a function of time, while the plot on the right shows the size of various astrohpysical hydrodynamics simulations. Both figures have Moore's Law plotted as a solid black line; on the left the doubling time is 14 months, on the right 20 months. As clusters have begun to incorporate more complex architectures, computational power has been outpacing simulation size. Right figure from www.illustris-project.org; supercomputing data from www.top500.org.

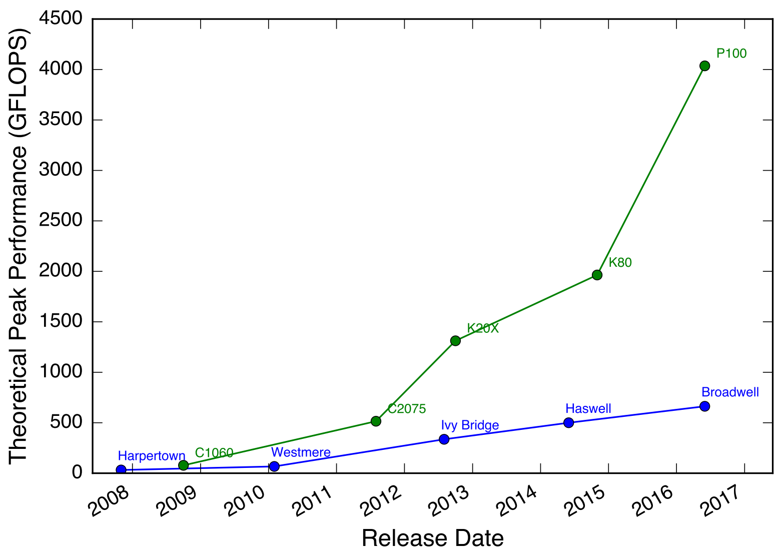

Figure 1.2: Theoretical peak performance (double precision) of representative Intel CPUs (blue) and NVIDIA general-purpose GPUs (green). Data from Intel and NVIDIA processor specifications.

In order to successfully harness this massive increase in computational power, a hydrodynamics code must be written with parallelism in mind at every step - simply porting an existing code designed for CPUs does not typically produce impressive results. At the time that this thesis work began, no GPU-based code for astrophysics was publicly available. This lack provided the motivation for the development of *Cholla*, an astrophysics code designed to run natively on GPUs. As detailed in Chapter 2, *Cholla* was built from the start with the ultimate goal of running on systems like Titan. Therefore, every design choice took into consideration the challenge of scaling well to thousands of GPUs - without sacrificing accuracy in the hydrodynamics algorithms. Given these requirements, a static-mesh, finite-volume code was a natural choice of architecture, as described in the following section.

1.1.2 Eulerian Versus Lagrangian Methods

Many techniques exist for numerically modeling the compressible fluid equations, and a full review of computational fluid dynamics is beyond the scope of this work. However, a brief accounting of popular methods employed in astrophysics is warranted. At present, the two most common choices are SPH, a Lagrangian method in which fluid variables like mass and energy are assigned to particles and evolved in their own rest-frame, and Eulerian grid codes, which track the flux of fluid quantities across cell interfaces. Each method has pros and cons. SPH is less susceptible to round-off errors in high-mach-number flows, naturally yields maximum resolution in the highest density regions, and more easily preserves hydrostatic equilibrium (see review by Springel, 2010b). Grid codes model shocks with higher accuracy, conserve fluxes of fluid variables across the simulation volume, and can be designed to yield high resolution in any desired region (see review by Teyssier, 2015). Because of their superior ability in modeling shocks and other discontinuities, Eulerian codes are often the better choice for resolving hydrodynamic instabilities and simulating turbulent processes. In addition, their gridded structure makes them a natural fit for the many-core architecture of GPUs.

In the last several years, a few groups have begun using codes that combine a Lagrangian and Eulerian approach. These techniques, including moving-mesh (Springel, 2010a) and meshless-finite-volume (Hopkins, 2015), employ the Riemann solvers of a grid code (which allows them to correctly model shocks), while also letting the reference frame move with the fluid in Lagrangian fashion. While promising, these methods involve a high degree of complexity and sructural overhead, making them more time-consuming to build and their numerical errors more difficult to quantify. In addition, these codes have been applied to computational problems that are already being studied by existing SPH or AMR codes. *Cholla* is designed to fill an entirely separate niche - the simulation of problems that require very high resolution at all locations across the volume - making a highly efficient static-mesh grid code an appropriate choice.

1.1.3 Finite-Volume Codes: Theoretical Framework[2]

Numerical modeling of the fluid equations in a grid code requires a discretization based on local volume elements, or cells. Within each cell, the conserved fluid variables of density ρ, momentum density, $\rho\boldsymbol{v}$, and energy density, E, are tracked over time. The energy density is a combination of the kinetic and internal energies, $E = \rho(\frac{1}{2}\boldsymbol{v}^2 + e)$, where $\boldsymbol{v}$ is the three-component velocity vector. The compressible fluid equations can be written in terms of the conserved variables:

$$\frac{\delta\rho}{\delta t} + \boldsymbol{\nabla}\cdot(\rho\boldsymbol{v}) = 0, \tag{1.1}$$

$$\frac{\delta(\rho\boldsymbol{v})}{\delta t} + \boldsymbol{\nabla}\cdot(\rho\boldsymbol{v}\otimes\boldsymbol{v} + p\mathbf{I}) = 0, \tag{1.2}$$

$$\frac{\delta E}{\delta t} + \boldsymbol{\nabla}\cdot(\boldsymbol{v}(E+p)) = 0, \tag{1.3}$$

with the addition of the pressure, p. (Here $\boldsymbol{I}$ is the identity matrix and $\otimes$ denotes the tensor product.) Adding an equation of state creates a closed system that can be modeled using standard numerical techniques for hyperbolic problems. In *Cholla*, we use the ideal gas equation of state, $p = \rho(\gamma-1)e$, where γ is the adiabatic index of the gas. The adiabatic sound speed, c_s, can then be determined from the pressure and the density, $c_s^2 = \gamma p/\rho$.

If the conserved variables are written as a vector, $\boldsymbol{u} = [\rho, \rho u, \rho v, \rho w, E]^{\mathrm{T}}$, and three flux vectors are defined

$$\boldsymbol{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (E+p)u \end{bmatrix}, \quad \boldsymbol{G} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ (E+p)v \end{bmatrix}, \quad \text{and} \quad \boldsymbol{H} = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ (E+p)w \end{bmatrix}, \tag{1.4}$$

then Equations 1.1 - 1.3 can be combined in a single expression,

$$\frac{\delta\boldsymbol{u}}{\delta t} + \frac{\delta\boldsymbol{F}}{\delta x} + \frac{\delta\boldsymbol{G}}{\delta y} + \frac{\delta\boldsymbol{H}}{\delta z} = 0. \tag{1.5}$$

[2]Material in this section draws from Laney (1998), LeVeque (2002), Stone et al. (2008), and Toro (2009).

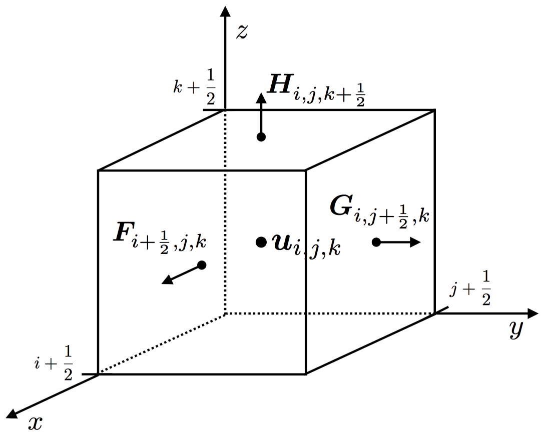
z
$k+\frac{1}{2}$
$\boldsymbol{H}_{i,j,k+\frac{1}{2}}$
$\boldsymbol{G}_{i,j+\frac{1}{2},k}$
$\boldsymbol{F}_{i+\frac{1}{2},j,k}$
$\boldsymbol{u}_{i,j,k}$
$j+\frac{1}{2}$
y
$i+\frac{1}{2}$
x

Figure 1.3: A single cell in a finite-volume simulation. Average values of the conserved variables, $\boldsymbol{u}$ are cell-centered, while fluxes are face-centered. This figure is based on Figure 1 of Stone et al. (2008).

Written in this form, the fluid equations are clearly seen to be conservation laws for mass, momentum, and energy. The change in any conserved quantity over time is exactly balanced by the flux of that variable in space (with pressure supplying an additional force term in the momentum equation and an additional work term in the energy equation). Thus, a finite-volume method can preserve the total values of the conserved quantities over time, provided that the flux of a conserved quantity exiting a cell matches the flux of that conserved quantity entering the next cell. If this requirement is satisfied, mass, momentum, and energy are conserved over the entire simulation volume.

Consider a cell centered at a location (i, j, k), as shown in Figure 1.3. The x-edges of the cell are located at $(i-\frac{1}{2}, j, k)$ and $(i+\frac{1}{2}, j, k)$, the y-edges at $(i, j-\frac{1}{2}, k)$ and $(i, j+\frac{1}{2}, k)$, and the z-edges at $(i, j, k-\frac{1}{2})$ and $(i, j, k+\frac{1}{2})$. The conserved variables at a given time t are defined as the volume integral over this cell:

$$\boldsymbol{u}^t_{(i,j,k)} = \frac{1}{\delta x \delta y \delta z} \times \int_{z-\frac{1}{2}}^{z+\frac{1}{2}} \int_{y-\frac{1}{2}}^{y+\frac{1}{2}} \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} \boldsymbol{u}(x, y, z, t) dx dy dz, \tag{1.6}$$

where δx, δy, and δz denote the length, width, and height of the cell. Similarly, the fluxes of the conserved variables can be defined as the integral over a time interval

δt and across a given cell face, e.g.

$$\boldsymbol{F}^{*}_{i+\frac{1}{2},j,k} = \frac{1}{\delta y \delta z \delta t} \times \int_{t}^{t+\delta t} \int_{z-\frac{1}{2}}^{z+\frac{1}{2}} \int_{y-\frac{1}{2}}^{y+\frac{1}{2}} \boldsymbol{F}(x_{i+\frac{1}{2}}, y, z, t')dydzdt', \tag{1.7}$$

for the flux at the $(i + \frac{1}{2}, j, k)$ interface (see Figure 1.3). Here, the $*$ superscript denotes an average over the time $t + \delta t$, not the value of the flux at a specific time. A conservative variable update for the cell from time t to time $t + \delta t$ can then be expressed,

$$\begin{aligned} \boldsymbol{u}^{t+\delta t}_{(i,j,k)} = \boldsymbol{u}^{t}_{(i,j,k)} &- \frac{\delta t}{\delta x}(\boldsymbol{F}^{*}_{i-\frac{1}{2},j,k} - \boldsymbol{F}^{*}_{i+\frac{1}{2},j,k}) \\ &- \frac{\delta t}{\delta y}(\boldsymbol{G}^{*}_{i,j-\frac{1}{2},k} - \boldsymbol{G}^{*}_{i,j+\frac{1}{2},k}) \\ &- \frac{\delta t}{\delta z}(\boldsymbol{H}^{*}_{i,j,k-\frac{1}{2}} - \boldsymbol{H}^{*}_{i,j,k+\frac{1}{2}}), \end{aligned} \tag{1.8}$$

by applying the divergence theorem to Equation 1.5. The average values of the conserved variables at the start time, t, are known. (At $t = 0$ they are the initial conditions of the simulation.) The key to a highly accurate finite-volume method, then, is a suitable approximation to the flux integrals over the cell faces. A thorough description of the methods used in *Cholla* to calculate these time-averaged fluxes is presented in Chapter 2, with additional material in Appendices A and B.

1.2 Galactic Winds

If a galaxy undergoes a period of rapid star formation (a starburst), energy input from stellar processes including stellar winds, radiation, and supernovae can result in the ejection of large quantities of gas from the galaxy. Similarly, an active galactic nucleus can inject a large quantity of energy that may also drive gas out of the galaxy. These outflows are collectively called galactic winds. In this thesis I will focus solely on the physics of winds driven by stellar feedback, and in particular, those driven by the injection of energy and momentum from supernovae during a starburst.

1.2.1 The Starburst-Driven Wind Model: Theoretical Perspective

When large numbers of supernovae occur within the interstellar medium (ISM) of a galaxy on a short timescale, their combined mass and energy deposition can create a hot region with significantly higher pressure than the surrounding ISM. If enough energy is deposited, the over-pressurized region will expand into a "superbubble" until its radius is comparable to the disk scale-height, at which point it will break out and flow freely into the surrounding halo (see Veilleux et al., 2005, and references therein). Alternatively, if the ISM is significantly porous, the hot fluid may leak out into the halo relatively quickly through under-dense regions, without needing to undergo a breakout (Cooper et al., 2008). Once the hot fluid created by the supernova ejecta has begun to stream into the halo, it becomes a wind.

Under the assumption that this mass and energy injection is approximately constant for the duration of the starburst, and the resulting wind expansion is approximately spherical, Chevalier and Clegg (1985) found an analytic solution for the properties of the wind as a function of radius. This solution is based on the fluid equations expressed with spherical symmetry:

$$\frac{1}{r^2}\frac{\mathrm{d}(r^2\rho v_r)}{\mathrm{d}r} = \frac{\dot{M}}{V}, \tag{1.9}$$

$$\rho v_r \frac{\mathrm{d}(v_r)}{\mathrm{d}r} = -\frac{\mathrm{d}p}{\mathrm{d}r} - \frac{\dot{M}}{V}v_r, \tag{1.10}$$

$$\frac{1}{r^2}\frac{\mathrm{d}}{\mathrm{d}r}\left[r^2\rho v_r\left(\frac{1}{2}v_r^2 + \frac{\gamma}{\gamma-1}\frac{p}{\rho}\right)\right] = \frac{\dot{E}}{V}, \tag{1.11}$$

where r is the radial coordinate, v_r is the radial velocity, V is the volume within the injection region $r < R_{\mathrm{starburst}}$, and all other symbols have the same meaning as in Section 1.1.3. The mass and energy injection rates, $\dot{M}$ and $\dot{E}$, appear as source terms in the equations. In this model, radiative cooling and gravity are neglected. Expressed as a function of the Mach number $M = v_r/c_s$, the solution is

$$\left(\frac{3\gamma + 1/M^2}{1+3\gamma}\right)^{-\frac{3\gamma+1}{5\gamma+1}}\left(\frac{\gamma - 1 + 2/M^2}{1+\gamma}\right)^{\frac{\gamma+1}{2(5\gamma+1)}} = \frac{r}{R_{\mathrm{starburst}}}, \quad (r < R_{\mathrm{starburst}}) \tag{1.12}$$

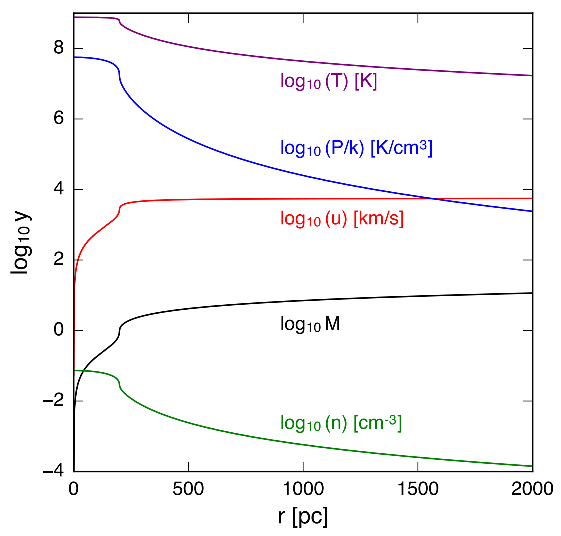

Figure 1.4: Solutions for the number density (n), radial velocity (u), pressure (P/k), temperature (T), and mach number (M) are given as a function of radius. These solutions assume $\dot{M} = 1\,\mathrm{M}_\odot/\mathrm{yr}$ and $\dot{E} = 10^{43}\,\mathrm{erg/s}$, the values given in the original Chevalier & Clegg paper solution as a model for the starburst galaxy M82. The radius of the starburst is $R = 300$ pc.

within the starburst region where mass and energy are injected, and

$$M^{\frac{2}{\gamma-1}} \left(\frac{\gamma - 1 + 2/M^2}{1+\gamma} \right)^{\frac{\gamma+1}{2(\gamma-1)}} = \left(\frac{r}{R_{\mathrm{starburst}}} \right)^2, \quad (r > R_{\mathrm{starburst}}) \tag{1.13}$$

outside. Figure 1.4 shows an example of this solution for the dimensionless Mach number as well as a number of physical variables as a function of radius. An important feature of the solution is the wind's transition to supersonic flow at $r = R_{\mathrm{starburst}}$, the edge of the mass and energy injection zone. Such supersonic outflows could potentially transport metal-rich material large distances from the galaxy.

Using stellar population synthesis models like Starburst99, the mass and energy injection rates due to supernovae (and stellar winds) can be estimated within the starburst region based on the star formation rate (SFR). For example, for a solar metallicity starburst, Leitherer et al. (1999) estimate that

$$\dot{M}_* = 0.26 \left(\frac{\mathrm{SFR}}{\mathrm{M}_\odot\,\mathrm{yr}^{-1}} \right) \mathrm{M}_\odot\,\mathrm{yr}^{-1} \tag{1.14}$$

and

$$\dot{E}_* = 7.0 \times 10^{41} \left(\frac{\mathrm{SFR}}{\mathrm{M}_\odot\,\mathrm{yr}^{-1}} \right) \mathrm{erg\,s^{-1}}. \tag{1.15}$$

Here, $\dot{M}_*$ and $\dot{E}_*$ refer to the mass and energy injection from supernovae and stellar winds. Additional uncertainty in the wind model arises from the fact that there may be mass-loading of the hot wind as it heats and incorporates cooler ISM gas, such that $\dot{M} = \beta\dot{M}_*$, where β represents the mass-loading factor[3]. Additionally, some fraction of the energy deposited by the supernovae may be radiated away by dense ISM gas, rather than being thermalized. With α representing the fraction of the injected energy that is thermalized, $\dot{E} = \alpha\dot{E}_*$. Given these caveats, the estimates above allow the Chevalier and Clegg (1985) model to be extrapolated to any galaxy for which there is an estimate of star formation rate and metallicity. Much work in recent years has gone into observationally constraining the values of α and β for winds in different galaxies, and determining whether there is a threshold SFR required to drive a wind (e.g. Heckman et al., 2015).

Within the driving region, the assumption of negligible radiative losses from the hot wind is reasonable. However, as pointed out by several authors including Wang (1995), the wind may be susceptible to significant radiative cooling at larger radii, especially if the values of β and α lead to a strongly mass-loaded wind. This can be understood as a result of the shape of the radiative cooling curve. At high temperatures, cooling of the wind is dominated by brehmsstrahlung, and gets less efficient as T decreases, such that the ratio of the cooling time t_{cool} to the advection time $t_{\mathrm{adv}} \sim r/v_r$ increases slowly with radius, $t_{\mathrm{cool}}/t_{\mathrm{adv}} \propto r^{1/3}$ (Thompson et al., 2016). As the wind expands, however, the gas cools adiabatically, and once the temperature drops below $T \leq 10^7\,\mathrm{K}$, cooling starts to become *more* efficient with decreasing T. As a result, the ratio $t_{\mathrm{cool}}/t_{\mathrm{adv}}$ begins to decrease with radius, $t_{\mathrm{cool}}/t_{\mathrm{adv}} \propto r^{-19/15}$ (Thompson et al., 2016). Thus, with the right initial conditions within the starburst region, winds will cool radiatively at large radii. This effect has recently been

[3]Note that the mass-loading factor β is also used within the galaxy evolution community to refer to the ratio of the total mass of the outflow to the star formation rate. That is *not* how β is used in this thesis.

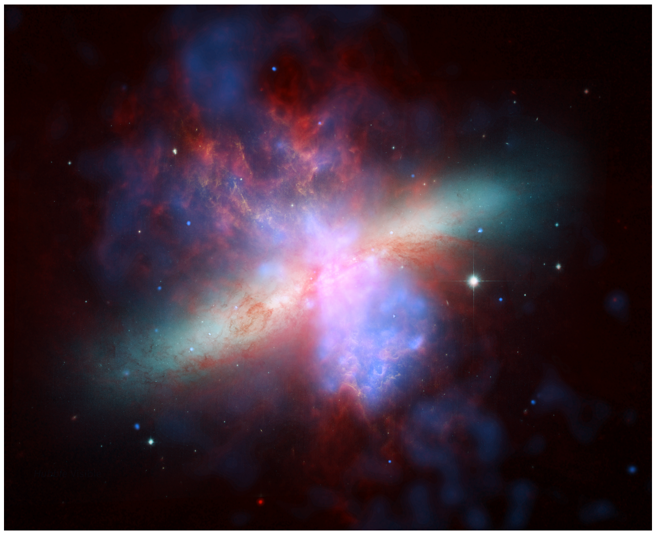

Figure 1.5: Composite image of starburst galaxy M82 in optical light from the Hubble Space Telescope (orange/green), infrared light from Spitzer (red), and x-ray light from the Chandra X-ray Observatory (blue). Image credits: X-ray: NASA/CXC/JHU/D.Strickland; Optical:NASA/ESA/STScI/AURA/The Hubble Heritage Team; IR: NASA/JPL-Caltech/UniversityofArizona/C.Engelbracht.

demonstrated in idealized numerical simulations by Scannapieco (2017).

1.2.2 Multiphase Galactic Winds: Observational Perspective

The possibility of efficient radiative cooling as the hot fluid escapes galaxies provides motivation for a study of winds that contain multiphase material. From a data-driven perspective, however, a model of starburst-driven winds that includes multiphase gas has been a necessity quite literally since their discovery. In 1963, the first observational evidence of a galactic wind in the nearby starburst galaxy M82 was demonstrated through spectacular Hα filaments (Lynds and Sandage, 1963). In the intervening years, that galaxy has been observed to have outflowing gas in every phase imaginable, from cold molecular gas (e.g. Leroy et al., 2015) to hot X-ray plasma (e.g. Strickland and Heckman, 2009). Figure 1.5 shows a composite image of this iconic starburst galaxy that highlights in red the Hα emission coming from warm ionized gas, which appears to be closely associated with the soft X-ray emission, and forms a biconical shell around the volume-filling hot X-ray gas.

In addition to observations of nearby galaxies, outflowing material has also been detected via absorption lines in high-redshift systems, both via stacking (e.g. Weiner et al., 2009), and in individually detected lensed systems (e.g. Pettini et al., 2000). The apparent ubiquity of outflows in rapidly star-forming galaxies has led some authors to posit a threshold star formation rate surface density beyond which outflow is inevitable (e.g. Heckman et al., 2015). The clear observational evidence for cool, warm, and hot gas in outflows, combined with the ubiquity of such winds in nature means that our theory of galaxy formation must somehow incorporate multiphase, starburst-driven winds. My investigation into the nature of such winds, including the interactions between different gas phases, comprises Chapter 3 of this thesis.

CHAPTER 2

CHOLLA: A NEW MASSIVELY-PARALLEL HYDRODYNAMICS CODE FOR ASTROPHYSICAL SIMULATION[†]

We present *Cholla* (Computational Hydrodynamics On ParaLLel Architectures), a new three-dimensional hydrodynamics code that harnesses the power of graphics processing units (GPUs) to accelerate astrophysical simulations. *Cholla* models the Euler equations on a static mesh using state-of-the-art techniques, including the unsplit Corner Transport Upwind (CTU) algorithm, a variety of exact and approximate Riemann solvers, and multiple spatial reconstruction techniques including the piecewise parabolic method (PPM). Using GPUs, *Cholla* evolves the fluid properties of thousands of cells simultaneously and can update over ten million cells per GPU-second while using an exact Riemann solver and PPM reconstruction. Owing to the massively-parallel architecture of GPUs and the design of the *Cholla* code, astrophysical simulations with physically interesting grid resolutions ($\gtrsim 256^3$) can easily be computed on a single device. We use the Message Passing Interface library to extend calculations onto multiple devices and demonstrate nearly ideal scaling beyond 64 GPUs. A suite of test problems highlights the physical accuracy of our modeling and provides a useful comparison to other codes. We then use *Cholla* to simulate the interaction of a shock wave with a gas cloud in the interstellar medium, showing that the evolution of the cloud is highly dependent on its density structure. We reconcile the computed mixing time of a turbulent cloud with a realistic density distribution destroyed by a strong shock with the existing analytic theory for spherical cloud destruction by describing the system in terms of its median gas density.

[†]This chapter has been published previously as Schneider & Robertson, 2015.

2.1 Introduction

Over the past fifty years, the field of computational hydrodynamics has grown to incorporate a wide array of numerical schemes that attempt to model a large range of astrophysical phenomena. From the pioneering work of Godunov (1959) and Courant et al. (1967), the sophistication of hydrodynamics solvers has steadily improved. Many astrophysical simulation codes now use high order reconstruction methods, implement very accurate or exact Riemann solvers, and model additional physics including gravity, cooling, magnetohydrodynamics, radiative transfer, and more (e.g. Kravtsov, 1999; Knebe et al., 2001; Fryxell et al., 2000; Teyssier, 2002; Hayes et al., 2006; Stone et al., 2008; Bryan et al., 2014). While these advanced techniques result in simulations of unprecedented physical accuracy, they can also be extremely computationally expensive. Given the detailed and expensive physical processes currently being modeled, new numerical approaches to modeling Eulerian hydrodynamics should be considered. This work presents a new, massively-parallel hydrodynamics code *Cholla* (Computational Hydrodynamics On ParaLLel Architectures) that leverages Graphics Processing Units (GPUs) to accelerate astrophysical simulations.

Historically, our ability to numerically model larger and more complex systems has benefitted from improvements in technology, especially increased storage and faster clock speeds for central processing units (CPUs). Algorithmic improvements such as adaptive mesh refinement (e.g., Berger and Oliger, 1984; Berger and Colella, 1989) have had a major impact on the ability of codes to achieve higher resolution, but much of the basic structure of static mesh grid codes has remained. In the last decade, computer speed has improved significantly as a result of increased parallelization, and the fastest supercomputers[1] now rely on hardware accelerators like GPUs or Intel Xeon Phi coprocessors to provide the bulk of their computational power. To leverage the full capabilities of these systems, multi-core CPU chips and accelerators must be used simultaneously in the context of a single hydrodynamics

[1]`http://www.top500.org`

code. While similar parallelization and vectorization techniques apply to a variety of hardware accelerators, *Cholla* utilizes GPUs to perform all its hydrodynamical calculations. Engineering *Cholla* to run natively on GPUs allows us to take advantage of the inherently parallel structure of grid-based Eulerian hydrodynamics schemes, and enables the substantial computational performance gain demonstrated in this paper.

Accelerators and other special purpose hardware have been used in astrophysical simulations for many years (e.g., Sugimoto et al., 1990; Aarseth, 1999; Spurzem, 1999; Portegies Zwart et al., 2004; Harfst et al., 2007; Portegies Zwart and Bédorf, 2014). Early work adapting Eulerian hydrodynamics solvers to the GPU indicated a promising avenue to accelerate simulations, with developers reporting speedups of 50× or more as compared to CPU-only implementations (e.g., Brandvik and Pullan, 2007; Pang et al., 2010; Bard and Dorelli, 2010; Kestener et al., 2010). These preliminary efforts clearly illustrated the substantial performance gains that could be achieved using a single GPU. More recently, a number of multi-device GPU-based hydrodynamical simulation methods have been presented, including AMR techniques (Schive et al., 2010; Wang et al., 2010), two dimensional Galerkin approaches (Chan et al., 2012), Smoothed Particle Hydrodynamics (SPH) codes (Sandalski, 2012; Domínguez et al., 2013), and hybrid schemes (Kulikov, 2014).

While these codes represent substantial advancement, the field of massively-parallel astrophysical hydrodynamics is still relatively new. All of the aforementioned methods have been restricted to second-order spatial reconstruction, and many would require considerable modification to run on a cluster. In contrast, the hydrodynamics solver implemented in *Cholla* is among the most complex and physically accurate of those that have been adapted to GPU hardware. Successfully implementing such a complex solver in a hybrid environment on cluster scales displays our ability to merge the state-of-the-art in CPU hydrodynamics with a new generation of computer hardware.

As is evidenced by the number of CPU codes presented in the literature, room exists for many different approaches optimized for different purposes. With *Cholla*, we have built a fast, GPU-accelerated static mesh hydrodynamics module that can be used efficiently on its own or in conjunction with a variety of additional physics. Beyond accelerating the hydrodynamics calculation, offloading the work onto the GPU frees the CPU to perform other tasks. This excess computational capacity makes *Cholla* an excellent bedrock for developing complex physical models that require hydrodynamics.

The large dynamic range of spatial scales in many astrophysical problems requires simulations with both high resolution and a high level of physical accuracy. The interaction of a high mach number shock with a gas cloud falls into this category (Klein et al., 1994a). Using the power of *Cholla*, we can efficiently run high resolution simulations of the cloud-shock problem to investigate how cloud density structure affects the destruction of high-density gas. Given the inhomogeneous nature of gas in galaxies, our results have wide-ranging implications, from the impact of supernovae on the gas in their immediate environment to the evolution of dense gas in galactic outflows.

In the following sections, we fully describe *Cholla*. The code models solutions to the equations of hydrodynamics using the Corner Transport Upwind (CTU) algorithm (Colella, 1990; Gardiner and Stone, 2008a), and includes multiple choices for both interface reconstruction and Riemann solvers. The CTU algorithm is presented in Section 2.2 along with brief descriptions of the reconstruction methods and Riemann solvers, which are fully documented in the Appendices. The code structure, including the simulation setup, CUDA functions, optimization strategies necessary to take advantage of the GPU architecture, and Message Passing Interface (MPI; Forum, 1994) implementation and scalability, is described in Section 2.3. We then demonstrate the excellent performance of *Cholla* on a suite of canonical hydrodynamics tests in Section 2.4. In Section 2.5, we derive new results describing the interaction of a high mach number shock with a turbulent gas cloud. We conclude in Section 2.6.

2.2 Hydrodynamics

Hydrodynamics is relevant to many astrophysical processes and represents one of the most computationally demanding parts of numerical simulations. Creating a fast hydrodynamics solver is therefore an important step in increasing the resolution and speed with which astrophysical calculations can be performed. In this section, we present the equations modeled by *Cholla*, and then describe the numerical algorithms used to model them. *Cholla* includes a variety of reconstruction techniques and Riemann solvers, each of which is described below.

In differential conservation law form (see, e.g., Toro, 2009), the multi-dimensional Euler equations can be written:

$$\frac{\delta \rho}{\delta t} + \frac{\delta(\rho u)}{\delta x} + \frac{\delta(\rho v)}{\delta y} + \frac{\delta(\rho w)}{\delta z} = 0, \tag{2.1}$$

$$\frac{\delta(\rho u)}{\delta t} + \frac{\delta(\rho u^2 + p)}{\delta x} + \frac{\delta(\rho uv)}{\delta y} + \frac{\delta(\rho uw)}{\delta z} = 0, \tag{2.2}$$

$$\frac{\delta(\rho v)}{\delta t} + \frac{\delta(\rho uv)}{\delta x} + \frac{\delta(\rho v^2 + p)}{\delta y} + \frac{\delta(\rho vw)}{\delta z} = 0, \tag{2.3}$$

$$\frac{\delta(\rho w)}{\delta t} + \frac{\delta(\rho uw)}{\delta x} + \frac{\delta(\rho vw)}{\delta y} + \frac{\delta(\rho w^2 + p)}{\delta z} = 0, \tag{2.4}$$

$$\frac{\delta E}{\delta t} + \frac{\delta[u(E + p)]}{\delta x} + \frac{\delta[v(E + p)]}{\delta y} + \frac{\delta[w(E + p)]}{\delta z} = 0. \tag{2.5}$$

Here ρ is the mass density, u, v, and w are the x-, y-, and z-components of velocity, p is the pressure, and E is the total energy per unit volume,

$$E = \rho \left(\frac{1}{2}\mathbf{V}^2 + e \right), \tag{2.6}$$

where $\mathbf{V} = [u, v, w]^T$ is the three-component velocity vector. The total energy includes the specific internal energy, e, and the specific kinetic energy,

$$\frac{1}{2}\mathbf{V}^2 = \frac{1}{2}\mathbf{V} \cdot \mathbf{V} = \frac{1}{2}\left(u^2 + v^2 + w^2\right). \tag{2.7}$$

Equation 2.1 describes the conservation of mass, Equations 2.2-2.4 the conservation of momentum, and Equation 2.5 the conservation of energy. To model solutions to this system of conservation laws, an equation of state is also necessary. We use the equation of state for an ideal gas,

$$p = (\gamma - 1)\rho e, \tag{2.8}$$

where γ is the ratio of specific heats. Incorporating a real gas equation-of-state model (Colella and Glaz, 1985) would not be incompatible with the structure of *Cholla*, though it is beyond the scope of our current work.

The Euler equations can also be written in vector notation. We define the vector of conserved quantities with components in three Cartesian dimensions,

$$\boldsymbol{u} = [\rho, \rho u, \rho v, \rho w, E]^{\mathrm{T}} \tag{2.9}$$

including density, the three components of momentum, and total energy. We will also refer to the vector of primitive variables,

$$\boldsymbol{w} = [\rho, u, v, w, p]^{\mathrm{T}} \tag{2.10}$$

that includes density, the three components of velocity, and pressure. We define three flux vectors

$$\boldsymbol{f} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u w \\ (E+p)u \end{bmatrix}, \quad \boldsymbol{g} = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho v w \\ (E+p)v \end{bmatrix}, \quad \text{and} \quad \boldsymbol{h} = \begin{bmatrix} \rho w \\ \rho u w \\ \rho v w \\ \rho w^2 + p \\ (E+p)w \end{bmatrix}. \tag{2.11}$$

Using these definitions, we can compactly write the three dimensional Euler equations in the conservative form:

$$\frac{\delta \boldsymbol{u}}{\delta t} + \frac{\delta \boldsymbol{f}}{\delta x} + \frac{\delta \boldsymbol{g}}{\delta y} + \frac{\delta \boldsymbol{h}}{\delta z} = 0. \tag{2.12}$$

The Euler equations can also be written using the primitive variables. In one dimension, the equations can be written as a set of linear hyperbolic equations of the form

$$\frac{\delta \boldsymbol{w}}{\delta t} + \mathbf{A}(\boldsymbol{w})\frac{\delta \boldsymbol{w}}{\delta x} = 0. \tag{2.13}$$

The matrix $\mathbf{A}(\boldsymbol{w})$ is diagonalizable and can be written

$$\mathbf{A} = \mathbf{R}\mathbf{\Lambda}\mathbf{L}, \tag{2.14}$$

where $\mathbf{R}$ is a matrix of right eigenvectors, $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues, and $\mathbf{L} = \mathbf{R}^{-1}$ is a matrix of left eigenvectors. The eigenvalues of $\mathbf{A}$ are real and correspond to the speeds at which information propagates for the fluid equations. If we further define the characteristic variables, $\boldsymbol{\xi}$, according to

$$d\boldsymbol{\xi} = \mathbf{L}d\boldsymbol{w}, \tag{2.15}$$

we can write the system a third way:

$$\frac{\delta\boldsymbol{\xi}}{\delta t} + \mathbf{\Lambda}\frac{\delta\boldsymbol{\xi}}{\delta x} = 0. \tag{2.16}$$

This description is called the characteristic form of the Euler equations. The characteristic variables are sometimes called wave strengths, because they describe the magnitude of the jump in the primitive variables across an associated wave. For an extensive treatment of the Euler equations and related subjects see, e.g., Laney (1998), LeVeque (2002), and Toro (2009).

2.2.1 The CTU Algorithm

Cholla models the Euler equations using a three-dimensional implementation of the Corner Transport Upwind (CTU) algorithm optimized for magnetohydrodynamics (MHD; Colella, 1990; Saltzman, 1994; Gardiner and Stone, 2008a). The six-solve version of CTU used in *Cholla* is fully adapted for MHD and documented in both Gardiner and Stone (2008a) and Stone et al. (2008). We will describe here an abbreviated version including only the steps relevant for hydrodynamics calculations.

The CTU algorithm is a Godunov-based method. A Godunov scheme uses a finite-volume approximation to model the Euler equations, evolving average values of the conserved quantities $\boldsymbol{u}$ in each cell using fluxes calculated at cell interfaces (Godunov, 1959). In three dimensions, the calculation to update the conserved

Table 2.1. Notation used in Chapter 2.

Symbol	Definition
$\boldsymbol{u}^n$	Conserved variable averages at time n
$\boldsymbol{U}_L$, $\boldsymbol{U}_R$	Reconstructed boundary values at the left and right of an interface
$\boldsymbol{U}_L^*$, $\boldsymbol{U}_R^*$	Initial time-evolved boundary values
$\boldsymbol{F}^*$, $\boldsymbol{G}^*$, $\boldsymbol{H}^*$	Initial one-dimensional fluxes
$\boldsymbol{U}_L^{n+1/2}$, $\boldsymbol{U}_R^{n+1/2}$	Transverse-flux-evolved boundary values
$\boldsymbol{F}^{n+1/2}$, $\boldsymbol{G}^{n+1/2}$, $\boldsymbol{H}^{n+1/2}$	CTU fluxes
$\boldsymbol{u}^{n+1}$	Updated conserved variable averages at time $n+1$

quantities $\boldsymbol{u}^n$ at timestep n can be written as

$$\begin{aligned}\boldsymbol{u}^{n+1}_{(i,j,k)} = \boldsymbol{u}^{n}_{(i,j,k)} &+ \frac{\Delta t}{\Delta x}[\boldsymbol{F}^{n+\frac{1}{2}}_{(i-\frac{1}{2},j,k)} - \boldsymbol{F}^{n+\frac{1}{2}}_{(i+\frac{1}{2},j,k)}] \\ &+ \frac{\Delta t}{\Delta y}[\boldsymbol{G}^{n+\frac{1}{2}}_{(i,j-\frac{1}{2},k)} - \boldsymbol{G}^{n+\frac{1}{2}}_{(i,j+\frac{1}{2},k)}] \\ &+ \frac{\Delta t}{\Delta z}[\boldsymbol{H}^{n+\frac{1}{2}}_{(i,j,k-\frac{1}{2})} - \boldsymbol{H}^{n+\frac{1}{2}}_{(i,j,k+\frac{1}{2})}].\end{aligned} \tag{2.17}$$

Here the superscript $n+1$ refers to the next time step, and $\boldsymbol{u}^{n+1}_{i,j,k}$ are the updated values of the conserved variables. The subscript (i,j,k) refers to the three-dimensional Cartesian index of the cell. Indices that are displaced by half an integer refer to interfaces. For example, $(i-\frac{1}{2},j,k)$ is the interface between cell $(i-1,j,k)$ and cell (i,j,k). The simulation time step is Δt, and Δx, Δy, and Δz refer to the cell widths in each dimension. We use lowercase versions of $\boldsymbol{u}$ and $\boldsymbol{w}$ when referring to a cell-averaged quantity, and uppercase versions when referring to an estimated value at a cell edge (e.g., $\boldsymbol{U}$ and $\boldsymbol{W}$). The fluxes in Equation 2.17 are averages in both space and time. Table 2.1 summarizes our notation.

When applied to every cell in the grid, Equation 2.17 conserves each of the quantities in $\boldsymbol{u}$. However, the physical accuracy of a method based on Equation 2.17 depends strongly on how the flux averages are calculated. Many hydrodynamics codes calculate the fluxes using only one-dimensional information as outlined in the method known as Strang (1968) splitting. While an elegant technique, Strang splitting may lead to asymmetries in hydrodynamics calculations and is not advantageous for some formulations of MHD (see, e.g., the discussion in Balsara, 2004). Therefore, we employ instead the unsplit CTU algorithm to improve on the one-dimensional calculation of fluxes by taking into account transverse fluxes that can cross cell interfaces in multi-dimensional simulations.

Before beginning a simulation the computational domain and boundaries must be initialized, as described in Section 2.3. Once the fluid properties on the grid have been initialized, the first simulation time step, Δt, is calculated using the equation

$$\Delta t = C_0 \frac{\Delta x}{|u| + a}, \tag{2.18}$$

where C_0 is the Courant-Friedrichs-Lewy (CFL) number and a is the average sound speed in the cell. In adiabatic hydrodynamics, the sound speed is a function of pressure and density

$$a = \sqrt{\gamma p / \rho}, \tag{2.19}$$

and can be calculated for each cell using the average values of the primitive variables. Thus, $|u| + a$ is the maximum wave speed in a cell with respect to the grid.

The minimum value of Δt across the entire grid is determined and used in the CTU calculation, constraining the time step for every cell to be equal. In two or three dimensions, Equation 2.18 is modified such that the minimum required timestep is computed for each direction in the cell. In three dimensions, this minimization is computed as

$$\Delta t = C_0 \mathrm{min} \left(\frac{\Delta x}{|u| + a}, \frac{\Delta y}{|v| + a}, \frac{\Delta z}{|w| + a} \right). \tag{2.20}$$

With a suitable choice of C_0, Equation 2.20 ensures that the Courant condition is satisfied in all three dimensions. Note that for the six-solve CTU algorithm the CFL number must be below 0.5 for the solution to remain stable (Gardiner and Stone, 2008a).

Once we have calculated the timestep Δt we carry out the following procedure:

1. Reconstruct values of the conserved variables on both sides of every interface using the average value of the conserved quantities in adjacent cells. These re-

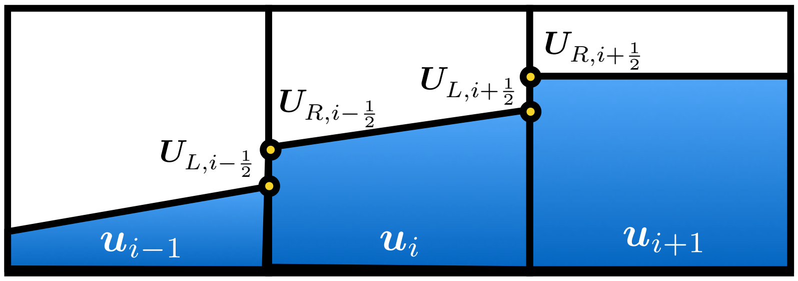
$\boldsymbol{U}_{R,i+\frac{1}{2}}$
$\boldsymbol{U}_{L,i+\frac{1}{2}}$
$\boldsymbol{U}_{R,i-\frac{1}{2}}$
$\boldsymbol{U}_{L,i-\frac{1}{2}}$
$\boldsymbol{u}_{i-1}$
$\boldsymbol{u}_{i}$
$\boldsymbol{u}_{i+1}$

Figure 2.1: Average values of the conserved variables, $\boldsymbol{u}$, are used to reconstruct boundary values on either side of each interface, $\boldsymbol{U}_L$ and $\boldsymbol{U}_R$, shown with circles. Shown is an example of piecewise linear reconstruction. The reconstructed boundary values are evolved in time to produce the initial time-evolved boundary values $\boldsymbol{U}_L^*$ and $\boldsymbol{U}_R^*$ (not shown). These $\boldsymbol{U}_L^*$ and $\boldsymbol{U}_R^*$ values are used as inputs to a Riemann problem whose solution is used to compute fluxes across cell interfaces.

constructed boundary values, denoted $\boldsymbol{U}_L$ and $\boldsymbol{U}_R$, represent an approximation to the true value of each conserved variable at the interface. For a multi-dimensional simulation, the reconstruction must be carried out in each dimension separately. We use additional subscripts to indicate which interface values we are calculating. For example, the left-hand reconstructed boundary value between cell (i, j, k) and cell $(i+1, j, k)$ is denoted $\boldsymbol{U}_{L,(i+\frac{1}{2},j,k)}$, while the right-hand value at that interface is $\boldsymbol{U}_{R,(i+\frac{1}{2},j,k)}$. *Cholla* includes several different options for the interface reconstruction, which we describe in Section 2.2.2. In order for the CTU algorithm to be second-order accurate in time, the reconstructed boundary values must be evolved half a time step before using them to calculate fluxes. The initial time evolution is considered part of the reconstruction, and is also inherently one-dimensional. We label the initial time-evolved boundary values $\boldsymbol{U}_L^*$ and $\boldsymbol{U}_R^*$. An example of a piecewise-linear reconstruction in the x-direction is shown in Figure 2.1.

2. Using the initial time-evolved boundary values as inputs, solve a Riemann problem at each cell interface in each direction. The solution to the Riemann problems yields a set of one-dimensional fluxes, $\boldsymbol{F}^*$, $\boldsymbol{G}^*$, and $\boldsymbol{H}^*$, corresponding to the x-,

y-, and z-interfaces, respectively. Like the boundary value arrays, the flux arrays contain five conserved value fluxes for each direction and interface. The Riemann solvers implemented in *Cholla* are described in Section 2.2.3.

3. Evolve the initial one-dimensional time-evolved boundary values half a time step using the transverse fluxes. For example, at the interface between cell (i, j, k) and cell $(i + 1, j, k)$ the transverse-flux-evolved boundary values are

$$\begin{aligned} \boldsymbol{U}^{n+\frac{1}{2}}_{L,(i+\frac{1}{2},j,k)} &= \boldsymbol{U}^{*}_{L,(i+\frac{1}{2},j,k)} \\ &+ \frac{1}{2}\frac{\Delta t}{\Delta y}[\boldsymbol{G}^{*}_{(i,j-\frac{1}{2},k)} - \boldsymbol{G}^{*}_{(i,j+\frac{1}{2},k)}] \\ &+ \frac{1}{2}\frac{\Delta t}{\Delta z}[\boldsymbol{H}^{*}_{(i,j,k-\frac{1}{2})} - \boldsymbol{H}^{*}_{(i,j,k+\frac{1}{2})}], \\ \boldsymbol{U}^{n+\frac{1}{2}}_{R,(i+\frac{1}{2},j,k)} &= \boldsymbol{U}^{*}_{R,(i+\frac{1}{2},j,k)} \\ &+ \frac{1}{2}\frac{\Delta t}{\Delta y}[\boldsymbol{G}^{*}_{(i+1,j-\frac{1}{2},k)} - \boldsymbol{G}^{*}_{(i+1,j+\frac{1}{2},k)}] \\ &+ \frac{1}{2}\frac{\Delta t}{\Delta z}[\boldsymbol{H}^{*}_{(i+1,j,k-\frac{1}{2})} - \boldsymbol{H}^{*}_{(i+1,j,k+\frac{1}{2})}]. \end{aligned} \tag{2.21}$$

For y and z interfaces, a cyclic permutation is applied. Thus, the x interface states are evolved using the y and z fluxes, the y interface states are evolved using the x and z fluxes, and the z interface states are evolved using the x and y fluxes. This step is only relevant for multidimensional problems. In one dimension, the CTU algorithm reduces to just the initial reconstruction step, a flux calculation, and a final conserved quantity update.

4. Use the transverse-flux-evolved boundary values, $\boldsymbol{U}_L^{n+\frac{1}{2}}$ and $\boldsymbol{U}_R^{n+\frac{1}{2}}$, as inputs for a new set of Riemann problems. The solution to these Riemann problems yields the CTU fluxes $\boldsymbol{F}^{n+\frac{1}{2}}$, $\boldsymbol{G}^{n+\frac{1}{2}}$, and $\boldsymbol{H}^{n+\frac{1}{2}}$. Each flux is calculated using a one-dimensional Riemann problem at a single interface, but includes contributions from adjacent perpendicular interfaces as a result of the evolution in Equation 2.21. The CTU fluxes are second-order accurate in time (Colella, 1990).

5. Use the CTU fluxes to update the conserved quantities in each cell as in Equa-

tion 2.17,

$$\begin{aligned} \boldsymbol{u}^{n+1}_{(i,j,k)} &= \boldsymbol{u}^{n}_{(i,j,k)} \\ &+ \frac{\Delta t}{\Delta x}[\boldsymbol{F}^{n+\frac{1}{2}}_{(i-\frac{1}{2},j,k)} - \boldsymbol{F}^{n+\frac{1}{2}}_{(i+\frac{1}{2},j,k)}] \\ &+ \frac{\Delta t}{\Delta y}[\boldsymbol{G}^{n+\frac{1}{2}}_{(i,j-\frac{1}{2},k)} - \boldsymbol{G}^{n+\frac{1}{2}}_{(i,j+\frac{1}{2},k)}] \\ &+ \frac{\Delta t}{\Delta z}[\boldsymbol{H}^{n+\frac{1}{2}}_{(i,j,k-\frac{1}{2})} - \boldsymbol{H}^{n+\frac{1}{2}}_{(i,j,k+\frac{1}{2})}]. \end{aligned} \tag{2.22}$$

The updated conserved quantities are used to calculate the next time step Δt^{n+1}.

6. Repeat the algorithm until the final simulation time is reached.

2.2.2 Interface Reconstruction

Cholla currently implements five methods for cell interface reconstruction. These include the piecewise constant method (PCM), two versions of the piecewise linear method (PLM), and two versions of the piecewise parabolic method (PPM). Differences between the versions of piecewise linear and piecewise parabolic reconstruction are demonstrated in the tests presented in Section 2.4. Access to multiple reconstruction options often proves useful, since lower-order methods are faster but higher-order methods are typically more accurate. Employing different versions of cell reconstruction also enables the impact of reconstruction on the evolution of a simulation to be quantified. Here, we give a brief overview of each of the reconstruction techniques implemented in *Cholla* . Detailed descriptions of the piecewise linear and piecewise parabolic options can be found in Appendix A.

Piecewise Constant Reconstruction

The simplest reconstruction technique is the piecewise constant method (Godunov, 1959; Courant et al., 1967). In PCM, the initial time-evolved boundary values $\boldsymbol{U}^*_L$ and $\boldsymbol{U}^*_R$ are set equal to the cell average quantities on either side of the interface, i.e.

$$\boldsymbol{U}^*_{R,(i-\frac{1}{2},j,k)} = \boldsymbol{u}^n_{(i,j,k)}, \tag{2.23}$$

and

$$\boldsymbol{U}^*_{L,(i+\frac{1}{2},j,k)} = \boldsymbol{u}^n_{(i,j,k)}. \tag{2.24}$$

Note that in this notation, the boundary value at the *right* of the interface is at the *left* side of the cell, and vice versa. While the piecewise constant method is generally too diffusive for practical applications, it has merit for code testing, and is useful as a comparison to higher order reconstruction techniques.

Piecewise Linear Reconstruction

The second and third reconstruction techniques implemented in *Cholla* are both forms of the piecewise linear method, a scheme that is second-order accurate in space and time (e.g. Toro, 2009). The PLMP reconstruction method detailed below primarily involves the primitive variables, while the PLMC method subsequently explicated involves projecting the primitive variables onto wave characteristics.

PLMP follows the method outlined in Chapter 13.4 of Toro (2009). First, the cell-average values of the primitive variables are used to calculate slopes of each variable across the left and right interface of each cell. We use the van Leer (1979) limiter to monotonize the slopes, thereby reducing the likelihood of spurious oscillations in a numerical solution. The limited slopes are used to calculate reconstructed values of the primitive variables at the cell interfaces, $\boldsymbol{W}_L$ and $\boldsymbol{W}_R$. To convert these reconstructed boundary values into input states for the Riemann problem, we need to evolve them by half a time step. For PLMP, we do this by converting primitive quantities back into conserved variables and calculating the associated fluxes using Equation 2.11. We use these fluxes to evolve the reconstructed boundary values half a time step, generating the initial time-evolved boundary states for the first set of Riemann problems, $\boldsymbol{U}^*_L$ and $\boldsymbol{U}^*_R$.

The second linear reconstruction technique, PLMC, is based on the method outlined in Stone et al. (2008). This reconstruction also uses a linear approximation to model the distribution of the conserved quantities in each cell, but limits the slopes of the characteristic variables (rather than the primitive quantities) and evolves the reconstructed boundary values differently. Rather than simply evolving the boundary values using the associated fluxes as in PLMP, we employ the more sophisticated approach first described and termed "characteristic tracing" in Colella and Woodward (1984). We calculate a first approximation to the time-evolved boundary values by integrating under the linear interpolation function used to calculate $\boldsymbol{W}_L$ and $\boldsymbol{W}_R$. The domain of dependence of the reconstructed boundary value integral is defined by the minimum (for the left-hand interface) or maximum (for the right-hand interface) wave speed. This integration is then corrected by including the contribution from each of the other characteristics approaching the interface. Once the corrections have been made, the calculation provides the initial time-evolved boundary values $\boldsymbol{U}_L^*$ and $\boldsymbol{U}_R^*$ that act as input states for the first set of Riemann problems. This process is more fully described in Appendix A.

Piecewise Parabolic Reconstruction

The remaining two reconstruction techniques implemented in *Cholla* are both versions of the piecewise parabolic method (PPM) originally described in Colella and Woodward (1984). We call the first method PPMP as it performs the reconstruction using primitive variables. Our PPMP implementation closely follows the FLASH code documentation (Fryxell et al., 2000). The second method, abbreviated PPMC, uses an eigenvalue decomposition to project onto the characteristic variables and is based on the Athena code documentation (Stone et al., 2008). Each PPM reconstruction method is described in detail in Appendix A.

The approach to slope limiting differs slightly between the two parabolic reconstruction techniques. PPMP calculates slopes at each interface the same way as PLMP, using van Leer (1979) limiting in the primitive variables. The slopes are limited in the characteristic variables for PPMC. In the parabolic methods, slopes are calculated using a stencil of five cells (two on either side of the cell for which we are calculating boundary values), which allows us to create a parabolic reconstruction of the primitive variables. This parabolic reconstruction makes PPM third-order accurate in space, though it remains only second-order accurate in time (Colella and Woodward, 1984).

Several differences between the two parabolic reconstruction methods warrant further discussion. PPMP identifies and steepens the slopes near contact discontinuities, which results in a method that is less diffusive for contact waves. Downsides to contact steepening include the necessity of empirically determined criteria for selecting contact discontinuities and increased oscillatory behavior in the solution near shocks. In *Cholla* the ability to turn off contact steepening is retained, allowing an explicit comparison of results obtained with and without the technique. PPMP also flattens shocks that have become too narrow to be treated accurately, which reduces the potential for severe post-shock oscillations. The more diffusive nature of PPMC renders a comparable correction unnecessary. Because the criteria for detecting a shock requires information from three cells on either side of an interface, the stencil for PPMP is larger than for PPMC. Both methods employ the characteristic tracing method of Colella and Woodward (1984) to translate from boundary extrapolated values based on the parabolic interpolation to input states for the Riemann problem, though the methods differ in detail (see Appendix A).

2.2.3 Riemann Solvers

Much effort has been devoted to finding efficient numerical algorithms to solve the Riemann problem (e.g., Toro, 2009), an initial value problem consisting of two constant states separated by a jump discontinuity. As displayed in Figure 2.2, the Riemann problem has an analytic solution that enables a numerical model for Eulerian hydrodynamics, as it allows for the calculation of the flux across an interface separating two initially discontinuous states. While Riemann solvers that calculate a numerical solution to the exact Riemann problem can be incorporated in hydrodynamics codes, the implicit nature of the Riemann solution requires an iterative step in the exact numerical solver. A large number of Riemann problems must be solved for every time step in a simulation, and the corresponding computational cost is substantial. As a result, a variety of approximate Riemann solvers have been engineered to quickly solve an approximation to the Euler equations at the expense

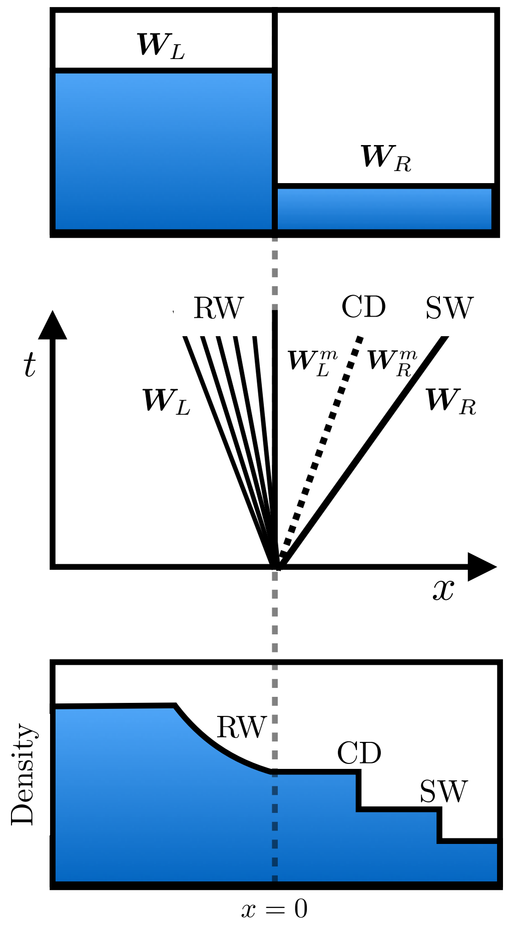

Figure 2.2: An example Riemann problem. Top: Two initial states, $\boldsymbol{W}_L$ and $\boldsymbol{W}_R$ are separated by a discontinuity at $x = 0$. Middle: The solution to this Riemann problem displays three important features, consisting of a rarefaction wave (RW) expanding to the left, a contact discontinuity (CD) moving right, and a shock wave (SW) moving right. Bottom: All three features can be seen in the solution for the density distribution.

of some physical accuracy.

Cholla computes numerical solutions to the Riemann problems on GPUs. Floating point operations are performed very efficiently on a GPU; additional factors like memory latency and data transfer contribute a larger share of the computational expense of the method. Thus, adding the extra operations needed for the iterative procedure in an exact solver versus an approximate one does not impact the performance speed of *Cholla* in the same way as for a CPU-based code. However, there are certain problems where the extra diffusion in an approximate solver is helpful, for example to deal with the well known carbuncle instability (Quirk, 1994) that affects grid-aligned shocks. For this reason, *Cholla* includes both an exact solver and the linearized solver first described by Roe (1981) that gives an exact solution to a linear approximation of the Euler equations. Detailed descriptions of our implementation of both solvers can be found in Appendix B.

The Exact Solver

Cholla implements the exact Riemann solver presented in Toro (2009). The solver uses a Newton-Raphson iteration to calculate the pressure of the gas in the intermediate state $\boldsymbol{W}^m$ of the Riemann solution that lies between the initial states on the left and right of the interface, as shown in Figure 2.2. Once the pressure in the intermediate state has been found, the exact solution for the primitive variables between the left and right initial states can be calculated explicitly at any later point in time. The pressure and velocity are used to determine the solution at the cell interface, and the values of the primitive variables at that point are used to calculate the fluxes of conserved variables at the interface according to Equation 2.11. Transverse velocities are passively advected as scalar quantities.

The Toro Riemann solver gives a numerically exact solution to the Riemann problem in one dimension, and will never return negative densities or pressures if the input states are physically self-consistent. However, the input states on either side of the cell interface are estimated quantities, and because of the extrapolation involved in the reconstruction techniques they could be physically invalid. In these situations, the solver may be presented with an initial value problem without a physically valid solution. To prevent artificial vacuum or negative pressure solutions owing to such a circumstance, a pressure floor of 10^{-20} in the adopted unit scheme is enforced in *Cholla* . In practice, when using an exact Riemann solver the pressure floor has proved necessary only when performing the Noh test described in Section 2.4.

The Roe Solver

One common alternative to calculating an exact solution to the Riemann problem is to linearize the non-linear conservations laws and solve the resulting approximate problem exactly. In one dimension, the non-linear Euler equations can be replaced with the following linearized equation

$$\frac{\delta \boldsymbol{u}}{\delta t} + \mathbf{A}(\tilde{\boldsymbol{u}})\frac{\delta \boldsymbol{u}}{\delta x} = 0, \tag{2.25}$$

where $\mathbf{A}$ is a constant Jacobian evaluated at some average state $\tilde{\boldsymbol{u}}$ that is a function of the initial states on either side of the cell interface. This method was employed by Roe (1981), and *Cholla* includes a linearized solver very similar to the original Roe solver.

The first step in the Roe solver is to calculate the average state $\tilde{\boldsymbol{u}}$. This average state, along with the eigenvalues, λ^{α}, and left and right eigenvectors of the Jacobian $\mathbf{A}$, $\mathbf{L}^{\alpha}$ and $\mathbf{R}^{\alpha}$, can be used to calculate the Roe fluxes at the interface:

$$\boldsymbol{F}_{\text{Roe}} = \frac{1}{2}\left(\boldsymbol{F_L} + \boldsymbol{F_R} + \sum_{\alpha=1}^{m} \xi^{\alpha}|\lambda^{\alpha}|\mathbf{R}^{\alpha}\right). \tag{2.26}$$

Here, $\alpha = 1, m$ are the m characteristics of the solution, and

$$\xi^{\alpha} = \mathbf{L}^{\alpha} \cdot \delta \boldsymbol{U} \tag{2.27}$$

are the characteristic variables, determined by projecting the differences in the initial left and right states, $\delta \boldsymbol{U} = \boldsymbol{U}_R - \boldsymbol{U}_L$, onto the left eigenvectors. $\boldsymbol{F}_L$ and $\boldsymbol{F}_R$ are fluxes calculated with the left and right input states using Equation 2.11. Expressions for the average state, $\tilde{\boldsymbol{u}}$, as well as the eigenvalues and eigenvectors are given in Roe (1981) and Appendix B. The matrix $\mathbf{A}$ is not actually needed in the calculation.

As pointed out by Einfeldt et al. (1991), there are certain Riemann problems that will cause any linearized solver to fail. In these cases, the linearized solution to the Riemann problem results in negative densities or pressures in the intermediate state calculated between the left and right input states. Because this intermediate state is used to calculate the fluxes returned by the solver, these unphysical solutions may lead to numerical pathologies. A failsafe is needed to deal with the case where the Roe solver produces negative pressures or densities. Following the method of Stone et al. (2008), we check the intermediate densities and pressures before returning the fluxes calculated with the Roe solver. Should any of them be negative, we revert to using the simpler HLLE Riemann solver, described below.

The HLLE Solver

The HLLE solver is a modification of the HLL solver first described by Harten et al. (1983) and later modified by Einfeldt (1988). Although the method is extremely diffusive for contact discontinuities, as demonstrated by Einfeldt et al. (1991) the HLLE solver is guaranteed to be positively conservative (that is, the density and internal energy remain positive). The HLLE solver calculates the interface flux using an average of the left and right state fluxes, together with bounding speeds comprising the largest and smallest physical signal velocities in the solution to the exact Riemann problem. If the Roe solver produces negative densities or pressures, we replace the Roe fluxes with a new numerical flux

$$\boldsymbol{F}_{\mathrm{HLLE}} = \frac{b^p \boldsymbol{F}_L - b^m \boldsymbol{F}_R}{b^p - b^m} + \frac{b^p b^m}{b^p - b^m} \delta \boldsymbol{U}. \tag{2.28}$$

The fluxes $\boldsymbol{F}_L$ and $\boldsymbol{F}_R$, and the slopes $\delta\boldsymbol{U}$ are calculated as in the Roe solver. The signal velocities b^p and b^m are calculated using the largest and smallest eigenvalues of the Roe matrix as described in Appendix B. Because the HLLE solver quickly allows contact discontinuities to diffuse, we do not use it as a standalone Riemann solver in *Cholla*.

2.3 Code Architecture

Cholla is a grid-based hydrodynamics code that takes advantage of the massively parallel computing power of GPUs. In order to harness this power, *Cholla* was designed with the operation of the GPU in mind. In this section, we describe the overall structure of *Cholla*, including optimization strategies necessary to benefit from the parallel architecture of GPUs. As is standard in GPU programming, we will use the term “host” to refer to the CPU, and “device” to refer to the GPU.

Cholla consists of a set of C/C++ routines that run on the host plus functions called *kernels* that execute on a device. The device kernels and the host functions that call them are written in CUDA C, an extension to the C language introduced by NVIDIA[2]. All of the CUDA functions are contained in a separate hydro module so that they can be compiled independently with the NVIDIA `nvcc` compiler. In addition, we have written a C/C++ version of the hydro module that performs the same calculations as all of the GPU kernels, so it is possible to run *Cholla* without using graphics cards. We use this mode for testing, but it is not recommended for performance since the structure of the code is optimized for use with GPUs.

2.3.1 Simulation Overview

Before detailing each piece of the code, we give a general overview of the steps followed by *Cholla* when a simulation is run. Given the power of a single GPU, small problems can easily be run on a single host/device pair. For large problems, *Cholla* can be run using the MPI library, and we describe our MPI implementation in Section 2.3.7. If MPI is enabled, the simulation volume will be split into subvolumes according to the number of processes. Each subvolume will then be treated as a self-contained simulation volume for the duration of each simulation time step. The main difference between an MPI and non-MPI simulation is the method for applying boundary conditions at the end of each time step; we describe that method in Section 2.3.7.

[2]`http://developer.nvidia.com`

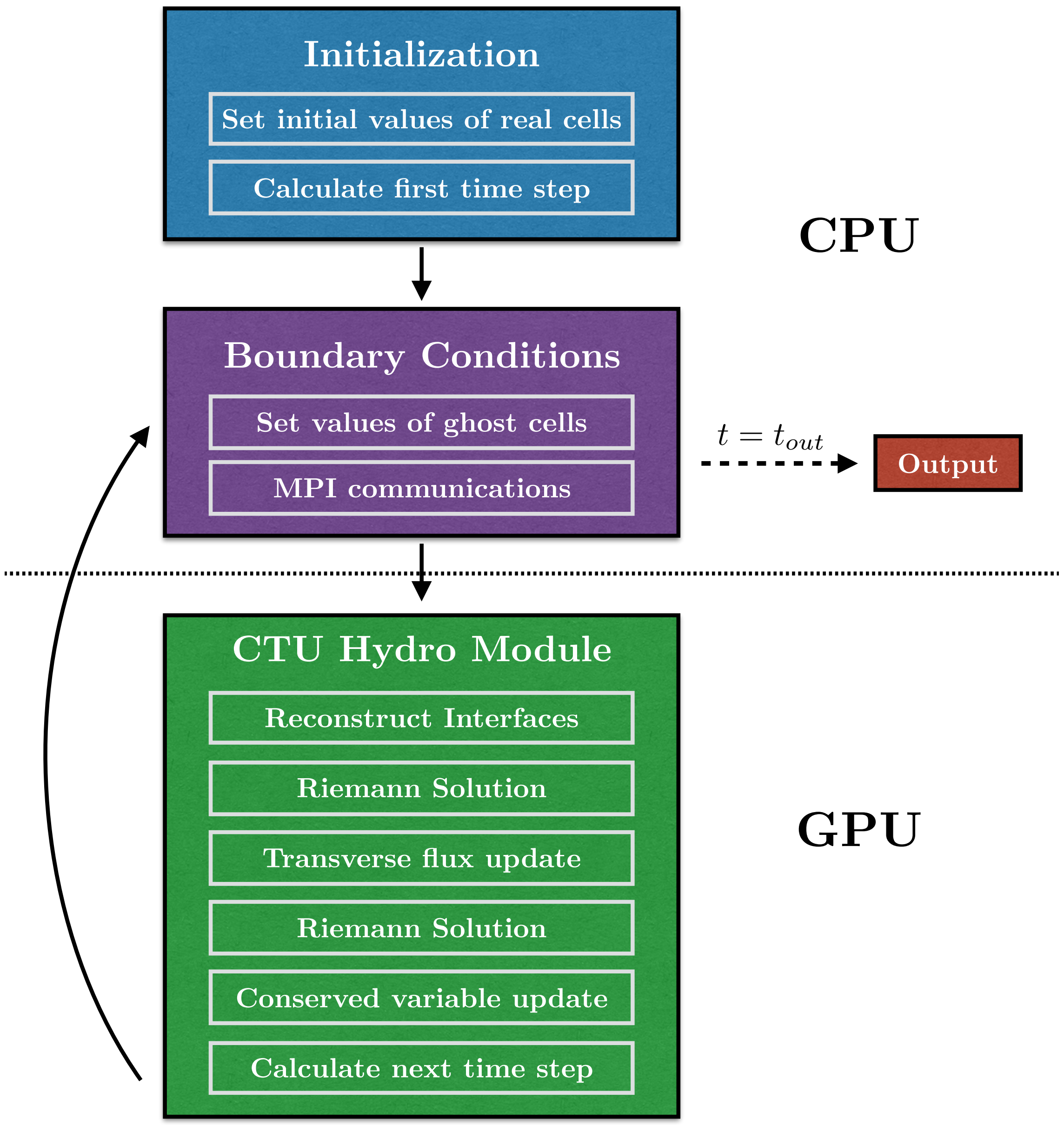

Figure 2.3: Algorithmic procedure of a *Cholla* simulation. The initialization and application of boundary conditions are done on the CPU. The conserved variable array is passed to the GPU, where the hydro calculation is done. The updated conserved variables must then be passed back to the CPU after each time step so that boundary cell information can be exchanged and the data output written (if necessary).

Portions of our algorithm that require information from potentially distant cells in the global simulation volume must be carried out on the host. The main host functions set initial conditions, apply boundary conditions, and perform any interprocess communications. Parts of the calculation that only require information from nearby cells can be carried out on the device. Because the bulk of the computational work resides in the CTU calculation that requires a stencil containing only local cells, essentially all of the hydrodynamical computations are performed on the GPU.

The steps in the *Cholla* algorithm are listed below and illustrated in Figure 2.3.

1. Initialize the simulation by setting the values of the conserved fluid quantities for all cells in the simulation volume, and calculate the first time step.

2. Transfer the array $\boldsymbol{u}$ of conserved variables to the GPU. The conserved variable array contains the values of each conserved quantity for every cell in the simulation volume.

3. Perform the CTU calculation on the GPU, including updating the conserved variable array and computing the next time step.

4. Transfer the updated conserved variable array back to the CPU.

5. Apply the boundary conditions. When running an MPI simulation, this step may require interprocess communication to exchange information for cells at the edges of subvolumes.

6. Output simulation data if desired.

The initialization of the simulation is carried out on the host. The initialization includes setting the values of the conserved variables for both the real and the ghost cells according to the conditions specified in a text input file. Ghost cells are a buffer of cells added to the boundaries of a simulation volume to calculate fluxes for real cells near the edges. The number of ghost cells reflects the size of the local stencil used to perform fluid reconstruction. Because updating the ghost cells at each time step may require information from cells that are not local in memory, the values of the ghost cells are set on the host before transferring data to the GPU.

Once the simulation volume has been initialized on the CPU, the hydrodynamical calculation begins. The host copies the conserved variable array onto the device. Because the GPU has less memory than the CPU, the conserved variable array associated with a single CPU may be too large to fit into the GPU memory at once. If so, *Cholla* uses a series of splitting routines described in Section 2.3.6 to copy smaller pieces of the simulation onto the GPU and carries out the hydrodynamics calculations on each subvolume. At the end of the hydro calculation the next time step is calculated on the device using a GPU-accelerated parallel reduction. The updated conserved variables and new time step are then transferred back to the host. The host updates the values of the ghost cells using the newly calculated values of the real cells, and Steps 2 - 5 repeat until the desired final simulation time is reached.

After each time step the values of the ghost cells are reset using the newly updated values of the conserved variables. *Cholla* includes three standard boundary conditions: periodic, reflective, and transmissive. These can be set in any combination on any of the borders of the simulation volume. For periodic and transmissive boundaries, the conserved variable values of each ghost cell are copied from the appropriate real cell. For reflective boundaries we follow the same process but reverse the sign of the perpendicular component of momentum. *Cholla* also includes the capability to define custom boundary conditions, such as the analytic boundaries specified in the Noh Strong Shock test (see Section 2.4.3). In a simulation performed using MPI communication, any necessary boundary regions are exchanged between relevant processes as described in Section 2.3.7.

2.3.2 Memory Structure

The data for a simulation in *Cholla* are contained in two structures. A header stores information about the size and shape of the domain, as well as global variables including the simulation time. A second structure contains the values of the conserved variables for each cell in the simulation. In an object oriented programming model, these values would often be stored in memory as an array of structures,

$$\begin{gathered}\mathrm{Cell}[0].\{\rho\ \rho u\ \rho v\ \rho w\ E\},\\ \vdots\\ \mathrm{Cell}[N-1].\{\rho\ \rho u\ \rho v\ \rho w\ E\},\end{gathered}$$

where N is the total number of cells in the grid. In a CPU-based simulation code, this configuration can improve the performance of memory accesses.

The object oriented model is intuitive, but the memory structure is not efficient when implemented on the GPU. On NVIDIA GPUs, calculations are performed simultaneously by thousands of individual computational elements called *cores*, analogous to but individually much less powerful than a typical CPU core. The set of instructions carried out on a single GPU core is called a *thread*. The efficiency of the GPU comes in part from its ability to efficiently schedule the execution of millions of threads requested in a single kernel call. The scheduling is organized by *streaming multiprocessors* on the device that schedule threads for execution in groups called *warps*. Each thread warp performs a given set of operations simultaneously in the execution model often referred to as Single Instruction Multiple Data (SIMD). Given that data operations across cores on the GPU are rapidly executed in a massively parallel manner via the SIMD approach, hardware timescales such as the GPU global memory access time can represent a considerable fraction of the total computational expense of a calculation. Techniques to reduce the expense of global memory accesses include the organization of data needed by each thread warp into adjacent regions in physical memory. To facilitate this advantageous data locality, *Cholla* organizes conserved variables into a structure of arrays:

$$\begin{gathered}\{\rho_0 \;...\; \rho_{N-1}\},\\ \{\rho u_0 \;...\; \rho u_{N-1}\},\\ \{\rho v_0 \;...\; \rho v_{N-1}\},\\ \{\rho w_0 \;...\; \rho w_{N-1}\},\\ \{E_0 \;...\; E_{N-1}\}.\end{gathered}$$

The thread warps can retrieve the conserved quantities within this structure of arrays in global memory with unit stride in memory accesses, reducing collisions in the access pattern.

The process of initiating a data transfer from the host to the device involves an associated computational overhead. Limiting the number of transfers required by the algorithm mitigates this overhead, as hardware latency may cause many small transfers to take longer than one large transfer. The allocation of a single structure containing the conserved variable arrays ensures a contiguous data layout in memory, and limits the required data transfers to two (single transfers to and from the device).

2.3.3 The GPU Grid

Once the simulation volume has been initialized on the host and the values of the ghost cells have been set, the array of conserved variables is transferred to global memory on the GPU. When kernels are then executed on the device, the GPU launches a *grid* of *thread blocks.* The GPU grid and thread block dimensions are set by the programmer and are application dependent. *Cholla* typically uses one or two dimensional grids of one dimensional thread blocks; the latter arrangement is illustrated in Figure 2.4. We emphasize that the dimensions of the GPU grid are not constrained to match the dimensions of the simulation, as the location of a cell in the simulation volume can always be mapped to a unique index within the GPU grid of thread blocks.

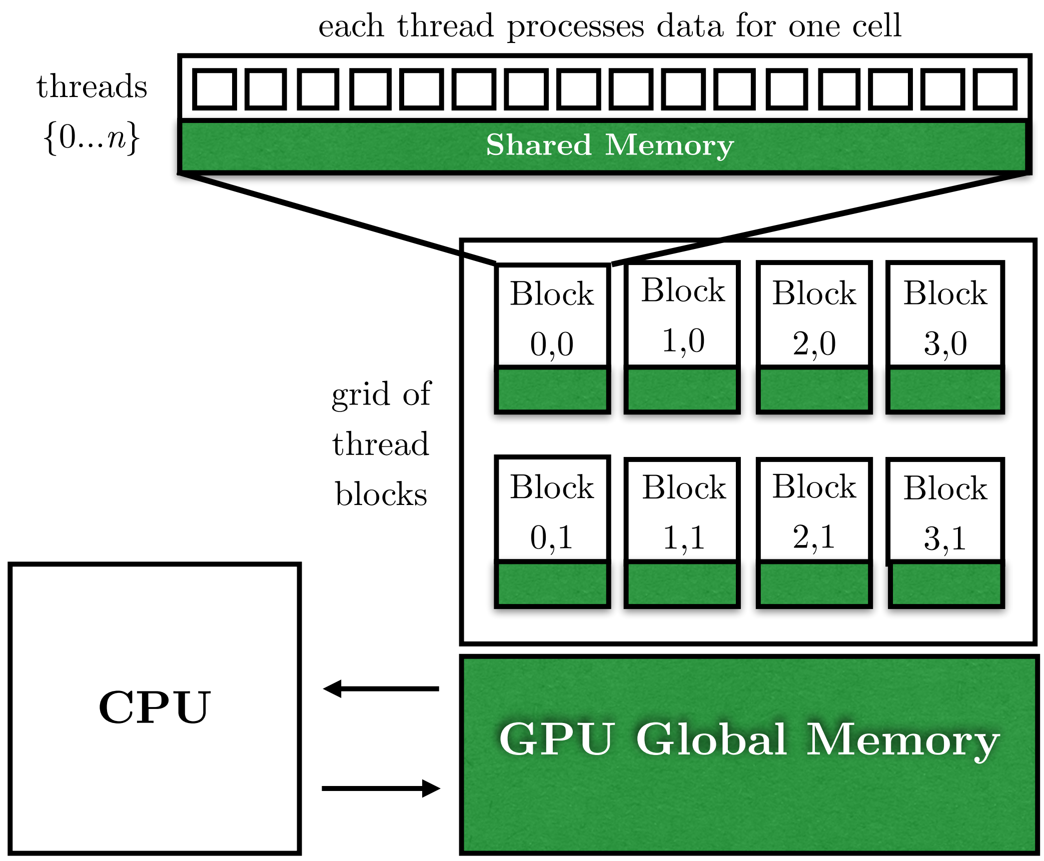

Figure 2.4: *Cholla* memory structure. After the conserved variable array is copied from the CPU into global memory on the GPU, the GPU initializes a grid of one dimensional thread blocks with set numbers of GPU threads. Each thread then calculates the information for a single grid cell. All threads can access global memory, but only threads in the same block have access to the much smaller amount of per-block shared memory.

The dimensions of the GPU grid can affect the efficiency with which the device performs calculations and dictate the mapping from a real-space cell index to a thread index. To define the thread index, the CUDA programming model includes built-in data elements that return specific values for each thread, as shown in the following pseudo-code:

```
tid = threadIdx.x + blockIdx.x * blockDim.x.
```

Here, `threadId` returns the ID of the thread within the block, `blockIdx` returns the ID of the thread block within the grid, and `blockDim` returns the dimensions of the block. By combining these pre-defined quantities, a unique global index can be calculated for each thread. This pseudo-code assumes a one dimensional grid of one dimensional blocks, but could easily be adjusted to create an equivalent mapping for two or three dimensional blocks or grids. We use the thread index to assign each thread the work of computing the conserved variable update for a single cell in the simulation. For *Cholla*, we choose a one dimensional block of threads because most of the kernels are one dimensional in nature. The PPM reconstruction, for example, requires only a one dimensional stencil and is carried out separately for the x, y, and z interfaces. Because a new GPU grid with different dimensions can be initiated every time a device function is called, the thread index calculation and subsequent mapping to a real cell index must be performed within each GPU kernel. No data needs to be transferred back to the CPU between kernels, as all information needed between kernels is stored in the GPU global memory.

2.3.4 The GPU Kernels

Cholla leverages a modular design that enables an easy selection of the reconstruction method or Riemann solver, and facilitates the incorporation of new features. Each reconstruction method and Riemann solver is performed through an associated kernel executed by the GPU. A routine implementing the CTU algorithm calls these kernels through a wrapper function that segregates CUDA calls from the rest of the code. This organization allows for a flexible compilation structure in which non-CUDA code (including MPI calls) can be compiled with standard C compilers. Within the CUDA wrapper for the CTU algorithm, the following steps are followed:

1. Allocate arrays within GPU global memory to hold the conserved variables $\boldsymbol{u}$, the initial time-evolved boundary values $\boldsymbol{U}^*$, the initial one-dimensional fluxes $\boldsymbol{F}^*$, the transverse-flux-evolved boundary values $\boldsymbol{U}^{n+\frac{1}{2}}$, and the CTU fluxes $\boldsymbol{F}^{n+\frac{1}{2}}$.
2. Transfer the conserved variable data from the host to the device and store in the newly allocated arrays in GPU global memory.
3. Call the reconstruction kernel for each dimension.
4. Call the Riemann solver kernel for each dimension.
5. Call the kernel to perform the transverse flux update (Equation 2.21).
6. Call the Riemann solver kernel for each dimension again.
7. Call the kernel to update the conserved variables and calculate the next time step (Equation 2.22).
8. Transfer the conserved variable arrays back to the CPU.

Step 1 involves the allocation of memory on the GPU, and it should be noted that the global GPU memory available is typically small compared with the CPU memory. Depending on the device, the simulation size, the number of MPI processes, and the domain decomposition, each process's conserved variable data may exceed the available GPU memory. An excess may occur even if the local grid governed by each process is small (e.g., 128^3). When necessary, *Cholla* uses a set of splitting routines to divide the simulation volume into more manageable subvolumes that are then treated according to the steps listed above. Section 2.3.6 describes these splitting routines in more detail.

The GPU kernel calls in Steps 3-7 resemble traditional C function calls, but kernels are implemented with additional variables that establish the dimensions of the grid of thread blocks launched by the GPU. For example, the syntax for calling *Cholla*'s PPM reconstruction function is:

```
PPM_reconstruction<<<BlocksPerGrid,
ThreadsPerBlock>>>(<function_parameters>),
```

where the triple chevron syntax, `<<<,>>>`, informs the CUDA-enabled compiler that this function should be executed on the GPU device. Since the amount of data processed at once by the GPU is limited by its available global memory, `BlocksPerGrid` can always be set large enough to assign a thread to each cell.

Separate kernels carry out different parts of the CTU algorithm, but each kernel shares common elements. Every kernel must begin with a calculation of the index of each thread, as described in Section 2.3.3. Using the appropriate mapping, the index of each thread of the kernel can be translated to a unique real-space cell index in the simulation volume. The threads within the kernel then retrieve necessary cell data from the GPU global memory. For the reconstruction function, these data would include the values of the conserved variables for the cell assigned to that thread, as well as those of the nearby cells within the reconstruction stencil. Once the data have been retrieved, the threads carry out any relevant calculations, and load the result into the relevant GPU global memory array. Once all of the threads have finished their calculations the kernel returns, and the process continues through each of the steps listed above.

2.3.5 Time Step Calculation

The implementation of most kernels described in the previous section (reconstruction, Riemann solver, and transverse flux update) follows closely the descriptions of the CTU calculation given in Section 2.2. However, the final conserved variable update kernel in each iteration of the algorithm is extended to include the calculation of the next simulation time step Δt^{n+1} via a parallel reduction operation performed on the GPU. Reductions on the GPU are a commonly used process, and examples of this operation can be found in e.g., the CUDA toolkit[3]. We include a brief explanation of our implementation of the parallel reduction operation here as a concrete example of the advantage of moving a given function from the CPU to the GPU.

Thread blocks on the GPU have a limited amount of "shared memory" that each thread in the block can access rapidly, as illustrated in Figure 2.4. At the end of the conserved variable computation, the updated conserved variable data for each cell are stored in the private register memory assigned to each thread. The updated values of the conserved variables are used by each thread to calculate the minimum time step associated with its cell, according to Equation 2.20. The individually calculated time steps are then loaded into an array the size of the thread block in shared memory - note that each thread block has its own array. The threads in the block then perform a tree-based parallel reduction on the time step array, finding the minimum time step for the entire block. This minimum value is uploaded into an array in the GPU global memory, and is then passed back to the CPU, where the final reduction is performed.

Calculating the time step on the GPU achieves a performance gain relative to a CPU since executing a large number of floating point operations is extremely efficient on the GPU. The shared memory reduction on the GPU reduces the number of loops needed on the CPU by a factor of `ThreadsPerBlock` (typically set to 128 for an NVIDIA Kepler K20X GPU). For reference, we find the parallel GPU reduction time step calculation for a 1920×1080 simulation can achieve a $100\times$ performance gain relative to a single CPU core depending on the architecture ($\sim$ 1ms vs. $\sim$ 100ms).

2.3.6 Subgrid Splitting

As mentioned in previous sections, the total amount of memory on a single device may be quite limited when compared to the memory available on the host. The memory footprint on the GPU for each cell in the simulation volume is of order 0.5 kilobytes, including conserved variables, interface values, and fluxes. At present, a typical GPU has only a few gigabytes of global memory, though this number has

[3]`https://developer.nvidia.com/cuda-toolkit`

been increasing with each new generation of devices. Therefore, current devices can typically only hold the information for a 3D hydrodynamical simulation of size $\sim 228^3$. In *Cholla*, slightly more cells can fit for 1D and 2D simulations owing to the reduced number of transverse interface states and fluxes that must be stored. The limited memory resources often require that the simulation volume associated with a local process may need to be successively subdivided to fit on a GPU. We term this subdivision process “subgrid splitting”. Similar methods have been used in CPU-based codes such as HERACLES (González et al., 2007).

In practice, subgrid splitting is typically only needed for multidimensional simulations or 1D simulations with millions of cells. A description of the 1D subgrid splitting is provided below as a straightforward example. First, the size of the simulation volume that will fit on the GPU at once given the global memory available is calculated and stored in a variable, e.g. `MaxVol`. The local volume governed by each local process is further split into subvolumes of size less than or equal to `MaxVol`. We refer to these subvolumes as “subgrid blocks”. The CTU calculations for each subgrid block are performed on the GPU sequentially, *including any necessary ghost cells from nearby subvolumes.* Memory buffers on the CPU are used to avoid overwriting grid cells that act as ghost cells for neighboring subgrid blocks. The procedure of copying data to the GPU, calculating, and transferring data back to the CPU is repeated until the hydro step for the entire simulation volume has completed. Because the conserved variables for the simulation are contiguous in memory on the host, copying them into buffers via `memcpy` contributes a negligible amount to the total simulation run time.

To illustrate the subgrid blocking method, Figure 2.5 displays a two-dimensional grid with an example subgrid blocking by a factor of four. Each of the four subgrid blocks (red, green, blue, purple) require real cells from adjacent subgrid blocks to act as *subgrid ghost cells* to form the full computational stencil for the fluid reconstruction and CTU calculations (indicated by colored dashed lines). Since these subgrid blocks also abut either local simulation boundaries between local processes or global boundaries at the edges of the illustrated region, they also require standard ghost cells (gray regions) to complete their computational stencils. The subgrid block regions outlined by the dashed lines are transferred sequentially to the GPU for the CTU calculation, taking care to preserve in memory the subgrid ghost cell boundary regions between subgrid blocks until the entire local volume has been processed. The standard ghost cells are updated after the CTU calculation, since they depend either on communication between MPI processes or the global boundary conditions of the simulation. Note that Figure 2.5 illustrates a very small grid for convenience, and the actual subgrid regions of a 2D simulation would be orders of magnitude larger.

We performed a variety of tests using subgrid blocks of different shapes in order to determine a method of division that helps minimize GPU communication overhead. For 2D simulations, splitting the volume into approximately square regions works well. For 3D simulations, we find that maintaining an aspect ratio that is approximately even in the y and z directions with a longer aspect ratio in the x-direction works well. In practice, we keep the aspect ratio even in y and z while maintaining an aspect ratio in x that is roughly 5 times the geometric mean of the y- and z-aspect ratios. Memory access overheads can be reduced by first copying multiple subgrid blocks into buffers on the CPU, and then transferring subarrays containing individual subgrid blocks to the GPU for computation. Even in simulations where local volumes must be subdivided into subgrid blocks dozens of times the overhead associated with copying the conserved variables into buffers on the CPU is insignificant, typically limited to 5% of the total time taken for the CTU calculation.

Transferring data to and from the GPU at each time step is time consuming, often taking $\sim 30\%$ of the entire GPU computation time for the hydro module. Therefore, strategies that reduce the fraction of the simulation volume transferred at each time step are desirable. For example, simulations that do not require subgrid splitting might achieve a performance boost by only transferring ghost cells that need to be updated via MPI. Such a strategy is beyond the scope of the current work, but is certainly worth exploring in future versions of *Cholla*.

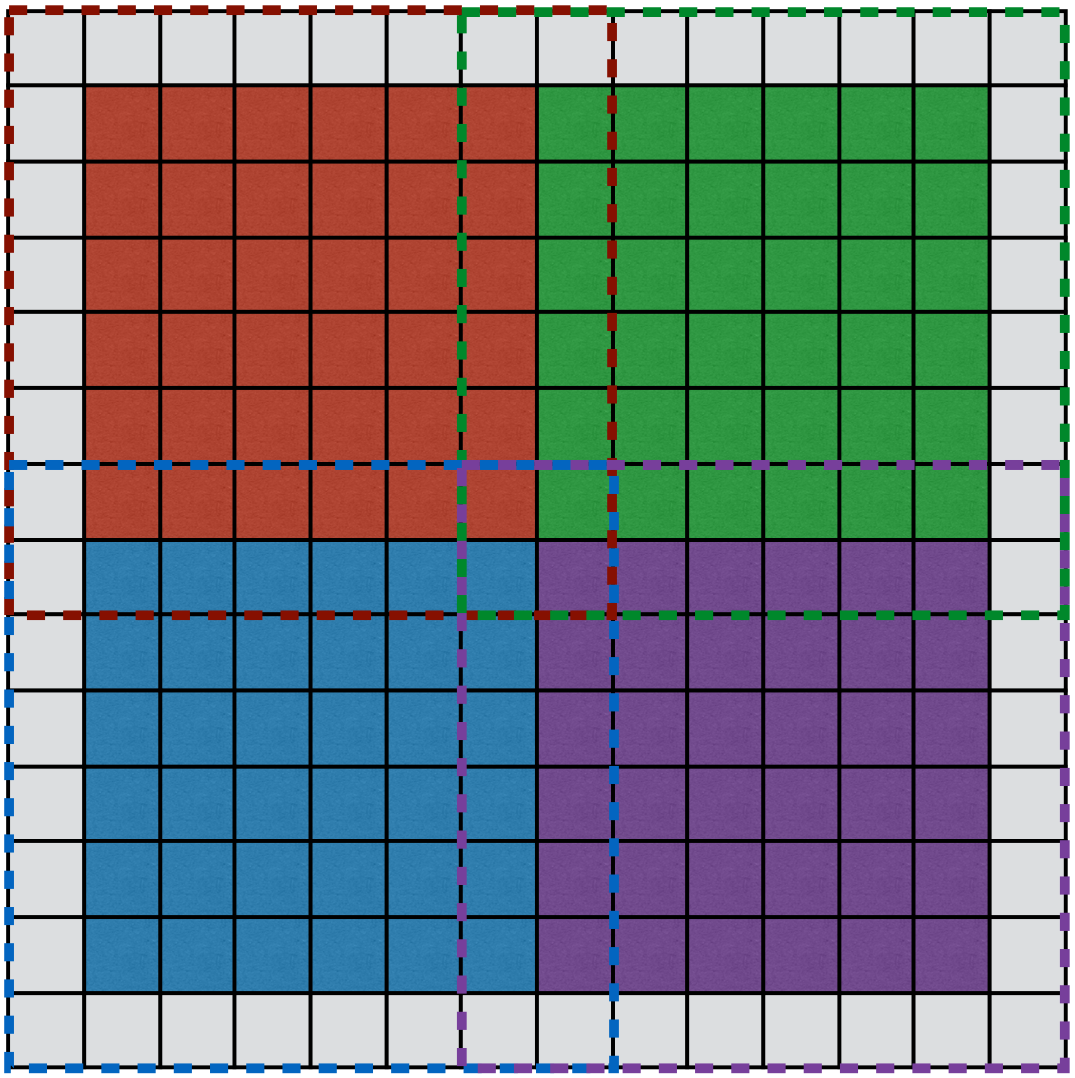

Figure 2.5: The well-known "ghost cell pattern" as applied to the subgrid blocking algorithm in *Cholla* . When the total area of a 2D simulation is too large to fit in global memory on the GPU, the simulation volume must be split into smaller subgrid blocks for GPU computation of the hydrodynamical calculation. When copying a subgrid block of the simulation onto the GPU, memory buffers are utilized such that each subgrid block can be copied to the device and the conserved variables updated without overwriting real cell data that will be needed as ghost cells for neighboring subgrid blocks. For this illustration, ghost cells on the global outer boundary of the simulation are shown in gray. The dashed lines outline the cells needed to perform the CTU calculation for each colored subgrid block. See e.g., Figure 9 of Kjolstad and Snir (2010).

2.3.7 MPI Implementation and Scaling

The massively parallel algorithm implemented by *Cholla* can be adapted to execute on multiple GPUs simultaneously. *Cholla* can thereby gain a multiplex advantage beyond the significant computation power afforded by a single GPU. The parallelization is implemented using the MPI library. The global simulation volume is decomposed into subvolumes, and the subvolumes are each assigned a single MPI process. In *Cholla*, each MPI process runs on a single CPU that has a single associated GPU, such that the number of MPI processes, CPUs, and GPUs are always equal. When the simulation volume is initialized, each process is assigned its simulation subvolume and surrounding ghost cells. Since the hydrodynamical calculation for every cell is localized to a finite stencil, only the ghost cells on the boundary of the volume may require updating from other processes via MPI communication every time step. Compared with a simulation done on a single CPU/GPU pair, additional overheads for a multi-process simulation can therefore include MPI communications needed to exchange information at boundaries and potential inefficiencies in the GPU computation introduced by the domain decomposition. While domain decomposition influences communications overheads in all MPI-parallelized codes by changing the surface area-to-volume ratio of computational subvolumes, domain decomposition additionally affects the performance of a GPU-accelerated code by changing the ratio of ghost to real cells in memory that must be transferred to the GPU. Since memory transfers from the CPU to the GPU involve considerable overhead, domain decompositions that limit the fraction of ghost cells on a local process are favorable. *Cholla* therefore allows for two different domain decompositions, described below.

Slab Decomposition

Following the domain decomposition utilized by the *Fastest Fourier Transform in the West* discrete Fourier transform library (FFTW; Frigo and Johnson, 2005), *Cholla* can use a slab-based decomposition in which the simulation volume is sliced only in one dimension. In the slab decomposition a maximum of two boundaries may be shared between processes, and because there are limited communications the slab decomposition proves efficient for simulations run with a small number of processes. With the addition of more processes the slabs grow narrower, the ratio of boundary ghost cells to real cells for each subvolume increases rapidly, and the time required to exchange boundary cells between processes remains nearly constant. Although these features cause a computational inefficiency that continues to degrade with increasing numbers of processes, *Cholla* nonetheless includes an optional slab decomposition for use with limited processes and in conjunction with FFTW.

The division of the simulation volume for *Cholla*'s slab decomposition is straightforward. When the FFTW library slab decomposition is used, the slab width on each process is optimized for accelerating discrete Fourier transform computations. Otherwise, the number of cells in the x-dimension spanning the total simulation volume is divided evenly across the number of processes, and any remaining cells are split as evenly among the processes as possible. Once the domains have been assigned, each process initializes the real cells associated with its volume and exchanges boundary cells. First, each process posts a receive request for each MPI boundary. If the process has a global simulation boundary along the x-direction, it posts either one receive request in the case of reflective, transmissive, or analytic global boundary conditions, or two in the case of global periodic boundary conditions. Processes that are surrounded by other processes will always have two MPI boundaries. The processes then send the initialized values of the real cells from their subvolume that are needed by other processes. While waiting for the cell exchange communications to complete, each process computes the cell values on its non-MPI boundaries (typically the y- and z boundaries). This asynchronous ordering of communication and boundary computation minimizes the amount of time the CPU must sit idle while waiting to receive boundary cell information. Once all the receives have completed, each process proceeds through the CTU step as though it were an independent simulation volume. At the end of each time step, boundary cells must again be exchanged along with the information from each local subvolume required to determine the global simulation time step.

Block Decomposition

In addition to being computationally inefficient, a slab decomposition limits the total number of processes that can be run as a result of the finite dimensions of the simulation volume. To improve upon both these factors, *Cholla* includes a block decomposition that seeks to minimize the aspect ratios of the simulation subvolume evolved by each process. For a block decomposition, up to six MPI communications for cell exchanges may be required per time step. Despite this increased number of communications, the reduction of the surface area-to-volume ratio of the block decomposition improves its efficiency beyond that achieved by a slab decomposition for large numbers of processes.

In the case of a block decomposition the x-, y-, and z-dimensions for each subvolume are kept as even as possible. Once a simulation volume is appropriately divided and local real cells initialized by each process, boundary cells between subvolumes must be exchanged. To keep the number of MPI communications to a maximum of six, the processes exchange boundary cells in a specific order. First, each process posts a receive request for any MPI boundaries on an x-face. While waiting for those data to arrive, the processes set x-ghost cells on any non-MPI x faces. Once the x-boundaries arrive, the processes post receive requests for MPI boundaries on y-faces, including the corner regions just transferred in the exchange along the x-direction. While waiting for the y-boundaries to arrive, each process computes the ghost cell values along non-MPI y-boundaries. By first sending the x-boundaries, processes can receive information needed for the y-boundary exchange from diagonally-adjacent processes without directly exchanging information with those processes. The same procedure is followed for the z-boundaries.

Figure 2.6 illustrates this process for a 2D simulation with periodic boundaries and a four-process decomposition. Each process first initializes its real cells, represented by the large colored squares. To perform the hydrodynamical simulation timestep in parallel across separate processes, each process must receive boundary ghost cells from the real cells hosted by surrounding processes. The boundary re-

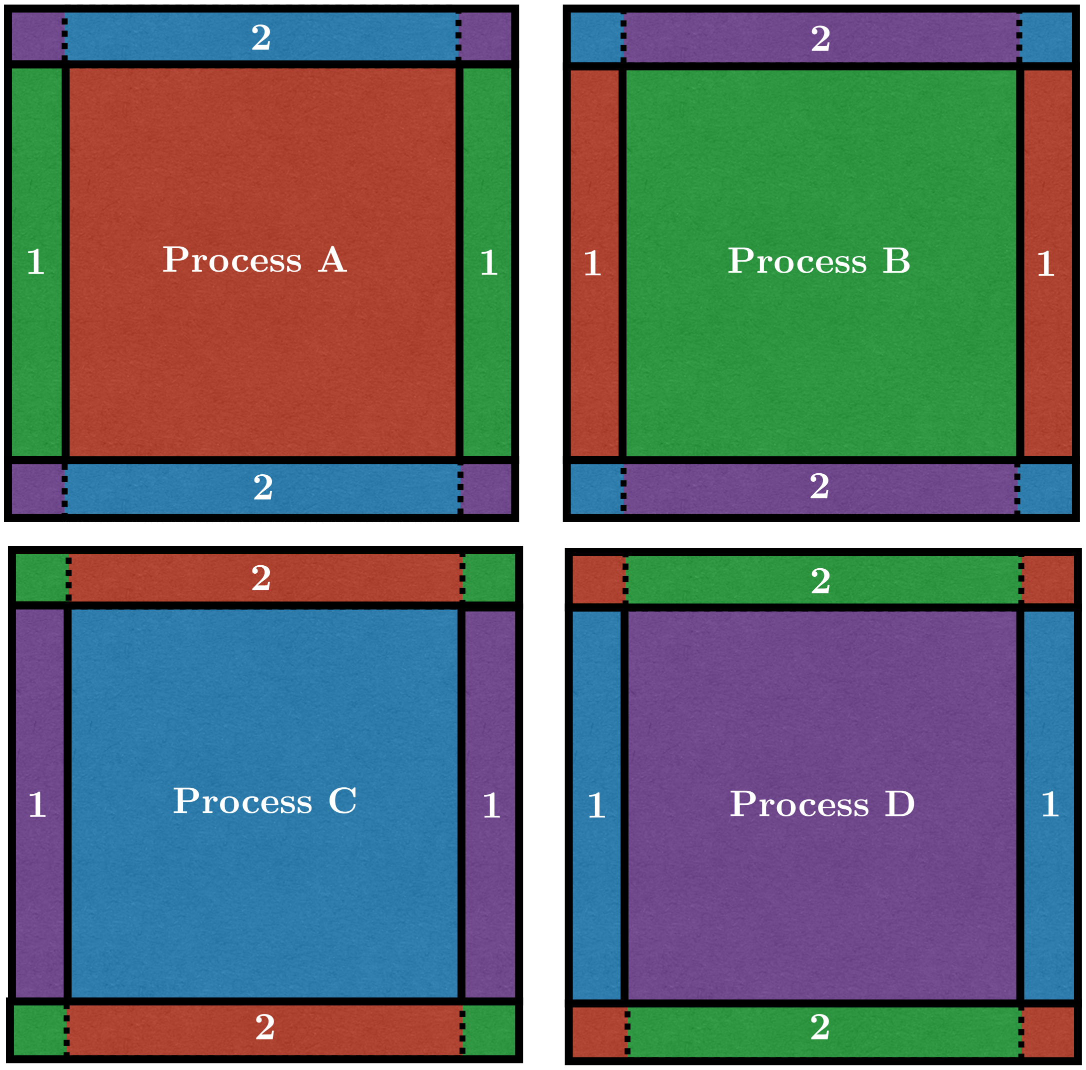

Figure 2.6: Ghost cell information is exchanged by MPI processes for the case of a 2D simulation with periodic boundaries. The real-cell domain of each of the four processes is represented by a colored square (A, B, C, and D). Each process needs information from the other three processes in order to set all of its ghost cells. By first exchanging x boundaries (represented by the outlined rectangles labeled step 1), then exchanging y boundaries (step 2), the processes are able to access the needed information without explicitly communicating with every other process. For example, on step 2, process D receives the red boundary cells from Process A that it needs to update its corner ghost cells, without ever communicating directly with process A.

gions for each process are outlined in Figure 2.6 and labelled 1 and 2. The processes first exchange x-boundary information (region 1) via two MPI communications. Once the x-boundary exchange is complete, y-boundary information (region 2) is exchanged, including the corner regions received from other processes. The same procedure is repeated for z-boundaries in a 3D simulation. Following this pattern keeps the required number of MPI communications to a maximum of six, instead of the potential twenty-six communications that would be required to separately exchange each face, edge, and corner boundary region for an interior subvolume in a 3D simulation. When using the block decomposition with a large number of GPUs, the MPI communications typically comprise only a few percent of the total computation time.

We note briefly that the block decomposition implemented in *Cholla* may also be adapted to enable the use of Fast Fourier Transform libraries that use a block decomposition, such as the Parallel FFT package written by Steve Plimpton[4].

Scaling

The scalability of the *Cholla* MPI implementation to more than one GPU warrants a discussion. To study the scaling of the code, the GPU-accelerated International Business Machines iDataplex cluster *El Gato* at the University of Arizona was used. Using *El Gato*, we have tested both the strong and weak scaling of *Cholla* using up to 64 GPUs. The results are shown in Figures 2.7 and 2.8. For both scaling tests, a three-dimensional sound wave perturbation problem with periodic boundary conditions and a block decomposition is used to maximize the number of MPI communications required per timestep. In both the strong and weak scaling tests, *Cholla* updates an average of 6.7×10^6 cells per second using the NVIDIA Kepler K20X GPUs available on *El Gato*. The 3D sound wave perturbation requires work to be done by every cell and uses third-order spatial reconstruction and an exact Riemann solver. The test is therefore relatively inefficient. By contrast, on a 2D sound wave test using second-order spatial reconstruction and a Roe solver, *Cholla* updates

[4]`http://www.sandia.gov/ sjplimp/docs/fft/README.html`

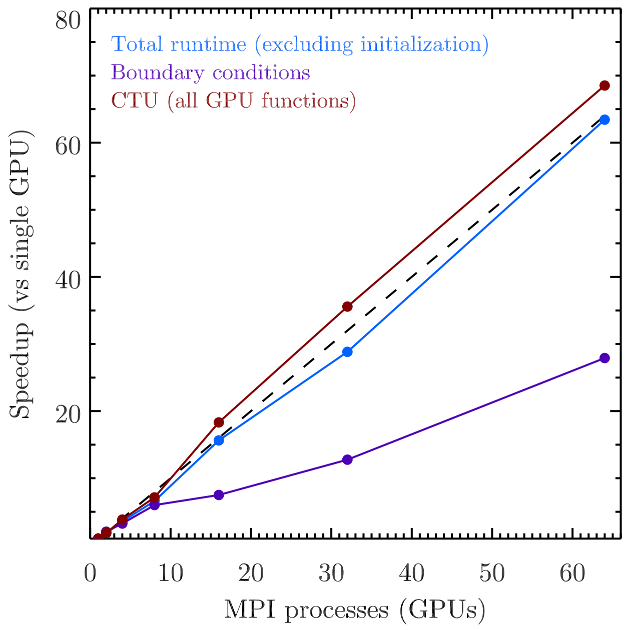

Figure 2.7: Strong scaling on a 512^3 double precision sound wave perturbation test with periodic boundaries measured relative to the calculation done on a single GPU (no MPI). Ideal strong scaling is shown by the dashed one-to-one line. The total runtime for the simulation remains close to ideal up to 64 processes, with non-ideal scaling coming primarily from the MPI communications needed to set boundary conditions. The portion of the code executing on the GPU is incorporated entirely within the CTU function, shown by the red points. We exclude the time taken to initialize the grid because the test was short and the initialization was a significant fraction of the total runtime, and therefore would heavily bias the total runtime results.

an average of 1.8×10^7 cells per second. All tests have been performed using double precision.

For the strong scaling test, a 512^3 grid is evolved for 10 time steps. The timing results for the total test runtime, the CTU algorithm (performed on the GPU), and the boundary computation including ghost cell exchange communication are tracked separately. We exclude the simulation initialization from the runtime, as it comprises a significant fraction of the runtime for these short tests and obscures the results. As Figure 2.7 shows, the overall scaling of *Cholla* is close to ideal, with the CTU step scaling slightly better than ideal beyond 8 processes. The increased efficiency at 16 processes or more owes to the decomposition decreasing the cells per process below the number that necessitates subgrid splitting, thereby reducing the CPU-GPU communications overhead. All of the GPU calculations contained within the CTU step scale better than ideal at 64 processes. That the boundary condition computation does not scale as well primarily owes to the reduced number of MPI communications needed in runs with small numbers of processes compared with tests utilizing large numbers of processes where every subvolume boundary requires an MPI communication per timestep. The scaling between an 8 process run and a 64 process run, both of which require MPI communications for all boundaries, is close to ideal. We achieve an effective bandwidth for the MPI ghost cell exchange of 2.4 gigabits per second in all runs.

The weak scaling performance of *Cholla* is shown in Figure 2.8. A double precision sound wave perturbation with periodic boundaries is again used, but in this test the size of the total computational volume is rescaled with the number of processes to keep the number of cells per process approximately constant at $\approx 322^3$ (note that each process uses its own distinct GPU during the simulation). The test is run for 10 time steps. The total runtime efficiency as a function of the number of processes remains roughly constant beyond a single process. The CTU algorithm comprises the majority of the computational cost of each timestep, and exhibits nearly perfect weak scaling. The boundary condition calculation for the serial case does not involve MPI communications and is correspondingly inexpensive when using a single process. With two or more processes, MPI communications induce an additional overhead beyond the single process case. However, the weak scaling of the boundary conditions is reasonably maintained to 64 processes.

2.4 Tests

A large variety of hydrodynamics tests exist in the literature, some of which have been used for several decades (e.g. Sod, 1978). Their ubiquity makes these canoni-

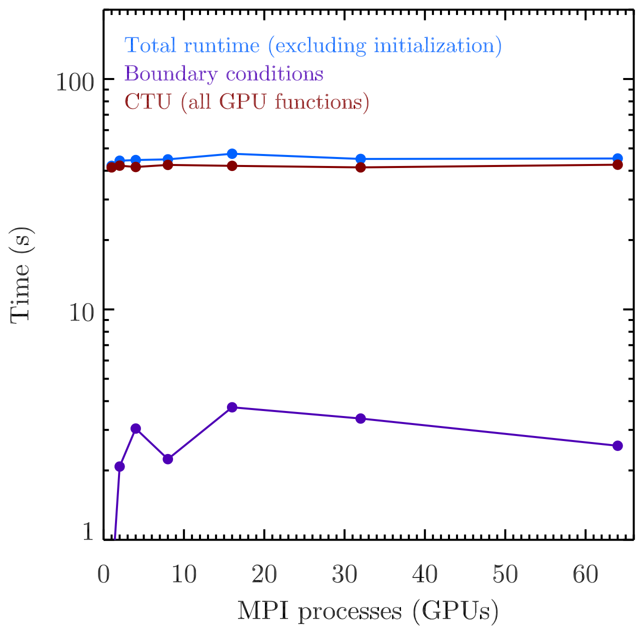

Figure 2.8: Weak scaling performance of *Cholla* . A double precision sound wave perturbation test is used, with the total number of cells calculated by each process is scaled to maintain $\approx 322^3$ cells for a single process. The test is run for 10 time steps. The total runtime (blue points) remains roughly constant up to 64 processes, as does the time taken for the GPU portion of the code (red points). The time taken for the boundary conditions (purple points) increases at low numbers of processes as the maximum number of MPI communications increases, but then remains flat as more processes are added.

cal tests an excellent way to compare the performance of *Cholla* with other codes. In addition, many tests have been designed to explicitly show the failings of hydrodynamics solvers in various environments or highlight the circumstances where they perform exceptionally well. In choosing the tests shown below, we attempt to demonstrate the breadth of problems *Cholla* can simulate. We also demonstrate the effects of changing reconstruction methods or Riemann solvers, and show differences in the outcomes of tests where they are relevant. If not otherwise specified, the following tests were performed using piecewise parabolic reconstruction with the characteristic variables (PPMC) and an exact Riemann solver. All tests were performed in double precision on GPUs.

Before delving into the specifics of each test, we make a note about the convergence rate of *Cholla*. Many shock-capturing methods revert to first order at shocks (see, e.g., Laney, 1998), so to test the convergence rate of *Cholla* the smooth perturbation test described in Stone et al. (2008) is employed. Both the PPMP and PPMC implementations in *Cholla* demonstrate second-order convergence in the L1 error norm out to grid resolutions of 1024 cells.

2.4.1 1D Hydrodynamics

Sod Shock Tube

The Sod problem (Sod, 1978) is often the first test performed by hydrodynamics codes, and we do not diverge from precedent here. Though the Sod problem is not a difficult test to run, the solution contains several important fluid features. We present the test here as an example of the ability of *Cholla* to resolve both shocks and contact discontinuities within a narrow region of just a few zones. The initial conditions are simply a Riemann problem, with density and pressure $\rho_L = P_L = 1.0$ on the left, $\rho_R = P_R = 0.1$ on the right, and an initial velocity $u_L = u_R = 0.0$. The initial discontinuity is at position $x = 0.5$. For this and all of the following one dimensional tests, orthogonal velocities are set to zero. We use an ideal gas equation of state with $\gamma = 1.4$ for all tests, unless otherwise noted.

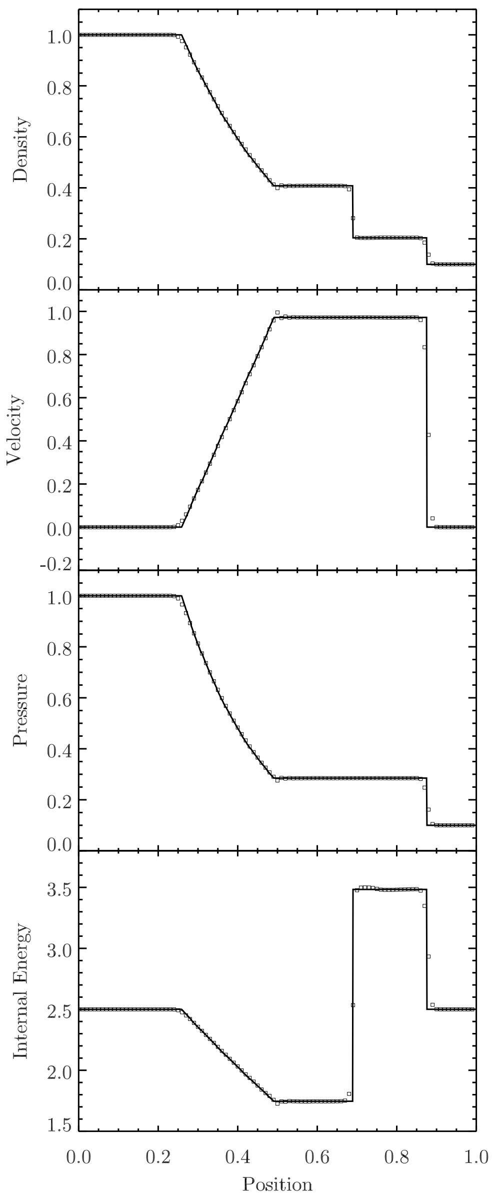

Figure 2.9: The solution to the Sod shock tube test using PPMP with a resolution of 100 cells. The exact solution is shown as a line with points from the *Cholla* simulation over plotted. Features seen in the density plot include a rarefaction wave expanding from the initial discontinuity at $x = 0.5$, a rightward moving contact discontinuity at $x \approx 0.7$, and a rightward moving shock at $x \approx 0.85$.

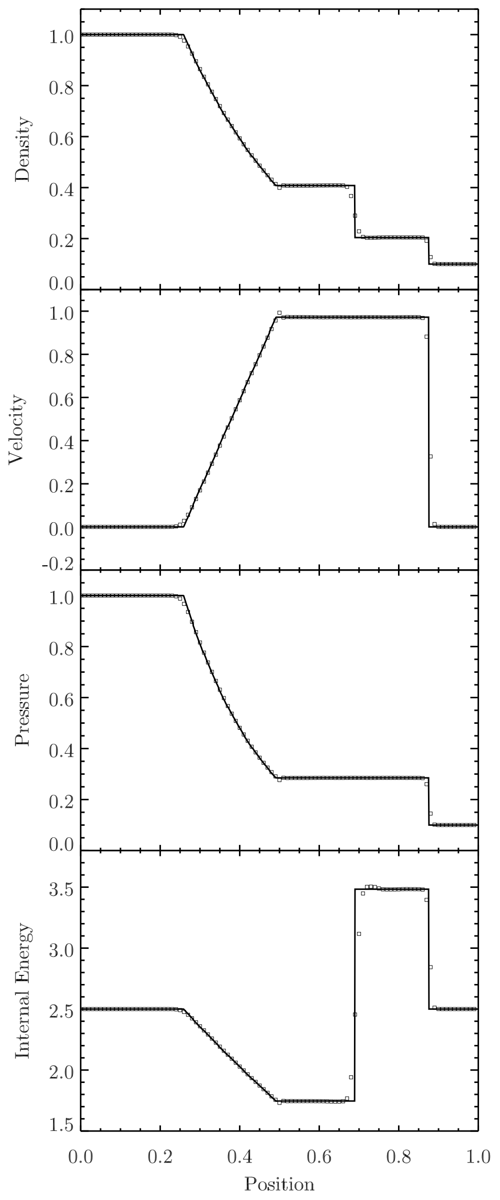

Figure 2.10: The solution to the Sod shock tube test using PPMC with a resolution of 100 cells. The exact solution is shown as a line with points from the *Cholla* simulation over plotted. The same fluid features are seen as in Figure 2.9, with the contact discontinuity slightly less narrowly resolved.

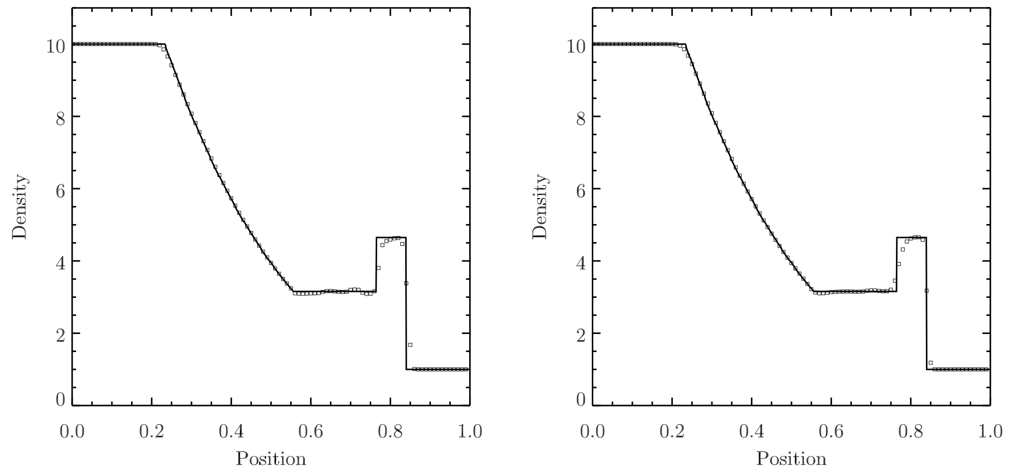

Figure 2.11: Numerical solution to the strong shock test at time $t = 0.2$ using PPMP (left) and PPMC (right) with a resolution of 100 cells, as compared to the exact solution shown by the solid line. The detailed initial conditions are described in the text. The contact discontinuity is better resolved with PPMP, but there fewer oscillations in the solution calculated with PPMC.

The test is computed on a grid of 100 cells until a final time of $t = 0.2$. By that time a shock, a contact discontinuity, and a rarefaction fan have formed and spread enough to be clearly visible as seen in Figures 2.9 and 2.10. As described in Section 2.2.2, *Cholla* has two versions of piecewise parabolic interface reconstruction. PPMP follows the FLASH code documentation (Fryxell et al., 2000) and includes contact discontinuity steepening and shock flattening, while PPMC is based on the Athena code documentation (Stone et al., 2008) and reconstructs the interface values using characteristics without explicit steeping or flattening. As can be seen in the density plot, the contact discontinuity is resolved over just two zones using PPMP, and over three to four zones using PPMC. Because of its explicit treatment of contacts, the PPMP method is slightly better at resolving contact discontinuities, but is also more susceptible to nonphysical oscillations as demonstrated in later tests.

Strong Shock Test

The strong shock test (Fryxell et al., 2000) resembles the Sod shock tube, but is more discriminating owing to the much more severe differences between the left and right initial states. This test starts with an initial discontinuity at $x = 0.5$, with left and right densities $\rho_L = 10.0$ and $\rho_R = 1.0$. Initial pressures are $P_L = 100$ and $P_R = 1.0$. The initial velocities are set to zero, as in the Sod test. The problem is calculated on a grid of 100 cells, and the resulting density in the numerical solution using both PPMP and PPMC is shown at time $t = 0.07$ in Figure 2.11.

As can be seen in Figure 2.11, both PPMP and PPMC do a decent job reproducing the exact solution on this difficult problem. However, the differences between the two reconstruction methods have more discernible effects in this test. The contact discontinuity at $x = 0.75$ is better resolved with PPMP, but the solution is more oscillatory in the region between the contact discontinuity and the tail of the rarefaction fan. In constructing the linear slopes across interfaces, both PPMP and PPMC use limiters that are designed to be total variation diminishing (TVD). However, the third-order reconstruction leads to added complications (Colella and Woodward, 1984). Despite attempts to preserve monotonicity (see Appendix A), problems with strong shocks are observed to cause oscillations in both methods. Due to its more diffusive nature, we find that PPMC is less susceptible to oscillations in regions with strong density and pressure contrasts. The inclusion of contact discontinuity steepening in PPMP keeps contacts sharp but tends to exacerbate the oscillations. The discontinuity detection relies on a number of heuristically determined constants, and the resulting slopes are not always TVD. The oscillations present in the upper panel of Figure 2.11 can be significantly reduced by lowering the value of the constant that determines whether a zone contains a density discontinuity or a shock (see Equation A.36). This constant is labeled "K_0" in Colella and Woodward 1984, not to be confused with "K", the coefficient used in their artificial dissipation scheme. When the discontinuity detection in PPMP is turned off entirely (equivalent to setting $K_0 = 0$) the two methods produce very similar results on the strong shock test.

Strong Rarefaction Test

The strong rarefaction test, or *123 problem*, was originally used by Einfeldt et al. (1991) to illustrate a scenario that causes a subset of approximate Riemann solvers to fail. Because the solution contains a region where the energy is largely kinetic and the pressure is close to vacuum, the Roe solver (or any other linearized solver) will produce negative densities or pressures in the numerical solution (Einfeldt et al., 1991). The initial conditions consist of a fluid with constant density and pressure but opposite receding velocity at the center. Specifically, we set $\rho_L = \rho_R = 1.0$, $P_L = P_R = 0.4$, $u_L = -2.0$, and $u_R = 2.0$. In Figure 2.12, we show the results of this test on a grid of 128 cells at time $t = 0.15$ using PPMP reconstruction and the exact solver. This test is not a challenge using the exact solver, but without modification the Roe solver would fail on this problem. For this reason, we test the density and pressure produced in the solution by the Roe solver and revert to the HLLE solver if necessary. With that fix we can run the problem with either solver, and in fact the HLLE fluxes are only needed on the first step of the simulation.

Shu and Osher Shocktube

The *Shu-Osher shocktube test* shows the tendency of PPM to cut off maxima in smoothly varying problems as a result of the slope limiters imposed in the reconstruction method (Shu and Osher, 1989). The test consists of a strong shockwave propagating into a region with a sinusoidally varying density. The initial conditions are $\rho_L = 3.857143$, $u_L = 2.629369$, and $p_L = 10.3333$; $\rho_R = 1 + 0.2\sin(5\pi x)$, $u_R = 0$, and $p_R = 1.0$. We run the problem on the domain $x = [-1, 1]$ with the initial discontinuity at $x = -0.8$. The results of the test using both 200 cells and 800 cells are shown in Figure 2.13. As can be seen, the low resolution solution does lose some of the amplitude of the peaks. Using newer versions of the limiting functions can help alleviate this problem (Colella and Sekora, 2008; Stone et al., 2008), although we have not yet implemented these limiters in *Cholla*.

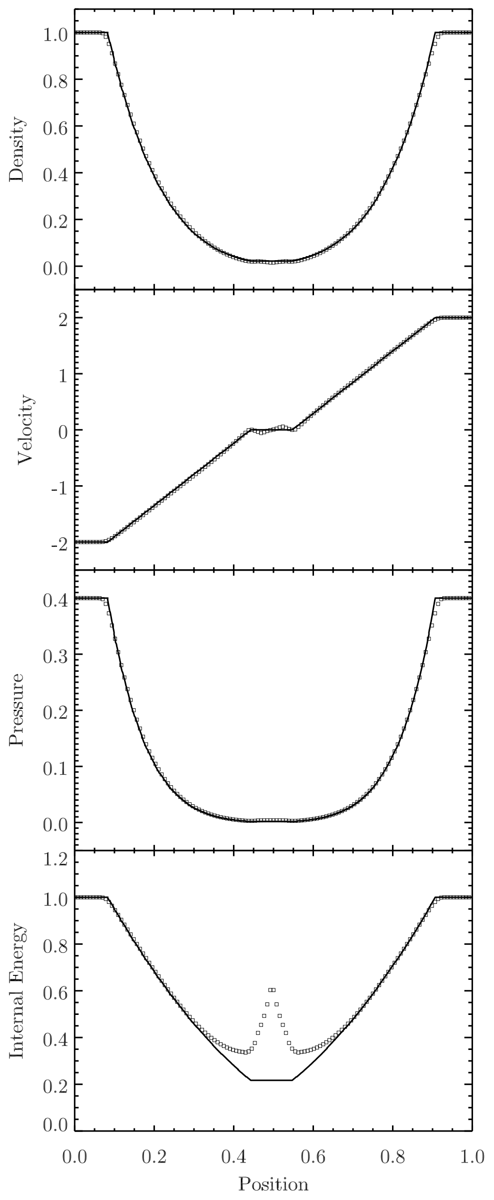

Figure 2.12: Numerical solution to the Einfeldt strong rarefaction test at $t = 0.07$ using PPMP and the exact Riemann solver with a resolution of 128 cells. The exact solution is shown as a line, with the solution from *Cholla* over plotted. This test will cause linearized solvers to fail without modification.

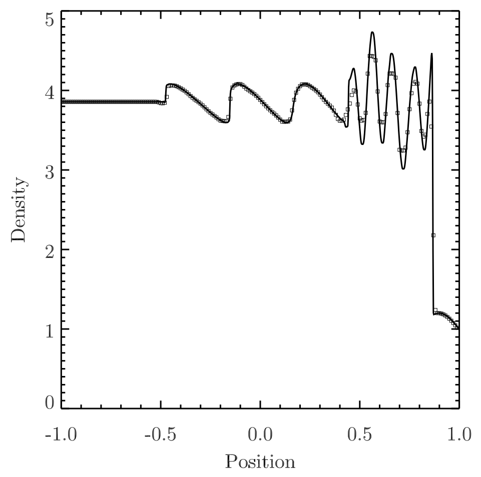

Figure 2.13: Numerical solution to the Shu & Osher shock tube problem. A low resolution solution with 200 cells (points) is plotted over a high resolution solution with 800 cells (line). This test shows the result when PPM limiters cut off maxima in low resolution models of smoothly varying solutions.

Interacting Blast Waves

Originally described in Colella and Woodward (1984), the *interacting blast wave test* helps quantify the behavior of a code near strong shocks and contact discontinuities. The test consists of a medium with initially constant density $\rho = 1.0$, with $\gamma = 1.4$ on the domain $x = [0, 1]$. Reflecting boundary conditions are used. Two shocks are initialized on either side of the domain, with $p = 1000$ for $x < 0.1$, $p = 100$ for $x > 0.9$, and $p = 0.01$ in between. The problem is run until time $t = 0.038$, at which point the shocks and rarefactions in the initial solution have interacted multiple times.

We show plots of the density computed with both PPMP and PPMC in Figure 2.14. We plot a low resolution solution with 400 grid cells over a high resolution reference solution computed with 9600 cells. As can be seen in the figure, PPMP does an excellent job keeping the contact discontinuities at $x = 0.6$ and $x = 0.8$ contained within just two zones, as compared to the solution computed with PPMC in which the contacts are smeared over many cells. In addition, PPMC tends to more severely cut off the maximum at $x = 0.75$, while PPMP does a decent job

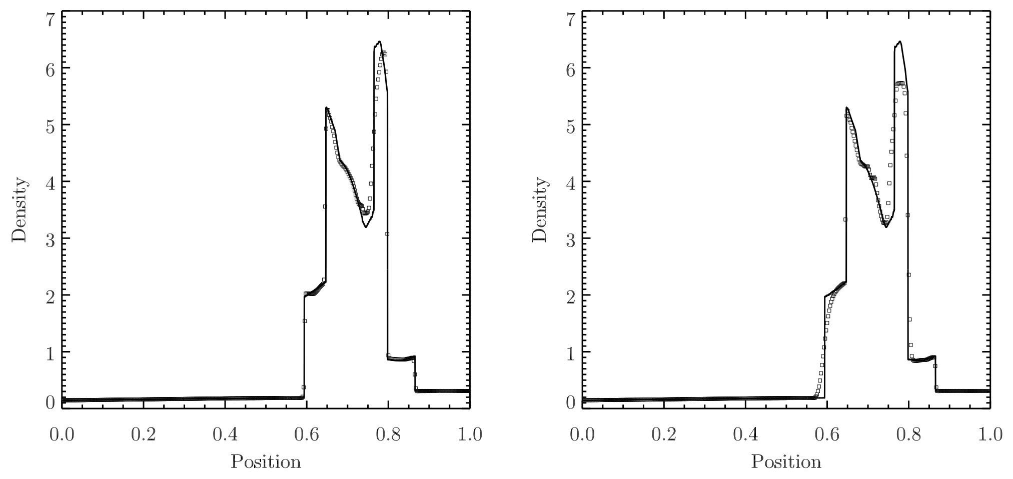

Figure 2.14: Numerical solution for the interacting blast wave test using PPMP (top) and PPMC (bottom) with a resolution of 400 cells, plotted over a reference solution with 9600 cells. This solution is shown at $t = 0.038$, when the original shocks and rarefactions have interacted several times. The test was designed to capture a code's ability to maintain narrow features.

of keeping the full height although the peak is slightly offset. Both reconstruction techniques do a good job reproducing the shocks and the rarefaction fan between $x = 0.65$ and $x = 0.7$.

2.4.2 2D Hydrodynamics

Implosion Test

The *implosion test* is a converging shock problem first presented in Hui et al. (1999). The version presented here is described in Liska and Wendroff (2003a), and begins with a square region of high density and pressure containing a diamond-shaped region of low density and pressure. These initial conditions evoke the traditional Sod shock tube problem extended to two dimensions, but inclined to the grid by 45 degrees rather than aligned as in the one dimensional case. As the test begins material moves inward rapidly toward the center, leading to an implosion. When run for a short amount of time, this test demonstrates the ability of a code to resolve contact discontinuities and other fluid features for a non-grid aligned shock tube.

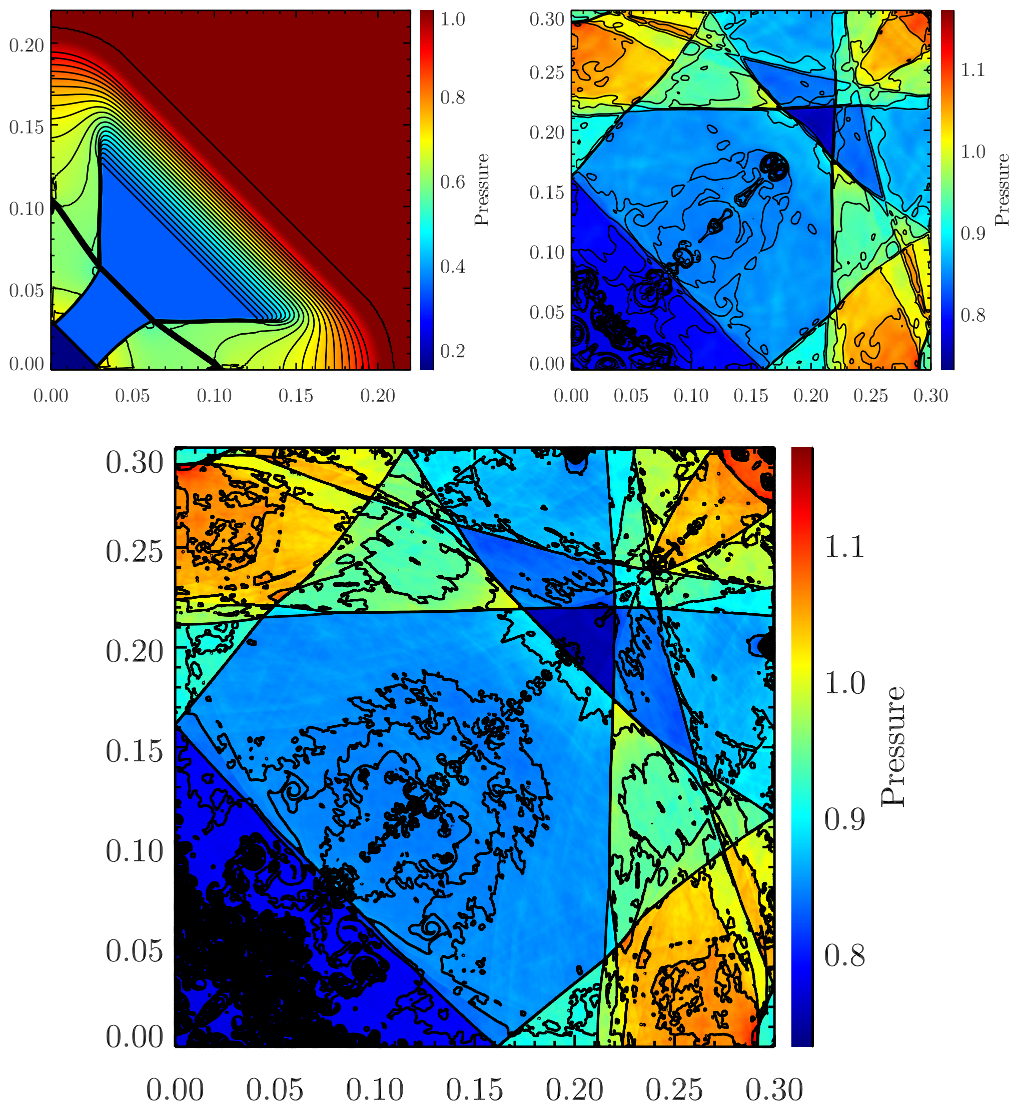

Figure 2.15: Numerical solutions for an implosion test. Top Left: The 400×400 implosion test at $t = 0.045$; 31 density contours from 0.35 to 1.1 are overlaid on a color-scale pressure map, with only the inner region of the computational domain shown, $x = [0, 0.22]$ and $y = [0, 0.22]$. Top Right: The same test at $t = 2.5$; 36 density contours from 0.125 to 1 are overlaid on a color-scale pressure map. Bottom: A 4096×4096 version of the implosion test at $t = 2.5$. The same contour levels are drawn.

When run for enough time to evolve well past the initial shock tube solution, the test illustrates the symmetry (or lack thereof) of a code.

Figure 2.15 shows the results of the implosion test run with PPMC and an exact solver at an early time $t = 0.045$ and a later time $t = 2.5$. The problem was run on a 400×400 grid with a domain $x = [0, 0.3]$, $y = [0, 0.3]$ and reflecting boundary conditions at every boundary, comprising the upper right quadrant of the axisymmetric test described above. The initial density and pressure within the diamond-shaped region are $\rho = 0.125$ and $p = 0.14$, while outside the density and pressure are $\rho = 1.0$ and $p = 1.0$. Initial velocities are zero, as in the Sod test. A discontinuous interface is located along the diagonal running from $(0.15, 0)$ to $(0, 0.15)$. In the upper panel of Figure 2.15 a rarefaction fan can be seen expanding outward from this interface. As the upper panel shows, *Cholla* does an excellent job resolving the contact at early times, as can be seen along the diagonal from $(0, 0.1)$ to $(0.1, 0)$.

At the later time a jet has appeared in the solution. The production of the jet is a direct result of ability to preserve symmetry in the *Cholla* solution to numerical accuracy. Liska and Wendroff (2003a) demonstrated that codes that employ non-symmetry preserving methods like Strang splitting may fail to produce the jet-like feature. The fact that this test is so sensitive to the symmetry of the problem makes it useful for diagnosing potential coding errors, but the test also demonstrates the extent to which a non-symmetric algorithm can impact the physical accuracy of the result. As this test shows, relatively large-scale features in the solution can be completely lost if a code fails to maintain a sufficient level of symmetric accuracy.

The bottom panel of Figure 2.15 shows the results of this test recomputed at a much higher resolution of 4096×4096. The same large-scale features are apparent in the solution, but as expected the small-scale density perturbations and shape of the jet have clearly not converged. However, this high resolution test serves as further evidence of the ability of *Cholla* to preserve axisymmetry even in a very difficult problem. At this extreme resolution, the code must perform over $200,000$ time steps and more than 3×10^{12} cell updates to reach time $t = 2.5$. At that point,

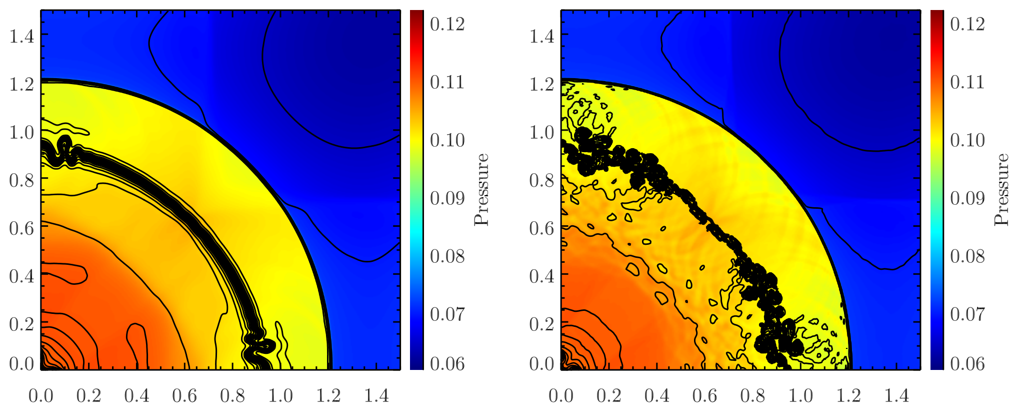

Figure 2.16: Numerical solution to the 2D explosion test at $t = 3.2$ using PLMP (left) and PPMP (right), both at a resolution of 400×400. We show a color map of the pressure overlaid by 27 density contours, from 0.08 to 0.21 with step 0.005. The lower order reconstruction method is more diffusive for the contact discontinuity, but is less susceptible to the instability that causes the contact interface to be unstable.

the results are still exactly symmetric (to floating-point precision), demonstrating that symmetry preservation in *Cholla* is a robust feature of the code.

Explosion Test

The *explosion test*, also from Liska and Wendroff (2003a), is designed to test the evolution of an unstable contact discontinuity and is highly sensitive to numerical noise in the initial conditions. This noise seeds an instability that grows as the solution evolves. The test starts with a domain $x = [0, 1.5]$, $y = [0, 1.5]$ that contains a circularly symmetric region of high density and pressure, with $\rho = 1$ and $p = 1$ inside a circle with radius $r = 0.4$. Reflecting inner boundaries and transmissive outer boundaries are used. Outside the circle the density and pressure are set to $\rho = 0.125$ and $p = 0.1$. The initial velocities are zero. Because the problem is sensitive to initial perturbations at the interface, the density and pressure for cells are area-weighted at the boundary. For each cell on the boundary of the circle the percentage of the area inside the radial boundary is computed, and the initial cell data weighted appropriately.

The test problem is performed on a grid of 400×400 cells, and Figure 2.16 shows the result of the calculation using PLMP and PPMP at $t = 3.2$. As expected, the higher order reconstruction method does a better job preserving the narrow structure of the contact, but is also more susceptible to structure along the interface as the instability develops. Thus, as the problem progresses, the lower order more diffusive method may result in a cleaner solution. We note that both methods preserve the exact symmetry of the problem, provided the initial conditions are symmetric.

Kelvin-Helmholtz Instability

A *Kelvin-Helmholtz instability* test demonstrates the extent to which a hydrodynamics code resolves mixing caused by shear flows. In this test, two fluids at different densities flow past each other and characteristic eddies appear and grow at the interface between the fluids. The growth of the eddies in the linear regime can be analytically described (Chandrasekhar, 1961) and depends on properties of the fluid and the interface itself. At a discontinuous interface, small eddies will develop first at the grid scale of the simulation, and these will gradually grow and combine into larger eddies as shown in Figure 2.17.

The exact nature of the instability depends sensitively on the resolution and initial conditions of the test (e.g., Robertson et al., 2010). The test shown in Figure 2.17 was run on a 1920×1080 grid, with a domain $x = [0, 1]$, $y = [0, 0.5625]$ in order to maintain square cells. The simulation initial conditions include a dense fluid with density $\rho = 2.0$ in the middle third of the box, surrounded by a less dense fluid with density $\rho = 1.0$ in the outer thirds. The denser fluid has a velocity $u = -0.5$, and the less dense fluid has a velocity $u = 0.5$; the y-velocities are initially $v = 0$. The entire simulation volume is initialized in pressure equilibrium with $p = 2.5$. A small-amplitude perturbation is added to the x- and y-velocities of every cell in the grid, in proportion to the x-position following $u = u + 0.01\sin(2\pi x)$ and $v = v + 0.01\sin(2\pi x)$. The simulation is evolved to $t = 1.0$, by which time the growth of the eddies has entered the non-linear regime.

As the eddies grow, more mixing between the high density and low density material occurs. Resolving this mixing is an important task for a hydrodynamics

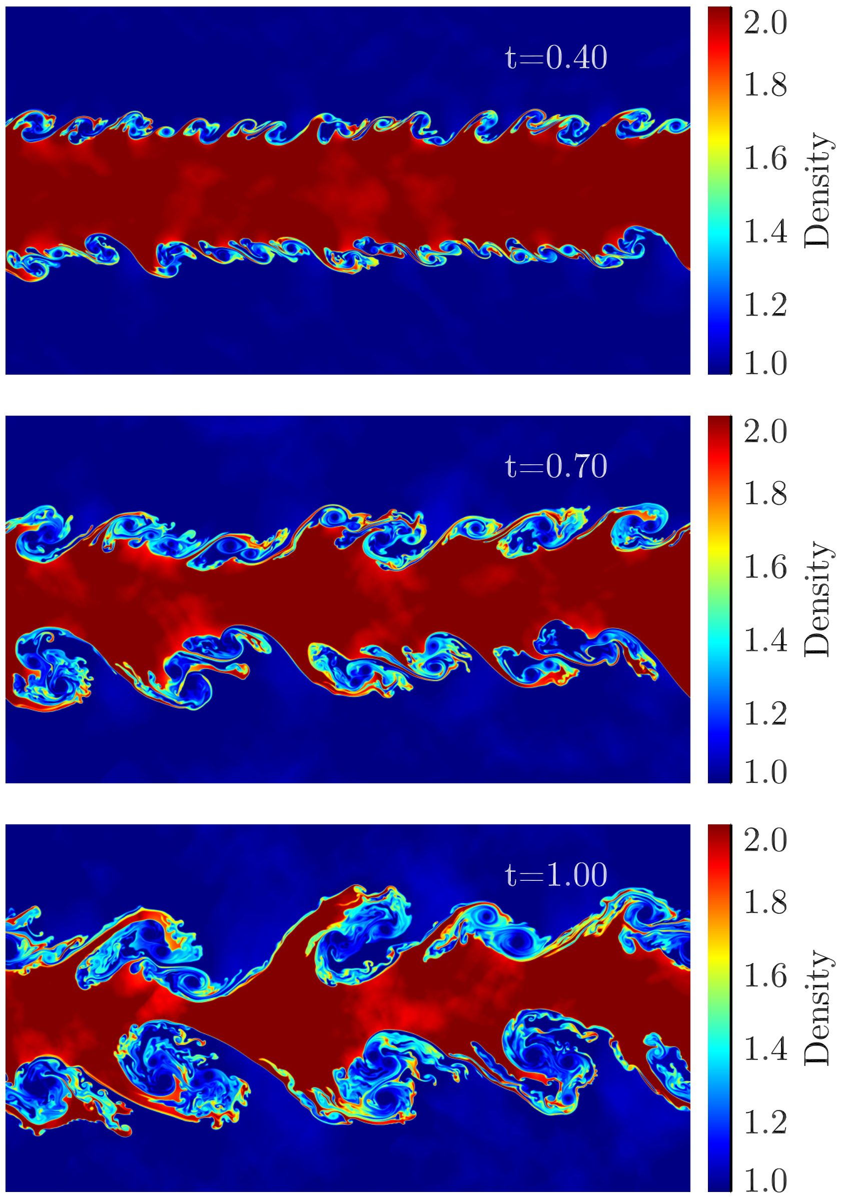

Figure 2.17: Snapshots from a 2D Kelvin-Helmholtz instability test on a 1920 x 1080 grid at t = 0.4, 0.7, and 1.0. The eddies at the discontinuous interface start out at the resolution scale of the simulation and grow in a predictable manner until they enter a non-linear regime.

code, as the amount of mixing can have a significant impact on broad features in the simulation outcome. The level of mixing tends to increase with resolution as well as with higher order reconstruction techniques. Therefore, Kelvin-Helmholtz instabilities highlight the importance of having an efficient high order reconstruction method and a fast code. As expected for a high resolution grid code with a high order reconstruction method, *Cholla* does an excellent job of resolving the shear mixing.

2.4.3 3D Hydrodynamics

Noh's Strong Shock

The *Noh strong shock test*, originally described in one dimension by Noh (1987), demonstrates how well a code can track a strong, high mach number shock. This test is considered difficult to perform in either two or three dimensions, as many hydrodynamics codes cannot run the test accurately and some fail completely (Liska and Wendroff, 2003a). The test starts with a constant density of $\rho_0 = 1.0$ throughout the grid, with zero pressure and constant velocity $|\mathbf{V}| = 1.0$ toward the origin. For this test, the adiabatic index is set to $\gamma = \frac{5}{3}$. These initial conditions result in a formally infinite strength shock reflecting outward from the origin with spherical symmetry. *Cholla* cannot be run with zero pressure, so we set the initial pressure to a low number, $p_0 = 10^{-6}$, but we note that the results are relatively insensitive to the initial pressures below $p_0 \sim 10^{-3}$.

The Noh test is initialized in an octant on the domain $[0, 1]$ with reflecting inner boundaries. The outer boundaries are set according to the analytic solution for the density and energy, which in two or three dimensions is

$$\rho(t) = \rho_0 \left(1 + \frac{t}{r}\right)^{n-1},$$

where r is the radius in polar or spherical coordinates, and n is the dimensionality of the problem. The momentum follows from the velocity and the solution for the

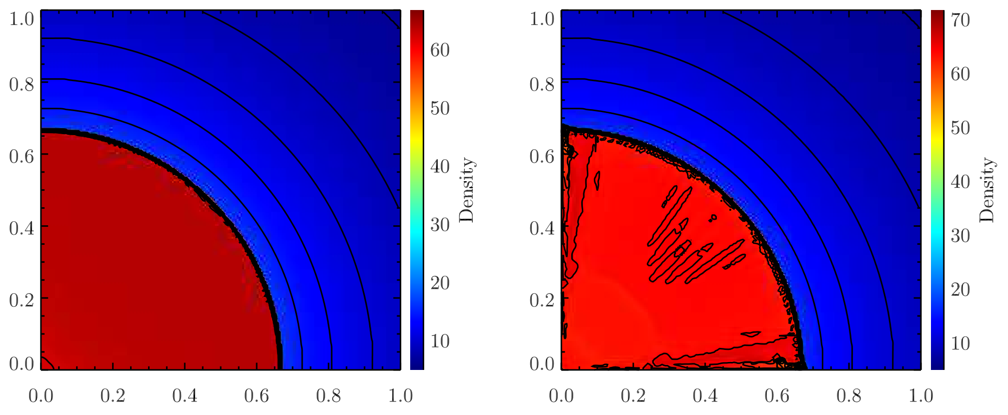

Figure 2.18: Numerical solution of the Noh strong shock test at $t = 2.0$ on a 200^3 grid. These figures show an xy slice through the $z = 0$ plane. 31 density contours from 4 to 64 are overlaid on a color-scale density map. Left: With h correction. Right: Without h correction.

density, and the total energy is set to

$$E(t) = \frac{p_0}{\gamma - 1} + 0.5\rho(t).$$

We evolve the solution to $t = 2.0$, by which time the shock has propagated through more than half of the computational domain. The density immediately in front of the shock as well as the density of the post shock gas can also be calculated analytically. In the 3D case, the gas immediately before the shock has a density of $\rho = 16$, and the post-shock gas has a corresponding density of $\rho = 64$.

Running the Noh test on a Cartesian grid creates strong, grid-aligned shocks that provoke a behavior in the numerical solution known as the carbuncle instability. The carbuncle instability arises as a result of oscillatory crossflow solutions to the Riemann problem near such shocks (Quirk, 1994). This problem is addressed by implementing a form of the H correction, as described in Sanders et al. (1998) and detailed in Appendix C. By incorporating information about the fastest transverse wave speeds, the H correction adds dissipation to the 1D fluxes calculated by the Roe Riemann solver that reduces the carbuncle strength.

The result of the Noh test using PPMC with and without the H correction can be seen in Figure 2.18. Without the H correction, the solution suffers from strong oscillatory behavior, particularly along the edges where the shock is aligned with the grid. In the version with the H correction applied, the unstable behavior along the axes is effectively absent. The region near the origin where the density dips down is a density error known as "wall heating" and is not related to the carbuncle instability. Wall heating is the feature that the 1D Noh test was originally designed to demonstrate. The slight noise along the shock front is a result of the strength of the shock, and is similar to the minor oscillations seen in the 1D strong shock test. We note that implementing the H correction increases the stencil required for CTU. Because the current version of *Cholla* is designed to accommodate a maximum of four ghost cells, we currently implement the H correction only with PPMC or lower-order reconstruction methods.

2.5 New Results for Astrophysical Phenomena: Shockwave-ISM Interactions

We now showcase the power of *Cholla* in an astrophysical context by simulating the interaction of supernova blast waves with clouds in the interstellar medium (ISM). Advancing theoretical understanding of the conditions in the ISM and the effects of supernova feedback is an active area of research (e.g. Agertz et al., 2013; Kim and Ostriker, 2014; Martizzi et al., 2014). Stellar feedback is thought to affect the evolution of galaxies on large scales by generating outflows and regulating star formation rates, but the scales on which this feedback couples to the ISM are unresolved in large cosmological simulations (Martin, 1999; Mac Low and Klessen, 2004). Using high resolution methods to constrain the physics of the ISM on sub-parsec scales is thus critical to improving the subgrid prescriptions applied in simulations of galaxy formation. In addition, simulating the ISM on smaller scales provides high-resolution numerical results that can be compared to observations of gas within our galaxy.

The superior shock-capturing abilities of grid-based hydrodynamic codes enables them to serve as an important tool for simulating the types of high-mach number shocks observed in star-forming regions of the ISM. In addition, simulating the interaction of these shocks with gas clouds benefits from high-order spatial reconstruction techniques that accurately trace the hydrodynamic instabilities that develop in ISM gas. Fast, physically accurate codes like *Cholla* are therefore well-suited for simulating problems like shockwave-ISM interactions.

Theoretical work describing the interaction between shocks and gas clouds has a long history extending back at least to the calculations of McKee and Cowie (1975). In order to treat the problem analytically, these authors presented early computations of a high mach number, planar shock hitting a spherical cloud. Using this simple setup, the speed of the shock within the cloud, $v_{\rm cs}$, can be calculated using only the density contrast between the cloud and the ambient medium, $\chi = n_{cl}/n_{\rm ism}$, and the speed of the shock in the ambient medium, v_s, as follows,

$$v_{\rm cs} = \chi^{-\frac{1}{2}} v_s. \tag{2.29}$$

Klein et al. (1994a) carried out a formative numerical study of the cloud-shock problem, in which they defined a characteristic timescale for the evolution of the cloud. This "cloud crushing time", $t_{\rm cc}$, corresponds roughly to the internal shock crossing time, i.e. $t_{\rm cc} = r_{\rm cl}/v_{\rm cs}$. Using Equation 2.29, the cloud crushing time can be related to the radius of the cloud, $r_{\rm cl}$, the density contrast χ, and v_s, via

$$t_{\rm cc} = r_{\rm cl}\chi^{\frac{1}{2}}/v_s. \tag{2.30}$$

Using this characteristic timescale, various stages of the cloud's evolution can be described. The mixing time of dense cloud gas and the ambient medium holds interest for both quantifying the impact of supernovae on their immediate environments and for the survival time of dense gas in galactic outflows. The shocked cloud experiences various hydrodynamic instabilities that cause its destruction, most importantly the Kelvin-Helmholtz instability (KHI). The long wavelength modes of the KHI tend to break the cloud apart, while the shorter wavelength modes mix cloud material with the surrounding medium. Using 2D adiabatic simulations with density contrasts in the range $10 < \chi < 100$, Klein et al. (1994a) demonstrated a spherical cloud is destroyed by large-scale instabilities in $t_{\rm dest} \simeq 3.5 t_{\rm cc}$, where the destruction time, $t_{\rm dest}$, is defined as the time it takes for the mass in the core of the cloud to be reduced to a fraction $1/e$ of the initial cloud mass. Meanwhile, small-scale instabilities efficiently mix cloud material with the ambient medium in a time of order $4 - 5t_{\rm cc}$. These results were corroborated in 3D simulations by Xu and Stone (1995a).

Subsequent work on the cloud-shock problem has examined how additional physics such as magnetic fields, radiative cooling, self-gravity, and thermal conduction affect the cloud's evolution (e.g. Mac Low et al., 1994a; Fragile et al., 2005a; Orlando et al., 2005a; Melioli et al., 2006; Shin et al., 2008a). These studies have produced many useful results, ranging from the stabilizing effects of magnetic fields in certain configurations to the structural properties of the cloud necessary for gravitational collapse. However, still missing from the literature is an attempt to connect the evolution of clouds with realistic density structures to the analytic theory derived for spheres. Almost all studies of the cloud-shock problem have investigated only spherical or elliptical over-densities, despite the approximately log-normal density distributions of ISM clouds (Padoan and Nordlund, 2002a). One exception is the work of Nakamura et al. (2006a), which showed that clouds with a steeply tapering density profiles were destroyed and mixed more quickly than those with a shallow density gradient. Another is the study by Cooper et al. (2009a), which included a simulation of a cloud with a fractal density distribution. However, that work focused primarily on the long-term survival of radiatively cooling cloud fragments in a galaxy-scale hot wind and did not attempt to produce an analytic timescale for cloud destruction or mixing for the fractal cloud case. A numerical study evaluating the evolution of a cloud with a realistic density distribution in the context of the analytic timescales defined by Klein et al. (1994a) remains to be performed.

In this section, we apply *Cholla* to a preliminary study of this problem. Our goal is to determine whether a realistic cloud with a given mean density is mixed with the ISM on a timescale comparable to a spherical cloud with the same mass and mean density. If not, we wish to adapt the analytical framework of Klein et al. (1994a) to more realistic clouds to characterize their destruction process. To address these issues we carry out a series of high-resolution, 3D hydrodynamic simulations comparing clouds with spherical density distributions to clouds with more realistic density distributions created using a Mach ~ 5 turbulence simulation, and we devise a simple alteration to the Klein et al. (1994a) formalism that enables an analytical description of the cloud evolution for both spherical and realistic density distributions.

2.5.1 The Simulations

We run three sets of simulations with low, medium, and high mean density contrasts, as listed in Table 2.2. Each simulation is run in a $512 \times 512 \times 1024$ box with side lengths $l = 10\,\mathrm{pc} \times 10\,\mathrm{pc} \times 20\,\mathrm{pc}$, corresponding to a resolution of $0.02\,\mathrm{pc}$. In each simulation, a cloud is placed with its center at $(0, 0, 2.5)$. In all simulations the cloud is initially at rest, and the temperature of the gas is set such that the cloud is in pressure equilibrium with the ambient medium. In the low density simulations, both the spherical cloud and the realistic cloud have a mean density that is 10 times the initial ambient density, $\chi = \hat{n}_{\mathrm{cl}}/n_{\mathrm{ism}} = 10$. In the intermediate-density simulations, $\chi = 20$, and in the high-density simulations, $\chi = 40$. The realistic cloud consists of a spherical region excised from a Mach ~ 5 turbulence simulation (Robertson and Goldreich, 2012) and has a log-normal density distribution that is truncated at densities below that of the ambient medium. Higher density regions are scaled such that the mean density of the realistic cloud matches that of the spherical cloud. The highest density regions in the realistic clouds are three orders of magnitude above the ambient density, and the lowest temperatures are of order 10 K. The radius of the spherical cloud in all simulations is $R_{\mathrm{cl}} = 1.07$ pc, and is set such that the total mass in the spherical cloud matches that of the realistic cloud. The low-density clouds (both spherical and realistic) have an initial mass $m_{\mathrm{cl,0}} \approx 0.13\mathrm{M}_{\odot}$, the clouds in the intermediate simulation have an initial mass $m_{\mathrm{cl,0}} \approx 0.25\mathrm{M}_{\odot}$ while the high-density clouds have an initial mass $m_{\mathrm{cl,0}} \approx 0.51\mathrm{M}_{\odot}$. The initial cloud masses and mean densities include all material above the ambient density. Figure 2.19 shows $x - z$ projections of the initial conditions and several later snapshots for the intermediate-density simulations.

We consider the interaction of small clouds with an old supernova blast wave, such that the radius of shock wave can be assumed to be infinite with respect to the cloud radius, and the shock can therefore be treated as planar. The simulation starts with a planar shock wave propagating upward through the box in the $+z$ direction through an ambient medium with an initial number density of hydrogen atoms $n_h = 0.1\,\mathrm{cm}^{-3}$, and initial temperature of $T = 10^4\,\mathrm{K}$. Using the shock jump conditions in the strong shock limit (Zeldovich and Raizer, 1966), the post-shock density, velocity, and pressure are given by

$$\begin{aligned} n_{\mathrm{psh}} &= \frac{\gamma+1}{\gamma-1} n_{\mathrm{ism}}, \\ v_{\mathrm{psh}} &= \frac{2}{\gamma+1} v_s, \\ p_{\mathrm{psh}} &\approx \frac{2\gamma}{\gamma+1} M^2 p_{\mathrm{ism}}, \end{aligned} \tag{2.31}$$

where $v_s = \mathcal{M} c_{\mathrm{ism}}$ is the shock speed, $\mathcal{M}$ is the Mach number of the shock, and c_{ism} is the sound speed in the interstellar medium. For the given initial ISM conditions and an adiabatic index $\gamma = \frac{5}{3}$, a Mach 50 shock travels at $v_s \approx 585\,\mathrm{km\,s}^{-1}$. The post-shock density is $n_h = 0.4\,\mathrm{cm}^{-3}$, $v_{\mathrm{psh}} \approx 440\,\mathrm{km\,s}^{-1}$, and the post-shock temperature is $T = 7.8 \times 10^6$ K. We set an inflowing $-z$ boundary with the post-shock quantities. All other boundaries are outflowing.

2.5.2 Results

Using Equation 3.2 we can calculate cloud-crushing times for the spherical clouds. For the the intermediate-density simulation, $t_{\mathrm{cc}} = 8.00\,\mathrm{kyr}$, while for the low- and high-density simulations $t_{\mathrm{cc}} = 5.65\,\mathrm{kyr}$ and $t_{\mathrm{cc}} = 11.33\,\mathrm{kyr}$, respectively. In order to analyze the mixing of the clouds in the following analysis, we define the cloud mass as the sum of material with a density $n > 2n_{\mathrm{psh}}$, where n_{psh} is the post-shock density of the ambient medium, which is $n_h = 0.8\,\mathrm{cm}^{-3}$ for these simulations. As mentioned previously, Klein et al. (1994a) defined a destruction time corresponding to the cloud breaking into large fragments. While this timescale is useful for the case of a

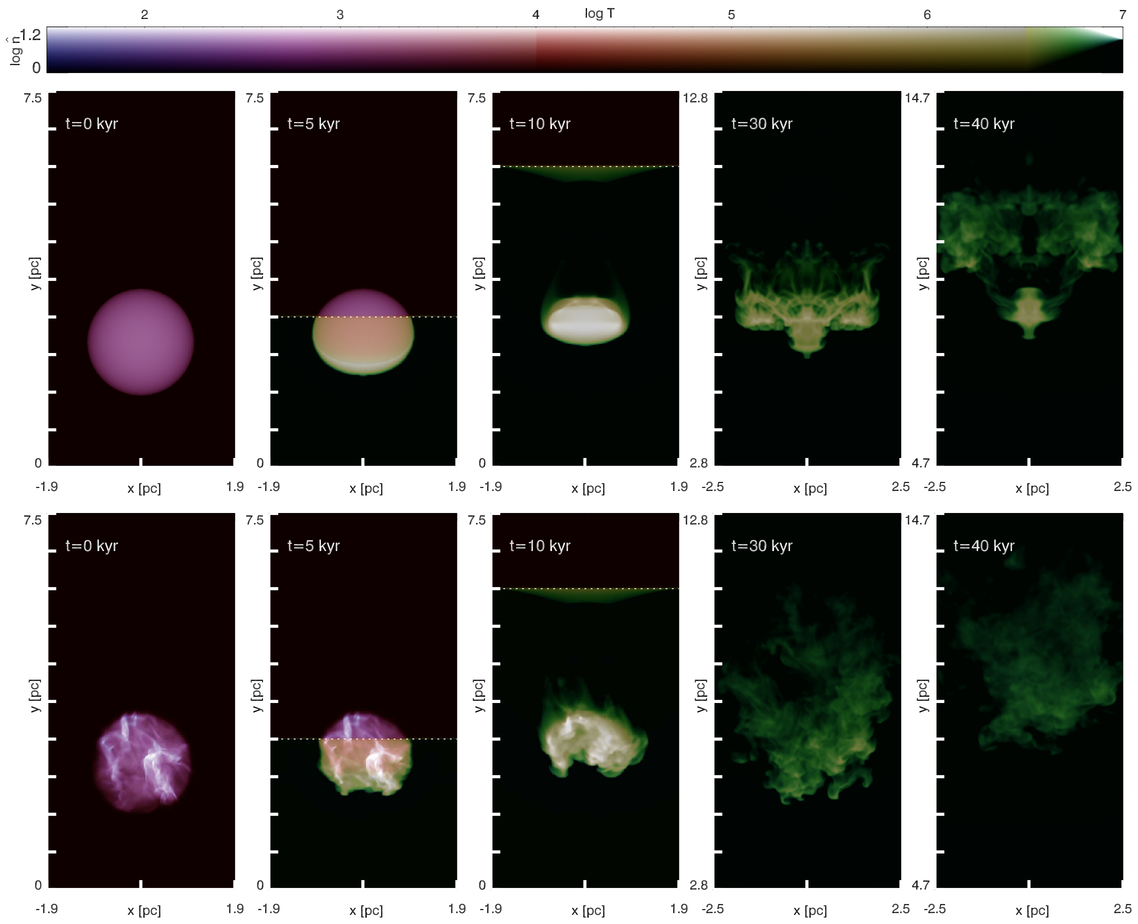

Figure 2.19: Snapshots at $t = 0$, $t = 5\,\mathrm{kyr} = 0.375\,t_{\mathrm{cc}}$, $t = 10\,\mathrm{kyr} = 1.0\,t_{\mathrm{cc}}$, $t = 30\,\mathrm{kyr} = 3.5\,t_{\mathrm{cc}}$, and $t = 40\,\mathrm{kyr} = 4.75\,t_{\mathrm{cc}}$ from the intermediate-density cloud-shock simulations. Each snapshot displays an $x - z$ density projection of the part of the domain containing the cloud. Yellow and green regions indicate high temperature shocked gas, while high intensity indicates higher density material. The density scale represents the average of the projected density, $\hat{n}$. At $t = 0$ the spherical cloud is at rest in a $10\,\mathrm{pc} \times 10\,\mathrm{pc} \times 20\,\mathrm{pc}$ simulation box. The cloud is in pressure equilibrium with the ambient medium, which has a temperature of 10^4 K. At $t = 0.375\,t_{\mathrm{cc}}$ a Mach 50 shock wave has propagated upward from the bottom of box and is sweeping over the cloud. The shock position of the shock wave is indicated by the white dotted line. At $t = 1.0\,t_{\mathrm{cc}}$ the internal shock wave has just crossed the spherical cloud, compressing it and increasing the mean density by a factor of 4. The realistic cloud has already begun to re-expand. At $t = 3.5\,t_{\mathrm{cc}}$ the cloud is being accelerated by the post-shock wind, and large-scale instabilities have disrupted the cloud. At $t = 4.75\,t_{\mathrm{cc}}$, 50% of the cloud material has been ablated and mixed with the ISM.

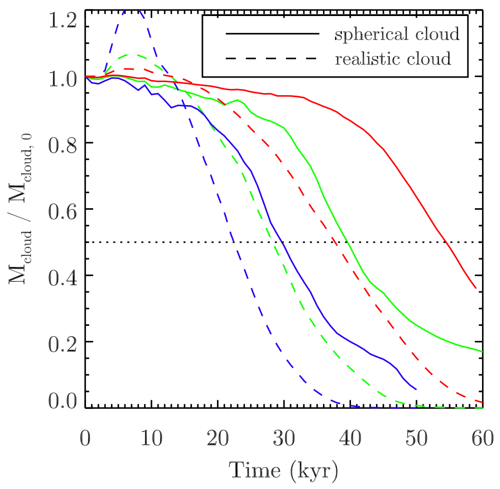

Figure 2.20: The fraction of each cloud's mass as compared to its initial mass, plotted as a function of time. Low-density (blue), intermediate density (green), and high-density (red) cases are shown. The dotted horizontal line indicates the mixing time, t_{mix}, when 50% of the cloud material has fallen below the density threshold, $n = 2n_{\mathrm{psh}}$. The shock wave hits at ~ 2 kyr.

spherical cloud, the realistic cloud contains many separate regions of high density, with no single well-defined core. Therefore, we investigate instead the mixing time t_{mix}, defined as the time at which the cloud mass (material with $n > 2n_{\mathrm{psh}}$) is reduced to 50% of the initial cloud mass. This definition leads to mixing times for the spherical clouds that agree to within a few percent of those quoted in Klein et al. (1994a), and an easy comparison between timescales in the spherical and realistic cloud simulations.

The mass fractions of the spherical and realistic clouds as a function of time for all cases are shown in Figure 2.20. Here we briefly discuss the overall evolution of the intermediate density cases. The shock wave hits at ~ 2 kyr, at which point the clouds start to compress. Owing to the definition of cloud mass, the compression of low-density cloud material causes the realistic cloud to initially gain mass. In contrast, the spherical cloud lacks internal low density material that can be compressed above the $n > 2n_{\mathrm{push}}$ threshold to add to the cloud mass, and the effective cloud mass begins to slowly decrease. At 10 kyr the shock wave has passed through the spherical cloud, which begins to re-expand (see Figure 2.19, middle panels). The shock wave finishes its passage through the realistic cloud about 1 kyr earlier. In both simulations, the cloud is accelerated by the post-shock wind, and hydrodynamic instabilities begin to mix cloud material into the ambient medium. Figure 2.19 shows the continued evolution of each cloud in snapshots at $t = 3.5\, t_{\rm cc}$, the typical cloud destruction time as defined by Klein et al. (1994a), and at $t = 4.75 t_{\rm cc}$, the measured mixing time for the intermediate density spherical cloud in our simulations.

As can be seen in Figure 2.20, the realistic cloud is mixed significantly earlier than the spherical cloud. In the intermediate-density case, the spherical cloud is mixed in 38 kyrs, or $t = 4.75 t_{\rm cc}$, comparable to the results from previous studies (e.g. Klein et al., 1994a; Nakamura et al., 2006a). Table 2.2 shows that the mixing time as a function of $t_{\rm cc}$ does not vary substantially as a function of density for the spherical clouds. In contrast, the realistic cloud is mixed in only 27 kyrs. Using a cloud-crushing time estimated from the mean density, this mixing time corresponds to $t_{\rm mix} = 3.125 t_{\rm cc}$, a much shorter timescale than in the spherical cloud simulation.

Evolution of the low- and high-density simulations is qualitatively similar (see Table 2.2). In each case, the realistic cloud is mixed much more quickly than the spherical cloud, despite having approximately the same initial mass and mean density. Over the density range explored in this work, the mixing time for the realistic clouds is $68 - 75\%$ that of a spherical cloud with the same mean density. Clearly, the definition of the cloud crushing time using the mean density does not fit the evolution of the realistic cloud very well. In our simulations, a realistic cloud is mixed on a timescale comparable to a spherical cloud with half its mass.

We can explain this new result in terms of the structural properties of the spherical and realistic clouds. While the mean density and mass of the realistic cloud matches that of the spherical cloud, 73% of the volume in the realistic cloud is filled with gas below the mean density. As a result, the internal shock propagates more quickly through the cloud, expediting the introduction of velocity shear between high density cloud material and the post-shock wind. Rather than a single reflected bow shock, the realistic clouds experience many reflected shocks from each high-

Table 2.2. Adiabatic cloud simulation parameters.

	Parameter	$\chi = 10$	$\chi = 20$	$\chi = 40$
Spherical Cloud	$\hat{n}_{\rm cl}$	9.98	19.97	39.93
	$t_{\rm cc}$ (kyr)	5.65	8.00	11.33
	$t_{\rm mix}$ (kyr)	28	38	53
	$t_{\rm mix}/t_{\rm cc}$	4.96	4.75	4.68
Realistic Cloud	$\hat{n}_{\rm cl}$	11.45	20.73	38.22
	$n_{\rm cl,med}$	6.11	10.68	18.95
	$t_{\rm cc,med}$ (kyr)	4.42	5.84	7.79
	$t_{\rm mix}$ (kyr)	21	27	36
	$t_{\rm mix}/t_{\rm cc,med}$	4.75	4.62	4.62

Note. — Average parameters for both the spherical and realistic clouds for the low-density, intermediate-density, and high-density simulations are listed. Parameters include average cloud density, $\hat{n}_{\rm cl}$, median cloud density, $n_{\rm cl,med}$, cloud crushing times using either the average (spherical cloud) or median (realistic cloud) density, $t_{\rm cc}$ or $t_{\rm cc,med}$, mixing times in kyr, and mixing times as a function of cloud-crushing time.

density region, as can be seen in the middle panel of Figure 2.19. The individual high-density regions fill a much smaller volume than the spherical cloud, effectively decreasing the cloud radius term that appears in Equation 3.2. The increased surface area for velocity shear for each high density region in the realistic cloud leads to material being more quickly ablated and mixed with the ambient medium. In contrast, the high-density core of the spherical cloud is protected by the outer material and survives beyond the mixing time.

The evolution of the realistic cloud can still be described within the analytic framework of Klein et al. (1994a) if a more appropriate density is used in the definition of the cloud crushing time. Rather than using the average density, we analyze the mixing time for the realistic cloud using the median density, as the median better represents the density of the volume-fillling gas within the cloud. Median cloud densities and the corresponding cloud-crushing times are given in the bottom half of Table 2.2. Using these cloud-crushing times, the mixing time for the realistic cloud matches much better the results for the spherical case.

Further examinations of this problem are left for future work, but potentially interesting investigations include a more detailed parameter study incorporating clouds that are not initially at rest (i.e. have a realistic momentum distribution), and shocks with a finite radius of curvature (i.e. a nearby supernova). Nonetheless, our results clearly show that significant quantitative and qualitative differences exist between the destruction and mixing of an idealized, spherical cloud and a cloud with a log-normal density distribution, even when each has a similar mean density. Further, we newly show that these differences can be understood by applying the Klein et al. (1994a) formalism adapted to use the median cloud density.

2.6 Conclusions

In this work we have presented *Cholla*, a new, massively-parallel, three-dimensional hydrodynamics code optimized for Graphics Processor Units (GPUs). *Cholla* uses the unsplit Corner Transport Upwind algorithm (Colella, 1990; Gardiner and Stone, 2008a), multiple Riemann solvers, and a variety of reconstruction methods to model numerical solutions to the Euler equations on a static mesh.

In writing the code, we have maintained a modular structure that allows for the implementation of different hydrodynamical schemes. *Cholla* features five methods for interface reconstruction, including the first-order piecewise constant method, two second-order linear reconstruction methods, and two third-order methods based on the original piecewise parabolic method developed by Colella and Woodward (1984). There are multiple Riemann solvers, including the exact solver from Toro (2009) and a linear solver based on the method of Roe (1981). Incorporating multiple reconstruction and Riemann solver methods provides the ability to test results for a dependence on the particular numerical techniques used, and the strengths and weaknesses of the different methods are discussed. *Cholla* also implements an optional diffusive correction called the H correction (Sanders et al., 1998) to suppress instabilities along grid-aligned shocks. The H correction adds additional diffusion to the Roe fluxes based on the fastest transverse velocities. The Appendices of this paper detail all of the equations used in the code, and supplement the discussion of each method presented in the main text.

The strategies employed in designing *Cholla* to run natively on GPUs are extensively detailed. The necessity of transferring data to and from the GPU with every time step requires a specific memory layout to improve efficiency. Once information has been transferred to the GPU, the CTU integration algorithm can be effectively divided into kernel functions that execute on the device, similar to functions in a traditional CPU code. Each of these kernels is self-contained and contributes to the modularity of the code. Because the GPU has limited global memory storage, the simulation volume must often be subdivided to optimize performance and compensate for GPU memory constraints. The strategy we employ to execute this "subgrid splitting" efficiently is also presented.

The architectural differences between CPUs and massively-parallel GPUs require the illucidation of several key concepts in GPU programming. When a device kernel is called, the GPU launches a large set of individual computational elements called threads. A single kernel call can launch millions of threads. In *Cholla*, each thread is assigned the work of computing the updated hydrodynamic conserved variables for a single real cell in the simulation volume. The streaming multiprocessors on the GPUs handle the work of assigning threads to GPU cores, which means that *Cholla* will be easily transportable to newer generations of GPU hardware or other coprocessors. Thousands of cores operating simultaneously on each device results in thousands of simultaneous cell updates, making the hydrodynamics solver in *Cholla* very fast. GPUs are designed to optimize for throughput as opposed to latency. As demonstrated in the GPU-based time step calculation presented in Section 2.3.5, adding additional floating point calculations to GPU kernels is relatively inexpensive. This feature can be exploited to incorporate more physics on the GPU, such as cooling, at relatively little computational cost.

The scalability of *Cholla* and its performance on a range of hydrodynamics problems was also documented. Using the Message Passing Interface (MPI) library (Forum, 1994), *Cholla* can be run across many GPUs in parallel. The code incorporates both slab-based and block-based domain decompositions and exhibits excellent strong and weak scaling in block-decomposed tests using at least 64 GPUs. We present the results of a suite of canonical hydrodynamics tests in one, two, and three dimensions. The state-of-the art hydrodynamics algorithms employed enables *Cholla* to perform accurately on a wide variety of tests, and the GPU architecture makes the computations fast without sacrificing physical accuracy. Since a single GPU can compute a simulation with nontrivial resolution, using *Cholla* with a cluster presents the option of running many problems to explore large parameter spaces rapidly. Further, the excellent weak scaling of *Cholla* suggests that very large problem sizes can be tackled on large GPU-enabled supercomputers.

We demonstrate the power of *Cholla* for astrophysics by addressing the classic numerical problem of a shock hitting a small gas cloud in the interstellar medium. Calculations of the evolution of such clouds require high numerical resolution and excellent shock-capturing abilities (Klein et al., 1994a), and *Cholla* was designed to address such astrophysical problems. Comparing the well-studied ideal case of spherical overdensities to realistic clouds with log-normal density distributions created by supersonic turbulence, we present new results showing that realistic clouds are destroyed more quickly than spherical clouds of the same mass and mean density. In our simulations, realistic clouds are mixed with the ambient medium on timescales comparable to spherical clouds with half their mean density. We posit that the faster destruction time is a result of the lower median density of the gas in the realistic cloud. The shock propagates more quickly through the realistic clouds, allowing the hydrodynamic instabilities responsible for disrupting the cloud and mixing the gas to develop sooner than in the spherical case. We further show that the Klein et al. (1994a) formalism can be used to describe the destruction of our realistic cloud simulations provided that the median density is used in place of the average cloud density. These results demonstrate the successful application of *Cholla* in the performance of high-resolution, physically-accurate astrophysical simulations. We plan to use similar calculations in the future to connect small-scale stellar feedback to galaxy evolution on larger scales.

Lastly, we note that in contrast to many MPI-enabled codes, we have designed *Cholla* to perform almost all hydrodynamics calculations on the GPU without transferring information to the CPU during the course of a single timestep, leaving the computational power of the CPU to address other tasks (see e.g. Figure 2.3). By performing calculations on the CPU and GPU simultaneously, additional physics could be modeled on the CPU during the hydrodynamical computation on the GPU. Logical extensions to *Cholla* include using Fourier transforms to solve gravity or drive turbulence. The addition of a magnetohydrodynamics module on the GPU is also an attractive possibility, as *Cholla* uses an unsplit integration algorithm that is optimized for MHD (Gardiner and Stone, 2008a).

CHAPTER 3

Hydrodynamical Coupling of Mass and Momentum in Multiphase Galactic Winds[†]

Using a set of high resolution hydrodynamical simulations run with the *Cholla* code, we investigate how mass and momentum couple to the multiphase components of galactic winds. The simulations model the interaction between a hot wind driven by supernova explosions and a cooler, denser cloud of interstellar or circumgalactic media. By resolving scales of $\Delta x < 0.1$ pc over > 100 pc distances our calculations capture how the cloud disruption leads to a distribution of densities and temperatures in the resulting multiphase outflow, and quantify the mass and momentum associated with each phase. We find the multiphase wind contains comparable mass and momenta in phases over a wide range of densities and temperatures extending from the hot wind ($n \approx 10^{-2.5}$ cm^{-3}, $T \approx 10^{6.5}$ K) to the coldest components ($n \approx 10^{2}$ cm^{-3}, $T \approx 10^{2}$ K). We further find that the momentum distributes roughly in proportion to the mass in each phase, and the mass-loading of the hot phase by the destruction of cold, dense material is an efficient process. These results provide new insight into the physical origin of observed multiphase galactic outflows, and inform galaxy formation models that include coarser treatments of galactic winds. Our results confirm that cool gas observed in outflows at large distances from the galaxy ($\gtrsim$ 1kpc) likely does not originate through the entrainment of cold material near the central starburst.

3.1 Introduction

Star-forming galaxies commonly feature a multiphase galactic wind, observed at a wide variety of densities, temperatures, and velocities (e.g. Lehnert and Heckman,

[†]This chapter has been published previously as Schneider & Robertson, 2017.

1996; Martin, 2005; Rupke et al., 2005; Strickland and Heckman, 2007; Tripp et al., 2011; Rubin et al., 2014), and over a large range of redshifts (e.g. Weiner et al., 2009; Coil et al., 2011; Nestor et al., 2011; Bouché et al., 2012; Kornei et al., 2012; Bordoloi et al., 2016). Despite their ubiquity, fully characterizing these winds can prove difficult. Spatially-resolved observations of the wind's many phases remain challenging, even for the nearest star-forming systems (Shopbell and Bland-Hawthorn, 1998; Westmoquette et al., 2009; Rich et al., 2010; Leroy et al., 2015). Different observational techniques and instruments are required for different phases, so amassing a complete picture for even a single galaxy represents a large coordinated effort. At higher redshifts, absorption line studies that trace outflowing gas in and around star-forming galaxies can be challenging to interpret as they require making assumptions about the wind's geometry (e.g. Rubin et al., 2011; Bouché et al., 2012). While much progress has been made in recent years thanks to the installation of the Cosmic Origins Spectrograph on the *Hubble Space Telescope*, large uncertainties still exist regarding the contributions of different phases of winds to the net mass, momentum, and energy content of outflows (Heckman et al., 2015).

Winds also play an important role in theoretical studies of galaxy evolution. Supernova-driven winds provide an attractive method of feedback in cosmological simulations, allowing galaxies to regulate their star formation rates and gas supply over cosmic time (e.g., Oppenheimer and Davé, 2008; Davé et al., 2011; Faucher-Giguère et al., 2011; Dalla Vecchia and Schaye, 2012; Muratov et al., 2015). Recent simulations have successfully reproduced the galaxy stellar mass function across a wide range of redshifts by including phenomenologically-motivated wind models (Vogelsberger et al., 2014; Schaye et al., 2015; Davé et al., 2016). However, the processes that launch winds and govern their evolution as they escape galaxies remain unresolved on the scale of cosmological simulations. We currently must turn to smaller-scale, higher-resolution simulations to learn more about the physical nature of the winds themselves.

On these smaller physical scales, idealized simulations of galactic winds have also presented a theoretical challenge. Both analytic studies and hydrodynamic simulations of winds have had difficulty accelerating cool gas to the velocities observed in winds, because the dense phases get destroyed by hydrodynamic instabilities too quickly (e.g., Zhang et al., 2015; Scannapieco and Brüggen, 2015; Brüggen and Scannapieco, 2016). Magnetic fields may play an important role in stabilizing the cool gas (McCourt et al., 2015), but without realistic comparisons to observations the most important physical processes at play in multiphase winds are difficult to ascertain. A detailed analysis of the momentum and energy budget of gas in different phases in these hydrodynamic simulations has not yet been conducted. This data would be valuable both for improving sub-grid prescriptions of winds in cosmological simulations, and for comparing with observations to better determine where our theoretical understanding of winds fails. However, such a study requires high resolution across a large simulation volume in order to track the gas in different phases for significant periods of time.

In this work, we aim to improve our theoretical understanding of multiphase galactic winds via high resolution, idealized simulations. Using the recently released Graphics Processor Unit (GPU)-based code *Cholla* [1] (Schneider and Robertson, 2015), we can perform hydrodynamic simulations of the interaction between cool and hot phases of a starburst-driven wind at high resolution (< 0.1pc) over a large volume (> 100pc). The code performs well enough to compute such simulations on a static mesh, and thus capture the interaction between the different phases of gas across a much larger region than any previous study (e.g. Cooper et al., 2009b; Scannapieco and Brüggen, 2015; Banda-Barragán et al., 2016). The ability to track gas in each phase over long periods of time allows a direct probe of the momentum coupling between the hot and cool phases of the wind. In addition, the calculations add an element of physical realism to the cool gas by changing the initial density structure of the multiphase clouds to better match the features seen in spatially-resolved outflows of dense gas.

Our simulations model a multiphase galactic wind as cold, dense interstellar or circumgalactic medium clouds embedded within a hot, rarified background flow

[1]A public version of the *Cholla* code is available at: http://github.com/cholla-hydro/cholla

driven by supernovae. Because the cool material starts at rest with respect to the background wind, the initial interaction between the two phases drives a shock into the dense cloud. While the current work focuses on the cloud densities, shock mach numbers, and physical scales relevant to galactic winds, the adopted numerical setup allows for comparisons with previous investigations of cloud-shock interactions.

Because of its ubiquity in the ISM, the shock-cloud interaction problem has been studied by many authors. Early numerical work by Klein et al. (1994b) investigated the case of a planar shock interacting with a spherical cloud using two-dimensional, adiabatic simulations. Their work indicated that clouds encountering a shock typically survive for a few "cloud crushing times," roughly the timescale for the initial shock to propagate through the cloud. For strong shocks, the cloud crushing time depends on the density contrast between the cloud and the ambient medium, the size of the cloud, and the speed of the shock. Earlier under-resolved numerical work came to similar conclusions (Bedogni and Woodward, 1990; Nittmann et al., 1982). These studies found that shocked clouds travel ~ 8 cloud radii before mixing with the ambient medium as a result of hydrodynamic instabilities. Adiabatic three-dimensional simulations (Stone and Norman, 1992; Xu and Stone, 1995b) corroborated the two-dimensional results, and additionally attempted to account for different cloud geometries. Cloud geometry and orientation in those simulations did not affect the timescale for cloud fragmentation, but did substantially affect the late-time morphology of the clouds before they were destroyed.

These early studies could reasonably ignore radiative cooling effects by limiting their studies to small clouds. In larger scale problems where the cooling timescale is smaller than the dynamical timescale, thermal energy losses must be included. Many authors have investigated this regime (e.g., Mellema et al., 2002; Fragile et al., 2004; Melioli et al., 2005; Cooper et al., 2009b), and demonstrated that radiative cooling inhibits destruction of the dense material and extends the lifetime of the cloud relative to the adiabatic case. Rather than efficiently mixing with the hot post-shock wind, radiatively-cooling clouds tend to get strung out into filaments containing individual "cloudlets" of dense gas that can survive much longer. Other authors have investigated the effects of conduction (e.g., Marcolini et al., 2005; Orlando et al., 2005b; Brüggen and Scannapieco, 2016; Armillotta et al., 2016) and magnetic fields (e.g., Mac Low et al., 1994b; Fragile et al., 2005b; Shin et al., 2008b; McCourt et al., 2015; Banda-Barragán et al., 2016) on the cloud-shock interaction, with varying results for the stabilization of the cloud.

While multiple previous works studied a range of potentially-important physics, few explored the impact of the initial structure of the cloud on the results of cloud-shock interactions. Early work focused on modeling supernova remnants in the ISM, and a simple spherical cloud provided a sufficient approximation for the initial conditions. In radiatively-cooling galactic winds, however, the initial morphology of the cloud may have a profound effect on its evolution. Only Cooper et al. (2009b) previously studied how the internal structure might influence the cloud destruction, using a fractal cloud as a proxy for a realistic cloud in a galactic wind. They found that fractal clouds survived for less time than initially spherical clouds. More recently, Schneider and Robertson (2015) examined how a turbulent interior cloud structure can alter the cloud crushing timescale in adiabatic simulations.

Our current study aims to better quantify the differences in the physical picture for inhomogeneous clouds, and more broadly describe the way the gas phases in the outflow evolve. Specifically, we attempt to capture the region of parameter space relevant for the cool ($\sim 10^4$ K) clouds observed in galactic winds near the disks of star-forming galaxies. In this regime, the wind can be adequately modeled as a hot ($\sim 10^6$ K), supersonic fluid containing a population of embedded clouds of denser, cooler, initially stationary material. Depending on the exact density contrast between the cool and hot phases, the cooling timescale may fall below the local dynamic timescale and the simulations therefore should include radiative cooling. Other potentially relevant effects, such as conduction and magnetic fields, we leave for future study.

An outline of our paper follows. We describe in Section 3.2 the model used to study the interaction between the multiple phases of the wind. In Section 3.3 we explain the setup of our wind simulations. Section 3.4 presents the qualitative evolution of the wind-cloud interaction, including the impact of the initial surface density of the cool gas on the cloud evolution. In Section 3.5, we describe in detail the density and temperature structure of the multiphase outflow. In Section 3.6 we study the velocities of the gas and describe how momentum distributes between different phases of the wind. Section 3.7 presents a resolution study focused on increasingly small-scale features in turbulent clouds. Section 3.8 contains our interpretation of these results, including a discussion of our findings in relation to previous work, possible effects of incorporating additional physical processes, and an analysis of the fate of dense gas within a gravitational potential. We summarize in Section 3.9.

3.2 A Multi-Component Wind Model

Theories of starburst galaxies have long suggested that the combination of stellar winds and supernovae should drive a hot ($\sim 10^8$ K) wind out of the starburst region (e.g. Chevalier and Clegg, 1985). This hot wind fluid remains difficult to observe directly, requiring high spatial resolution X-ray spectra. In the nearby starburst galaxy M82 where such observations are possible, the detection of a diffuse $\sim 4 \times 10^7$ K plasma in the central region indicates the presence of a hot wind (e.g. Griffiths et al., 2000; Strickland and Heckman, 2007). Our current study assumes that stellar feedback can drive such a wind, and that the coupling of energy and momentum from the hot wind fluid with cooler gas leads to the multiphase winds seen in many starburst systems (see the review by Veilleux et al., 2005). In this work, we seek to better quantify the coupling between the hot, rarified phase of galactic winds, and the cooler, denser outflowing gas that is nearly ubiquitously observed in rapidly star-forming systems.

We attempt to account for the origin of a multiphase wind by modeling the interaction between a hot fast outflow and the cold, dense "clouds" it may entrain. Adequately capturing the hydrodynamic processes occurring in such a scenario requires resolutions of ≈ 2 orders of magnitude smaller than the size of the cool clouds in question. Even with a tool like *Cholla*, modeling clouds with sizes of ~ 10 pc requires an idealized set of simulations to probe sufficiently the interactions between the hot and cool gas phases in a wind. Correspondingly our simulations examine dense clouds in a box with a background wind (see Figure 3.2), representing cool material exposed to the hot phase of a wind. In the following two subsections, we detail our models for both the hot background and cool cloud components of these multiphase winds.

3.2.1 Hot Wind Component

We seek to model the impact of a hot, supernovae driven outflow on cooler material. Given a small set of assumptions including spherical symmetry and negligible radiative cooling, the hot phase of the wind can be modeled analytically at distances close to the plane of a galaxy. All of our simulations use an analytic model of the hot wind as a constant background state with properties set using the adiabatic wind model of Chevalier and Clegg (1985, hereafter, CC85). The CC85 model envisions a hot wind driven by central energy and mass input from stellar feedback processes. Three input parameters determine the solutions to the model: the energy input as a function of time, $\dot{E}$, the mass input as a function of time, $\dot{M}$, and the size of the driving region within which energy and mass are injected, R_*. By $\dot{M}$ we mean the total mass injection rate to the wind due to supernovae and mass-loading, *not* the star formation rate. With these parameters, a solution to the set of spherical hydrodynamic equations can be found that smoothly transitions from subsonic within the driving region, to supersonic at further radii. The solutions cross the sonic point at $r = R_*$.

We choose the input parameters for our version of the CC85 model according to the fits derived by Strickland and Heckman (2009) using *Chandra* X-ray observations of the nearby starburst galaxy M82. In particular, we set $\dot{E} = 10^{42}$ erg s^{-1}, $\dot{M} = 2$ M$_\odot$ yr^{-1}, and $R_* = 300$ pc. In interpreting their results, we have made additional assumptions about the wind mass-loading factor, β and the supernova thermalization fraction α for which they give a range of correlated values. Here we are using the Strickland and Heckman (2009) interpretation of β meaning

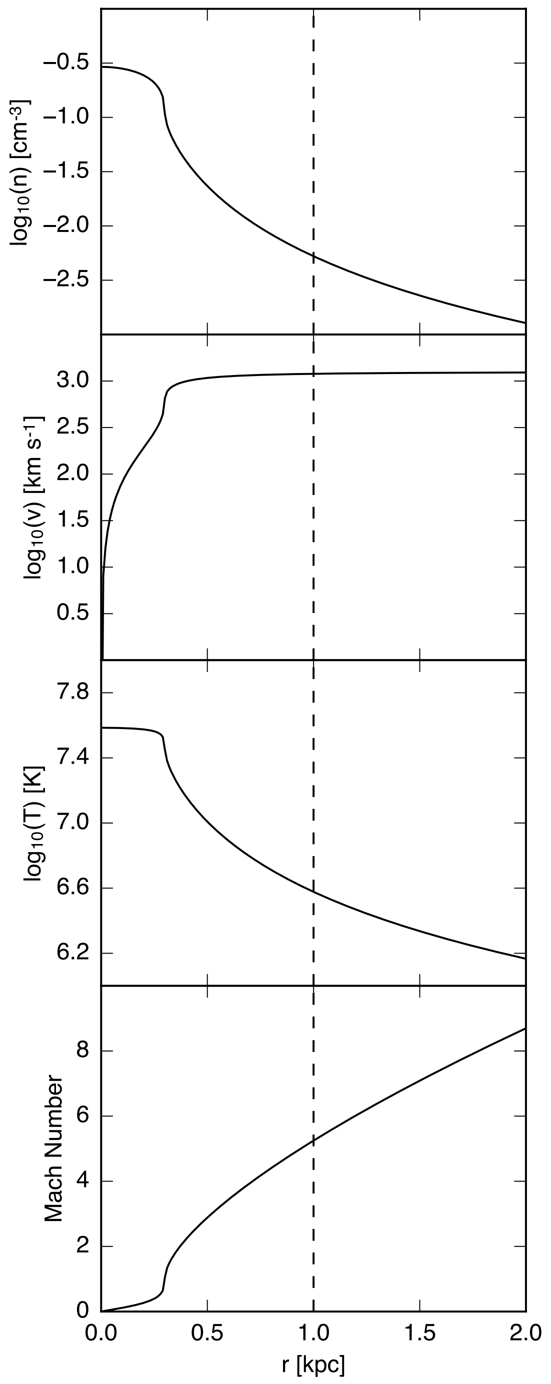

Figure 3.1: The adiabatic wind model used in all simulations. The top three panels display physical values of number density, radial velocity, and temperature as a function of radius. The fourth panel shows the dimensionless mach number of the wind, which crosses the sonic point at $r = R_* = 300$ pc in our model. The wind-cloud simulations use values at $r = 1$ kpc for the background wind, shown with the dashed vertical line in each panel.

the fraction of total mass injected into the wind as compared to the mass injected by supernovae and stellar winds, $\beta = \dot{M}/\dot{M}_{\mathrm{SN+SW}}$. Likewise, our definition of α corresponds to their ϵ, and refers to the fraction of the supernova energy that is deposited in low-density gas and does not suffer large radiative losses before being incorporated into the wind. We take $\beta = 1.42$ and $\alpha = 0.33$, values near the middle of the acceptable range of fits reported by Strickland and Heckman (2009). These choices give a central temperature of $T_c \gtrsim 10^{7.5}$ K in the driving region, consistent with estimates made from the X-ray emission (see Strickland and Heckman, 2009, Table 2). The resulting values for number density, velocity, temperature, and mach number as a function of radius for this model are displayed in Figure 3.1.

For the background flow in our multiphase wind simulations, we use the physical parameters of the CC85 wind model shown in Figure 3.1 at $r = 1$ kpc. These values are

$$n_{\mathrm{wind}} = 5.2626 \times 10^{-3}\ \mathrm{cm}^{-3},$$

$$v_{\mathrm{wind}} = 1.1962 \times 10^{3}\ \mathrm{km\ s}^{-1} = 1.2225\ \mathrm{pc\ kyr}^{-1},$$

$$P_{\mathrm{wind}}/k = 1.9881 \times 10^{4}\ \mathrm{cm}^{-3}\ \mathrm{K},$$

where k is the Boltzmann constant. At $r = 1$ kpc, the wind pressure corresponds to a temperature of $T_{\mathrm{wind}} = 3.7778 \times 10^{6}$ K, as displayed in Figure 3.1. We choose a radius of 1 kpc to set the background wind properties in our simulations for several reasons. First, we wish to capture the mass-loading of the wind outside the driving region, which restricts us to hot wind properties at radii $r > 300$ pc. Second, we will model the interactions between cool and hot material in a simulation volume with a physical length of 160 pc. Given that the wind properties in our simulations remain approximately constant across this volume and the hot wind density, temperature, and mach number changes most rapidly just outside the driving region (see Figure 3.1), we favored an initial radius of $r \sim 1$ kpc over smaller radii. Further, the best observations of a multiphase wind come from M82, where cool material clearly resides at $r > 1$ kpc above the disk (Leroy et al., 2015). The fits derived by Strickland and Heckman (2009) suggest that the hot wind should not yet have suffered

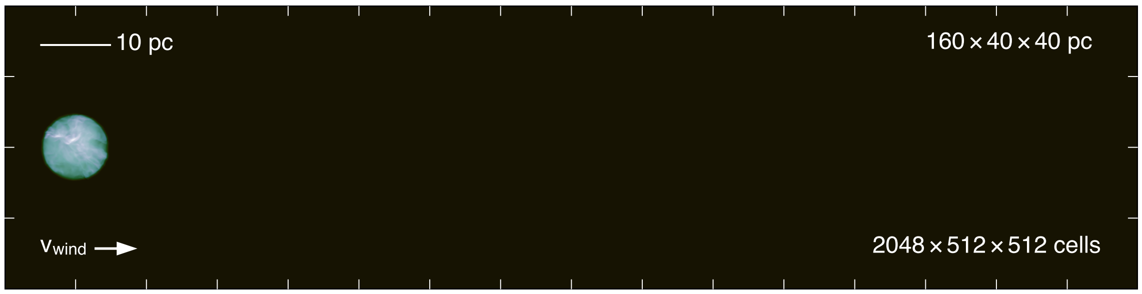

Figure 3.2: The initial conditions for one of our cloud-wind simulations. Each simulation box is much larger than the initial size of the cloud, so we can track the long-term evolution of cloud material, even after it has been stripped from the main body of the cloud. This density projection shows the $\tilde{n} = 1\ \mathrm{cm}^{-3}$ turbulent cloud. The initial density distribution for the cloud material in this simulation is displayed in Figure 3.3.

serious radiative losses at $r \sim 1$kpc, which would invalidate the CC85 model (Zhang et al., 2015; Thompson et al., 2016).

3.2.2 Cool Cloud Component

To capture the multiphase nature of galactic winds, our simulations also include a cool component representing interstellar or circumgalactic material. As with previous studies of galactic winds, we model this second component as dense clouds initially at rest with respect to the hot wind (e.g. Scannapieco and Brüggen, 2015; Brüggen and Scannapieco, 2016). Our aim is to extend previous studies by examining the detailed momentum and energy coupling between different wind phases, so we consider both idealized spherical clouds and more realistic turbulent clouds with a distribution of interior densities set by turbulent processes. Observations have revealed cool gas in outflows at a variety of densities and temperatures (see references in Section 3.1). By varying the median density of the cold gas, $\tilde{n}$, and its interior density structure, we are able to model a range of properties for the dense component of the wind. Both the median number density and the cloud morphology affect the integrated surface density of the cold gas, $\Sigma_{\rm cl} = M_{\rm cl}/\pi R_{\rm cl}^2$, where $M_{\rm cl}$ is the total mass of the cloud and $R_{\rm cl}$ is the cloud radius. The surface density influences how momentum transfers from the hot wind to the cold component, as discussed below.

As the hot wind destroys the clouds, cool material will both heat and rarify. This material will accelerate as momentum transfers from the hot wind, and enter the outflowing wind with a range of velocities. To adequately track all of this material, we perform simulations using boxes with long aspect ratios. The simulation boxes feature transverse dimensions equal to 8 $R_{\rm cl}$, and a long dimension along the wind direction of 32 $R_{\rm cl}$. Figure 3.2 displays an example of an entire box, showing a density projection of the $\tilde{n} = 1$ cm^{-3} turbulent cloud at time $t = 0$. The clouds are initially centered 2 $R_{\rm cl}$ from the left boundary of the box to capture the resulting bow shock. The long dimension of the box enables the simulations to follow the bulk of the cloud as it accelerates *and* track material stripped from the main cloud body.

Cloud and Sphere Models

Each cloud initially sits at rest and in thermal pressure equilibrium with the surrounding hot wind. The clouds in our study are not dense enough to be gravitationally confined, allowing us to neglect self-gravity. The density of the cloud material therefore determines its initial temperature. We initialize the spherical clouds with constant interior temperature, and initialize the interior temperatures of regions within the turbulent clouds along an appropriate isobar. We list the temperatures and median and mean densities of the cloud initial conditions in Table 3.1. We list the full range of temperatures for the turbulent clouds - their median temperature matches the spheres. For both the spheres and turbulent clouds we taper the densities at the edge, starting at a radius of 4.5 pc, such that the cloud density smoothly transitions from the median to the wind density. The density taper has the form

$$n(r)_{\rm cl} = \tilde{n} \exp[(\ln(n_{\rm wind}/\tilde{n})/(R_{\rm cl} - 4.5)]|r - 4.5|, \quad (3.1)$$

where r is the distance from the center of the cloud, and $\tilde{n}$ is the median cloud density. $R_{\rm cl} = 5$ pc for all of our clouds. To create the turbulent clouds, we excise a region from a Mach 5 isothermal turbulence simulation (Robertson and Goldreich,

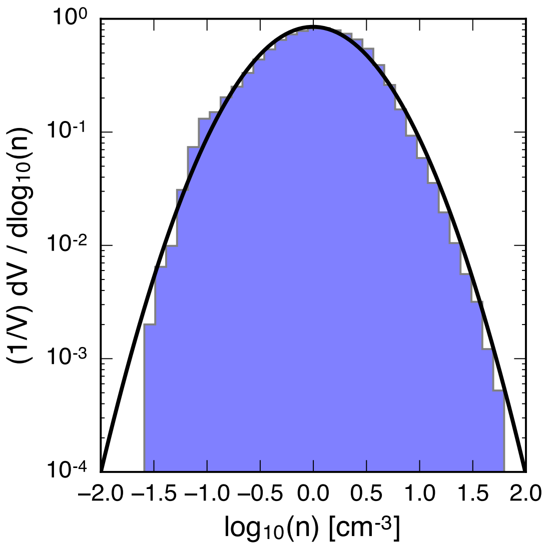

Figure 3.3: A normalized histogram of the initial density distribution for the $\tilde{n} = 1$ turbulent cloud is shown in blue, fit with a gaussian in log-space. Note: this distribution does not include the density taper at the edges of the cloud, as its purpose is to illustrate the lognormal distribution of the bulk of the cloud's mass.

2012), and scale the density linearly to match the desired median. Figure 3.3 shows a histogram of the initial gas densities for the $\tilde{n} = 1$ cm^{-3} turbulent cloud simulation.

Cloud Crushing Time

To interpret our results in relation to previous studies of cloud-shock interactions, we make use of the "cloud-crushing time" $t_{\rm cc}$ initially devised by Klein et al. (1994b). When the hot wind first interacts with the cool cloud, a shock drives into the over-dense gas (and a reverse shock reflects into the oncoming wind). The cloud-crushing time estimates how long the initial shock takes to pass through and compress the cloud. This timescale can be calculated in terms of known quantities by relating the initial density contrast of the cloud to the wind $\chi = n_{\rm cl}/n_{\rm wind}$ along with the radius of the cloud $R_{\rm cl}$ and the wind velocity $v_{\rm wind}$ as

$$t_{\rm cc} = R_{\rm cl}\chi^{\frac{1}{2}}/v_{\rm wind}. \tag{3.2}$$

We use the cloud-crushing time as an evolutionary timescale for the clouds in our simulations, and report $t_{\rm cc}$ for each of our simulations in Table 3.1. The derivation leading to Equation 3.2 assumes that the pre-shocked gas in the cloud evolves adiabatically, allowing for an estimate of the shock speed within the cloud from the density contrast and wind speed. In a radiatively-cooling cloud the pre-shock conditions change as the cloud cools, decreasing the sound speed and slowing the shock. The cloud crushing times listed in Table 3.1 therefore represent imperfect estimates of the cloud compression timescale, and our simulations show they underestimate the duration of this phase. In our simulations, maximum compression is typically reached at $\sim 2\ t_{\rm cc}$.

Cloud Cooling Time

The cooling time provides another important evolutionary timescale for the clouds we simulate. If the cloud cooling time greatly exceeds the cloud crushing time, we expect the clouds to evolve similarly to the well-studied adiabatic case (Klein et al., 1994b; Xu and Stone, 1995b; Poludnenko et al., 2002; Nakamura et al., 2006b). If the cloud crushing time grows much longer than the cooling time, the radiative loss of energy may affect the cloud properties substantially before the initial shock can disrupt it. We can estimate the cooling time of the clouds in our simulation as their thermal energy divided by the rate at which energy is lost owing to radiative cooling as

$$t_{\rm cool} = \frac{3 n_{\rm cl} k T_{\rm cl}}{2\Lambda(T_{\rm cl})}, \tag{3.3}$$

where k is the Boltzmann constant and $\Lambda(T)$ is the cooling rate in erg s^{-1} cm^{-3}, evaluated at the cloud temperature (see Appendix F for information on our calculation of cooling rates). In theory, we might like to use $T_{\rm cl}$ and n_{cl} post-shock to calculate the cooling time. Given the complex nature of the cooling function, these quantities often prove impossible to calculate analytically (Creasey et al., 2011). Additionally, the lognormal gas distribution in our turbulent clouds means that n and T vary for different parts of the cloud. Instead, we calculate the cloud cooling times using the median pre-shock density and temperature to compare most directly with the cloud-crushing time. These cooling times appear in Table 3.1. Our simulations sample the transition from clouds dominated by adiabatic evolution to those in the radiatively cooling regime. As discussed in Section 3.4, our simulations reproduce this transition and thereby justify our assumptions used to compute the cooling time.

3.3 Simulations

We ran a total of six “production” simulations with initial parameters displayed in Table 3.1. The first parameter we varied in these simulations was the cloud density distribution. Half the simulations modeled constant density spherical clouds to compare with previous studies. The other half modeled clouds with a lognormal density distribution appropriate for a turbulent gas (e.g. Padoan and Nordlund, 2002b; Kritsuk et al., 2007). We also varied the median density of the clouds from $\tilde{n} = 0.1$ cm^{-3} to $\tilde{n} = 1.0$ cm^{-3} to sample a range of density and temperature phase space. At the low end, this density range samples the transition from adiabatic to radiative evolution of the cool gas. At the higher densities, the evolution of the clouds is very similar when scaled by the cloud-crushing time (See Section 3.4), but higher density clouds have increasingly lengthy lifetimes. As a result, our upper limit on cloud density is set by computational expense.

Throughout this paper, we refer to the production simulations by the names given in Table 3.1. Simulations with spherical clouds begin with an ‘S’, and those with turbulent clouds are denoted ‘T’. The remainder of the name references the median density of the cloud. E.g. the simulation featuring a turbulent cloud with a median number density of $\tilde{n} = 1$ cm^{-3} is ‘T1’; the spherical cloud with a median number density of $\tilde{n} = 0.1$ cm^{-3} is ‘S01’, etc. We will sometimes refer to the spherical clouds as “spheres”; “cloud” alone will always refer to a turbulent cloud.

Each of the simulations listed in Table 3.1 was run in a volume with $N = 2048 \times 512 \times 512$ cells, and physical dimensions of $160 \times 40 \times 40$ pc, yielding a physical

Table 3.1. Radiative cloud simulation parameters.

Model Name	$\tilde{n}_{\rm cl}$[a] [cm^{-3}]	$\bar{n}_{\rm cl}$[b] [cm^{-3}]	$\Sigma_{\rm cl}$[c] [$M_\odot$ pc^{-2}]	N_H[d] [cm^{-2}]	$T_{\rm cl}$[e] [K]	$M_{\rm cl}$[f] [M$_\odot$]	χ[g]	$t_{\rm cc}$[h] [kyr]	$t_{\rm cool}$[i] [kyr]
S01	0.1	0.086	0.013	2.87×10^{18}	1.988×10^{5}	1.05	1.9×10^{1}	17.8	26.0
S05	0.5	0.45	0.064	1.43×10^{19}	3.976×10^{4}	5.04	9.5×10^{1}	39.8	3.96
S1	1.0	0.88	0.13	2.85×10^{19}	1.988×10^{4}	10.0	1.9×10^{2}	56.4	1.30
T01	0.1	0.16	0.022	2.05×10^{19}	2.2×10^{3}	1.73	1.9×10^{1}	17.8	26.0
T05	0.5	0.83	0.11	1.02×10^{20}	4.4×10^{2}	8.61	9.5×10^{1}	39.8	3.96
T1	1.0	1.7	0.24	2.23×10^{20}	2.0×10^{2}	18.8	1.9×10^{2}	56.4	1.30

[a]Median density of the cloud, calculated including all material with a density greater than 1/10 $\tilde{n}$.

[b]Mean density of the cloud, calculated including all material with a density greater than 1/10 $\tilde{n}$.

[c]Surface density of the cloud, calculated as $M_{\rm cl}/(\pi R_{\rm cl}^2)$.

[d]Maximum initial column density sight-line through the cloud (excluding the hot wind).

[e]Initial temperature of the cloud material, in pressure equilibrium with the wind. The median temperature is provided for the spherical clouds, and the temperature of the highest density regions is provided for the turbulent clouds.

[f]Initial mass of the cloud, calculated including all material with a density greater than 1/10 $\tilde{n}$.

[g]Initial density contrast of the cloud with the background wind, calculated using $\chi = \tilde{n}_{\rm cl}/n_{\rm wind}$.

[h]Cloud crushing time, calculated using Equation 3.2 with the median density and a cloud radius of 5.0 pc.

[i]Cloud cooling time, calculated using Equation 3.3 with the median cloud density and temperature.

Note. — Simulations with a spherical cloud begin with an 'S'; those with a turbulent cloud begin with a 'T'. All simulations listed in Table 3.1 are run in a volume with a numerical resolution of $2048 \times 512 \times 512$ cells, corresponding to a physical box size of $160 \times 40 \times 40$ pc.

resolution for these simulations of $\Delta x = 0.07825$ pc/cell. We also carried out a resolution study (see Section 3.7), with higher and lower resolution simulations for the $\tilde{n} = 0.5$ turbulent cloud. Previous work on this problem has relied on techniques such as adaptive mesh refinement and cloud-tracking (moving the reference frame of the simulation to follow the main body of the cloud) to reduce computational costs. In contrast, we have constant resolution across our entire volume, and as a result we capture the evolution of both low and high density material. Our long boxes also allow us to track material at distances further from the cloud than in previous studies.

We ran our simulations with the *Cholla* hydrodynamics code (Schneider and Robertson, 2015), using piecewise parabolic reconstruction, an HLLC Riemann solver, a simple unsplit integrator, and a dual energy scheme. Optically-thin radiative cooling with a photoionizing UV background was assumed, and implemented using pre-computed Cloudy tables (Ferland et al., 2013). Details of the integration scheme, Riemann solver, dual energy, and radiative cooling appear in the Appendices. We used outflow boundaries for all sides of the box except the left x-face, where the inflow was set according to the wind properties described in Section 3.2.1. All of the production simulations were run for a minimum of 25 t_{cc}. The ultra high resolution simulation used in our resolution study was only run for 12 t_{cc}, as it is extremely expensive.

The six simulations listed in Table 3.1 have high enough resolution to adequately capture the hydrodynamic instabilities that disrupt the spherical clouds, 64 cells/R_{cl}. These simulations were used to produce the majority of the results in this paper. In addition, we have run low resolution versions of each of the simulations in Table 3.1 in a volume with half as many cells ($N = 1024 \times 256 \times 256$). The primary purpose of these low resolution runs was to verify initial conditions, estimate runtimes, and test for convergence. We have also run a low resolution simulation with a different mean molecular weight (see Appendix F) to test the effect of the cooling implementation on our results.

In total, we have run 14 simulations for this paper. The 7 low resolution simulations were carried out on the *El Gato* cluster at the University of Arizona. The high resolution production simulations and ultra high resolution comparison simulation were carried out on the *Titan* supercomputer at the Oak Ridge Leadership Computing Facility via a director's discretionary time allocation. In sum, these high resolution simulations took ≈ 1.5 million *Titan* core-hours, which corresponds to 50,000 GPU-hours. The time each simulation required varied greatly based on the total length of cloud survival, from the lowest density $\tilde{n} = 0.1$ spherical cloud simulation that required $\approx$ 1,500 GPU-hours, to the $\tilde{n} = 1.0$ spherical cloud simulation that required $\approx$ 9,400 GPU-hours. The $\tilde{n} = 0.5$ ultra high resolution turbulent cloud simulation required $\approx$ 14,000 GPU-hours despite only running to 12 $t_{\rm cc}$.

3.4 Cool Cloud Evolution

We begin our results with a general description of the evolution of the cool gas in the clouds, focusing on the $\tilde{n} = 1$ turbulent cloud simulation (T1), our fiducial case. The evolution of cool material when exposed to a hot wind is of interest, given the uncertainty in the theoretical community about whether it is possible to accelerate cool material to the hot wind speed without destroying it (e.g. Scannapieco and Brüggen, 2015; Zhang et al., 2015). If cool gas can be efficiently entrained, the process could provide an explanation for the presence of cool material observed at large distances from galaxies (e.g. McCourt et al., 2015). The efficiency with which cool material is destroyed will also affect the mass-loading factor of the hot wind, which may in turn affect the chances of thermal instability and rapid cooling of the hot wind (Wang, 1995; Thompson et al., 2016). In this section, we describe the overall evolution of the cool gas in our simulations as a function of both cloud morphology and initial surface density. In particular, we focus on the destruction time of the cool material.

The early stages of adiabatic shock-cloud interactions have been thoroughly described in the literature, particularly beginning with the comprehensive study of Klein et al. (1994b). Cloud evolution is often described in terms of the cloud-crushing time, $t_{\rm cc} = \chi R_{\rm cl}/v_{\rm wind}$ (see Section 3.2.2). In an adiabatic simulation, the initial compression of the cloud is followed by a downstream expansion, after which the cloud quickly fragments and dense material is destroyed. The entire cloud destruction process typically takes 4 - 5 $t_{\rm cc}$ for spheres. If the cloud-crushing time is calculated using the median gas density, adiabatic turbulent clouds are destroyed in a similar number of crushing times (Schneider and Robertson, 2015).

When dense clouds are able to cool radiatively, their evolution follows a different path. The early cloud crushing phase is much more effective as the heat generated by compression is radiated away. In a radiative cloud, gas densities can reach over an order of magnitude above their pre-shock levels. Though radiative clouds still get disrupted within 10 $t_{\rm cc}$, the individual dense "cloudlets" that result can take as long as 40 $t_{\rm cc}$ to mix into the hot wind (Scannapieco and Brüggen, 2015).

3.4.1 Turbulent Clouds vs Spheres

In Figures 3.4 and 3.5 we illustrate the general evolution of the $\tilde{n} = 1$ cm^{-3} turbulent and spherical cloud simulations (models T1 and S1, Table 3.1). These figures show the column density of gas in the box (including the hot wind), in both a y-axis projection with the hot wind entering the box from the left, and an x-axis projection in the direction of the wind. The figures show snapshots of the simulations at 5 representative times - the initial conditions, and after 2, 5, 10, and 20 $t_{\rm cc}$.

In both simulations, the maximum average cloud density is reached at $t = 2$ $t_{\rm cc}$. The maximum average density is $\bar{n} = 5.2$ cm^{-3} for T1, and $\bar{n} = 6.8$ cm^{-3} for S1. (We calculate the mean density using material with density greater than 1/10th the initial median density, as in Table 3.1.) Despite the similarity of this compression timescale, Figures 3.4 and 3.5 show a drastic difference in cloud morphology at $t = 2$ $t_{\rm cc}$. While the initial shock has propagated at different speeds through regions of different density for the turbulent cloud, the shock has very effectively compressed the sphere into a single flat pancake. As a result, the evolution of the two types of clouds diverges strongly at later times. Low density material is quickly accelerated between $t = 5-20$ $t_{\rm cc}$ in the turbulent cloud, leaving only a few high density cloudlets

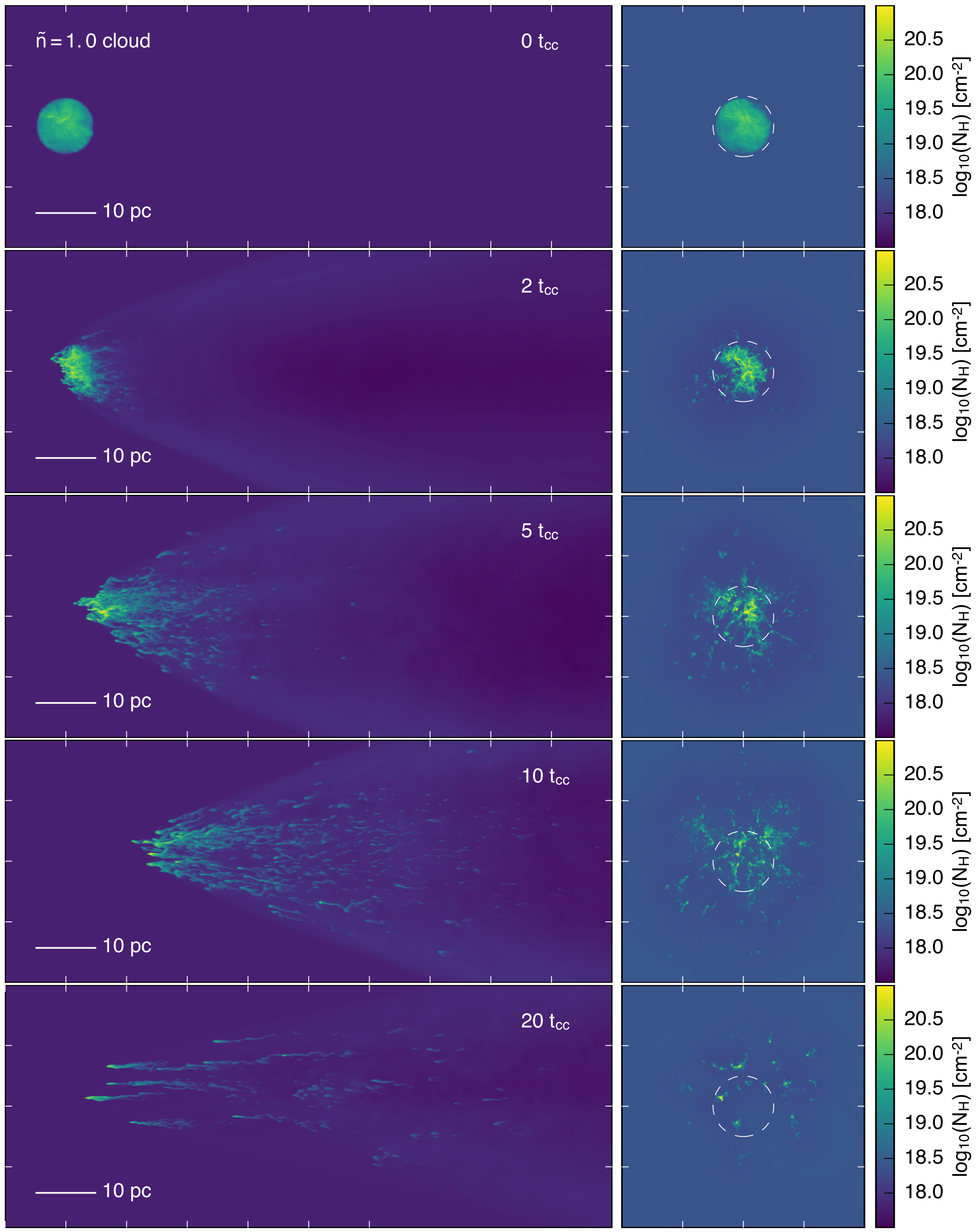

Figure 3.4: Time series evolution of our fiducial simulation, the $\tilde{n} = 1$ cm^{-3} turbulent cloud (model T1). Plots on the left show surface density projected along the y-axis, with the wind entering the box from the left, while the right column shows the surface density projected in the direction of the wind velocity. The scale of the axes ticks is 10 pc. Snapshots are shown at $t = 0$, 2, 5, 10, and 20 t_{cc}. The dashed circle in the right column shows the original extent of the cloud. Note: the y-projection at $t = 20$ t_{cc} has been shifted by 30 pc to re-center the cloud material.

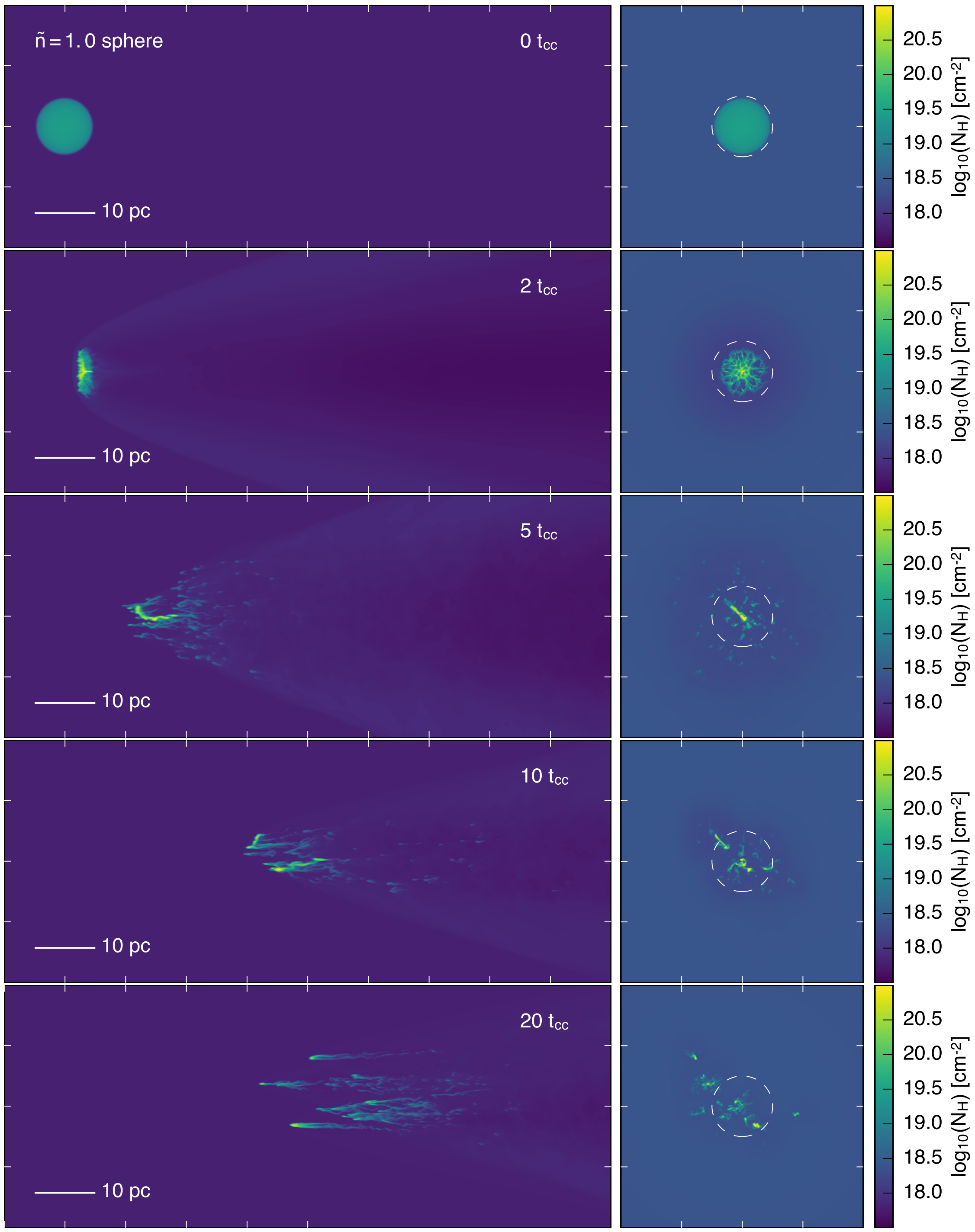

Figure 3.5: Time series evolution of the $\tilde{n} = 1$ cm^{-3} sphere simulation (model S1). As in Figure 3.4, the left column shows surface density projected along the y-axis, and the right column shows the surface density projected along the wind axis. Snapshots are shown at $t = 0$, 2, 5, 10, and 20 $t_{\rm cc}$. Note: the reference frame of the y-projection at $t = 20$ $t_{\rm cc}$ has been shifted by 40 pc to re-center the cloud.

at late times. We quantify this rapid mass loss in Figure 3.6, which shows the normalized cloud mass as a function of cloud crushing time. The mass is calculated using material at or above 1/3 the initial median density. By $t = 20\ t_{\rm cc}$, nearly 80% of the original mass of the turbulent cloud has been mixed into the hot wind and fallen below the density threshold of $n = 0.33$ cm^{-3}.

The loss of material proceeds more rapidly for turbulent clouds than for spheres, even with the cloud-crushing time calculated as a function of the median density. As can be seen in Figure 3.5, panel 3, after the initial pancake stage, the spherical cloud compresses into a single core, with a small surface area and high column density in the wind direction. This morphology is a result of the original shock moving in the wind direction combined with the compression from shocks on the sides of the cloud, which push material toward the center. This high density core presents a smaller surface area for ablation of material than the many small high density regions in the turbulent cloud (compare the third panels of Figures 3.4 and 3.5). The single bow shock at the front of the spherical cloud protects the core, while also resulting in a pressure gradient through the cloud. Cloud elongation in the direction of the wind gradually breaks up the original core into smaller fragments, which each have individual bow shocks and lose material. However, as demonstrated in Figure 3.6, the overall process is much slower for spheres than for turbulent clouds. Only $\sim$40% of the original spherical cloud material has been lost by $t = 20\ t_{\rm cc}$.

3.4.2 Median Density and Cloud Lifetimes

Figures 3.4 and 3.5 show the evolution of clouds with a median density of $\tilde{n} = 1$ cm^{-3}. We have also run four high resolution simulations with lower median densities. Models T05 and S05, the turbulent and spherical clouds with a median density of $\tilde{n} = 0.5$, show a mass and morphology evolution similar to their higher density counterparts. Figure 3.6 shows that the mass loss as a function of cloud crushing time is nearly identical for the $\tilde{n} = 0.5$ cm^{-3} and $\tilde{n} = 1$ cm^{-3} clouds out to 25 $t_{\rm cc}$. However, as the original median density continues to decrease, the evolution begins to follow a qualitatively different path. The original shock that passes through

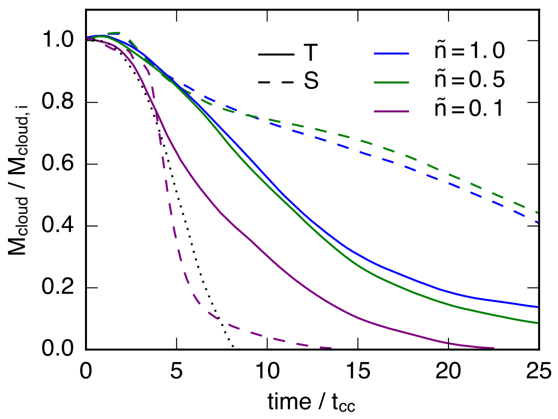

Figure 3.6: The mass evolution (cloud mass divided by initial cloud mass as a function of time) of each of our high resolution simulations. Cloud mass is calculated as the sum of all gas in the simulation with a density greater than 1/3 the initial median density. Tracks for turbulent clouds are shown as solid lines, while spherical clouds are shown with dashed lines. The mass-loss track for an adiabatic cloud simulation is also plotted (dotted line) for comparison.

the cloud does not compress the gas enough for it to reach densities where it can cool efficiently. The warm cloud gas quickly gets rarified and accelerated with the hot wind, causing the mass-loss of the clouds to proceed much more rapidly at lower original median densities. This difference is clearly visible in the $\tilde{n} = 0.1$ evolutionary tracks in Figure 3.6.

In model S01, almost none of the gas reaches the requisite density to cool. In fact, only a small ring of gas within the cloud reaches the threshold density. This ring is visible in Figure 3.7, which shows surface density projections of the $\tilde{n} = 0.1$ sphere simulation at 2 t_{cc}. The ring structure is caused by the shock moving in the wind direction colliding with the oblique shocks coming in from the sides of the cloud. While the densities in this collision do not get large enough to result in a single compact core as seen in the higher density models, this small amount of dense gas is enough to cause the extended tail seen in the mass evolution track of Figure 3.6. However, most of the cloud is destroyed in less than 5 t_{cc}, consistent with adiabatic simulations of cloud-shock interactions (Schneider and Robertson, 2015).

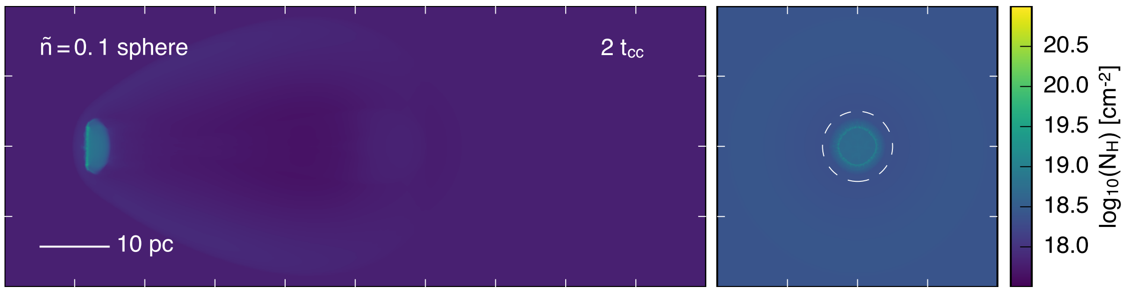

Figure 3.7: Surface density projections of the $\tilde{n} = 0.1$ sphere simulation at $t = 2$ $t_{\rm cc}$. The left panel shows a y-projection and the right panel shows an x-projection, as in Figures 3.4 and 3.5. A small ring of high surface density material is formed by the collision of shocks within the cloud, but most of the cloud remains at densities too low to cool effectively. The dashed circle in the right panel shows the original extent of the cloud.

The evolution of the $\tilde{n} = 0.1$ cm^{-3} turbulent cloud (model T01) is less dramatically different as compared to its higher density counterparts. Although the median cloud density is below that required for effective cooling, the lognormal density distribution of initial cloud material spans a range up to $n \approx 10$ cm^{-3}. As a result, parts of the cloud that were initially above the median density are still able to cool effectively after being shocked. As Figure 3.6 shows, the post-shock mass loss for the $\tilde{n} = 0.1$ turbulent cloud proceeds faster than the $\tilde{n} = 1$ or $\tilde{n} = 0.5$ clouds, but more slowly than the $\tilde{n} = 0.1$ sphere. We expect that the speed of mass loss for turbulent clouds would continue to increase as the initial median density is lowered, until eventually the evolution proceeded on an adiabatic track (shown by the dotted black line in Figure 3.6.

3.5 Phase Structure of the Wind

In this section, we investigate the physical state of the gas in our simulations. The typical temperature of gas of a given density is useful in interpreting the cloud evolution. The density threshold for rapid cooling is also an important feature that determines whether any dense gas will survive for many cloud-crushing times. The high resolution of our simulations across the entire volume enables this study of the

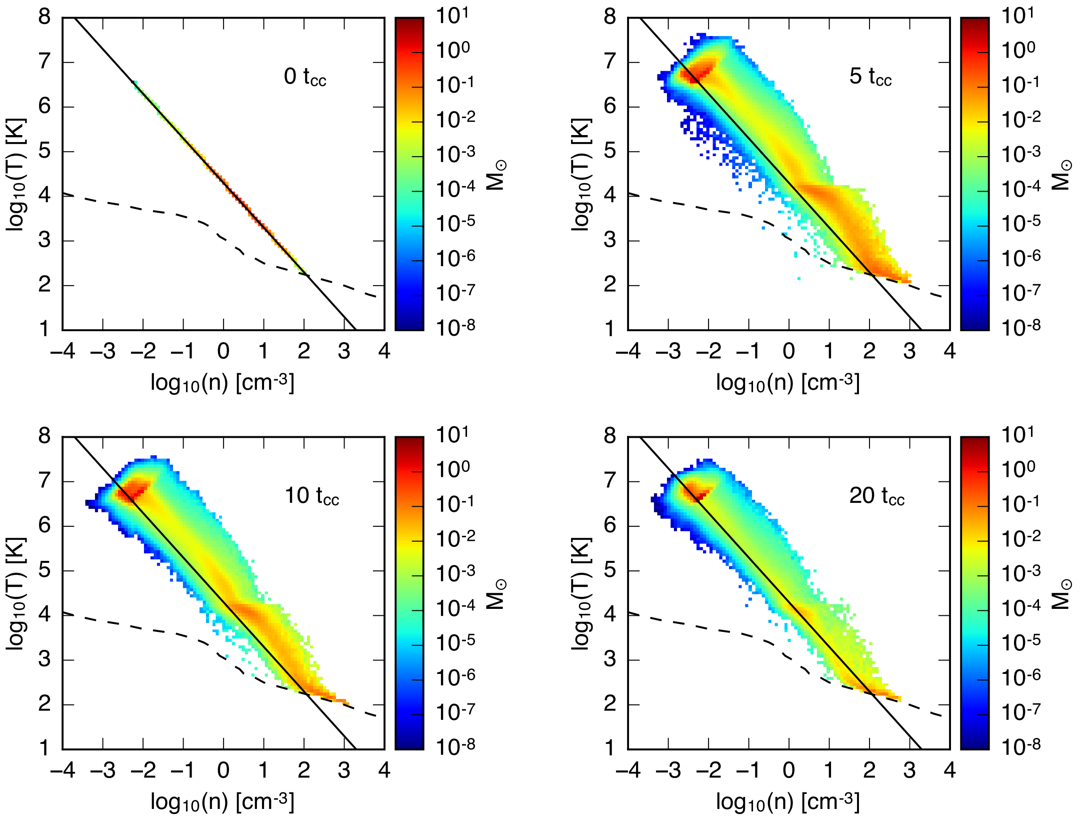

Figure 3.8: Mass-weighted density-temperature phase diagrams at $t = 0$, 5, 10, and 20 $t_{\rm cc}$ for the $\tilde{n} = 1$ cloud simulation (model T1). The initial equilibrium pressure is plotted as a solid line, and the temperature at which heating equals cooling as a function of density is plotted as a dotted line.

detailed phase structure of the gas as it evolves.

3.5.1 Density and Temperature Structure

Figure 3.8 shows density-temperature phase diagrams for the $\tilde{n} = 1$ turbulent cloud at 4 different times in its evolution - the initial conditions, and $t = 5$, 10, and 20 $t_{\rm cc}$. Each bin is colored according to the total mass it contains, to better focus attention on the locations with the majority of the cloud material. Throughout the simulation, most of the mass is in the hot wind, visible as the red region at the upper left of the distribution in each panel (and as a single bin in the initial conditions). At $t = 0$ $t_{\rm cc}$, the cloud's mass is distributed across a range of densities and temperatures, with most of the mass in regions at or above the median density of $\tilde{n} = 1$ cm^{-3}. Each panel also contains an isobar showing the original pressure of the gas (recall that the cloud's pressure is matched to the wind in the initial conditions), as well as a dashed line that shows the equilibrium location between heating and cooling in our simulations.

After the cloud has been shocked, the density-temperature phase diagram takes on a characteristic shape. At early times, most of the gas is at higher pressure than the initial conditions. As the simulation proceeds, the cloud gas slowly evolves back toward thermal pressure equilibrium with the incoming wind. Cloud material quickly fills out the entire range of densities between the original cloud density and the wind. A large amount of mass has been compressed to high densities, much of it over an order of magnitude higher than the initial densities. The high density material cools effectively, with much of the cloud mass located just above the equilibrium cooling temperature, at the lower right in each panel. There also exists a mass concentration around $\log(n) = 0.5$ and $\sim 2 \times 10^4$ K. This buildup reflects the shape of the cooling curve, with maximally efficient cooling around 10^5 K and a steep falloff around 10^4 K. These features remain throughout the subsequent evolution, as the mass in high density bins is gradually reduced.

The phase diagrams for simulations T01 and S01 yield an estimate of the density threshold required for efficient cloud cooling in our simulations. Figure 3.9 shows the mass-weighted density-temperature phase diagrams for both simulations at $t = 2$ $t_{\rm cc}$. The turbulent cloud, displayed in the left panel, has a large amount of mass in the cooling sweet spot just above $\log_{10}(n) = 0$. In contrast, most of the shocked sphere material never crosses this density threshold, and as a result, the gas remains at high temperatures and low densities where it quickly mixes with the hot wind. Animations of the phase diagrams show clearly the path that the cool gas takes in the turbulent cloud simulation. Cloud gas originally above the median density shocks to densities just above $\log_{10}(n) = 0$, after which the gas begins to cool rapidly down to 10^4 K. Only a small fraction of the sphere material reaches densities of $\log_{10}(n) = 0$, and as a result, most of the mass of the cloud is lost at early times.

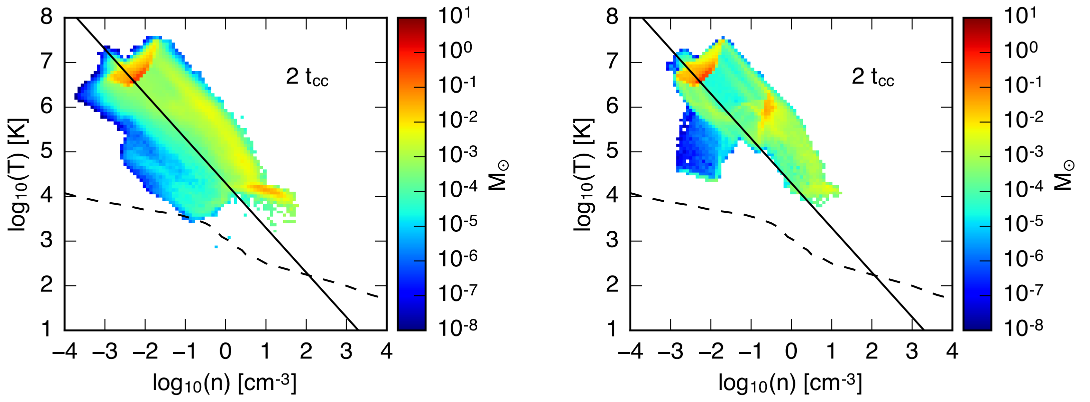

Figure 3.9: Mass-weighted density-temperature phase diagrams for the $\tilde{n} = 0.1$ turbulent (left) and spherical (right) cloud simulations at 2 $t_{\rm cc}$. While much of the mass in the turbulent cloud has crossed the $\log(n) = 0$ threshold and is able to cool efficiently, much of the mass of the spherical cloud remains below this density. The low-density gas is unable to cool and is quickly mixed into the hot wind.

3.6 Momentum Coupling

While the phase diagrams of the cloud gas are enlightening, in this section we seek to quantify several less obvious aspects of the cloud-wind interaction. The first area we address is cool cloud entrainment - the acceleration of cool material by the wind. We investigate entrainment in our simulations by studying 2D density-velocity histograms, to determine the typical velocities attained by gas of a given density. We follow our discussion of entrainment with an investigation of the detailed coupling of momentum between the hot wind and the cool cloud material. We assess how cloud mass transitions from one phase to another, and how the momentum in different phases changes over time. As in previous sections, we will focus on the fiducial $\tilde{n} = 1$ cm^{-3} turbulent cloud simulation.

3.6.1 Cool Cloud Entrainment

While cool clouds have been observed in galactic outflows at a variety of distances and velocities, the primary physical mechanism responsible for fast-moving cool gas continues to be debated. In this section, we investigate the velocity of the cool gas

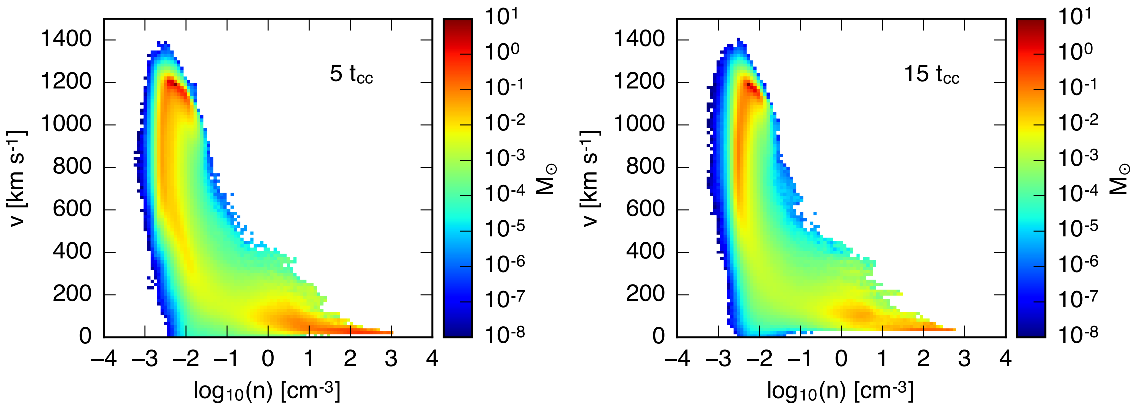

Figure 3.10: Mass-weighted density-velocity phase diagrams at 5 and 15 $t_{\rm cc}$ for the $\tilde{n} = 1$ cm^{-3} turbulent cloud simulation. High density material is not effectively accelerated in turbulent clouds.

in our simulations, and show that in the purely hydrodynamic case, hot winds are unlikely to accelerate cool gas to the high velocities seen in many outflows.

Figure 3.10 shows mass-weighted density-velocity histograms for model T1. The figure shows two representative snapshots, at $t = 5$ and $t = 15$ $t_{\rm cc}$, with the velocity in the wind direction plotted on the y-axis. As in the density-temperature diagrams, the hot wind appears as a mass concentration at the upper left in each panel, with a small spread around the initial wind density and velocity. While cloud material has clearly acquired a range of densities, only low density material travels at high speeds. Only 3% of the cloud mass above the original median density of $\tilde{n} = 1$ cm^{-3} is moving faster than 200 km s^{-1} by 15 $t_{\rm cc}$. The average velocity of the dense material ($n > 1$ cm^{-3}) is only $v_x \approx 120$ km s^{-1}.

One of the goals of this work was to investigate how different cloud density structures change the effectiveness of the hot wind in accelerating high density material. This question is addressed in Figure 3.11, which compares the density-velocity histograms for the $\tilde{n} = 1$ cm^{-3} turbulent cloud at $t = 10$ $t_{\rm cc}$ to the sphere. This figure indicates accelerating high density material proves more difficult in the turbulent cloud case. This difference results from the different initial column densities of the cloud material, as we describe below.

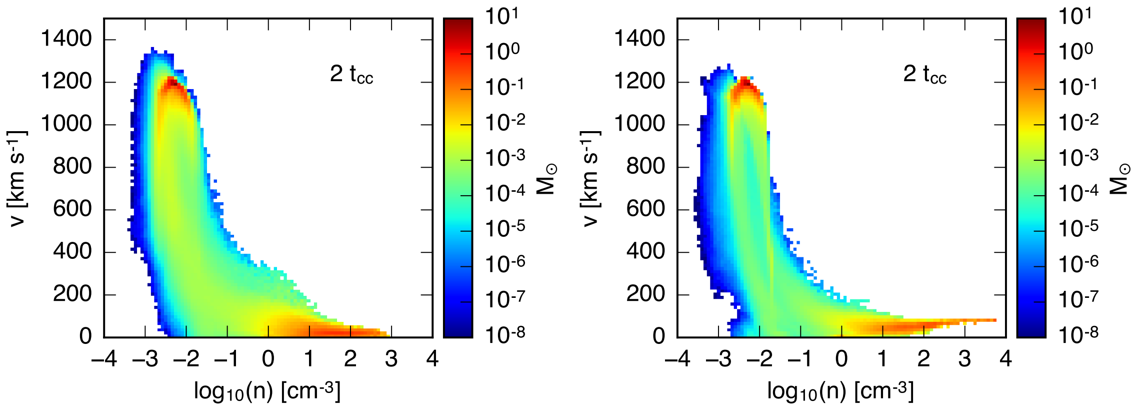

Figure 3.11: Mass-weighted density-velocity phase diagrams for the $\tilde{n} = 1$ cloud (left) and sphere (right) at 2 $t_{\rm cc}$. Acceleration of dense gas is much more effective for spherical clouds, which has lower initial column density than the dense regions of turbulent clouds with the same initial median density.

In a simple model, the ram pressure of the hot wind will accelerate cold cloud material (we neglect other forces in the following analysis). The ram pressure of the hot wind scales as

$$P_{\rm ram} = n_{\rm wind} v_{\rm wind}^2, \tag{3.4}$$

which gives a constant ram pressure per unit mass of $P_{\rm ram}/m_{\rm H} = 7.2 \times 10^{13}\ {\rm cm^{-1}\,s^{-2}}$ for our background wind conditions. The associated acceleration of the cloud, $g_{\rm cl}$, will then be

$$g_{\rm cl} = \frac{P_{\rm ram}}{m_{\rm H}} \frac{m_{\rm H}}{\Sigma_{\rm cl}} = \frac{P_{\rm ram}}{m_{\rm H}} \frac{1}{N_{\rm H}}, \tag{3.5}$$

where $\Sigma_{\rm cl}$ is the average surface density of the cloud, and $N_{\rm H} = n_{\rm cl} L$ is the column density of a particular region of the cloud. If the cloud is a constant density sphere, then L can be approximated as twice the cloud radius. For the $\tilde{n} = 1\ {\rm cm^{-3}}$ spherical cloud with a radius of $r \approx 5$ pc, the column density is approximately $N_{\rm H} \approx 3 \times 10^{19}\,{\rm cm^{-2}}$ across the whole cloud, and the resulting acceleration is

$$g_{\rm cl} \sim 2.3 \times 10^{-6}\,{\rm cm\,s^{-2}} \left[\left(\frac{n_{\rm cl}}{1{\rm cm^{-3}}} \right) \left(\frac{L}{10\,{\rm pc}} \right) \right]^{-1}, \tag{3.6}$$

or

$$g_{\rm cl} \sim 0.75\,{\rm km\,s^{-1}\,kyr^{-1}} \left[\left(\frac{n_{\rm cl}}{1\,{\rm cm^{-3}}} \right) \left(\frac{L}{10\,{\rm pc}} \right) \right]^{-1}. \tag{3.7}$$

The cloud crushing time for the $\tilde{n} = 1$ clouds is $t_{\rm cc} \approx 56.4$ kyr. The acceleration in Equation 3.7 gives a cloud velocity of $v_{\rm cl} \sim 85\,{\rm km\,s^{-1}}$ after 2 $t_{\rm cc}$ (113 kyr), which is roughly consistent with our results for the velocities of the densest gas after 2 $t_{\rm cc}$, shown in the right panel of Figure 3.11. In fact, the average velocity of gas denser than $n = 500$ cm^{-3} at 2 $t_{\rm cc}$ for the $\tilde{n} = 1$ sphere is $77\,{\rm km\,s^{-1}}$. After the cloud has been crushed Equation 3.7 is no longer an adequate estimate of the cloud acceleration, because the column densities have increased by over an order of magnitude (compare the first and second panels in Figure 3.5).

In contrast, at $t = 0$ the densest sight lines through the $\tilde{n} = 1$ cm^{-3} turbulent cloud have column densities as high as $N_{\rm H} = 2 \times 10^{20}\,{\rm cm^{-2}}$, almost an order of magnitude larger than the sphere (see the first panel in Figure 3.4). At these column densities, the cloud acceleration is only $g_{\rm cl} \approx 0.11\,{\rm km\,s^{-2}\,kyr^{-1}}$, yielding a dense gas velocity of $v_{\rm cl} \sim 12\,{\rm km\,s^{-1}}$ at $t = 2$ $t_{\rm cc}$. As a result, the regions of high column density are accelerated less in the turbulent cloud, and the high density gas at $t = 2$ $t_{\rm cc}$ is traveling considerably slower than in the spherical cloud case. The average velocity of gas with $n > 500$ cm^{-3} is only $v \approx 16\,{\rm km\,s^{-1}}$ at $t = 2$ $t_{\rm cc}$.

In both the turbulent cloud and sphere simulations, much less acceleration occurs at late times than predicted by Equation 3.7 (see right panel, Figure 3.10). As mentioned previously, this minimal acceleration can be explained by the increased column densities that result from the cloud crushing. In both the turbulent and spherical clouds, compression towards the center caused by the initial shock results in column densities at $t = 2$ $t_{\rm cc}$ that are an order of magnitude higher than the original values (compare the first and second panels in both Figure 3.4 and 3.5). Thus, we conclude that entrainment of cool material to high velocities in a hot wind remains very inefficient under these circumstances, and the problem only compounds for clouds with more realistic shapes and density distributions.

These simulations also show that the average surface density of the cloud, $\Sigma_{\rm cl} = M_{\rm cl}/\pi R_{\rm cl}^2$, as given in Table 3.1 fails to adequately predict how much dense gas will be accelerated. This fact can be demonstrated by comparing the dense gas acceleration for the $\tilde{n} = 0.5$ cm^{-3} turbulent cloud (not shown) and the $\tilde{n} = 1$ cm^{-3}

sphere. The $\tilde{n} = 0.5$ cm^{-3} turbulent cloud originally has a lower average surface density than the $\tilde{n} = 1$ sphere - $\Sigma_{\rm cl} = 0.11$ and $\Sigma_{\rm cl} = 0.13$ M$_\odot$ pc^{-2}, respectively. However, the spatial coherence of the densest regions within the turbulent cloud leads to individual column densities several times higher than the average column density of the sphere, around $N_{\rm H} \approx 10^{20}\,{\rm cm}^{-2}$ as compared to $N_{\rm H} = 3 \times 10^{19}\,{\rm cm}^{-2}$. As a result, the average velocity of densest material in the $\tilde{n} = 0.5$ cm^{-3} turbulent cloud simulation is $v_x \approx 30$ km s^{-1}, less than half the speed of the dense gas in the $\tilde{n} = 1$ cm^{-3} spherical cloud simulation.

3.6.2 Integrated Mass and Momentum

Having established that in our simulations momentum does not transfer efficiently enough to accelerate the dense phase to the wind velocity, we would like to better quantify the momentum gained by other phases of the outflow. Figure 3.6 showed the evolution of cloud mass for each of our high resolution simulations as an integrated quantity, the total sum of material above a given density threshold. To better understand the way that gas evolves within the cloud, we now divide this evolution into multiple density bins. Figure 3.12 shows this binned mass evolution plot for the $\tilde{n} = 1$ cloud. The black line matches the one displayed in Figure 3.6, comprising all the material above a density threshold of 1/3 $\tilde{n}$ but no longer normalized by the initial cloud mass. Colored lines in Figure 3.12 show the evolution of total cloud mass in four density bins: low, from $0.02 < n < 0.2$ cm^{-3}; medium, from $0.2 < n < 2.0$ cm^{-3}; high, from $2.0 < n < 20$ cm^{-3}, and very high, $n > 20$ cm^{-3}. The other three panels show 1D histograms of the mass as a function of density. Integrating any of the colored regions in the histogram yields the value represented as a circle on the line of the same color in the top left panel.

Much of the evolution in Figure 3.12 takes place early on, with the most drastic shift occurring before 2 $t_{\rm cc}$ as the cloud is crushed. Initially, much of the cloud mass (12 $M_\odot$) is in the high density bin, $2.0 < n < 20$ cm^{-3}, with a significant amount (5.9 $M_\odot$) also in the medium density bin that surrounds the initial median density of $n = 1$ cm^{-3}. Only 0.64 $M_\odot$ is at $n > 20.0$ cm^{-3} initially. As the cloud is crushed,

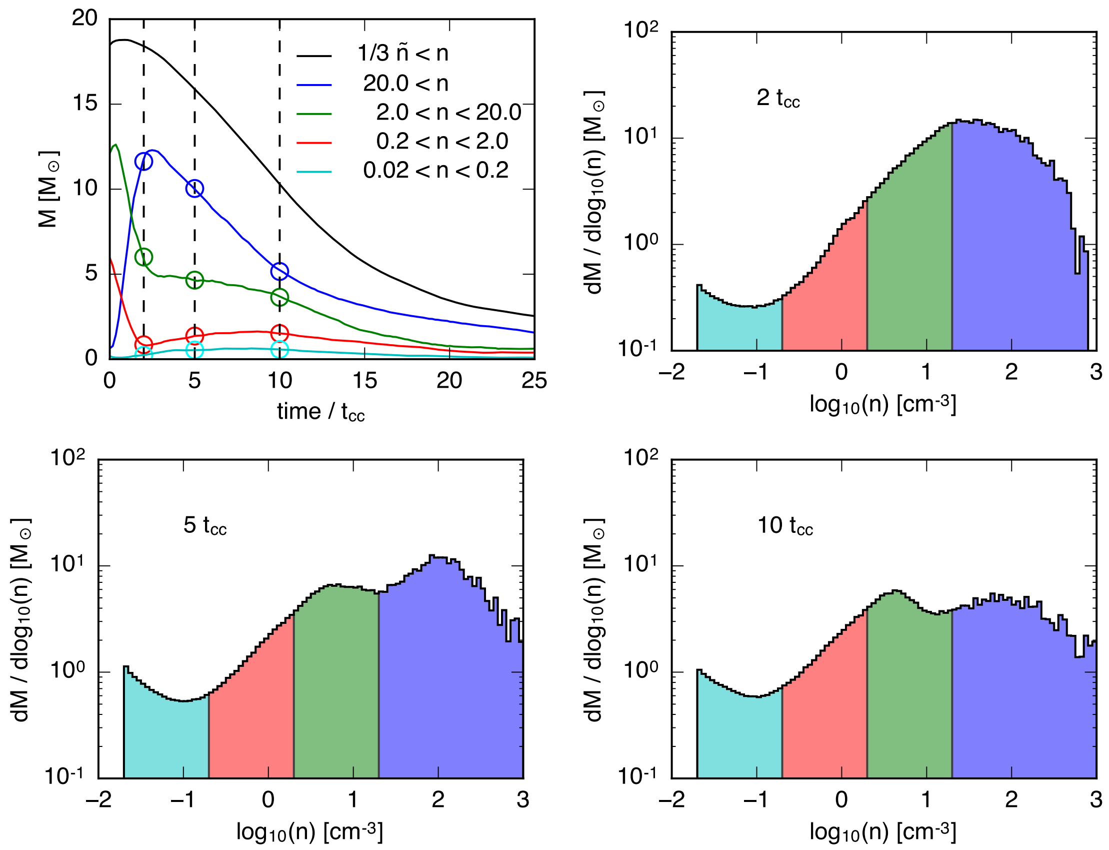

Figure 3.12: Evolution of cloud mass in different density bins for the $\tilde{n} = 1$ cm^{-3} turbulent cloud. In the upper left panel we show the total mass in different density bins; colors of the lines in the upper left panel correspond to density bins in the 1D histograms shown in the other three panels. Circles in the first panel correspond to the times displayed in the other three panels. The 1D histograms are normalized such that the integral over each density bin yields the total mass indicated by the circles.

almost all of the material increases by about an order of magnitude in density, such that at 2 $t_{\rm cc}$ the medium density bin has only 0.85 $M_\odot$, the high density bin has 6.0 $M_\odot$, and 11.6 $M_\odot$ is in the very high density bin. These values are highlighted by circles at 2 $t_{\rm cc}$ in the first panel of Figure 3.12. Very little mass is in the low density bin at this time, and no cloud mass has been lost.

After 2 $t_{\rm cc}$, the evolution proceeds more gradually, with material slowly moving from higher density bins to lower ones. At 5 $t_{\rm cc}$ mass is concentrated in the same locations evident in the density-temperature phase diagram - there is a significant amount of mass (10.0 $M_\odot$) at very high densities, as well as a bump around $\log_{10}(n) = 0.5$ as a result of the shape of the cooling curve. These features are still present at 10 $t_{\rm cc}$, though the mass in the highest density bin has been substantially reduced. At all times in our simulation after 2 $t_{\rm cc}$, the majority of the cloud mass is in the highest density bin. Eventually we expect this mass to shift to lower density bins as the last of the high density gas is destroyed.

In the same way that we have integrated the total cloud mass, we can integrate total cloud momentum in the simulations. Figure 3.13 shows how the total momentum is divided amongst the same four density bins. The top left panel shows the evolution of the total momentum in the wind direction, computed as

$$p_{\rm tot} = \sum_{n=n_{\rm low}}^{n=n_{\rm high}} M_i v_{x,i} \tag{3.8}$$

for each cell, i, in the simulation, where M_i is the integrated mass in that cell and v_x is the direction of the wind. The sum is taken over the same density ranges shown in Figure 3.12. In the other three panels, we show histograms of the momentum as a function of density at $t = 2$, 5, and 10 $t_{\rm cc}$. Integrals of the histograms are again shown as circles on the colored lines in the top left panel. At $t = 2\ t_{\rm cc}$ the momentum is distributed in a similar way to the mass, with most of the momentum in the two highest density bins. At $t = 5\ t_{\rm cc}$, however, the distribution of total momentum has shifted. The three lower density bins have continued to gain momentum, but the highest density bin actually loses some, despite the fact that the highest density bin still contains the majority of the cloud mass. At this point, the total momentum in

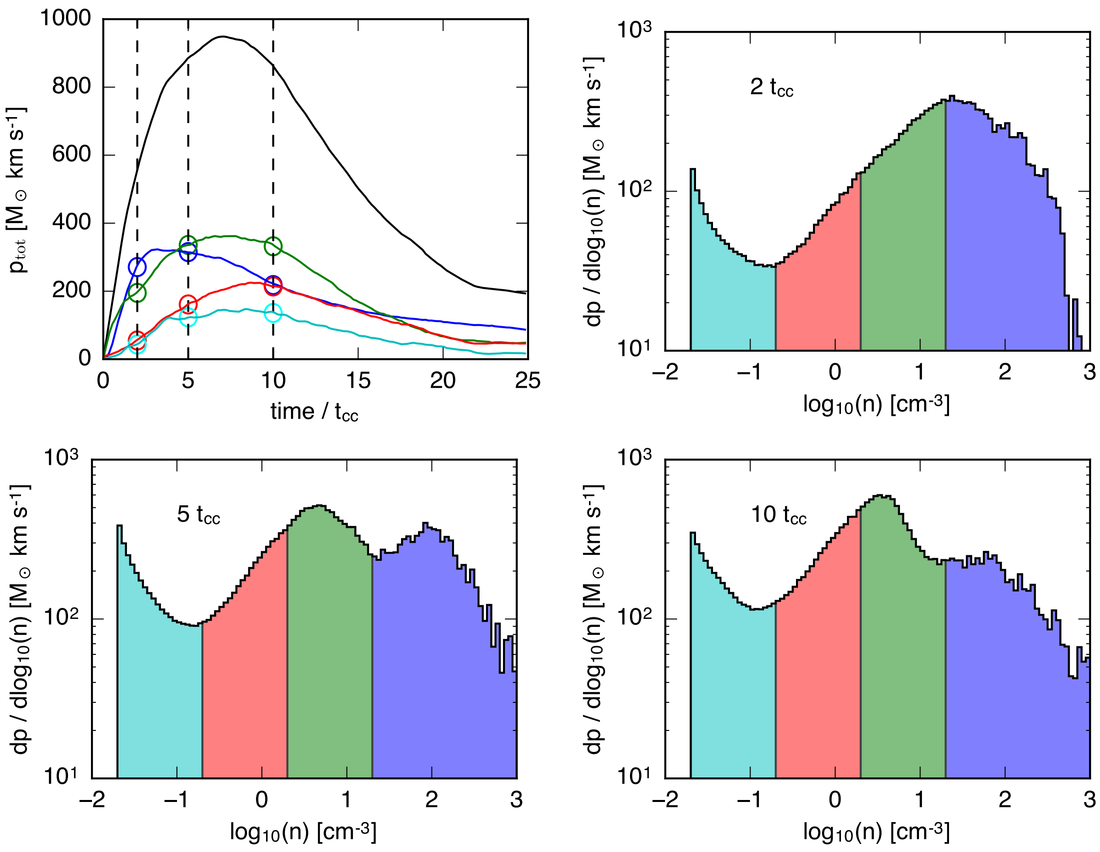

Figure 3.13: Evolution of total momentum in the wind direction in different density bins for the $\tilde{n} = 1$ turbulent cloud. The first panel shows the total momentum as a function of cloud crushing time, calculated by summing Mv_x over all cells with a density in the given range (see Eqn 3.8). Colors represent the same density bins shown in Figure 3.12. Other panels show 1D histograms of momentum as a function of density at 2, 5, and 10 t_{cc} . Integrating under the histograms gives the values shown as circles in the first panel.

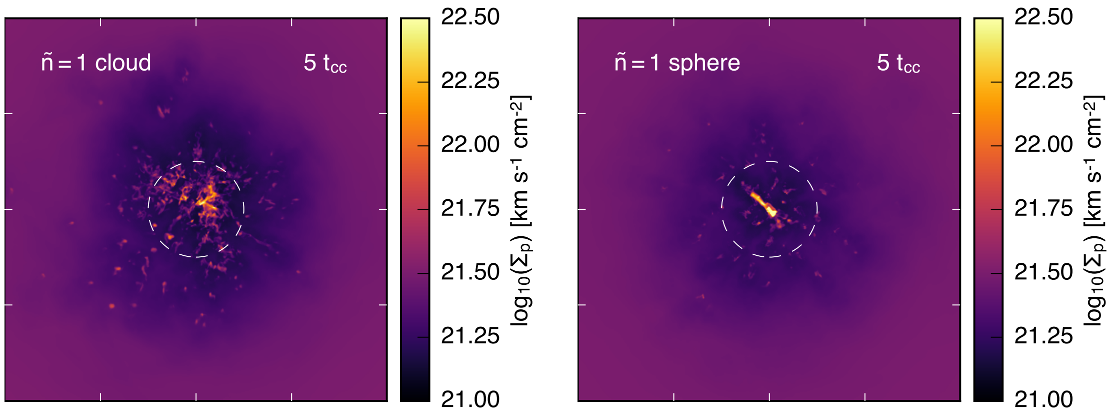

Figure 3.14: Comparison of the momentum column density ($\Sigma_p = nv_x$ integrated in the wind direction) for the $\tilde{n} = 1$ turbulent and spherical cloud simulations at 5 $t_{\rm cc}$. The scale of the axes ticks is 10 pc. White dashed circles indicate the original extent of the cloud. Much higher momentum column densities are visible for the spherical cloud, which has been crushed to a single high density core.

the medium and high density bins ($0.2\,{\rm cm}^{-3} < n < 20\,{\rm cm}^{-3}$) is $p_{\rm tot} = 495\ M_\odot$ km s^{-1}, 50% more than is in the highest density bin ($n > 20\,{\rm cm}^{-3}$), $p_{\rm tot} = 315\ M_\odot$ km s^{-1}.

This shift indicates that momentum is transferred efficiently from the hot wind to lower density gas. Gas with densities between $1\,{\rm cm}^{-3} < n < 10\,{\rm cm}^{-3}$ is in a relatively long-lived phase, giving it more time to gain momentum from the hot wind. The total momentum in the warm phase likely also increases as gas from higher density bins moves to lower density. Gas in the highest density bin with the most momentum may be the most likely to decrease in density. Without tracer particles, this shift is difficult to quantify in our simulations, but the highest density bin clearly loses mass at every time $t > 2\ t_{\rm cc}$ and at least some high density material with significant momentum shifts to the lower density bins. Overall, the distribution of total momentum across the different gas phases in the cloud is remarkably equal. The density-velocity histograms shown in Figure 3.10 also indicate this equality, showing that lower density material moves more quickly than higher density material. At very late times, $t > 17\ t_{\rm cc}$ the highest density bin again has the most total momentum - this reflects the lack of mass in the lower bins by this time.

We can also compare the distribution of momentum between the turbulent and spherical clouds by looking at the integrated momentum in the wind direction, a "momentum surface density":

$$\Sigma_p = \sum_{i=0}^{i=N_x} n_i v_{x,i}, \tag{3.9}$$

where N_x is the total number of cells in the wind direction, n_i is the number density of the gas in cell i, and $v_{x,i}$ is the velocity in the wind direction in cell i. We show projections of this quantity for the $\tilde{n} = 1$ turbulent cloud and sphere at $t = 2\ t_{\rm cc}$ in Figure 3.14. As one would expect given the differences in the density-velocity phase diagrams, the momentum surface density is much higher for the sphere, reaching a maximum of 2.4×10^{23} km s^{-1} cm^{-2} in the highest column. The momentum surface density of the background wind is 3.1×10^{21} km s^{-1} cm^{-2}. In contrast, the momentum has a larger spatial extent and is spread over a wider range of gas column densities in the turbulent cloud, with the maximum column of 4.3×10^{22} km s^{-1} cm^{-2}. Thus, even though the average momentum surface density at this time is similar between the two simulations, the momentum is clearly much more concentrated in the high density gas in the spherical cloud simulation. This result is consistent with the total momentum being shared across every density bin in Figure 3.13.

3.7 Resolution Effects

At our fiducial resolution of 64 cells/$R_{\rm cl}$, we expect the global properties of cloud evolution to be qualitatively correct (Gregori et al., 2000; Poludnenko et al., 2002; Melioli et al., 2005). These properties include the morphologies seen in the cloud destruction in Section 3.4, the general features seen in the phase diagrams in Section 3.5, and the multiphase distribution of momentum demonstrated in Section 3.6.2. However, many authors have suggested that a higher resolution of at least 128 cells/$R_{\rm cl}$ is required for convergence of quantitative measurements of properties like the mass loss rate and destruction time (Klein et al., 1994b; Nakamura et al., 2006b; Scannapieco and Brüggen, 2015). The effects of resolution may be especially important for turbulent cloud simulations, where increasingly dense structures are captured as the resolution of the simulation is increased. Therefore, we have carried out a numerical study to test the dependence of our results on the resolution of our simulations. We emphasize that this is *not* a convergence study in the classical sense, because the initial conditions of the cloud change with resolution (as described below).

The study compares three versions of the $\tilde{n} = 0.5$ turbulent cloud simulation: an ultra high resolution simulation run with 128 cells/$R_{\rm cl}$ (hereafter R_{128}), the production version with 64 cells/$R_{\rm cl}$ (R_{64}), and a low resolution version with 32 cells/$R_{\rm cl}$ (R_{32}). Because of the large computational expense of the R_{128} simulation, we use a physical box of size $80 \times 30 \times 30$ pc, as compared to the production simulations which ran in boxes of size $160 \times 40 \times 40$ pc. The R_{128} simulation volume contains $2048 \times 784 \times 784$ cells, which yields a resolution of $\Delta x = 0.039$ pc. The reduced time step required by the Courant condition and the computational expense of the additional cells mean that we can only afford to run the simulation for $\sim 12\ t_{\rm cc}$. In practice, using the shorter box length results in the material leaving the computational domain at later times. The R_{32} simulation is run in a box with the same physical size as the production simulation.

Each of the three simulations uses a cloud drawn from the same region of a Mach 5 isothermal turbulence simulation (Robertson and Goldreich, 2012). To create the lower resolution clouds, the highest resolution simulation was resampled using a cubic spline interpolation. The bulk physical properties of the initial conditions are identical, including the median number density and total mass. As the resolution increases, we allow the cloud initial conditions to include progressively more small-scale structures and higher density regions. The initial conditions of the higher resolution simulations therefore do not simply reflect a better resolved version of low-resolution run initial conditions. The comparison presented below does not aim to act as convergence study, but instead attempts to capture how increasingly smaller scale features of the cloud initial conditions might influence the evolution of

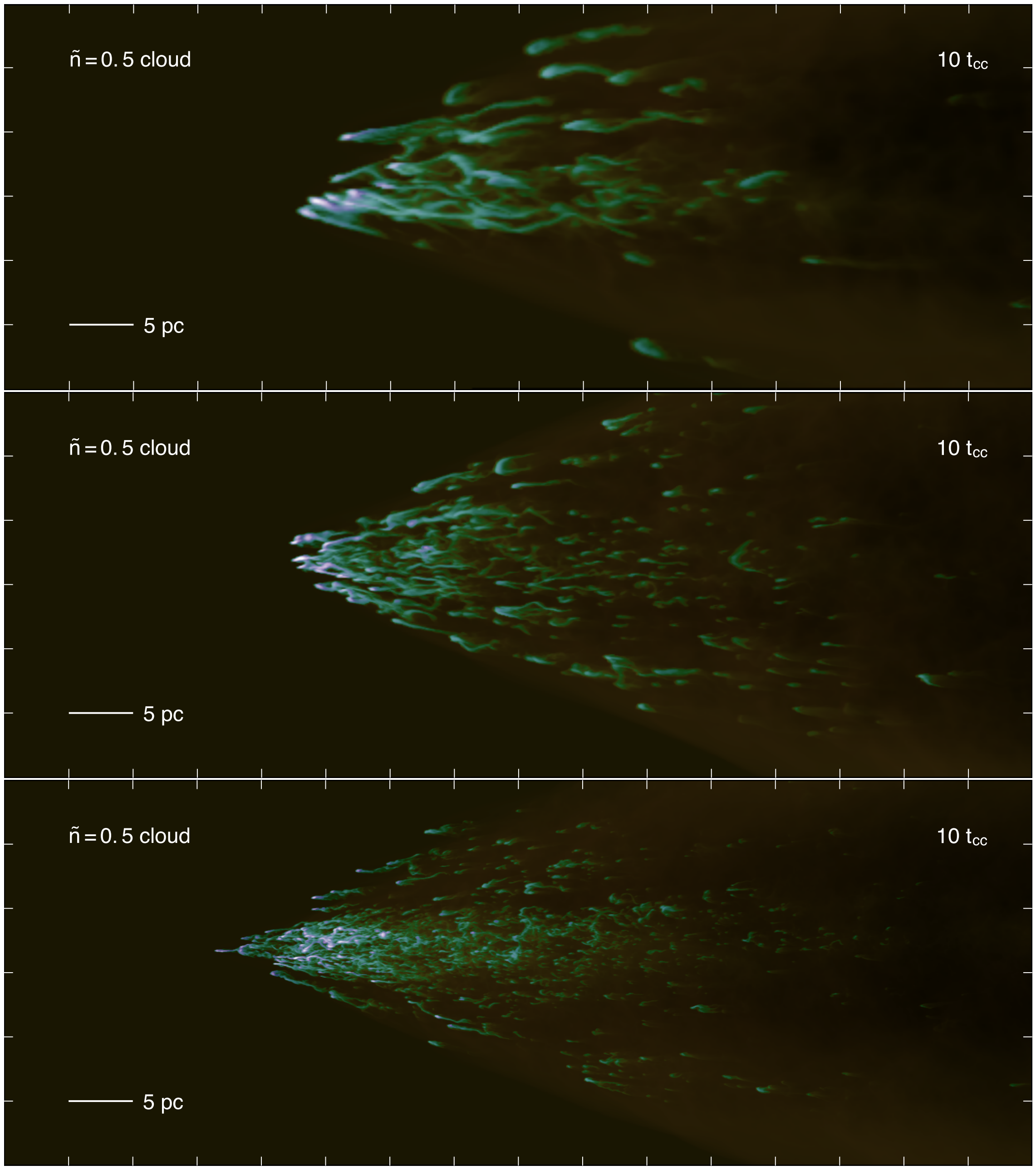

Figure 3.15: A comparison at 10 $t_{\rm cc}$ between $\tilde{n} = 0.5$ turbulent cloud simulations with three different resolutions: 32 cells/$R_{\rm cl}$, 64 cells/$R_{\rm cl}$, and 128 cells/$R_{\rm cl}$. The intensity in each panel corresponds to the projected number density, and color reflects the temperature of the gas (purple is cold, green is warm, and red is hot). As the resolution of the simulations is increased, the high-density features are resolved into smaller structures. As a result, the densest gas in the cloud is accelerated less efficiently with increasing resolution.

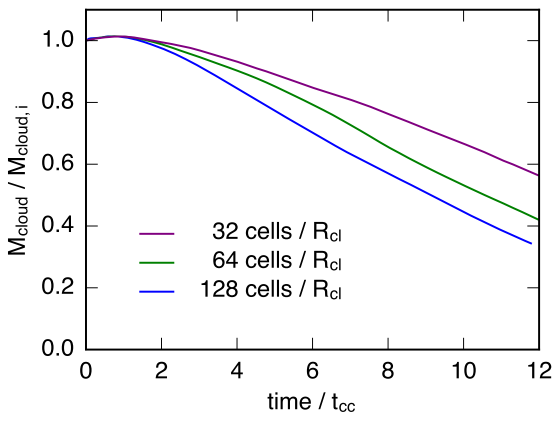

Figure 3.16: Cloud mass is displayed as a function of time for the three $\tilde{n} = 0.5$ turbulent clouds in the resolution study. The resolution of each simulation in terms of cells / cloud radius is displayed in the lower left. Mass loss initially proceeds more quickly for the higher resolution simulations.

the wind-cloud system.

Figure 3.15 shows snapshots of the three simulations at 10 t_{cc}, with the simulation time limited by the expense of the R_{128} run. The intensity of the image in each panel scales logarithmically with the projected number density, and the color reflects the gas temperature. The R_{32} and R_{64} simulations have been cropped to display the same region as the R_{128} simulation, for which the full box is shown. As expected, with each increase in resolution, finer-scale structure emerges. While in the R_{32} simulation only ~ 10 cloudlets form, the increased resolution and the more detailed initial conditions of the R_{128} simulation result in far more. At low resolutions the wind has successfully pushed the dense gas further, as evidenced by the bulk of the cloud material shifting further to the right in the upper panels. In the R_{128} simulation, some of the dense gas has travelled less than $2R_{\text{cl}}$ in 10 t_{cc}.

For a more quantitative measure of the difference between these simulations, we plot in Figure 3.16 the mass evolution of each cloud. Even at our highest resolution, the results have not yet converged. Figure 3.16 shows a general trend toward more efficient mass loss as the resolution is increased. To better understand this trend, we examine the mass evolution in the separate density bins used in Section 3.6.2. The resulting mass-loss curves for the R_{128} and R_{64} simulations are shown in Figure 3.17.

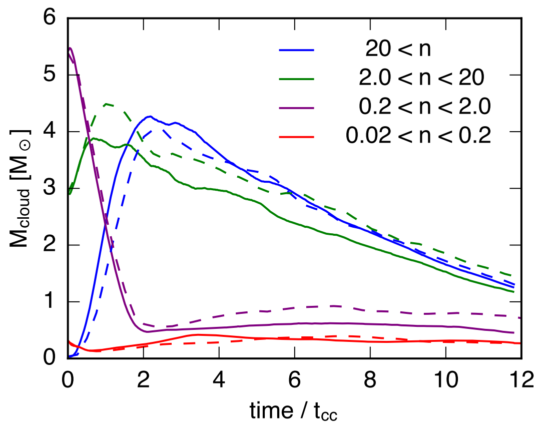

Figure 3.17: The evolution of cloud mass in four density bins for the R_{128} (dashed) and R_{64} (solid) simulations. The decreased mass in the $2\,\mathrm{cm}^{-3} < n < 20\,\mathrm{cm}^{-3}$ density bin results in faster mass loss at early times for the higher resolution simulation, as the lower density cloud gas is quickly accelerated by the wind.

At early times, the primary difference between the two simulations appears in the two highest mass bins. The R_{128} gains slightly more mass in the highest density bin ($n > 20\,\mathrm{cm}^{-3}$), but contains significantly less mass in the bin with $2\,\mathrm{cm}^{-3} < n < 20\,\mathrm{cm}^{-3}$ than the R_{64} simulation.

At early times ($t < 2\,t_{\mathrm{cc}}$) the R_{128} simulation generates less high density gas ($n > 1$ cm^{-3}) - the density threshold required for efficient cooling. A higher fraction of the cloud gas in the R_{128} simulation resides at densities and temperatures susceptible to quick destruction by the hot wind. This process of incorporation into the hot wind happens quickly, as evidenced by the lack of mass in the low density, slow ($v < 200$ km s^{-1}) region in the density-velocity diagrams (see Figure 3.10). Because cloud gas does not spend much time in this region of phase-space, the low density curves in Figure 3.17 look similar. At later times, mass-loss proceeds similarly for the high resolution simulations (compare the slope of the blue and green curves after ~ 5 t_{cc} in Figure 3.16).

3.8 Discussion

The results presented in the previous sections have important implications for the current theoretical understanding of galactic winds. First, we have showed that the structure of the dense clouds plays a significant role in their evolution when exposed to a hot wind. Clouds that start with lognormal density distributions set by turbulent processes are mixed more quickly into the hot wind than spherical clouds with similar masses or median densities. This result implies that the mass-loading of galactic winds in the regions near the base of a hot outflow is an efficient process that likely results in significant mass-loading factors when hot gas interacts with denser clouds.

While our simulations indicate that mass-loading likely proves more efficient when the dense clouds have a turbulent structure, we have also demonstrated that this same structure in clouds tends to inhibit dense gas entrainment. The total acceleration of cloud material by the hot wind closely relates to the initial column density of the cool gas. In a turbulent cloud, the spatial coherence of high density regions leads to large column densities along individual sight lines, which makes dense gas difficult to accelerate. While individual dense clumps can remain long-lived as a result of efficient cooling, dense gas ($n > 1\ \mathrm{cm}^{-3}$) in our simulations rarely achieves velocities higher than $\approx 200\ \mathrm{km\ s}^{-1}$. Cloudlets tend to get ablated and mixed into the hot wind before traveling more than $\sim 30\ R_{\mathrm{cl}}$.

These findings support a picture of galactic winds where mass-loading into the hot phase operates efficiently near the base of an outflow, and any surviving dense gas accelerates very little. However, this efficient mass-loading will result in hot winds that are more susceptible to thermal instability as they expand and cool. As calculated in Thompson et al. (2016), high mass loading factors will decrease the radius at which gas can cool out of the hot wind. If the gas that cools out of the hot phase retains significant velocity, it could explain the high velocity neutral gas seen at distances $1 - 100$ kpc from starbursting galaxies without a need to resort to entrainment of cool gas directly associated with the ISM.

Furthermore, we have calculated in detail the distribution of mass and momentum associated with the clouds in our simulations. Far from being a simple two-phase medium, these winds are characterized by gas that spans a large range of densities and temperatures. Momentum from the hot wind couples to these phases with different efficiencies, such that the total momentum in each phase tends to be similar even if the mass is not. In cosmological simulations, where resolution limits the ability to capture these features of the multiphase galactic outflows, our calculations could be leveraged to improve treatments of the temperature and momentum distribution of the wind phases.

We now compare our findings to similar work in the literature. While many authors have studied the cloud-shock problem, relatively few have investigated the parameter space relevant to galactic winds and we will focus our attention on these results. We also discuss the potential effects that additional physics could have on our results, including different cooling rates, conduction, and magnetic fields. Finally, we include an analysis of the fate of the dense gas in our simulations in the presence of a gravitational potential.

3.8.1 Cloud structure

Cooper et al. (2009b) presented the only previous study that has investigated the destruction of clouds with a density distribution that is not symmetric along multiple axes. In that work, the authors compared the destruction of a radiatively-cooling fractal cloud to that of a radiatively cooling spherical cloud. The background hot wind properties in Cooper et al. (2009b) resembled ours. Our results agree well with theirs, in that they found that fractal clouds were more susceptible to fast destruction than spheres. However, their computational volume was too small to follow the evolution of the gas for many cloud-crushing times, and their resolution was relatively poor, and thus the fate of the small dense clumps that result from the destruction of an inhomogeneous cloud remained unclear. In our work, we find that while these small cores can survive for tens of cloud-crushing times as a result of efficient cooling, they are very difficult to accelerate due to their high column density. Additionally, higher resolution tends to amplify these effects. As a result, the cloudlets in our simulations are gradually destroyed over the course of $t \sim 20$ $t_{\rm cc}$ as the dense gas gets eaten away by the hot wind.

3.8.2 Entrainment and Mass Loading

In the past several years, several studies have investigated the ability of hot winds to entrain cool gas and carry it large distances from a galaxy. Scannapieco and Brüggen (2015) studied cloud-wind interactions using adaptive mesh refinement simulations with a maximum resolution equivalent to the fixed resolution of our simulations. The primary purpose of their work was to explore the effects that different background wind parameters had on cool cloud lifetimes and acceleration. Using spherical clouds of $T \approx 10^4$ K the authors derived a scaling for cloud destruction time that only depended on the mach number of the hot wind, with different density contrasts accounted for in the cloud-crushing time. Here, we compare our results to that scaling at t_{50}, defined as the time when 50% of the cloud material is at or above 1/3 the initial cloud density. Scannapieco and Brüggen (2015) find

$$t_{50} = 4t_{\rm cc}\sqrt{1 + M_{wind}}, \tag{3.10}$$

which for our background wind would correspond to $t_{50} = 10$ $t_{\rm cc}$. Looking at Figure 3.6, we see that this actually fits the turbulent clouds quite well, but our spherical clouds have a much longer lifetime.

At first glance, this result may seem contradictory. However, the longer spherical cloud lifetimes may be explained by the different treatment of cooling in our simulations. As explained in Appendix F, we allow gas in our simulations to cool to temperatures as low as $T = 10$ K using cooling rates calculated for solar metallicity gas. We also include the effects of a photoionizing background. The resulting heating keeps low density gas ($n < 1$ cm^{-3}) warm, but is much less effective at higher densities. In contrast, Scannapieco and Brüggen (2015) simulated larger-scale clouds with the assumption of complete ionization, and therefore only allowed cooling above $T = 10^4$ K. The ability of gas to cool to temperatures well below $T = 10^4$ K in our simulations results in greater cloud compression. The resulting smaller surface area decreases the efficiency with which material ablates and correspondingly increases the cloud lifetime.

We find further evidence for this explanation by performing simulations with a different mean molecular weight μ. All the simulations presented in this paper used $\mu = 1$ when converting from mass density to number density,

$$n = \rho/\mu m_{\rm p}, \tag{3.11}$$

where $m_{\rm p}$ is the mass of a proton. However, we also performed a low resolution turbulent cloud simulation with $\mu = 0.6$, the value appropriate for fully ionized solar metallicity gas. In comparing the lifetime of the cloud in this simulation with the same cloud in the standard simulation we find that the $\mu = 0.6$ cloud is destroyed more slowly, though the effect is small. This slight difference can be explained by the higher number densities for a given mass density in the $\mu = 0.6$ simulation. These higher number densities lead to more efficient cooling, which in turn leads to higher average densities at early times. The resulting high density clumps of gas prove more difficult for the hot wind to destroy than in the equivalent $\mu = 1$ model. Hence, the $\mu = 0.6$ cloud lives longer.

Scannapieco and Brüggen (2015) also find higher final velocities for their clouds than we do, even when comparing only spherical clouds. For clouds with similar background wind conditions they find velocities $v \sim 200 - 300$ km s^{-1} at $t = 15$ $t_{\rm cc}$ (see their Figure 7), while we never see velocities above 200 km s^{-1}. Again, this difference is likely a result of lower temperature gas in our simulations. Because our clouds can compress to higher densities in the initial stages of the wind interaction, their column densities increase more and acceleration is more difficult. This effect is compounded in the turbulent cloud case with higher initial column densities.

3.8.3 Additional Physics

Other studies of cool gas in the context of galactic winds have investigated the effects of additional physics in cloud-wind interaction simulations. Brüggen and

Scannapieco (2016) perform a similar study to Scannapieco and Brüggen (2015) but incorporate the effects of conduction. They find that conduction can result in complete evaporation of the cloud at early times in cases where the column density of cloud material is below $N \simeq 1.5 \times 10^{18}$ cm^{-2}. The clouds we simulate do not start with column densities this low, and we therefore do not expect conduction to result in their rapid, complete destruction. We note the initial conditions of our hot wind most resembles their Mach 6.4 run that shows the least amount of difference in cloud evolution between the conduction and non-conduction models.

Brüggen and Scannapieco (2016) demonstrate that when clouds in their simulations do survive, conduction tends to decrease the cross-section of the cloud presented to the hot wind. The decreased cross-section provides less surface area for ram pressure to act on the dense gas, which in turn decreases the efficiency of cloud acceleration. This effect is qualitatively similar to the impact of inhomogeneous column densities for turbulent clouds described in Section 3.6.1, though the origin is completely different. We suggest that the inhomogeneous cloud structure may work in concert with conduction to further increase the early destruction of low column density material, and decrease the efficiency with which high column material accelerates.

Magnetic fields may also play a role in cool cloud evolution. A number of cloud-shock studies investigated the effects of planar magnetic fields in spherical clouds, with inconclusive results regarding whether the presence of field lines increased or decreased the time until cloud destruction (Gregori et al., 2000; Fragile et al., 2005b; Shin et al., 2008b). More recently, McCourt et al. (2015) performed wind-cloud simulations testing the effects of tangled magnetic fields incorporated within a spherical cloud. They showed that the presence of the magnetic field drastically increased the lifetime of the resulting cloudlets and increased their acceleration, allowing them to reach the hot wind speed without destruction.

The McCourt et al. (2015) simulations help motivate a future extension of our high-resolution wind-cloud simulations with realistic initial conditions to include MHD. The simulations by McCourt et al. (2015) use a resolution of 32 cells/$R_{\rm cl}$.

Scannapieco and Brüggen (2015) demonstrated that spherical cloud simulations with resolution less than 64 cells per cloud radius tend to prolong cloud lifetimes. Our simulations of turbulent clouds show a trend toward faster mass loss with increasing resolution that has not yet converged at 128 cells/R_{cl}. In addition, we have shown that spherical clouds have longer lifetimes than those with more realistic internal structure. While the results of McCourt et al. (2015) suggest that including a tangled magnetic field would likely prolong the lifetime of the dense components of the turbulent clouds we simulate, the combined effects of resolution and density structure make a quantitative prediction difficult. Incorporating magnetic fields in wind simulations with turbulent clouds is an avenue that we aim to pursue in future work.

In addition to potentially important additional physics, parameters such as the metallicity of the cloud gas and nature of the background wind in our simulations will affect the quantitative results we have presented. In this work we have focused exclusively on the effects of cloud structure and surface density. The primary effect of changing the metallicity of the gas would be to change the cooling rates. As noted in Section 3.8.2, tests with a different value of μ indicate that more rapid cooling leads to longer cloud survival, and our comparison with the simulations of Scannapieco and Brüggen (2015) indicates that less efficient cooling reduces cloud survival time. However, this is not an order-of-magnitude effect, and we therefore do not expect a change in metallicity to drastically alter our results.

On the other hand, as Figure 3.1 shows, the state of the background wind in the Chevalier and Clegg (1985) model changes rapidly with radius as the wind escapes the galaxy. Given our choice of a single background wind state, we do not regard our simulations as completely generic with respect to galactic winds, though we do expect the general result of more rapid destruction of turbulent clouds to hold. Scannapieco and Brüggen (2015) demonstrated a scaling with background wind mach number that indicates clouds survive longer with increasing mach number (see their Equation 22). In the future, we would like to investigate the relationship between turbulent cloud lifetime and the background wind parameters, sampling a variety of distances from the galaxy that would inform the initial conditions for clouds at each distance.

3.8.4 Ram Pressure vs Gravity

As a final note, we consider the final fate of the dense gas in our simulations. Our simulations do not include gravity, and if the simulations continued running indefinitely eventually all of the cool material would mix into the outflowing hot wind. However, we can estimate the effect of the host galaxy's gravitational potential. Using an analysis similar to that in Section 3.6.1, we can compare the expected acceleration of dense cloud regions owing to the wind's ram pressure, $P_{\rm ram}$, to their expected deceleration owing to gravity.

We can estimate the ram pressure acceleration as a function of column density, $N_{\rm H} = n_{\rm cl}L$, via

$$g_{\rm ram} \sim 2.3\,{\rm km\,s^{-1}\,kyr^{-1}} \left(\frac{N_{\rm H}}{10^{19}\,{\rm cm^{-2}}}\right)^{-1}.$$

(This expression is equivalent to Equation 3.7.) Similarly, we can estimate the gravitational acceleration as a function of column density. At 1 kpc, the gravitational acceleration of M82 is

$$\begin{aligned} g_{\rm grav} &\sim 1.4\times10^{-7}\,{\rm cm\,s^{-2}} \left(\frac{M_{\rm M82}}{10^{10}M_\odot}\right)\left(\frac{R}{1\,{\rm kpc}}\right)^{-2}, \\ g_{\rm grav} &\sim 0.044\,{\rm km\,s^{-1}\,kyr^{-1}}. \end{aligned} \tag{3.12}$$

Given these estimates, we would expect cloud regions with column densities greater than $N_{\rm H} \approx 5\times10^{20}$ cm^{-2} to begin to fall back toward the galaxy. None of the clouds in our simulations initially have column densities quite this high - the densest sight lines in the $\tilde{n} = 1$ cm^{-3} turbulent cloud reach $N_{\rm H} \approx 3\times10^{20}$ cm^{-2}. Nonetheless, the acceleration due to ram pressure and deceleration due to gravity for these column densities are very similar, so in the presence of gravity the densest gas in our turbulent cloud simulations would likely be accelerated very little or possibly fall back toward the central galaxy.

3.9 Summary and Conclusions

In this work, we have modeled the hydrodynamic evolution of radiatively cooling clouds in the context of galactic winds with very high numerical resolution. Our study investigated two main parameters relevant to cold cloud survival - the initial structure of the cool gas and the median density of the cloud. We varied the cloud structure in our simulations between a lognormal density distribution with large-scale structure as set by turbulent processes and an idealized spherical distribution of gas. The median densities of our clouds ranged from $\tilde{n} = 0.1 - 1.0$ cm^{-3}. The median density affects the overall destruction time of the cool gas via the cloud crushing time, as well as the efficiency of cooling within the cloud.

We find that clouds with a turbulent density structure are destroyed more quickly than clouds with a homogeneous spherical density distribution. This efficient destruction results in faster mass-loading of the hot wind, as intermediate- and low-density regions of turbulent clouds are quickly heated, rarified, and accelerated to the hot wind velocity. The entrainment of dense gas within cool turbulent clouds proves extremely inefficient, and much less efficient than for idealized spherical initial conditions. The varying column densities present in turbulent clouds result in very little acceleration of the densest regions, which are the only regions that survive for many cloud-crushing times. These effects are amplified as the resolution of the simulations is increased and the clouds are allowed to become increasingly realistic. We therefore conclude that entrainment of turbulent ISM clouds in hot supernova winds does not explain the neutral gas observed at large distances from starburst galaxies, unless other physical processes (such as magnetic fields) substantially alter the results from the hydrodynamic case.

We have also provided an extensive description of the phase structure of the gas in the wind. Shortly after being shocked the gas associated with the turbulent clouds spreads over a large range of densities and temperatures, with the densest regions cooling down to temperatures of $T \sim 100$ K. Each phase of gas remains close to thermal pressure equilibrium with the hot ($\gg 10^6$ K) wind. Interestingly, though the majority of the mass remains in the densest phases ($n > 20$ cm^{-3}) for much of the cloud evolution, the total momentum distributes fairly evenly across densities. Roughly the same amount of momentum transfers to cold neutral (100's of K), cool ionized ($\sim 10^4 K$), and warm ionized ($\sim 10^5$ K) gas.

CHAPTER 4

SUMMARY AND FUTURE PROSPECTS

In the previous two chapters, I have described the massively-parallel hydrodynamics code, *Cholla*, and presented a high resolution but small physical scale study of the interactions between different phases of gas within a multiphase galactic wind. However, as discussed in Chapter 1, *Cholla* was designed to be able to tackle many problems. Additionally, the simulations discussed in Chapters 2 and 3 raised many questions. As a result, there exist many future prospects for further investigation into the nature of multiphase galactic winds using *Cholla*, some of which I briefly describe in this section.

4.0.1 Global Disk Simulations of Galactic Winds

Idealized simulations of the type presented in Chapter 3 are useful for probing physical scales in winds that cannot be resolved when the entire galaxy is simulated. On this sub-parsec scale, I have demonstrated that the structure of dense clouds affects the timescale for their disruption and incorporation into the hot wind. However, such simulations require a variety of assumptions, particularly the fixed nature of the background flow. While the background could be varied from simulation to simulation in order to probe different regions of parameter space, no one simulation can self-consistently reproduce the physical conditions of the changing background as hot gas is escaping a galaxy, and therefore, the inferences about acceleration of dense gas made from such simulations must be limited (particularly near the galaxy where the hot wind state changes most rapidly). These limitations can be overcome by simulating the full galaxy and allowing the wind to expand over many kiloparsecs.

As mentioned in Section 1, galactic winds that are sufficiently mass-loaded are expected to cool radiatively as they expand to large radii. In order to capture this cooling, a simulation must cover a large enough volume that the wind reaches the cooling radius, which Thompson et al. (2016) estimate to be 1 - 10 kpc, depending on the star formation rate surface density and the hot wind mass-loading. In addition, the simulation must have sufficient resolution to capture the characteristic scale of the clouds formed, which McCourt et al. (2016) estimate is roughly equal to the sound speed of the dense gas times the cooling time, $r_{\mathrm{cloud}} \sim c_{\mathrm{s}} t_{\mathrm{cool}} \sim (0.1\,\mathrm{pc}) \left(\frac{n}{\mathrm{cm}^{-3}}\right)^{-1}$. In addition, the initial density perturbations in the hot flow caused by the destruction of cool clouds in or near the starburst region ($r < R_{\mathrm{starburst}}$) will affect the way that the hot gas cools. Therefore, a simulation that can capture the multiphase nature of radiatively-cooling starburst-driven winds must include cool disk gas, a starburst region where energy and mass are input according to a star formation rate, extend out to $\approx 10\,\mathrm{kpc}$, and have a resolution of a few parsecs. With *Cholla*, such simulations are possible for the first time. Over the next several years, I will be carrying out a suite of these global-disk simulations that will test the effects of different star formation rates and ISM structures on the large scale properties of galactic winds..

4.0.2 Additional Physics in *Cholla*

As with any major code, the development period for *Cholla* will never be over. Tackling problems in a range of fields requires a range of physics, and continuing to add further physics modules to *Cholla* will make it a more flexible and useful code. In the sections below, I describe two extensions to the current publicly-available version of *Cholla* described in Chapter 2 and the appendices. The first of these modifications, the incorporation of static gravity, is already complete.

Static Gravity

In order to carry out the simulations described above, some additional physics must be added to *Cholla*. Because the simulations will extend well outside the mass and energy injection region of the starburst, gravity is expected to play an important role in the kinematics of the wind, particularly for the denser gas (Wang, 1995). Additionally, the initial conditions will consist of a stable gas disk in vertical hydrostatic equilibrium. Therefore, *Cholla* needs to include the effects of a static gravitational potential.

Gravity appears as source terms in the momentum and energy fluid equations, $\boldsymbol{S}_{\rho\boldsymbol{v}} = \rho\nabla\phi$ and $S_E = \rho\boldsymbol{v}\cdot\nabla\phi$, where $\nabla\phi = \mathbf{g}$ is the gradient of the gravitational potential ϕ (i.e. the gravitational acceleration). In *Cholla*, these source terms are included via an the operator-split update at the end of the hydro step. This takes the form (in 1D):

$$(\rho v)^{n+1} = (\rho v)^{*} + \frac{\Delta t}{2} g(\rho^{n} + \rho^{*}) \tag{4.1}$$

and

$$E^{n+1} = E^{*} + \frac{\Delta t}{4} g(\rho^{n} + \rho^{*})(v^{n} + v^{*}), \tag{4.2}$$

where n refers to the values of the density and velocity at the beginning of the timestep (before the hydro update), $*$ refers to the values after the hydro update but before the gravitational source terms are added, and $n+1$ are the values after the complete hydro plus gravity update. Because the acceleration is applied to an average of the variables before and after the hydro update (and the potential is static), this update is second-order in time.

Magnetic Fields

Another natural extension to the work presented in this thesis is testing the effects of magnetic fields on the structure and dynamics of galactic winds. McCourt et al. (2015) have suggested that magnetic fields may slow the destruction of dense gas clouds in galactic winds, making them potentially quite important for a complete theory of multiphase winds. In order to carry out such a study, the purely hydrodynamic integration algorithms in *Cholla* will need to be extended to MHD. Fortunately, *Cholla* was designed so that this extension would be straightforward. While the details are beyond the scope of this work, the CTU algorithm presented in Chapter 2 was developed for MHD, so no new algorithm development will be required. The primary concern in extending *Cholla* to MHD has been the memory constraints of the GPUs that the code uses. However, the most recent generation of NVIDIA GPUs, namely the Pascal architecture, have addressed this concern, as they roughly doubled the amount of global memory available per core compared to the Kepler GPUs available when *Cholla* was designed. Thus, despite the additional B-fields that must be tracked in an MHD integration, the extension to MHD should not cause a problem for *Cholla*, and may in fact lead it to perform even better in comparison to CPU-based codes.

APPENDIX A

RECONSTRUCTION METHODS IN CHOLLA

To calculate the input states for the Riemann solvers used in *Cholla*, appropriate values of the conserved variables at each interface must be reconstructed using the cell-averaged quantities. Each Riemann problem requires input states at either side of a cell interface, referred to as $\boldsymbol{U}_L^*$ and $\boldsymbol{U}_R^*$ in the context of the CTU algorithm presented in Section 2.2.1. While previously L and R indicated the left and right of the interface, in this Appendix a cell-centered labeling is used to document the procedure for computing the input states at the left and right boundaries of a single cell.

The input states calculated at the left and right sides of the cell will be labeled $\boldsymbol{U}_L^*$ and $\boldsymbol{U}_R^*$ in the conserved variables, or $\boldsymbol{W}_L^*$ and $\boldsymbol{W}_R^*$ in the primitive variables. As described in Table 2.1, the asterisk indicates the time-evolved input state. The boundary values reconstructed before time evolution will be labeled $\boldsymbol{W}_L$ and $\boldsymbol{W}_R$. For each method described below, only the steps involved in the reconstruction for the x-interfaces are shown. The y- and z-reconstructions proceed in the same manner but with an appropriate change of stencil. The notation will drop the i, j, and k subscripts unless they are needed for clarification.

A.0.1 PLMP

The simplest practical reconstruction method implemented in *Cholla* is PLMP, a piecewise linear reconstruction with slope limiting applied in the primitive variables. The stencil to calculate the boundary values for cell i contains cells $i-1$ to the left and $i+1$ to the right. The first step in the method converts the cell-averaged values of the conserved variables into the primitive variables, $\boldsymbol{w} = [\rho, u, v, w, p]^T$. The cell-averaged primitive values are then used to reconstruct boundary values in the primitive variables, $\boldsymbol{W}_L$ and $\boldsymbol{W}_R$ at the left and right sides of cell i using a local, piece-wise linear reconstruction (Toro, 2009):

$$\begin{aligned} \boldsymbol{W}_L &= \boldsymbol{w}_i - \frac{1}{2}\delta\boldsymbol{w}_i, \\ \boldsymbol{W}_R &= \boldsymbol{w}_i + \frac{1}{2}\delta\boldsymbol{w}_i, \end{aligned} \tag{A.1}$$

where $\delta\boldsymbol{w}_i$ is a vector containing the slope of each primitive variable across cell i. To compute $\delta\boldsymbol{w}_i$, we first calculate the left, right, and centered differences in the primitive variables across each of the cell interfaces:

$$\delta\boldsymbol{w}_L = \boldsymbol{w}_i - \boldsymbol{w}_{i-1}, \quad \delta\boldsymbol{w}_R = \boldsymbol{w}_{i+1} - \boldsymbol{w}_i, \quad \delta\boldsymbol{w}_C = 0.5(\boldsymbol{w}_{i+1} - \boldsymbol{w}_{i-1}). \tag{A.2}$$

A monotonized central-difference limiter (van Leer, 1977) is then used to compute $\delta\boldsymbol{w}_i$:

$$\delta\boldsymbol{w}_i = \begin{cases} \mathrm{sgn}(\delta\boldsymbol{w}_C)\min(|\delta\boldsymbol{w}_C|, 2|\delta\boldsymbol{w}_L|, 2|\delta\boldsymbol{w}_R|), & \delta\boldsymbol{w}_L\delta\boldsymbol{w}_R > 0 \\ 0, & \text{otherwise}, \end{cases} \tag{A.3}$$

where sgn is defined as

$$\mathrm{sgn}(x) = \begin{cases} -1, & x < 0 \\ 1, & \text{otherwise}. \end{cases} \tag{A.4}$$

Each of the primitive variables is treated independently in the limiting process, so the vector of primitive variable slopes (for extrapolations to the left cell face) can be simply written as

$$\delta\boldsymbol{w}_L = \{\delta\rho_L, \delta u_L, \delta v_L, \delta w_L, \delta p_L\}^{\mathrm{T}}. \tag{A.5}$$

The primitive variable slopes for extrapolating to the right cell face can be similarly defined.

The last step in computing input states for the Riemann problem is to evolve the reconstructed boundary values by half a time step $\Delta t/2$. The time evolution is modeled using the conserved form of the Euler equations, and $\boldsymbol{W}_L$ and $\boldsymbol{W}_R$ are therefore converted back into conserved variables, $\boldsymbol{U}_L$ and $\boldsymbol{U}_R$ and used to calculate the associated fluxes via Equation 2.11. These fluxes are used to evolve the reconstructed boundary values and obtain input states appropriate for the Riemann problem:

$$\boldsymbol{U}_L^* = \boldsymbol{U}_L + 0.5\frac{\Delta t}{\Delta x}[\boldsymbol{F}(\boldsymbol{U}_L) - \boldsymbol{F}(\boldsymbol{U}_R)] \tag{A.6}$$

$$\boldsymbol{U}_R^* = \boldsymbol{U}_R + 0.5\frac{\Delta t}{\Delta x}[\boldsymbol{F}(\boldsymbol{U}_L) - \boldsymbol{F}(\boldsymbol{U}_R)]. \tag{A.7}$$

A.0.2 PLMC

The second reconstruction method, PLMC, is also based on a piecewise linear reconstruction but with the slope limiting computed using the characteristic variables. The stencil again contains one cell to the left and right of cell i. After converting the cell-averaged quantities from conserved to primitive variables, an eigenvector decomposition of the Euler equations is performed using the characteristic variables as described in Section 2.2. First, the eigenvalues of the linear system of equations for cell i are calculated. For adiabatic hydrodynamics, the eigenvalues correspond to the three wave speeds,

$$\lambda^m = u_i - a_i, \quad \lambda^0 = u_i, \quad \lambda^p = u_i + a_i, \tag{A.8}$$

where a_i is the average sound speed in cell i. The quantities λ^m and λ^p are speeds of the acoustic waves and λ^0 is the speed of the contact wave. The corresponding eigenvalue for any advected scalar quantity (such as the transverse velocities in multidimensional problems) is simply the speed of the fluid in the normal direction u_i.

The left ($\delta\boldsymbol{w}_L$), right ($\delta\boldsymbol{w}_R$), and centered ($\delta\boldsymbol{w}_C$) differences in the primitive variables shown in Equation A.2 are then calculated. These differences are projected onto the characteristic variables, $\delta\boldsymbol{\xi}$, using the left eigenvectors given in Appendix A of Stone et al. (2008). Rather than reproduce the the expressions for each eigenvector, equations describing the final projections are shown since they are actually used in the GPU kernel. The projection of the left difference is

$$\delta\boldsymbol{\xi}_L = \begin{bmatrix} -0.5(\rho_i \delta u_L/a_i + \delta p_L/a_i^2) \\ \delta\rho_L - \delta p_L/a_i^2 \\ \delta v_L \\ \delta w_L \\ 0.5(\rho_i \delta u_L/a_i + \delta p_L/a_i^2) \end{bmatrix}, \tag{A.9}$$

where $\delta\rho_L$, δu_L, δv_L, δw_L, and δp_L are the components of the primitive variable difference vector $\delta\boldsymbol{w}_L$. The projections for the right and central differences are calculated in the same manner, yielding $\delta\boldsymbol{\xi}_R$ and $\delta\boldsymbol{\xi}_C$.

The characteristic differences are then monotonized using the van Leer (1977) limiter, computed as

$$\delta\boldsymbol{\xi} = \begin{cases} \mathrm{sgn}(\delta\boldsymbol{\xi}_C)\min(|\delta\boldsymbol{\xi}_C|, 2|\delta\boldsymbol{\xi}_L|, 2|\delta\boldsymbol{\xi}_R|), & \delta\boldsymbol{\xi}_L\delta\boldsymbol{\xi}_R > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{A.10}$$

We project the monotonized differences in the characteristic variables back onto the primitive variables, providing slopes in each variable that are analogous to the limited slopes described in PLMP:

$$\delta\boldsymbol{w}_i = \begin{bmatrix} \delta\xi_0 + \delta\xi_1 + \delta\xi_4 \\ (a_i/\rho_i)(-\delta\xi_0 + \delta\xi_4) \\ \delta\xi_2 \\ \delta\xi_3 \\ a_i^2(\delta\xi_0 + \delta\xi_4) \end{bmatrix}. \tag{A.11}$$

Here, the numeric subscripts to refer to the components of the vector $\boldsymbol{\xi}$.

As in PLMP, these slopes are subsequently used to create a linear interpolation for reconstructing boundary values of the primitive variables:

$$\begin{aligned} \boldsymbol{W}_{L,A} &= \boldsymbol{w}_i - \frac{1}{2}\delta\boldsymbol{w}_i, \\ \boldsymbol{W}_{R,A} &= \boldsymbol{w}_i + \frac{1}{2}\delta\boldsymbol{w}_i. \end{aligned} \tag{A.12}$$

The primitive variable boundary values are further monotonized to ensure that they are numerically bounded by the neighboring cell values:

$$\begin{aligned} \boldsymbol{W}_{L,B} &= \max[\min(\boldsymbol{w}_i, \boldsymbol{w}_{i-1}), \boldsymbol{W}_{L,A}] \\ \boldsymbol{W}_{L} &= \min[\max(\boldsymbol{w}_i, \boldsymbol{w}_{i-1}), \boldsymbol{W}_{L,B}] \\ \boldsymbol{W}_{R,B} &= \max[\min(\boldsymbol{w}_i, \boldsymbol{w}_{i+1}), \boldsymbol{W}_{R,A}] \\ \boldsymbol{W}_{R} &= \min[\max(\boldsymbol{w}_i, \boldsymbol{w}_{i+1}), \boldsymbol{W}_{R,B}], \end{aligned} \tag{A.13}$$

enabling a slope vector to be computed from these adjusted boundary values as

$$\delta\boldsymbol{w}_i = \boldsymbol{W}_R - \boldsymbol{W}_L. \tag{A.14}$$

The reconstructed boundary values must be evolved in time to calculate appropriate input states for the Riemann problem. Instead of simply evolving the reconstructed values using associated fluxes as in Equations A.6 and A.7, the characteristic tracing method of Colella and Woodward (1984) is employed. To obtain a first approximation for the input states, an integration under the linear interpolation function is performed using the minimum wave speed to define the domain of dependence for the left side of the cell, λ^m, and the maximum wave speed for the right side of the cell, λ^p:

$$\begin{aligned} \tilde{\boldsymbol{W}}_L^* &= \boldsymbol{W}_L - 0.5\frac{\Delta t}{\Delta x}\min(\lambda^m, 0)\delta\boldsymbol{w}_i \\ \tilde{\boldsymbol{W}}_R^* &= \boldsymbol{W}_R - 0.5\frac{\Delta t}{\Delta x}\max(\lambda^p, 0)\delta\boldsymbol{w}_i \end{aligned} \tag{A.15}$$

The input states are then corrected by accounting for the portion of each wave that does not reach the interface over a time $\Delta t/2$ as a result of the presence of the other waves. Correction terms are only needed for characteristics propagating toward each interface. The eigenvector projection and correction for each element of $\boldsymbol{W}^*$ is shown below, tracking the correction terms in the vectors $\boldsymbol{s}_L$ and $\boldsymbol{s}_R$. For the left side of cell i,

$$\boldsymbol{W}_L^* = \tilde{\boldsymbol{W}}_L^* + 0.5\frac{\Delta t}{\Delta x}\boldsymbol{s}_L \tag{A.16}$$

with

$$\boldsymbol{s}_L = \begin{bmatrix} (\lambda^m - \lambda^0)(\delta\rho_i - \delta p_i/a_i^2) \\ 0 \\ (\lambda^m - \lambda^0)\delta v_i \\ (\lambda^m - \lambda^0)\delta w_i \\ 0 \end{bmatrix} + \begin{bmatrix} 0.5(\lambda^m - \lambda^p)(\rho_i \delta u_i/a_i + \delta p_i/a_i^2) \\ 0.5(\lambda^m - \lambda^p)[\delta u_i + \delta p_i/(a_i\rho_i)] \\ 0 \\ 0 \\ 0.5(\lambda^m - \lambda^p)(\rho_i \delta u_i a_i + \delta p_i) \end{bmatrix}. \quad \text{(A.17)}$$

The first term is associated with the contact wave and is added only if $\lambda^0 < 0$. The second term is associated with the right acoustic wave and is added only if $\lambda^p < 0$. If both λ^0 and λ^p are greater than 0, there is no need for a correction because those waves cannot affect the left interface of the cell. For the right side of cell i,

$$\boldsymbol{W}_R^* = \tilde{\boldsymbol{W}}_R^* + 0.5\frac{\Delta t}{\Delta x}\boldsymbol{s}_R \quad \text{(A.18)}$$

with

$$\boldsymbol{s}_R = \begin{bmatrix} 0.5(\lambda^p - \lambda^m)(-\rho_i \delta u_i/a_i + \delta p_i/a_i^2) \\ 0.5(\lambda^p - \lambda^m)[\delta u_i - \delta p_i/(a_i\rho_i)] \\ 0 \\ 0 \\ 0.5(\lambda^p - \lambda^m)(-\rho_i \delta u_i a_i + \delta p_i) \end{bmatrix} + \begin{bmatrix} (\lambda^p - \lambda^0)(\delta\rho_i - \delta p_i/a_i^2) \\ 0 \\ (\lambda^p - \lambda^0)\delta v_i \\ (\lambda^p - \lambda^0)\delta w_i \\ 0 \end{bmatrix}. \quad \text{(A.19)}$$

Here the first term is associated with the left acoustic wave and is added only if $\lambda^m > 0$, while the second term is associated with the contact wave and is added only if $\lambda^0 > 0$. As with the left interface, the corrections only apply if the waves are moving toward the interface. Once the corrections have been made, $\boldsymbol{W}_L^*$ and $\boldsymbol{W}_R^*$ can be used as inputs to the Riemann problem.

A.0.3 PPMC

Cholla also includes implementations of third-order spatial reconstruction techniques, including two varieties of the piecewise parabolic method developed by Colella and Woodward (1984). The piecewise parabolic method with slope limiting applied in the characteristic variables (PPMC) is presented first as it shares several steps with PLMC. Our implementation of this method closely follows that outlined in Stone et al. (2008).

The first step in PPMC is to calculate monotonized slopes for cells $i-1$, i, and $i+1$. These slopes are labeled $\delta \boldsymbol{w}_{i-1}$, $\delta \boldsymbol{w}_i$, and $\delta \boldsymbol{w}_{i+1}$. The limited slopes for each cell are calculated in a manner identical to that described in PLMC, following Equations A.8 - A.11. Since slopes must be calculated for all three cells, the stencil for PPMC contains two cells to the left and right of cell i. Once the limited slope vectors for all three cells have been calculated, the algorithm proceeds as follows.

Using the monotonized linear slopes, a parabolic interpolation is computed and used to calculate the reconstructed boundary values:

$$\begin{aligned} \boldsymbol{W}_{L,A} &= \frac{1}{2}(\boldsymbol{w}_i + \boldsymbol{w}_{i-1}) - \frac{1}{6}(\delta\boldsymbol{w}_i - \delta\boldsymbol{w}_{i-1}), \\ \boldsymbol{W}_{R,A} &= \frac{1}{2}(\boldsymbol{w}_{i+1} + \boldsymbol{w}_i) - \frac{1}{6}(\delta\boldsymbol{w}_{i+1} - \delta\boldsymbol{w}_i). \end{aligned} \tag{A.20}$$

Monotonicity constraints are applied to the reconstructed boundary values to ensure that they lie between the average values in neighboring cells. If the cell contains a local minimum or maximum, both interface values are set equal to the cell average:

$$\boldsymbol{W}_{L,A} = \boldsymbol{W}_{R,A} = \boldsymbol{w}_i, \quad \text{if } (\boldsymbol{W}_{R,A} - \boldsymbol{w}_i)(\boldsymbol{w}_i - \boldsymbol{W}_{L,A}) \leq 0 \tag{A.21}$$

If the parabolic interpolation violates monotonicity as a result of a steep gradient, the interface values are modified as

$$\begin{aligned} \boldsymbol{W}_{L,A} &= 3\boldsymbol{w}_i - 2\boldsymbol{W}_{R,A}, \quad \text{if } (\boldsymbol{W}_{R,A} - \boldsymbol{W}_{L,A})[\boldsymbol{W}_{L,A} - (3\boldsymbol{w}_i - 2\boldsymbol{W}_{R,A})] < 0 \\ \boldsymbol{W}_{R,A} &= 3\boldsymbol{w}_i - 2\boldsymbol{W}_{L,A}, \quad \text{if } (\boldsymbol{W}_{R,A} - \boldsymbol{W}_{L,A})[(3\boldsymbol{w}_i - 2\boldsymbol{W}_{L,A}) - \boldsymbol{W}_{R,A}] < 0 \end{aligned} \tag{A.22}$$

These reconstructed boundary values are further adjusted using the minmod limiter operation:

$$\begin{aligned} \boldsymbol{W}_{L,B} &= \max[\min(\boldsymbol{w}_i, \boldsymbol{w}_{i-1}), \boldsymbol{W}_{L,A}] \\ \boldsymbol{W}_{L} &= \min[\max(\boldsymbol{w}_i, \boldsymbol{w}_{i-1}), \boldsymbol{W}_{L,B}] \\ \boldsymbol{W}_{R,B} &= \max[\min(\boldsymbol{w}_i, \boldsymbol{w}_{i+1}), \boldsymbol{W}_{R,A}] \\ \boldsymbol{W}_{R} &= \min[\max(\boldsymbol{w}_i, \boldsymbol{w}_{i+1}), \boldsymbol{W}_{R,B}] \end{aligned} \tag{A.23}$$

At this point, a monotonized parabolic interpolation can be reconstructed. New slopes are computed that account for the adjusted boundary values:

$$\delta \boldsymbol{w}_i = \boldsymbol{W}_R - \boldsymbol{W}_L. \tag{A.24}$$

These slopes are used to compute the time-evolved left and right boundary values by integrating under a parabolic interpolation function:

$$\begin{aligned} \tilde{\boldsymbol{W}}_L^* &= \boldsymbol{W}_L - \frac{1}{2}\alpha^m \left[\delta \boldsymbol{w}_i + \boldsymbol{w}_6(1 + \frac{2}{3}\alpha^m)\right], \\ \tilde{\boldsymbol{W}}_R^* &= \boldsymbol{W}_R - \frac{1}{2}\beta^p \left[\delta \boldsymbol{w}_i - \boldsymbol{w}_6(1 - \frac{2}{3}\beta^p)\right], \end{aligned} \tag{A.25}$$

where

$$\boldsymbol{w}_6 = 6\boldsymbol{w}_i - 3(\boldsymbol{W}_L - \boldsymbol{W}_R). \tag{A.26}$$

Here we have borrowed from the notation of Colella and Woodward (1984) to define α^m and β^p, unit-free variables associated with the characteristic speeds

$$\begin{aligned} \alpha^m &= \frac{\Delta t}{\Delta x}\min(\lambda^m, 0), \quad \beta^m = \frac{\Delta t}{\Delta x}\max(\lambda^m, 0), \\ \alpha^0 &= \frac{\Delta t}{\Delta x}\min(\lambda^0, 0), \quad \beta^0 = \frac{\Delta t}{\Delta x}\max(\lambda^0, 0), \\ \alpha^p &= \frac{\Delta t}{\Delta x}\min(\lambda^p, 0), \quad \beta^p = \frac{\Delta t}{\Delta x}\max(\lambda^p, 0). \end{aligned} \tag{A.27}$$

As in PLMC, the minimum characteristic speed, λ^m, is used to define the domain of dependence for the left interface, and the maximum characteristic speed, λ^p, is used for the right interface. The primitive variable time-evolved boundary values $\tilde{\boldsymbol{W}}_L^*$ and $\tilde{\boldsymbol{W}}_R^*$ are first approximations to the input states for the Riemann problem.

The input states must now be corrected by accounting for the other characteristics propagating toward the interface. At the left side of the cell, we compute

$$\boldsymbol{s}_L = \begin{bmatrix} \boldsymbol{E}_0 - \boldsymbol{E}_4/a_i^2 \\ 0 \\ \boldsymbol{E}_2 \\ \boldsymbol{E}_3 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.5(\rho_i \boldsymbol{B}_1/a_i + \boldsymbol{B}_4/a_i^2) \\ 0.5\left[\boldsymbol{B}_1 + \boldsymbol{B}_4/(a_i\rho_i)\right] \\ 0 \\ 0 \\ 0.5(\rho_i \boldsymbol{B}_1 a_i + \boldsymbol{B}_4) \end{bmatrix}, \tag{A.28}$$

where the first term is added only if $\lambda^0 < 0$, and the second only if $\lambda^p < 0$ - otherwise there is no correction. In the above,

$$\boldsymbol{E} = \frac{1}{2}\alpha^m \left[\delta \boldsymbol{w}_i + \boldsymbol{w}_6(1 + \frac{2}{3}\alpha^m)\right] - \frac{1}{2}\alpha^0 \left[\delta \boldsymbol{w}_i + \boldsymbol{w}_6(1 + \frac{2}{3}\alpha^0)\right], \tag{A.29}$$

is associated with the contact wave, and

$$\boldsymbol{B} = \frac{1}{2}\alpha^m \left[\delta \boldsymbol{w}_i + \boldsymbol{w}_6(1 + \frac{2}{3}\alpha^p)\right] - \frac{1}{2}\alpha^p \left[\delta \boldsymbol{w}_i + \boldsymbol{w}_6(1 + \frac{2}{3}\alpha^p)\right]. \tag{A.30}$$

is associated with the right-most acoustic wave. Subscripts denote the elements of $\boldsymbol{E}$ and $\boldsymbol{B}$. Similarly, for the right side of the cell,

$$\boldsymbol{s}_R = \begin{bmatrix} 0.5(-\rho_i \boldsymbol{C}_1/a_i + \boldsymbol{C}_4/a_i^2) \\ 0.5\left[\boldsymbol{C}_1 - \boldsymbol{C}_4/(a_i\rho_i)\right] \\ 0 \\ 0 \\ 0.5(-\rho_i \boldsymbol{C}_1 a_i + \boldsymbol{C}_4) \end{bmatrix} + \begin{bmatrix} (\boldsymbol{D}_0 - \boldsymbol{D}_4/a_i^2) \\ 0 \\ \boldsymbol{D}_2 \\ \boldsymbol{D}_3 \\ 0 \end{bmatrix}. \tag{A.31}$$

The first term accounts for the correction owing to the left-most acoustic wave and is added only if $\lambda^m > 0$, and the second term accounts for the correction from the contact wave and is added only if $\lambda^0 > 0$. In this case,

$$\begin{aligned} \boldsymbol{C} &= \frac{1}{2}\beta^p \left[\delta \boldsymbol{w}_i - \boldsymbol{w}_6(1 - \frac{2}{3}\beta^p)\right] - \frac{1}{2}\beta^m \left[\delta \boldsymbol{w}_i - \boldsymbol{w}_6(1 - \frac{2}{3}\beta^m)\right], \\ \boldsymbol{D} &= \frac{1}{2}\beta^p \left[\delta \boldsymbol{w}_i - \boldsymbol{w}_6(1 - \frac{2}{3}\beta^p)\right] - \frac{1}{2}\beta^0 \left[\delta \boldsymbol{w}_i + \boldsymbol{w}_6(1 + \frac{2}{3}\beta^0)\right]. \end{aligned} \tag{A.32}$$

With these correction terms input states for the Riemann problem can be calculated as

$$\boldsymbol{W}_L^* = \tilde{\boldsymbol{W}}_L^* + \boldsymbol{s}_L \quad \text{and} \quad \boldsymbol{W}_R^* = \tilde{\boldsymbol{W}}_R^* + \boldsymbol{s}_R. \tag{A.33}$$

A.0.4 PPMP

The *Cholla* implementation of the piecewise parabolic method computed in the primitive variables closely follows the original description in Colella and Woodward (1984), with some additional notation adapted from Fryxell et al. (2000). For convenience, the following description assumes a uniform cell size. PPMP is the most complicated of the reconstruction methods implemented in *Cholla*, and the algorithm follows this brief outline:

1. Reconstruct boundary values using parabolic interpolation.
2. Steepen contact discontinuities, if necessary.
3. Flatten shocks, if necessary.
4. Ensure that the parabolic distribution is monotonic.
5. Integrate under the parabolic interpolation to determine input states for the Riemann problem.
6. Use characteristic tracing to correct the input states.

Parabolic Interpolation

The first step in PPMP is to reconstruct the boundary values using a parabolic interpolation with limited slopes. The interpolation is identical to that shown in Equation A.20:

$$\begin{aligned}\boldsymbol{W}_L &= \frac{1}{2}(\boldsymbol{w}_i + \boldsymbol{w}_{i-1}) - \frac{1}{6}(\delta\boldsymbol{w}_i - \delta\boldsymbol{w}_{i-1}),\\ \boldsymbol{W}_R &= \frac{1}{2}(\boldsymbol{w}_{i+1} + \boldsymbol{w}_i) - \frac{1}{6}(\delta\boldsymbol{w}_{i+1} - \delta\boldsymbol{w}_i).\end{aligned} \tag{A.34}$$

However, in PPMP the slopes are limited in the primitive variables using the van Leer (1977) limiter. The slopes $\delta\boldsymbol{w}_{i-1}$, $\delta\boldsymbol{w}_i$, and $\delta\boldsymbol{w}_{i+1}$ are all calculated following Equations A.2 and A.3.

Contact Discontinuity Steepening

Once the interface values have been reconstructed, contact discontinuity steepening is applied to the interface values for the density, $\boldsymbol{W}_L[0] = \rho_L$ and $\boldsymbol{W}_R[0] = \rho_R$. Whether steepening is applied depends on a number of necessary criteria. First, to avoid steepening density jumps owing to numerical noise, steepening is only applied if the density difference between cells exceeds a minimum relative size:

$$\frac{|\rho_{i+1} - \rho_{i-1}|}{\min(\rho_{i+1}, \rho_{i-1})} > 0.01. \tag{A.35}$$

If the density jump is large enough, we further require that the pressure jump across the cell be sufficiently small:

$$\frac{|p_{i+1} - p_{i-1}|}{\min(p_{i+1}, p_{i-1})} < 0.1\gamma\frac{|\rho_{i+1} - \rho_{i-1}|}{\min(\rho_{i+1}, \rho_{i-1})}. \tag{A.36}$$

Next, the second derivative of the density distribution across cells $i-1$ and $i+1$ is estimated as

$$\delta^2\rho_i = \frac{\rho_{i+1} - 2\rho_i + \rho_{i-1}}{6\Delta x^2}. \tag{A.37}$$

The product of the second derivatives then determines whether the local density profile on either side of the cell i has the same concavity by requiring

$$\delta^2\rho_{i-1}\delta^2\rho_{i+1} > 0. \tag{A.38}$$

Assuming the three conditions listed in Equations A.35, A.36, and A.38 are satisfied, a dimensionless quantity involving the first and third derivatives of density is calculated as

$$\tilde{\eta}_i = -\frac{(\delta^2\rho_{i+1} - \delta^2\rho_{i-1})\Delta x^2}{\rho_{i+1} - \rho_{i-1}}. \tag{A.39}$$

This quantity is used to compute the *steepening coefficient* η_i from the parameters determined heuristically in Colella and Woodward (1984):

$$\eta_i = \max\left[0, \min(20\tilde{\eta}_i - 1, 1)\right]. \tag{A.40}$$

The steepening coefficient η_i and the monotonized slopes $\delta\boldsymbol{w}_{i-1}[0] = \delta\rho_{i-1}$ and $\delta\boldsymbol{w}_{i+1}[0] = \delta\rho_{i+1}$ are then used to steepen the left and right interface density values, providing

$$\begin{aligned} \rho_L &= \rho_L(1-\eta_i) + (\rho_{i-1} + 0.5\delta\rho_{i-1})\eta_i \\ \rho_R &= \rho_R(1-\eta_i) + (\rho_{i-1} - 0.5\delta\rho_{i+1})\eta_i \end{aligned} \tag{A.41}$$

Shock Flattening

Because of their self-steepening property, shocks in PPMP can become under-resolved, i.e. narrow enough to be contained within a single cell. Tests have demonstrated that shocks contained within a single cell tend to lead to severe oscillations near the shock front, while those spread over two or more cells do not pose a problem (Colella and Woodward, 1984; Fryxell et al., 2000). A solution is to flatten numerically the interpolation near problematic shocks, reverting to a first-order reconstruction in circumstances where the shock is empirically deemed too narrow. To determine whether a shock needs flattening, a shock steepness parameter S is calculated to compare the pressure gradient across two and four cells:

$$S_i = \frac{p_{i+1} - p_{i-1}}{p_{i+2} - p_{i-2}} \tag{A.42}$$

The steepness parameter is used to construct a dimensionless coefficient

$$\tilde{F}_i = \max(0, \min[1, 10(S_i - 0.75)]) \tag{A.43}$$

that may cover the range $\tilde{F}_i = [0, 1]$. This formulation is designed to ensure that only shocks contained within fewer than two cells are steepened. Further, we set $\tilde{F}_i = 0$ if the relative pressure jump is not large and the shock is not steep, when

$$\frac{|p_{i+1} - p_{i-1}|}{\min(p_{i+1}, p_{i-1})} < \frac{1}{3} \tag{A.44}$$

or if the velocity gradient is positive (indicating that the fluid is not being compressed in the direction along which we are reconstructing boundary values), when

$$u_{i+1} - u_{i-1} > 0. \tag{A.45}$$

The same rules are applied to the dimensionless parameters $\tilde{F}_{i-1}$ and $\tilde{F}_{i+1}$. This procedure means that PPMP requires a stencil with three cells on both sides of cell i. Here we are calculating shocks in the x-direction; in the y- and z-directions, the components v and w are used to test the velocity gradient. The final flattening coefficient for cell i is set as

$$F_i = \begin{cases} \max(\tilde{F}_i, \tilde{F}_{i+1}), & \text{if } p_{i+1} - p_{i-1} < 0, \\ \max(\tilde{F}_i, \tilde{F}_{i-1}), & \text{otherwise.} \end{cases} \tag{A.46}$$

We use this value of F_i to modify the interface values. Unlike in discontinuity steeping, for shock flattening every primitive variable is modified:

$$\begin{aligned} \boldsymbol{W}_L &= F_i \boldsymbol{w}_i + (1 - F_i)\boldsymbol{W}_L \\ \boldsymbol{W}_R &= F_i \boldsymbol{w}_i + (1 - F_i)\boldsymbol{W}_R \end{aligned} \tag{A.47}$$

If $F_i = 0$, the expression has no effect on the interface values. If $F_i = 1$ the zone average values are used for the interface values, effectively replacing the original limited slope for cell i with a flatter slope.

Monotonicity

The next step in the reconstruction is to ensure that the parabolic distribution of each of the variables is monotonic by checking for local maxima and minima and modifying steep gradients, as in Equations A.21 and A.22:

$$\begin{aligned} \boldsymbol{W}_L = \boldsymbol{W}_R = \boldsymbol{w}_i, &\quad \text{if } (\boldsymbol{W}_R - \boldsymbol{w}_i)(\boldsymbol{w}_i - \boldsymbol{W}_L) <= 0 \\ \boldsymbol{W}_L = 3\boldsymbol{w}_i - 2\boldsymbol{W}_R, &\quad \text{if } (\boldsymbol{W}_R - \boldsymbol{W}_L)[\boldsymbol{W}_L - (3\boldsymbol{w}_i - 2\boldsymbol{W}_R)] < 0 \\ \boldsymbol{W}_L = 3\boldsymbol{w}_i - 2\boldsymbol{W}_L, &\quad \text{if } (\boldsymbol{W}_R - \boldsymbol{W}_L)[(3\boldsymbol{w}_i - 2\boldsymbol{W}_L) - \boldsymbol{W}_R] < 0. \end{aligned} \tag{A.48}$$

Calculation of the Input States

By this stage, reconstruction of the boundary values has been completed and the input states for the Riemann problem can be calculated. Once again, the characteristic tracing method of Colella and Woodward (1984) is used. First, the speeds of the three characteristics are defined as in PLMC and PPMC:

$$\lambda^m = u_i - a_i, \quad \lambda^0 = u_i, \quad \lambda^p = u_i + a_i. \tag{A.49}$$

Again, a_i is the sound speed in cell i calculated using average values of the density and pressure. Because we have adjusted the boundary values from the original parabolic interpolation, we must adjust the values of the slopes across the cell so that the parabolic interpolation retains the correct average value:

$$\delta \boldsymbol{w}_i = \boldsymbol{W}_R - \boldsymbol{W}_L, \quad \boldsymbol{w}_6 = 6\left[\boldsymbol{w}_i - 0.5(\boldsymbol{W}_L + \boldsymbol{W}_R)\right]. \tag{A.50}$$

We now define α and β, two variables that are associated with the characteristic wave speeds approaching the left and right interfaces,

$$\begin{aligned}
\alpha^m &= \frac{\Delta t}{\Delta x}\min(\lambda^m, 0), \quad \beta^m = \frac{\Delta t}{\Delta x}\max(\lambda^m, 0),\\
\alpha^0 &= \frac{\Delta t}{\Delta x}\min(\lambda^0, 0), \quad \beta^0 = \frac{\Delta t}{\Delta x}\max(\lambda^0, 0),\\
\alpha^p &= \frac{\Delta t}{\Delta x}\min(\lambda^p, 0), \quad \beta^p = \frac{\Delta t}{\Delta x}\max(\lambda^p, 0).
\end{aligned} \tag{A.51}$$

We use these variables to calculate a time-evolved boundary value associated with each characteristic:

$$\begin{aligned}
\boldsymbol{W}_L^m &= \boldsymbol{W}_L - \frac{1}{2}\alpha^m\left[\delta\boldsymbol{w}_i + \boldsymbol{w}_6(1 + \frac{2}{3}\alpha^m)\right]\\
\boldsymbol{W}_L^0 &= \boldsymbol{W}_L - \frac{1}{2}\alpha^0\left[\delta\boldsymbol{w}_i + \boldsymbol{w}_6(1 + \frac{2}{3}\alpha^0)\right]\\
\boldsymbol{W}_L^p &= \boldsymbol{W}_L - \frac{1}{2}\alpha^p\left[\delta\boldsymbol{w}_i + \boldsymbol{w}_6(1 + \frac{2}{3}\alpha^p)\right]\\
\boldsymbol{W}_R^m &= \boldsymbol{W}_R - \frac{1}{2}\beta^m\left[\delta\boldsymbol{w}_i - \boldsymbol{w}_6(1 - \frac{2}{3}\beta^m)\right]\\
\boldsymbol{W}_R^0 &= \boldsymbol{W}_R - \frac{1}{2}\beta^0\left[\delta\boldsymbol{w}_i - \boldsymbol{w}_6(1 - \frac{2}{3}\beta^0)\right]\\
\boldsymbol{W}_R^p &= \boldsymbol{W}_R - \frac{1}{2}\beta^p\left[\delta\boldsymbol{w}_i - \boldsymbol{w}_6(1 - \frac{2}{3}\beta^p)\right]
\end{aligned} \tag{A.52}$$

For example, $\boldsymbol{W}_L^m$ is the time-evolved boundary value at the left interface of the cell obtained by integrating under the characteristic associated with the left acoustic wave. If the fluid flow is supersonic toward the right, the left acoustic wave is not approaching the interface, and $\boldsymbol{W}_L^m$ is simply equal to the reconstructed boundary value. The same integration appeared in Equation A.25, though in this case we explicitly integrate under every characteristic and not just the characteristic approaching the interface with the greatest speed. For the density, normal velocity, and pressure, we refer to the value calculated using the characteristic approaching the interface at the greatest speed as the "reference state", e.g. $\boldsymbol{W}_L^m$ for the left interface and $\boldsymbol{W}_R^p$ for the right. As with PPMC, this reference state is our first guess at the input state for the Riemann problem. For the transverse velocities, we use the states associated with the advection speed, $\boldsymbol{W}_L^0$ and $\boldsymbol{W}_R^0$.

These reference states are only first-order accurate approximations and the input states can be further corrected to account for the presence of other characteristics approaching the interface. Following the notation in Colella and Woodward (1984), the description of the algorithm continues in terms of the primitive variables. The sound speeds for the reference states on the left and right of the cell are computed

$$a_L = \sqrt{\frac{\gamma p_L^m}{\rho_L^m}} \quad \text{and} \quad a_R = \sqrt{\frac{\gamma p_R^p}{\rho_R^p}}, \tag{A.53}$$

along with correction terms that are added to the reference state,

$$\begin{aligned} \chi_L^p &= -\frac{1}{2\rho_L^m a_L}\left(u_L^m - u_L^p - \frac{p_L^m + p_L^p}{\rho_L^m a_L}\right), \\ \chi_R^m &= \frac{1}{2\rho_s^p a_R}\left(u_R^p - u_R^m - \frac{p_R^p - p_R^m}{\rho_R^p a_R}\right), \\ \chi_L^0 &= \frac{p_L^m - p_L^0}{(\rho_L^m a_L)^2} + \frac{1}{\rho_L^m} - \frac{1}{\rho_L^0}, \\ \chi_R^0 &= \frac{p_R^p - p_R^0}{(\rho_R^p a_R)^2} + \frac{1}{\rho_R^p} - \frac{1}{\rho_R^0}. \end{aligned} \tag{A.54}$$

In the event that the characteristic is not traveling toward the interface these correction terms are set to zero,

$$\begin{aligned} \chi_L^0 &= 0 \quad \text{if} \lambda^0 > 0, \\ \chi_L^p &= 0 \quad \text{if} \lambda^p > 0, \\ \chi_R^m &= 0 \quad \text{if} \lambda^m < 0, \\ \chi_R^0 &= 0 \quad \text{if} \lambda^0 < 0. \end{aligned} \tag{A.55}$$

The correction terms are then used with the reference state integration to calculate the final input states for the Riemann problem on each side of the cell:

$$\begin{aligned} \rho_L = \left(\frac{1}{\rho_L^m} - \chi_L^0 - \chi_L^p\right)^{-1}, &\quad \rho_{R,i} = \left(\frac{1}{\rho_R^p} - \chi_R^m - \chi_R^0\right)^{-1}, \\ u_L = u_L^m + \rho_L^m a_L \chi_L^p, &\quad u_R = u_R^p - \rho_R^p a_R \chi_R^m, \\ p_L = p_L^m + (\rho_L^m a_L)^2 \chi_L^p, &\quad p_R = p_R^p + (\rho_R^p a_R)^2 \chi_R^m, \\ v_L = v_L^0, &\quad v_R = v_R^0 \\ w_L = w_L^0, &\quad w_R = w_R^0. \end{aligned} \tag{A.56}$$

APPENDIX B

RIEMANN SOLVERS IN CHOLLA

B.0.1 The Exact Solver

The exact solver used in *Cholla* follows the solver presented in Chapter 4 of Toro (2009), adapted for implementation in CUDA C. The algorithm to solve the Riemann problem is presented below, using an x-interface as an example. In the following section the CTU notation from Section 2.2.1 is used, where states are labeled at the left and right of the interface.

First, the input states at the left and right of the interface are converted to the primitive variables, $\boldsymbol{W}_L^*$ and $\boldsymbol{W}_R^*$. (Between GPU kernels like the interface reconstruction and the Riemann problem, calculated values are stored in the conserved form.) These vectors are used to compute the corresponding sound speed on either side of the interface

$$a_L = \sqrt{\frac{\gamma p_L}{\rho_L}} \quad \text{and} \quad a_R = \sqrt{\frac{\gamma p_R}{\rho_R}}. \tag{B.1}$$

To determine the Riemann solution, the exact pressure p^m and velocity u^m in the intermediate state must be computed (see Figure 2.2). We use the Toro (2009) primitive variable Riemann solver to provide an initial approximation to the intermediate state pressure, given by

$$\tilde{p} = 0.5(p_L + p_R) + 0.125(u_L - u_R)(\rho_L + \rho_R)(a_L + a_R). \tag{B.2}$$

Because $\tilde{p}$ is an approximation and the solution for p cannot be negative, we set $\tilde{p} = 0$ if the calculated pressure is below zero. A two-shock Riemann solver is then used to calculate a more accurate estimate,

$$p_0 = \frac{g_L p_L + g_R p_R - (u_R - u_L)}{g_L + g_R}, \tag{B.3}$$

with

$$g_k = \sqrt{\frac{A_k}{\tilde{p} + B_k}}, \quad A_k = \frac{2}{(\gamma+1)\rho_k}, \quad B_k = \frac{\gamma - 1}{\gamma + 1} p_k, \quad \text{where } k = L,\ R. \tag{B.4}$$

To maintain positivity, a pressure floor of $p_0 \geq 10^{-6}$ is enforced in this estimate. The pressure p_0 is then used as a starting point in a Newton-Raphson iteration to compute the exact solution for the pressure in the intermediate region. We define the pressure functions f_L and f_R,

$$f_k = \begin{cases} \frac{2a_k}{\gamma-1}\left[\left(\frac{p_0}{p_k}\right)^{\frac{\gamma-1}{2\gamma}} - 1\right] & \text{if } p_0 \leq p_k \text{ (rarefaction)}, \\ (p_0 - p_k)\left(\frac{A_k}{p_0 + B_k}\right)^{\frac{1}{2}} & \text{if } p_0 > p_k \text{ (shock)}, \end{cases} \tag{B.5}$$

and their first derivatives f_L' and f_R',

$$f_k' = \begin{cases} \frac{1}{\rho_k a_k}\left(\frac{p_0}{p_k}\right)^{-\frac{\gamma+1}{2\gamma}} & \text{if } p_0 \leq p_k \text{ (rarefaction)}, \\ \left(\frac{A_k}{B_k + p_0}\right)^{\frac{1}{2}}\left[1 - \frac{p_0 - p_k}{2(B_k + p_0)}\right] & \text{if } p_0 > p_k \text{ (shock)}. \end{cases} \tag{B.6}$$

Again, $k = L$ or R, and A_k and B_k are as defined above. These quantities are used to calculate the pressure in the intermediate state, p^m,

$$p^m = p_0 - \frac{f_L + f_R + u_R - u_L}{f_L' + f_R'}. \tag{B.7}$$

We then compare the newly computed pressure, p^m, to the previously computed pressure,

$$\Delta = 2\frac{|p^m - p_0|}{|p^m + p_0|}. \tag{B.8}$$

If Δ is greater than a relative tolerance (e.g., 10^{-6}), the values of f_k, f_k', and p^m are recomputed using the updated value of p^m in place of p_0 in Equations B.5 - B.8. When Δ is less than the tolerance, the procedure halts. Having calculated a suitably accurate pressure p^m, the pressure can then be used to compute the velocity in the intermediate state as

$$u^m = 0.5(u_L + u_R + f_R - f_L). \tag{B.9}$$

Once the values of p^m and u^m in the intermediate state have been computed, the values for each of the primitive variables can be calculated at the cell interface.

To do this, a number of conditions are tested to determine where in the Riemann solution the interface lies. For pure hydrodynamics there are ten possible outcomes.

If $u^m \geq 0$, the contact discontinuity is to the right of the cell interface. We then check to see if there is a rarefaction wave on the left, i.e. if $p^m \leq p_L$. If so, there are three possible solutions.

- If $u_L - a_L \geq 0$ the interface is in the left data state, and the solution is simply the input data on the left:

$$\rho = \rho_L, \quad u = u_L, \quad p = p_L. \tag{B.10}$$

- If $u^m - a_L \left(\frac{p^m}{p_L}\right)^{\frac{\gamma-1}{2\gamma}} < 0$ the interface is in the intermediate data state to the right of the fan, but left of the contact:

$$\rho = \rho_L \left(\frac{p^m}{p_L}\right)^{\frac{1}{\gamma}}, \quad u = u^m, \quad p = p^m. \tag{B.11}$$

- Otherwise, the interface is within the rarefaction fan:

$$\begin{aligned} &\rho = \rho_L \left(\frac{a}{a_L}\right)^{\frac{2}{\gamma-1}}, \quad u = a, \quad p = p_L \left(\frac{a}{a_L}\right)^{\frac{2\gamma}{\gamma-1}}, \\ &\text{where } a = \frac{2}{\gamma+1}\left(a_L + \frac{\gamma-1}{2}u_L\right). \end{aligned} \tag{B.12}$$

If there is a shock to the left, rather than a rarefaction, i.e. if $p^m > p_L$, the shock speed is calculated as

$$s_L = u_L - a_L\sqrt{\frac{(\gamma+1)}{2\gamma}\frac{p^m}{p_L} + \frac{\gamma-1}{2\gamma}}. \tag{B.13}$$

- If $s_L \geq 0$ the interface samples the left data state:

$$\rho = \rho_L, \quad u = u_L, \quad p = p_L. \tag{B.14}$$

- Otherwise, the interface samples the intermediate data state to the left of the contact:

$$\rho = \rho_L \left(\frac{p^m}{p_L} + \frac{\gamma-1}{\gamma+1}\right) / \left(\frac{p^m}{p_L}\frac{\gamma-1}{\gamma+1} + 1\right), \quad u = u^m, \quad p = p^m. \tag{B.15}$$

If instead $u^m < 0$, the contact discontinuity is to the left of the cell interface and there is a similar set of five possible outcomes. If there is a rarefaction wave on the right of the Riemann solution, i.e. if $p^m \le p_R$ there are three possibilities:

- If $u_R + a_R \le 0$ the interface samples the right data state:

$$\rho = \rho_R, \quad u = u_R, \quad p = p_R. \tag{B.16}$$

- If $u^m + a_R \left(\frac{p^m}{p_R}\right)^{\frac{\gamma-1}{2\gamma}} \ge 0$ the interface samples the intermediate state to the left of the fan, but right of the contact:

$$\rho = \rho_R \left(\frac{p^m}{p_R}\right)^{\frac{1}{\gamma}}, \quad u = u^m, \quad p = p^m. \tag{B.17}$$

- Otherwise, the interface samples the rarefaction fan:

$$\begin{aligned} &\rho = \rho_R \left(\frac{a}{a_R}\right)^{\frac{2}{\gamma-1}}, \quad u = -a, \quad p = p_R \left(\frac{a}{a_R}\right)^{\frac{2\gamma}{\gamma-1}}, \\ &\text{where } a = \frac{2}{\gamma+1}\left(a_R + \frac{\gamma-1}{2}u_R\right). \end{aligned} \tag{B.18}$$

If $p^m > p_R$ there is a shock to the right rather than a rarefaction, the shock speed is calculated as

$$s_L R = u_R + a_R\sqrt{\frac{(\gamma+1)}{2\gamma}\frac{p^m}{p_R} + \frac{\gamma-1}{2\gamma}}. \tag{B.19}$$

- If $s_R \le 0$ the interface samples the right data state:

$$\rho = \rho_R, \quad u = u_R, \quad p = p_R. \tag{B.20}$$

- Otherwise, the interface samples the intermediate data state to the right of the contact:

$$\rho = \rho_R \left(\frac{p^m}{p_R} + \frac{\gamma-1}{\gamma+1}\right) / \left(\frac{p^m}{p_R}\frac{\gamma-1}{\gamma+1} + 1\right), \quad u = u^m, \quad p = p^m. \tag{B.21}$$

After determining where in the Riemann solution the interface samples, the evolved primitive variables ρ, u, and p at the cell interface are determined. The fluxes of the conserved variables can then be calculated following Equation 2.11:

$$\boldsymbol{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v_k \\ \rho u w_k \\ (E+p)u \end{bmatrix}, \tag{B.22}$$

where $k = L$ or R; L if $u \geq 0$, and R if $u < 0$. These conditions reflect the fact that transverse velocities are simply advected with the flow.

B.0.2 The Roe Solver

Rather than using an expensive iterative procedure to calculate the exact solution to the Riemann problem, the Roe (1981) Riemann solver calculates an exact solution to a linearized version of the Euler equations. Below the procedure for calculating the Roe fluxes at an x-interface is detailed. The y- and z-interface calculations are identical modulo an appropriate change of variables. Should the approximate Roe solver fail, we include a failsafe based on the method of Stone et al. (2008) where HLLE fluxes (Harten et al., 1983; Einfeldt, 1988) are substituted. The Roe and HLLE solvers are described below.

The Roe solver starts with the calculated input vectors of primitive variables at the left and right of an interface, $\boldsymbol{W}_L^*$ and $\boldsymbol{W}_R^*$. Fluxes corresponding to the left and right state, $\boldsymbol{F}_L$ and $\boldsymbol{F}_R$, are calculated following Equation 2.11. The Roe average state, $\tilde{\boldsymbol{u}} = (\tilde{\rho}, \tilde{u}, \tilde{v}, \tilde{w}, \tilde{H})^T$, is then computed:

$$\begin{aligned} \tilde{\rho} &= \sqrt{\rho_L}\sqrt{\rho_R} \\ \tilde{u} &= (\sqrt{\rho_L}u_L + \sqrt{\rho_R}u_R)/(\sqrt{\rho_L} + \sqrt{\rho_R}) \\ \tilde{v} &= (\sqrt{\rho_L}v_L + \sqrt{\rho_R}v_R)/(\sqrt{\rho_L} + \sqrt{\rho_R}) \\ \tilde{w} &= (\sqrt{\rho_L}w_L + \sqrt{\rho_R}w_R)/(\sqrt{\rho_L} + \sqrt{\rho_R}) \\ \tilde{H} &= (\sqrt{\rho_L}H_L + \sqrt{\rho_R}H_R)/(\sqrt{\rho_L} + \sqrt{\rho_R}), \end{aligned} \tag{B.23}$$

with the enthalpy, $H = (E + p)/\rho$, used instead of the pressure. The average sound speed

$$\tilde{a} = \sqrt{(\gamma - 1)(\tilde{H} - 0.5\tilde{\mathbf{V}}^2)}, \quad \text{where } \tilde{\mathbf{V}}^2 = \tilde{u}\tilde{u} + \tilde{v}\tilde{v} + \tilde{w}\tilde{w} \tag{B.24}$$

is also needed. These Roe average states are used to calculate the eigenvalues of the Roe Jacobian,

$$\lambda^m = \tilde{u} - \tilde{a}, \quad \lambda^0 = \tilde{u}, \quad \text{and} \quad \lambda^p = \tilde{u} + \tilde{a}. \tag{B.25}$$

If the flow is supersonic ($\lambda^m \geq 0$ to the right, or $\lambda^p \leq 0$ to the left), the solver returns the appropriate left or right state fluxes $\boldsymbol{F}_L$ or $\boldsymbol{F}_R$. If flow is subsonic, the calculation of the Roe fluxes proceeds.

Differences in the conserved variables between the left and right states are computed:

$$\begin{aligned}
\delta\rho &= \rho_R - \rho_L \\
\delta m_x &= m_{x,R} - m_{x,L} \\
\delta m_y &= m_{y,R} - m_{y,L} \\
\delta m_z &= m_{z,R} - m_{z,L} \\
\delta E &= E_R - E_L.
\end{aligned} \tag{B.26}$$

These differences are projected onto the characteristics by multiplying by the left eigenvector associated with each eigenvalue. The resulting characteristics are the wave strengths, $\boldsymbol{\xi}$, from Equation 2.27:

$$\begin{aligned}
\xi_0 &= \delta\rho N_a(0.5\gamma'\tilde{\mathbf{V}}^2 + \tilde{u}\tilde{a}) - \delta m_x N_a(\gamma'\tilde{u} + \tilde{a}) - \delta m_y N_a\gamma'\tilde{v} - \delta m_z N_a\gamma'\tilde{w} + \delta E N_a\gamma' \\
\xi_1 &= -\delta\rho\tilde{v} + \delta m_y \\
\xi_2 &= -\delta\rho\tilde{w} + \delta m_z \\
\xi_3 &= \delta\rho(1 - N_a\gamma'\tilde{\mathbf{V}}^2) + \delta m_x\gamma'\tilde{u}/\tilde{a}^2 + \delta m_y\gamma'\tilde{v}/\tilde{a}^2 + \delta m_z\gamma'\tilde{w}/\tilde{a}^2 - \delta E\gamma'/\tilde{a}^2 \\
\xi_4 &= \delta\rho N_a(0.5\gamma'\tilde{\mathbf{V}}^2 - \tilde{u}\tilde{a}) - \delta m_x N_a(\gamma'\tilde{u} - \tilde{a}) - \delta m_y N_a\gamma'\tilde{v} - \delta m_z N_a\gamma'\tilde{w} + \delta E N_a\gamma'
\end{aligned} \tag{B.27}$$

where $N_a = 1/(2\tilde{a}^2)$ and $\gamma' = \gamma - 1$. Numeric subscripts denote the elements of $\boldsymbol{\xi}$. Each characteristic variable is multiplied by its associated eigenvalue, yielding a vector of coefficients, $\boldsymbol{C}$:

$$C_0 = \xi_0|\lambda^m|, \quad C_1 = \xi_1|\lambda^0|, \quad C_2 = \xi_2|\lambda^0|, \quad C_3 = \xi_3|\lambda^0|, \quad C_4 = \xi_4|\lambda^p| \tag{B.28}$$

The product of these coefficients with the right eigenvectors are then summed, keeping track of the summation in the vector $\boldsymbol{s}$:

$$\begin{aligned}
s_0 &= C_0 + C_3 + C_4 \\
s_1 &= C_0(\tilde{u} - \tilde{a}) + C_3\tilde{u} + C_4(\tilde{u} + \tilde{a}) \\
s_2 &= C_0\tilde{v} + C_1 + C_3\tilde{v} + C_4\tilde{v} \\
s_3 &= C_0\tilde{w} + C_2 + C_3\tilde{2} + C_4\tilde{w} \\
s_4 &= C_0(\tilde{H} - \tilde{u}\tilde{a}) + C_1\tilde{v} + C_2\tilde{w} + 0.5C_3\tilde{\mathbf{V}}^2 + C_4(\tilde{H} + \tilde{u}\tilde{a})
\end{aligned} \tag{B.29}$$

By this stage, all information needed to compute the Roe fluxes has been obtained. The Roe fluxes are then computed as

$$\boldsymbol{F}_{Roe} = \frac{1}{2}\left(\boldsymbol{F}_L + \boldsymbol{F}_R - \boldsymbol{s}\right) \tag{B.30}$$

Before returning these fluxes, however, the intermediate states are examined for possible negative densities or pressures. The intermediate states, labeled $\boldsymbol{U}_L^m$ and $\boldsymbol{U}_R^m$ in analogy with the exact Riemann solver, are calculated by projecting the characteristic variables onto the conserved variables. Each characteristic variable is multiplied by its associated right eigenvector and the results summed, in turn, to the left state. The left intermediate state, $\boldsymbol{U}_L^m$, is calculated as

$$\begin{aligned}
\rho_L^m &= \rho_L + \xi_0 \\
\rho u_L^m &= \rho u_L + \xi_0(\tilde{u} - \tilde{a}) \\
\rho v_L^m &= \rho v_L + \xi_0\tilde{v} \\
\rho w_L^m &= \rho w_L + \xi_0\tilde{w} \\
E_L^m &= E_L + \xi_0(\tilde{H} - \tilde{u}\tilde{a}).
\end{aligned} \tag{B.31}$$

If $\lambda^0 > \lambda^m$, we check for negative density and pressure. We then move on to the right intermediate state, $\boldsymbol{U}_R^m$:

$$\begin{aligned} \rho_R^m &= \rho_L^m + \xi_3 \\ \rho u_R^m &= \rho u_L^m + \xi_3 \tilde{u} \\ \rho v_R^m &= \rho v_L^m + \xi_1 + \xi_3 \tilde{v} \\ \rho w_R^m &= \rho w_L^m + \xi_2 + \xi_3 \tilde{w} \\ E_R^m &= E_L^m + \xi_1 \tilde{v} + \xi_2 \tilde{w} + 0.5 \xi_3 \tilde{\mathbf{V}}^2. \end{aligned} \tag{B.32}$$

If $\lambda^p > \lambda^0$ then negative densities and pressures are possible and, if present, the Roe fluxes are replaced with HLLE fluxes. Otherwise the Roe fluxes calculated in Equation B.30 are returned.

Little additional work must be done to calculate the HLLE fluxes since many of the required quantities are computed for the Roe fluxes. The HLLE solver constructs a single average state between the right and left input states, neglecting the contact wave. Ignoring the contact wave means that density discontinuities diffuse quickly, but the HLLE solver has the advantage of always producing positive intermediate densities and pressures, as shown in Einfeldt et al. (1991). The HLLE flux algorithm starts with the computation of the sound speed for the left and right input states, a_L and a_R. Bounding speeds are then calculated, defined by the minimum and maximum Roe eigenvalues and the left and right acoustic waves:

$$b^m = \min[\min(\lambda^m, u_L - a_L), 0], \quad b^p = \max[\max(\lambda^p, u_R + a_R), 0] \tag{B.33}$$

These speeds are used to compute left and right fluxes:

$$\boldsymbol{F}_L = \boldsymbol{F}(\boldsymbol{W}_L) - b^m \boldsymbol{U}_L \quad \text{and} \quad \boldsymbol{F}_R = \boldsymbol{F}(\boldsymbol{W}_R) - b^p \boldsymbol{U}_R, \tag{B.34}$$

where $\boldsymbol{F}(\boldsymbol{W})$ is calculated using Equation 2.11, and $\boldsymbol{U}$ refers to the conserved variables on either side of the interface, i.e. the input states for the Riemann problem. These intermediate fluxes are then employed to compute the HLLE fluxes, given by

$$\boldsymbol{F}_{\text{HLLE}} = \frac{1}{2}\left[(\boldsymbol{F}_L + \boldsymbol{F}_R) + (\boldsymbol{F}_L - \boldsymbol{F}_R)\left(\frac{b^p + b^m}{b^p - b^m}\right)\right]. \tag{B.35}$$

If the Roe solver fails during the transverse flux or conserved variable update steps of the CTU algorithm, these HLLE fluxes may be used.

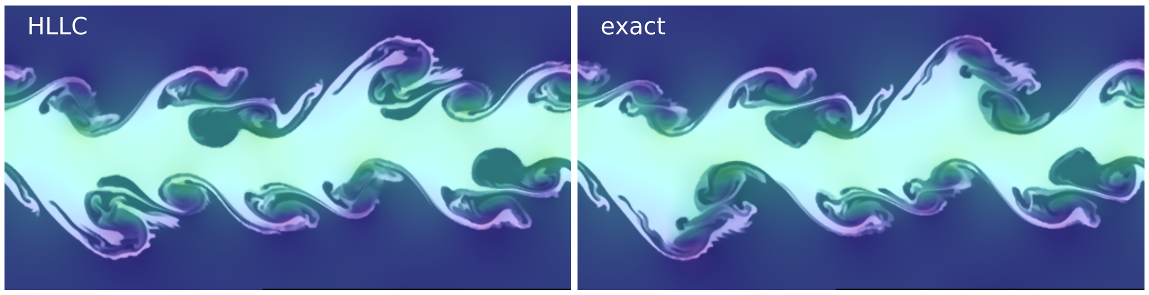

Figure B.1: We compare the results of a 2D Kelvin-Helmholtz simulation, starting with a discontinuous interface, on the left evolved using an HLLC Riemann solver, and on the right using an exact solver. Although the HLLC solver is an approximation, the contact discontinuity is well resolved with both methods.

B.0.3 The HLLC Solver

We also employ an HLLC Riemann solver for this work (see Harten et al., 1983; Toro et al., 1994). The *Cholla* implementation of the HLLC solver follows the description in Batten et al. (1997), which revises the original HLLC solver presented in Toro et al. (1994) with updated maximum and minimum wave speeds that account for the possibility of colliding shocks within a cell. The HLLC solver has several advantages over the exact and Roe Riemann solvers presented in the previous sections. Unlike the exact solver, the HLLC solver does not require an iterative procedure to calculate the intermediate-state pressure. This feature makes it a more natural fit to the GPU thread model, which works best on algorithms without thread divergence (i.e. fewer "if" statements). Also, like all Riemann solvers in the HLL family, the HLLC is positive-definite (Batten et al., 1997), meaning that it never produces negative pressures or densities in the intermediate state solution (Einfeldt et al., 1991). These unphysical solutions can arise using linearized solvers like the Roe, which require "fallback" techniques in failure scenarios (see, e.g., Lemaster and Stone, 2009). Standard hydrodynamic tests show that the HLLC solver provides a comparable level of accuracy to the Roe and exact solvers, including problems that involve contact discontinuities (Batten et al., 1997). We qualitatively illustrate this in Figure B.1, which compares well-resolved contact interfaces in Kelvin-Helmholtz simulations using the HLLC solver vs an exact solver.

APPENDIX C

THE H CORRECTION

As mentioned in the main text, strictly upwind multidimensional integration algorithms like CTU are susceptible to a particular type of numerical pathology, referred to as the carbuncle instability owing to its appearance (Quirk, 1994). The carbuncle instability arises in problems containing strong, grid-aligned shocks as a result of insufficient dissipation when inserting a one dimensional Riemann flux into a multidimensional problem. In the scenario when a planar shock is present perpendicular to the direction of the fluid flow, crossflow oscillations arise that grow and lead to severe inaccuracies in the numerical model. To correct this phenomenon, *Cholla* implements a solution devised by Sanders et al. (1998) that accounts for the perpendicular velocities when calculating the Riemann flux normal to an interface. The correction thereby adds sufficient dissipation to mitigate the carbuncle instability. This method for suppressing the carbuncle instability is called *H correction* in reference to the shape of the required stencil.

In *Cholla*, H Correction is implemented in conjunction with the Roe Riemann solver. The correction is only applied to the fluxes calculated during the second set of Riemann problems, between the transverse flux update and the final conserved variable update of the CTU algorithm. To apply the H correction, at every interface a quantity η is computed that depends on the velocity normal to the interface and the sound speed. For the x-interface at $(i+\frac{1}{2}, j)$ in a two dimensional simulation, this quantity can be computed as

$$\eta_{(i+\frac{1}{2},j)} = \frac{1}{2}|(u_R + a_R) - (u_L + a_L)|, \tag{C.1}$$

where u_R and u_L are the normal velocity at the interface calculated using the right and left input states, and a_L and a_R are the corresponding sound speeds. For y- and z-interfaces, the velocity components v and w would be used. Once a value of

η has been calculated for each interface, a further quantity η^H is computed for each interface by finding the maximum value of η for each of the surrounding interfaces:

$$\eta^H_{(i+\frac{1}{2},j)} = \max[\eta_{(i-1,j+\frac{1}{2})}, \eta_{(i-1,j-\frac{1}{2}}, \eta_{(i+\frac{1}{2},j)}, \eta_{(i+1,j+\frac{1}{2})}, \eta_{(i+1,j-\frac{1}{2})}]. \tag{C.2}$$

For x-interfaces, information about the maximum wavespeeds at the adjacent y-interfaces that could affect the Riemann solution are thereby incorporated. In three dimensions, the calculation of η^H uses an appropriate nine-point stencil. The η^H quantity is used in conjunction with the Roe Riemann solver when calculating the CTU fluxes for an interface ($\boldsymbol{F}^{n+\frac{1}{2}}$, $\boldsymbol{G}^{n+\frac{1}{2}}$, and $\boldsymbol{H}^{n+\frac{1}{2}}$) by replacing the eigenvalues λ^α in Equation B.28 with

$$|\tilde{\lambda}^\alpha| = \max(|\lambda^\alpha|, \eta^H) \tag{C.3}$$

This substitution serves to add crossflow dissipation to the one-dimensional Roe fluxes and mitigates the carbuncle instability, as seen in Figure 2.18. Using the H correction comes at an additional cost, as its stencil requires adding an extra ghost cell on each side of the simulation in every dimension.

APPENDIX D

A SIMPLE INTEGRATION METHOD

In Chapter 2, we presented the *Cholla* implementation of the 6-step corner transport upwind (CTU) integration scheme originally described in Gardiner and Stone (2008b). The unsplit CTU integrator preserves symmetry and minimizes numerical diffusion, making the scheme a good choice for many magnetohydrodynamics problems (see Stone et al. 2008). Despite performing well in the standard Liska and Wendroff (2003b) tests, we found that the transverse flux correction step of CTU led to estimates for the 1D density and energy fluxes that produced negative densities and pressures after the final update, particularly when used to model multidimensional strong radiative shocks. In fact, our implementation of the CTU integration method always fails for the simulations in Chapter 3, regardless of the choice of interface reconstruction method or Riemann solver. Therefore, we instead employed a very simple, robust integration scheme for the simulations presented in Chapter 3, described below.

The integrator we use follows the initial steps of CTU, including piecewise parabolic reconstruction of the interface values in the primitive variables, characteristic time evolution of those interface values, and one-dimensional Riemann solutions at each interface to calculate fluxes. We implemented each of these initial steps as described in Chapter 2. However, rather than updating the interface values using the transverse fluxes, we simply skip to the final update and use only the one-dimensional fluxes to evolve the conserved quantities. Because the fluxes do not contain corrections for transverse directions, we found this method requires a very low CFL number to be stable - we use $cfl = 0.1$ for all the simulations in Chapter 3. The low CFL number makes the integrator expensive despite its simplicity, but when combined with a dual energy formalism the method is quite robust. Figure D.1 shows the results of this new integration scheme as compared to the

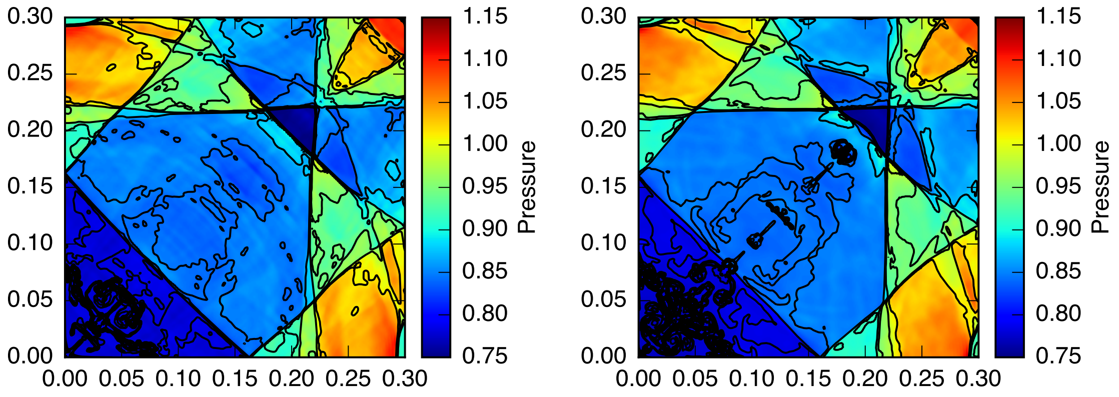

Figure D.1: A 2D implosion test, as presented in Chapter 2. A colored pressure map is overlaid with 36 density contours ranging from 0.125 to 1.0. On the left are the results at $t = 2.5$ for the integration scheme used for the simulations in the paper, on the right are the results at the same time using the CTU integrator. Both simulations used a CFL number of 0.1, piecewise parabolic reconstruction, and an HLLC Riemann solver. The additional diffusion in the integration scheme presented on the left prevents as clear a jet from forming, but also allows the turbulent cloud simulations presented in this work to run without failing (which proved impossible with CTU).

CTU method on the implosion test described in Liska and Wendroff (2003b). As the implosion test shows, this integrator is slightly more diffusive than CTU. The additional diffusion allows this simple integrator to succeed in circumstances where CTU fails.

APPENDIX E

DUAL ENERGY IN CHOLLA

The conservative nature of Godunov methods enables shock capturing and allows them to correctly model shock heating, both highly desirable qualities in a hydrodynamics modeling scheme. However, in scenarios where the kinetic energy of the gas is large relative to the internal energy, the total energy formulation can lead to negative values of the internal energy. These scenarios may arise in the physical models we investigate in Chapter 3, where high mach number turbulent flows interact with rapidly cooling gas. A negative value of the internal energy does not necessarily affect the dynamic behavior of the gas in such scenarios, which is dominated by the much larger kinetic energy. However, an accurate estimate of the internal energy is required to correctly calculate the gas temperature, which is needed to determine the correct radiative cooling source terms. Therefore, in such scenarios, a modification of the total energy update may improve the physical realism of the model. So-called "dual-energy" schemes that evolve both the internal energy and total energy simultaneously have proven to be a robust approach to fixing this problem (Ryu et al., 1993; Bryan et al., 1995). We employ a dual-energy formulation in *Cholla* that roughly follows the descriptions in Bryan et al. (2014) and Teyssier (2015). Below, we briefly outline the 1D version of the dual-energy update.

Without the dual-energy model, *Cholla* stores and evolves the conserved hydrodynamic variables: density ρ, the three components of the momentum vector, $\rho\mathbf{v} = (\rho v_x, \rho v_y, \rho v_z)^{\mathrm{T}}$, and the total energy, $E_{\mathrm{tot}} = \frac{1}{2}\rho\mathbf{v}^2 + e$, where e is the internal energy. With the dual-energy model, *Cholla* also explicitly tracks an estimated internal energy. For the majority of steps in the hydrodynamic update, including interface reconstruction and Riemann solutions, the internal energy is treated as a passively advected scalar in the same manner as the transverse velocities. However, during the final update of the conserved variables *Cholla* evolves the separately tracked internal energy in cell i according to the following non-conservative equation:

$$e_i^{n+1} = e_i^n + \frac{\delta t}{\delta x}\left[(F(e)_{i-\frac{1}{2}} - F(e)_{i+\frac{1}{2}}) + \frac{1}{2}P_i(v_{i-1} - v_{i+1})\right]. \tag{E.1}$$

The second term on the right hand side of this update captures the flux of internal energy at the cell interfaces. The third term on the right hand side encompasses the change in internal energy due to pressure forces. We use $1/2(v_{i-1} - v_{i+1})/\delta x$ as a one-dimensional estimate of the velocity derivative. Both the pressure P and velocities v_{i-1} and v_{i+1} used in Equation E.1 are calculated at time n, making the update only first-order accurate in time. However, in cases where the conservative total energy update results in a negative estimate of the internal energy this first order estimate is more accurate.

Once the new total energy and internal energy have been calculated at time $n+1$, we must determine which of the two internal energy calculations to use, the conservative estimate obtained by subtracting the kinetic energy from the total energy or the non-conservative estimate obtained with Equation E.1. In addition, the internal energy must be synchronized with the total energy after the update. Here we follow the decision tree outlined in Bryan et al. (2014). At the end of the hydrodynamic update, the conservative estimate of the internal energy in cell i, $e_{\text{cons}} = E_{\text{tot}} - \frac{1}{2}\rho\mathbf{v}^2$, is compared to the total energy in that cell. If the conservatively calculated internal energy is large enough (i.e., $e_{\text{cons}} > 0.001E_{\text{tot}}$), we use the conservative estimate for the updated internal energy: $e^{n+1} = e_{\text{cons}}$. In addition, to prevent the use of the nonconservative update in shocks, we compare the conservatively calculated internal energy to the maximum nearby total energy, $E_{\text{max}} = \max(E_{i-1}, E_i, E_{i+1})$. If $e_{\text{cons}} > 0.1E_{\text{max}}$, we again use the conservative estimate for the internal energy update, $e^{n+1} = e_{\text{cons}}$. However, if neither condition is met, we keep the non-conservative estimate for the separately tracked internal energy following Equation B1.

The last step in the dual-energy formulation is to synchronize the updated total energy with the updated internal energy. If the non-conservative estimate for e^{n+1} is used, the value of the total energy must be corrected by subtracting off the old conservatively calculated energy, and adding the new non-conservative estimate, i.e. $E_{\rm tot}^{n+1} = E_{\rm tot}^{n+1} - e_{\rm cons} + e^{n+1}$.

APPENDIX F

OPTICALLY-THIN RADIATIVE COOLING IN CHOLLA

The addition of radiative cooling requires the introduction of source terms to the right-hand side of the energy equation

$$\frac{\delta E}{\delta t} + \nabla \cdot [\mathbf{v}(E + P)] = \Gamma - \Lambda. \tag{F.1}$$

Here, $E = \frac{1}{2}\rho \mathbf{v}^2 + e$ is again the total fluid energy per unit volume of the gas, ρ is the density, $\mathbf{v}$ the velocity vector, and P the gas pressure related to the internal energy e via an ideal gas equation of state, $P = (\gamma - 1)e$ with adiabatic index γ. The quantity Γ is a source term that accounts for heating of the gas, and Λ represents radiative losses (see, e.g., Katz et al., 1996). In the following subsections, we describe how *Cholla* couples the source terms to the adiabatic equations and how we calculate Γ and Λ.

F.0.1 Coupling of Source Terms

We implement radiative cooling in *Cholla* using an operator-split approach. After the hydrodynamic quantities have been updated for a given time step according to the ideal hydrodynamic equations, we add a source term to the internal energy to account for losses (or gains) as a result of radiative cooling (or heating) of the gas, such that

$$e^{n+1} = \tilde{e}^n + \dot{e}\Delta t_{\mathrm{ad}}, \tag{F.2}$$

where $\dot{e}$ is the rate of change of the internal energy, and Δt_{ad} is the hydrodynamic time step. Here, $\tilde{e}^n$ represents the updated internal energy after the hydrodynamic time step. The internal energy is calculated either from the gas pressure assuming an ideal gas equation of state, or tracked directly via the dual energy formalism. Because the adiabatic time step can be large compared to the radiative cooling time $e/\dot{e}$, the rate of change in internal energy is often a nonlinear function of the temperature over the course of a single $\Delta t_{\rm ad}$. Thus, we employ the common approach of subcycling the radiative cooling steps (e.g. Smith et al., 2008; Gray and Scannapieco, 2010), calculating a new $\dot{e}$ many times over the course of a single adiabatic time step so as to limit the change in internal energy for a radiative sub step, $\Delta t_{\rm rad}$, to less than five percent of the current internal energy:

$$\frac{\Delta e}{e} < 0.05. \tag{F.3}$$

In practice, this update means that at the end of each hydrodynamic time step, we calculate the temperature for a given cell's number density and internal energy according to the ideal gas law

$$T = \frac{P}{nk}, \tag{F.4}$$

where n is the number density of the gas in cm^{-3} calculated assuming $n = \rho/(\mu m_{\rm p})$, $m_{\rm p}$ is the mass of a proton, and k is Boltzmann's constant. We have assumed a mean molecular weight of $\mu = 1$ for all calculations, and have verified this does not influence our results compared with values of $\mu = 0.6$, as described in Section 3.8.

We next look up the tabulated net cooling rate associated with this density and temperature using a bilinear interpolation (described below), and calculate the resulting change in temperature over the full adiabatic time step given the cooling rate, $\Delta T = \dot{T}\ \Delta t_{\rm ad}$. If the change in temperature is greater than five percent, we shrink the radiative sub step such that $\Delta t_{\rm rad}$ results in a temperature change of five percent. We then update the temperature with this smaller time step, and repeat the process until we have synchronized with the full adiabatic time step.

F.0.2 Calculating Cooling and Heating Rates

We tabulate the cooling and heating rates per unit volume, Λ and Γ (measured in erg s^{-1} cm^{-3}), using the Cloudy code, version 13.03 (Ferland et al., 2013). For the calculations presented in this work, we assume that the gas is in photoionization equilibrium, subject to the $z = 0$ cosmic microwave background and the HM05 UV/X-ray background, as described in Hazy, the Cloudy documentation. The gas metallicity

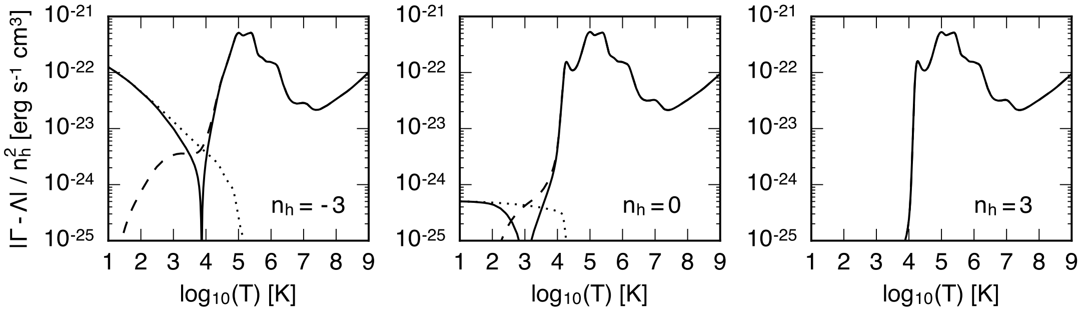

Figure F.1: Examples of cooling and heating rates as a function of temperature for solar metallicity gas calculated using Cloudy. The cooling function (dashed line), heating function (dotted line), and net energy gain or loss (solid) are shown for gas of three different densities. The gas is assumed to be in photoionization equilibrium while being exposed to the $z = 0$ CMB and a Haardt & Madauu UV/X-ray background. At low densities, photoionization heating leads to an equilibrium temperature $\sim 10^4$ K, while at higher densities heating is less effective.

is assumed to be solar, using the GASS10 abundances in Cloudy (Grevesse et al., 2010). Examples of the absolute cooling rates for gas of several different densities are shown in Figure F.1.

Our cooling and heating rates are tabulated on a grid, with points calculated at equally spaced logarithmic intervals of 0.1 between 10 K $< T <$ 10^9 K, and 10^{-6} cm^{-3} $< n <$ 10^6 cm^{-3}. To perform the interpolation necessary to calculate Λ and Γ at an arbitrary n and T, we take advantage of a novel feature of GPUs - texture memory. This special memory space allows a user to copy a 1, 2, or 3 dimensional array, or "texture", onto the GPU, and then use built-in GPU functions to quickly retrieve arbitrary values from that texture using bilinear interpolation. We have tested this method against a similar CPU-based method using GSL bilinear interpolation functions, and find that the GPU texture memory approach typically speeds up the radiative cooling calculation by orders of magnitude as compared to a CPU function. In fact, when implemented using the operator-splitting approach described above for simulations run with $\sim 128^3$ cells/GPU, the time spent calculating radiative cooling is completely negligible compared to the hydrodynamic calculations.

REFERENCES

Aarseth, S. J. (1999). From NBODY1 to NBODY6: The Growth of an Industry. *PASP*, **111**, pp. 1333–1346. doi:10.1086/316455.

Agertz, O., A. V. Kravtsov, S. N. Leitner, and N. Y. Gnedin (2013). Toward a Complete Accounting of Energy and Momentum from Stellar Feedback in Galaxy Formation Simulations. *ApJ*, **770**(1), p. 25.

Armillotta, L., F. Fraternali, and F. Marinacci (2016). Efficiency of gas cooling and accretion at the disc-corona interface. *MNRAS*. doi:10.1093/mnras/stw1930.

Balsara, D. S. (2004). Second-Order-accurate Schemes for Magnetohydrodynamics with Divergence-free Reconstruction. *ApJS*, **151**, pp. 149–184. doi:10.1086/381377.

Banda-Barragán, W. E., E. R. Parkin, C. Federrath, R. M. Crocker, and G. V. Bicknell (2016). Filament formation in wind-cloud interactions - I. Spherical clouds in uniform magnetic fields. *MNRAS*, **455**, pp. 1309–1333. doi:10.1093/mnras/stv2405.

Bard, C. and J. Dorelli (2010). GPU Accelerated Hall Magnetohydrodynamics. *AGU Fall Meeting Abstracts*, p. A1349.

Batten, P., N. Clarke, C. Lambert, and D. M. Causon (1997). On the choice of wavespeeds for the HLLC Riemann Solver. *SIAM Journal on Scientific Computing*, **18**, pp. 1553–1570.

Bedogni, R. and P. R. Woodward (1990). Shock wave interactions with interstellar clouds. *A&A*, **231**, pp. 481–498.

Berger, M. J. and P. Colella (1989). Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, **82**, pp. 64–84. doi:10.1016/0021-9991(89)90035-1.

Berger, M. J. and J. Oliger (1984). Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, **53**, pp. 484–512. doi:10.1016/0021-9991(84)90073-1.

Bordoloi, R., J. R. Rigby, J. Tumlinson, M. B. Bayliss, K. Sharon, M. G. Gladders, and E. Wuyts (2016). Spatially resolved galactic wind in lensed galaxy RCSGA 032727-132609. *MNRAS*, **458**, pp. 1891–1908. doi:10.1093/mnras/stw449.

Bouché, N., W. Hohensee, R. Vargas, G. G. Kacprzak, C. L. Martin, J. Cooke, and C. W. Churchill (2012). Physical properties of galactic winds using background quasars. *MNRAS*, **426**, pp. 801–815. doi:10.1111/j.1365-2966.2012.21114.x.

Brandvik, T. and G. Pullan (2007). Acceleration of a two-dimensional Euler flow solver using commodity graphics hardware. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, **221**(12), pp. 1745–1748. doi:10.1243/09544062JMES813FT.

Brüggen, M. and E. Scannapieco (2016). The Launching of Cold Clouds by Galaxy Outflows. II. The Role of Thermal Conduction. *ApJ*, **822**, 31. doi:10.3847/0004-637X/822/1/31.

Bryan, G. L., M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman, B. Smith, R. P. Harkness, J. Bordner, J.-h. Kim, M. Kuhlen, H. Xu, N. Goldbaum, C. Hummels, A. G. Kritsuk, E. Tasker, S. Skory, C. M. Simpson, O. Hahn, J. S. Oishi, G. C. So, F. Zhao, R. Cen, Y. Li, and The Enzo Collaboration (2014). ENZO: An Adaptive Mesh Refinement Code for Astrophysics. *ApJS*, **211**, 19. doi:10.1088/0067-0049/211/2/19.

Bryan, G. L., M. L. Norman, J. M. Stone, R. Cen, and J. P. Ostriker (1995). A piecewise parabolic method for cosmological hydrodynamics. *Computer Physics Communications*, **89**, pp. 149–168. doi:10.1016/0010-4655(94)00191-4.

Chan, C.-k., D. Mitra, and A. Brandenburg (2012). Dynamics of saturated energy condensation in two-dimensional turbulence. *Phys. Rev. E*, **85**(3), 036315. doi:10.1103/PhysRevE.85.036315.

Chandrasekhar, S. (1961). *Hydrodynamic and Hydromagnetic Stability.* Oxford University Press.

Chevalier, R. A. and A. W. Clegg (1985). Wind from a starburst galaxy nucleus. *Nature*, **317**, p. 44. doi:10.1038/317044a0.

Coil, A. L., B. J. Weiner, D. E. Holz, M. C. Cooper, R. Yan, and J. Aird (2011). Outflowing Galactic Winds in Post-starburst and Active Galactic Nucleus Host Galaxies at 0.2 ¡ z ¡ 0.8. *ApJ*, **743**, 46. doi:10.1088/0004-637X/743/1/46.

Colella, P. (1990). Multidimensional upwind methods for hyperbolic conservation laws. *Journal of Computational Physics*, **87**, pp. 171–200. doi:10.1016/0021-9991(90)90233-Q.

Colella, P. and H. M. Glaz (1985). Efficient solution algorithms for the Riemann problem for real gases. *Journal of Computational Physics*, **59**, pp. 264–289. doi:10.1016/0021-9991(85)90146-9.

Colella, P. and M. D. Sekora (2008). A limiter for PPM that preserves accuracy at smooth extrema. *Journal of Computational Physics*, **227**, pp. 7069–7076. doi: 10.1016/j.jcp.2008.03.034.

Colella, P. and P. R. Woodward (1984). The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations. *Journal of Computational Physics*, **54**, pp. 174–201. doi:10.1016/0021-9991(84)90143-8.

Cooper, J. L., G. V. Bicknell, R. S. Sutherland, and J. Bland-Hawthorn (2008). Three-Dimensional Simulations of a Starburst-driven Galactic Wind. *ApJ*, **674**, 157-171. doi:10.1086/524918.

Cooper, J. L., G. V. Bicknell, R. S. Sutherland, and J. Bland-Hawthorn (2009a). Starburst-Driven Galactic Winds: Filament Formation and Emission Processes. *ApJ*, **703**, pp. 330–347. doi:10.1088/0004-637X/703/1/330.

Cooper, J. L., G. V. Bicknell, R. S. Sutherland, and J. Bland-Hawthorn (2009b). Starburst-Driven Galactic Winds: Filament Formation and Emission Processes. *ApJ*, **703**, pp. 330–347. doi:10.1088/0004-637X/703/1/330.

Courant, R., K. Friedrichs, and H. Lewy (1967). On the Partial Difference Equations of Mathematical Physics. *IBM Journal of Research and Development*, **11**, pp. 215–234. doi:10.1147/rd.112.0215.

Creasey, P., T. Theuns, R. G. Bower, and C. G. Lacey (2011). Numerical overcooling in shocks. *MNRAS*, **415**, pp. 3706–3720. doi:10.1111/j.1365-2966.2011.19001.x.

Dalla Vecchia, C. and J. Schaye (2012). Simulating galactic outflows with thermal supernova feedback. *MNRAS*, **426**, pp. 140–158. doi:10.1111/j.1365-2966.2012.21704.x.

Davé, R., B. D. Oppenheimer, and K. Finlator (2011). Galaxy evolution in cosmological simulations with outflows - I. Stellar masses and star formation rates. *MNRAS*, **415**, pp. 11–31. doi:10.1111/j.1365-2966.2011.18680.x.

Davé, R., R. J. Thompson, and P. F. Hopkins (2016). MUFASA: Galaxy Formation Simulations With Meshless Hydrodynamics. *ArXiv e-prints*.

Domínguez, J. M., A. J. C. Crespo, D. Valdez-Balderas, B. D. Rogers, and M. Gómez-Gesteira (2013). New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. *Computer Physics Communications*, **184**, pp. 1848–1860. doi:10.1016/j.cpc.2013.03.008.

Einfeldt, B. (1988). On Godunov-Type Methods for Gas Dynamics. *SIAM Journal of Numerical Analysis*, **25**, pp. 294–318.

Einfeldt, B., P. L. Roe, C. D. Munz, and B. Sjogreen (1991). On Godunov-type methods near low densities. *Journal of Computational Physics*, **92**, pp. 273–295. doi:10.1016/0021-9991(91)90211-3.

Faucher-Giguère, C.-A., D. Kereš, and C.-P. Ma (2011). The baryonic assembly of dark matter haloes. *MNRAS*, **417**, pp. 2982–2999. doi:10.1111/j.1365-2966.2011.19457.x.

Ferland, G. J., R. L. Porter, P. A. M. van Hoof, R. J. R. Williams, N. P. Abel, M. L. Lykins, G. Shaw, W. J. Henney, and P. C. Stancil (2013). The 2013 Release of Cloudy. *RMXAA*, **49**, pp. 137–163.

Forum, M. P. (1994). MPI: A Message-Passing Interface Standard. Technical report, Knoxville, TN, USA.

Fragile, P. C., P. Anninos, K. Gustafson, and S. D. Murray (2005a). Magnetohydrodynamic Simulations of Shock Interactions with Radiative Clouds. *ApJ*, **619**, pp. 327–339. doi:10.1086/426313.

Fragile, P. C., P. Anninos, K. Gustafson, and S. D. Murray (2005b). Magnetohydrodynamic Simulations of Shock Interactions with Radiative Clouds. *ApJ*, **619**, pp. 327–339. doi:10.1086/426313.

Fragile, P. C., S. D. Murray, P. Anninos, and W. van Breugel (2004). Radiative Shock-induced Collapse of Intergalactic Clouds. *ApJ*, **604**, pp. 74–87. doi:10.1086/381726.

Frigo, M. and S. G. Johnson (2005). The Design and Implementation of FFTW3. *Proceedings of the IEEE*, **93**(2), pp. 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

Fryxell, B., K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo (2000). FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *ApJS*, **131**, pp. 273–334. doi:10.1086/317361.

Gardiner, T. A. and J. M. Stone (2008a). An unsplit Godunov method for ideal MHD via constrained transport in three dimensions. *Journal of Computational Physics*, **227**, pp. 4123–4141. doi:10.1016/j.jcp.2007.12.017.

Gardiner, T. A. and J. M. Stone (2008b). An unsplit Godunov method for ideal MHD via constrained transport in three dimensions. *Journal of Computational Physics*, **227**, pp. 4123–4141. doi:10.1016/j.jcp.2007.12.017.

Gingold, R. A. and J. J. Monaghan (1977). Smoothed particle hydrodynamics - Theory and application to non-spherical stars. *MNRAS*, **181**, pp. 375–389. doi:10.1093/mnras/181.3.375.

Godunov, S. K. (1959). A Difference Scheme for Numerical Solution of Discontinuous Solution of Hydrodynamic Equations. *Math. Sbornik*, **47**, pp. 271–306.

González, M., E. Audit, and P. Huynh (2007). HERACLES: a three-dimensional radiation hydrodynamics code. *A&A*, **464**, pp. 429–435. doi:10.1051/0004-6361:20065486.

Gray, W. J. and E. Scannapieco (2010). Formation of Compact Stellar Clusters by High-Redshift Galaxy Outflows. I. Non-Equilibrium Coolant Formation. *ApJ*, **718**, pp. 417–432. doi:10.1088/0004-637X/718/1/417.

Gregori, G., F. Miniati, D. Ryu, and T. W. Jones (2000). Three-dimensional Magnetohydrodynamic Numerical Simulations of Cloud-Wind Interactions. *ApJ*, **543**, pp. 775–786. doi:10.1086/317130.

Grevesse, N., M. Asplund, A. J. Sauval, and P. Scott (2010). The chemical composition of the Sun. *Ap&SS*, **328**, pp. 179–183. doi:10.1007/s10509-010-0288-z.

Griffiths, R. E., A. Ptak, E. D. Feigelson, G. Garmire, L. Townsley, W. N. Brandt, R. Sambruna, and J. N. Bregman (2000). Hot Plasma and Black Hole Binaries in Starburst Galaxy M82. *Science*, **290**, pp. 1325–1328. doi:10.1126/science.290.5495.1325.

Harfst, S., A. Gualandris, D. Merritt, R. Spurzem, S. Portegies Zwart, and P. Berczik (2007). Performance analysis of direct N-body algorithms on special-purpose supercomputers. *New Astron.*, **12**, pp. 357–377. doi:10.1016/j.newast.2006.11.003.

Harten, A., P. D. Lax, and B. Van Leer (1983). On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws. *SIAM Review*, **25**, pp. 35–61.

Hayes, J. C., M. L. Norman, R. A. Fiedler, J. O. Bordner, P. S. Li, S. E. Clark, A. ud-Doula, and M.-M. Mac Low (2006). Simulating Radiating and Magnetized Flows in Multiple Dimensions with ZEUS-MP. *ApJS*, **165**, pp. 188–228. doi:10.1086/504594.

Heckman, T. M., R. M. Alexandroff, S. Borthakur, R. Overzier, and C. Leitherer (2015). The Systematic Properties of the Warm Phase of Starburst-Driven Galactic Winds. *ApJ*, **809**, 147. doi:10.1088/0004-637X/809/2/147.

Hopkins, P. F. (2015). A new class of accurate, mesh-free hydrodynamic simulation methods. *MNRAS*, **450**, pp. 53–110. doi:10.1093/mnras/stv195.

Hui, W. H., P. Y. Li, and Z. W. Li (1999). A Unified Coordinate System for Solving the Two-Dimensional Euler Equations. *Journal of Computational Physics*, **153**, pp. 596–637. doi:10.1006/jcph.1999.6295.

Katz, N., D. H. Weinberg, and L. Hernquist (1996). Cosmological Simulations with TreeSPH. *ApJS*, **105**, p. 19. doi:10.1086/192305.

Kestener, P., F. Château, and R. Teyssier (2010). *Accelerating Euler Equations Numerical Solver on Graphics Processing Units*, pp. 281–288. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-13136-3.

Kim, C.-G. and E. C. Ostriker (2014). Momentum Injection by Supernovae in the Interstellar Medium. *ArXiv e-prints*.

Kjolstad, F. B. and M. Snir (2010). Ghost Cell Pattern. In *Proceedings of the 2010 Workshop on Parallel Programming Patterns*, ParaPLoP '10, pp. 4:1–4:9. ACM, New York, NY, USA. ISBN 978-1-4503-0127-5. doi:10.1145/1953611.1953615.

Klein, R. I., C. F. McKee, and P. Colella (1994a). On the hydrodynamic interaction of shock waves with interstellar clouds. 1: Nonradiative shocks in small clouds. *ApJ*, **420**, pp. 213–236. doi:10.1086/173554.

Klein, R. I., C. F. McKee, and P. Colella (1994b). On the hydrodynamic interaction of shock waves with interstellar clouds. 1: Nonradiative shocks in small clouds. *ApJ*, **420**, pp. 213–236. doi:10.1086/173554.

Knebe, A., A. Green, and J. Binney (2001). Multi-level adaptive particle mesh (MLAPM): a c code for cosmological simulations. *MNRAS*, **325**, pp. 845–864. doi:10.1046/j.1365-8711.2001.04532.x.

Kornei, K. A., A. E. Shapley, C. L. Martin, A. L. Coil, J. M. Lotz, D. Schiminovich, K. Bundy, and K. G. Noeske (2012). The Properties and Prevalence of Galactic Outflows at z ˜ 1 in the Extended Groth Strip. *ApJ*, **758**, 135. doi:10.1088/0004-637X/758/2/135.

Kravtsov, A. V. (1999). *High-resolution simulations of structure formation in the universe*. Ph.D. thesis, NEW MEXICO STATE UNIVERSITY.

Kritsuk, A. G., M. L. Norman, P. Padoan, and R. Wagner (2007). The Statistics of Supersonic Isothermal Turbulence. *ApJ*, **665**, pp. 416–431. doi:10.1086/519443.

Kulikov, I. (2014). GPUPEGAS: A New GPU-accelerated Hydrodynamic Code for Numerical Simulations of Interacting Galaxies. *ApJS*, **214**, 12. doi:10.1088/0067-0049/214/1/12.

Laney, C. B. (1998). *Computational Gasdynamics*. Cambridge University Press.

Lehnert, M. D. and T. M. Heckman (1996). Ionized Gas in the Halos of Edge-on Starburst Galaxies: Evidence for Supernova-driven Superwinds. *ApJ*, **462**, p. 651. doi:10.1086/177180.

Leitherer, C., D. Schaerer, J. D. Goldader, R. M. G. Delgado, C. Robert, D. F. Kune, D. F. de Mello, D. Devost, and T. M. Heckman (1999). Starburst99: Synthesis Models for Galaxies with Active Star Formation. *ApJS*, **123**, pp. 3–40. doi:10.1086/313233.

Lemaster, M. N. and J. M. Stone (2009). Dissipation and Heating in Supersonic Hydrodynamic and MHD Turbulence. *ApJ*, **691**, pp. 1092–1108. doi:10.1088/0004-637X/691/2/1092.

Leroy, A. K., F. Walter, P. Martini, H. Roussel, K. Sandstrom, J. Ott, A. Weiss, A. D. Bolatto, K. Schuster, and M. Dessauges-Zavadsky (2015). The Multi-phase Cold Fountain in M82 Revealed by a Wide, Sensitive Map of the Molecular Interstellar Medium. *ApJ*, **814**, 83. doi:10.1088/0004-637X/814/2/83.

LeVeque, R. J. (2002). *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press.

Liska, R. and B. Wendroff (2003a). Comparison of several difference schemes on 1D and 2D test problems for the Euler equations. *SIAM Journal of Scientific Computing*, **25**, pp. 995–1017. doi:10.1137/S1064827502402120.

Liska, R. and B. Wendroff (2003b). Comparison of several difference schemes on 1D and 2D test problems for the Euler equations. *SIAM Journal of Scientific Computing*, **25**, pp. 995–1017. doi:10.1137/S1064827502402120.

Lucy, L. B. (1977). A numerical approach to the testing of the fission hypothesis. *AJ*, **82**, pp. 1013–1024. doi:10.1086/112164.

Lynds, C. R. and A. R. Sandage (1963). Evidence for an Explosion in the Center of the Galaxy M82. *AJ*, **68**, p. 284. doi:10.1086/109098.

Mac Low, M.-M. and R. S. Klessen (2004). Control of star formation by supersonic turbulence. *Reviews of Modern Physics*, **76**, pp. 125–194. doi:10.1103/RevModPhys.76.125.

Mac Low, M.-M., C. F. McKee, R. I. Klein, J. M. Stone, and M. L. Norman (1994a). Shock interactions with magnetized interstellar clouds. 1: Steady shocks hitting nonradiative clouds. *ApJ*, **433**, pp. 757–777. doi:10.1086/174685.

Mac Low, M.-M., C. F. McKee, R. I. Klein, J. M. Stone, and M. L. Norman (1994b). Shock interactions with magnetized interstellar clouds. 1: Steady shocks hitting nonradiative clouds. *ApJ*, **433**, pp. 757–777. doi:10.1086/174685.

Marcolini, A., A. D'Ercole, D. Strickland, and T. Heckman (2005). Evolution of thermally conducting clouds embedded in a galactic wind. In de Grijs, R. and R. M. González Delgado (eds.) *Starbursts: From 30 Doradus to Lyman Break Galaxies*, volume 329 of *Astrophysics and Space Science Library*, p. P46.

Martin, C. L. (1999). Properties of Galactic Outflows: Measurements of the Feedback from Star Formation. *ApJ*, **513**, pp. 156–160. doi:10.1086/306863.

Martin, C. L. (2005). Mapping Large-Scale Gaseous Outflows in Ultraluminous Galaxies with Keck II ESI Spectra: Variations in Outflow Velocity with Galactic Mass. *ApJ*, **621**, pp. 227–245. doi:10.1086/427277.

Martizzi, D., C.-A. Faucher-Giguère, and E. Quataert (2014). Supernova Feedback in an Inhomogeneous Interstellar Medium. *ArXiv e-prints*.

McCourt, M., S. P. Oh, R. M. O'Leary, and A.-M. Madigan (2016). A Characteristic Scale for Cold Gas. *ArXiv e-prints*.

McCourt, M., R. M. O'Leary, A.-M. Madigan, and E. Quataert (2015). Magnetized gas clouds can survive acceleration by a hot wind. *MNRAS*, **449**, pp. 2–7. doi:10.1093/mnras/stv355.

McKee, C. F. and L. L. Cowie (1975). The interaction between the blast wave of a supernova remnant and interstellar clouds. *ApJ*, **195**, pp. 715–725. doi:10.1086/153373.

Melioli, C., E. M. de Gouveia Dal Pino, R. de La Reza, and A. Raga (2006). Star formation triggered by SN explosions: an application to the stellar association of β Pictoris. *MNRAS*, **373**, pp. 811–818. doi:10.1111/j.1365-2966.2006.11076.x.

Melioli, C., E. M. de Gouveia dal Pino, and A. Raga (2005). Multidimensional hydrodynamical simulations of radiative cooling SNRs-clouds interactions: an application to starburst environments. *A&A*, **443**, pp. 495–508. doi:10.1051/0004-6361:20052679.

Mellema, G., J. D. Kurk, and H. J. A. Röttgering (2002). Evolution of clouds in radio galaxy cocoons. *A&A*, **395**, pp. L13–L16. doi:10.1051/0004-6361:20021408.

Muratov, A. L., D. Kereš, C.-A. Faucher-Giguère, P. F. Hopkins, E. Quataert, and N. Murray (2015). Gusty, gaseous flows of FIRE: galactic winds in cosmological simulations with explicit stellar feedback. *MNRAS*, **454**, pp. 2691–2713. doi:10.1093/mnras/stv2126.

Nakamura, F., C. F. McKee, R. I. Klein, and R. T. Fisher (2006a). On the Hydrodynamic Interaction of Shock Waves with Interstellar Clouds. II. The Effect of

Smooth Cloud Boundaries on Cloud Destruction and Cloud Turbulence. *ApJS*, **164**, pp. 477–505. doi:10.1086/501530.

Nakamura, F., C. F. McKee, R. I. Klein, and R. T. Fisher (2006b). On the Hydrodynamic Interaction of Shock Waves with Interstellar Clouds. II. The Effect of Smooth Cloud Boundaries on Cloud Destruction and Cloud Turbulence. *ApJS*, **164**, pp. 477–505. doi:10.1086/501530.

Nestor, D. B., B. D. Johnson, V. Wild, B. Ménard, D. A. Turnshek, S. Rao, and M. Pettini (2011). Large-scale outflows from z 0.7 starburst galaxies identified via ultrastrong Mg II quasar absorption lines. *MNRAS*, **412**, pp. 1559–1572. doi:10.1111/j.1365-2966.2010.17865.x.

Nittmann, J., S. A. E. G. Falle, and P. H. Gaskell (1982). The dynamical destruction of shocked gas clouds. *MNRAS*, **201**, pp. 833–847.

Noh, W. F. (1987). Errors for calculations of strong shocks using an artificial viscosity and an artificial heat flux. *Journal of Computational Physics*, **72**, pp. 78–120. doi:10.1016/0021-9991(87)90074-X.

Oppenheimer, B. D. and R. Davé (2008). Mass, metal, and energy feedback in cosmological simulations. *MNRAS*, **387**, pp. 577–600. doi:10.1111/j.1365-2966.2008.13280.x.

Orlando, S., G. Peres, F. Reale, F. Bocchino, R. Rosner, T. Plewa, and A. Siegel (2005a). Crushing of interstellar gas clouds in supernova remnants. I. The role of thermal conduction and radiative losses. *A&A*, **444**, pp. 505–519. doi:10.1051/0004-6361:20052896.

Orlando, S., G. Peres, F. Reale, F. Bocchino, R. Rosner, T. Plewa, and A. Siegel (2005b). Crushing of interstellar gas clouds in supernova remnants. I. The role of thermal conduction and radiative losses. *A&A*, **444**, pp. 505–519. doi:10.1051/0004-6361:20052896.

Padoan, P. and Å. Nordlund (2002a). The Stellar Initial Mass Function from Turbulent Fragmentation. *ApJ*, **576**, pp. 870–879. doi:10.1086/341790.

Padoan, P. and Å. Nordlund (2002b). The Stellar Initial Mass Function from Turbulent Fragmentation. *ApJ*, **576**, pp. 870–879. doi:10.1086/341790.

Pang, B., U.-l. Pen, and M. Perrone (2010). Magnetohydrodynamics on Heterogeneous architectures: a performance comparison. *ArXiv e-prints*.

Pettini, M., C. C. Steidel, K. L. Adelberger, M. Dickinson, and M. Giavalisco (2000). The Ultraviolet Spectrum of MS 1512-CB58: An Insight into Lyman-Break Galaxies. *ApJ*, **528**, pp. 96–107. doi:10.1086/308176.

Poludnenko, A. Y., A. Frank, and E. G. Blackman (2002). Hydrodynamic Interaction of Strong Shocks with Inhomogeneous Media. I. Adiabatic Case. *ApJ*, **576**, pp. 832–848. doi:10.1086/341886.

Portegies Zwart, S. and J. Bédorf (2014). Computational Gravitational Dynamics with Modern Numerical Accelerators. *ArXiv e-prints*.

Portegies Zwart, S. F., H. Baumgardt, P. Hut, J. Makino, and S. L. W. McMillan (2004). Formation of massive black holes through runaway collisions in dense young star clusters. *Nature*, **428**, pp. 724–726. doi:10.1038/nature02448.

Quirk, J. J. (1994). A contribution to the great Riemann solver debate. *International Journal for Numerical Methods in Fluids*, **18**, pp. 555–574. doi: 10.1002/fld.1650180603.

Rich, J. A., M. A. Dopita, L. J. Kewley, and D. S. N. Rupke (2010). NGC 839: Shocks in an M82-like Superwind. *ApJ*, **721**, pp. 505–517. doi:10.1088/0004-637X/721/1/505.

Robertson, B. and P. Goldreich (2012). Adiabatic Heating of Contracting Turbulent Fluids. *ApJ*, **750**, L31. doi:10.1088/2041-8205/750/2/L31.

Robertson, B. E., A. V. Kravtsov, N. Y. Gnedin, T. Abel, and D. H. Rudd (2010). Computational Eulerian hydrodynamics and Galilean invariance. *MNRAS*, **401**, pp. 2463–2476. doi:10.1111/j.1365-2966.2009.15823.x.

Roe, P. L. (1981). Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *Journal of Computational Physics*, **43**, pp. 357–372. doi: 10.1016/0021-9991(81)90128-5.

Rubin, K. H. R., J. X. Prochaska, D. C. Koo, A. C. Phillips, C. L. Martin, and L. O. Winstrom (2014). Evidence for Ubiquitous Collimated Galactic-scale Outflows along the Star-forming Sequence at z ˜ 0.5. *ApJ*, **794**, 156. doi: 10.1088/0004-637X/794/2/156.

Rubin, K. H. R., J. X. Prochaska, B. Ménard, N. Murray, D. Kasen, D. C. Koo, and A. C. Phillips (2011). Low-ionization Line Emission from a Starburst Galaxy: A New Probe of a Galactic-scale Outflow. *ApJ*, **728**, 55. doi:10.1088/0004-637X/728/1/55.

Rupke, D. S., S. Veilleux, and D. B. Sanders (2005). Outflows in Active Galactic Nucleus/Starburst-Composite Ultraluminous Infrared Galaxies1,. *ApJ*, **632**, pp. 751–780. doi:10.1086/444451.

Ryu, D., J. P. Ostriker, H. Kang, and R. Cen (1993). A cosmological hydrodynamic code based on the total variation diminishing scheme. *ApJ*, **414**, pp. 1–19. doi: 10.1086/173051.

Saltzman, J. (1994). An Unsplit 3D Upwind Method for Hyperbolic Conservation Laws. *Journal of Computational Physics*, **115**, pp. 153–168. doi:10.1006/jcph.1994.1184.

Sandalski, S. (2012). *Neptune: An astrophysical smooth particle hydrodynamics code for massively parallel computer architectures*. Ph.D. thesis, California State University, Long Beach.

Sanders, R., E. Morano, and M.-C. Druguet (1998). Multidimensional Dissipation for Upwind Schemes: Stability and Applications to Gas Dynamics. *Journal of Computational Physics*, **145**, pp. 511–537. doi:10.1006/jcph.1998.6047.

Scannapieco, E. (2017). The Production of Cold Gas Within Galaxy Outflows. *ApJ*, **837**, 28. doi:10.3847/1538-4357/aa5d0d.

Scannapieco, E. and M. Brüggen (2015). The Launching of Cold Clouds by Galaxy Outflows. I. Hydrodynamic Interactions with Radiative Cooling. *ApJ*, **805**, 158. doi:10.1088/0004-637X/805/2/158.

Schaye, J., R. A. Crain, R. G. Bower, M. Furlong, M. Schaller, T. Theuns, C. Dalla Vecchia, C. S. Frenk, I. G. McCarthy, J. C. Helly, A. Jenkins, Y. M. Rosas-Guevara, S. D. M. White, M. Baes, C. M. Booth, P. Camps, J. F. Navarro, Y. Qu, A. Rahmati, T. Sawala, P. A. Thomas, and J. Trayford (2015). The EAGLE project: simulating the evolution and assembly of galaxies and their environments. *MNRAS*, **446**, pp. 521–554. doi:10.1093/mnras/stu2058.

Schive, H.-Y., Y.-C. Tsai, and T. Chiueh (2010). GAMER: A Graphic Processing Unit Accelerated Adaptive-Mesh-Refinement Code for Astrophysics. *ApJS*, **186**, pp. 457–484. doi:10.1088/0067-0049/186/2/457.

Schneider, E. E. and B. E. Robertson (2015). CHOLLA: A New Massively Parallel Hydrodynamics Code for Astrophysical Simulation. *ApJS*, **217**, 24. doi:10.1088/0067-0049/217/2/24.

Shin, M.-S., J. M. Stone, and G. F. Snyder (2008a). The Magnetohydrodynamics of Shock-Cloud Interaction in Three Dimensions. *ApJ*, **680**, pp. 336–348. doi: 10.1086/587775.

Shin, M.-S., J. M. Stone, and G. F. Snyder (2008b). The Magnetohydrodynamics of Shock-Cloud Interaction in Three Dimensions. *ApJ*, **680**, pp. 336–348. doi: 10.1086/587775.

Shopbell, P. L. and J. Bland-Hawthorn (1998). The Asymmetric Wind in M82. *ApJ*, **493**, pp. 129–153. doi:10.1086/305108.

Shu, C.-W. and S. Osher (1989). Efficient Implementation of Essentially Non-oscillatory Shock-Capturing Schemes, II. *Journal of Computational Physics*, **83**, pp. 32–78. doi:10.1016/0021-9991(89)90222-2.

Smith, B., S. Sigurdsson, and T. Abel (2008). Metal cooling in simulations of cosmic structure formation. *MNRAS*, **385**, pp. 1443–1454. doi:10.1111/j.1365-2966.2008.12922.x.

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

Springel, V. (2005). The cosmological simulation code GADGET-2. *MNRAS*, **364**, pp. 1105–1134. doi:10.1111/j.1365-2966.2005.09655.x.

Springel, V. (2010a). E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh. *MNRAS*, **401**, pp. 791–851. doi:10.1111/j.1365-2966.2009.15715.x.

Springel, V. (2010b). Smoothed Particle Hydrodynamics in Astrophysics. *ARA&A*, **48**, pp. 391–430. doi:10.1146/annurev-astro-081309-130914.

Spurzem, R. (1999). Direct N-body Simulations. *Journal of Computational and Applied Mathematics*, **109**, pp. 407–432.

Stone, J. M., T. A. Gardiner, P. Teuben, J. F. Hawley, and J. B. Simon (2008). Athena: A New Code for Astrophysical MHD. *ApJS*, **178**, pp. 137–177. doi:10.1086/588755.

Stone, J. M. and M. L. Norman (1992). The three-dimensional interaction of a supernova remnant with an interstellar cloud. *ApJ*, **390**, pp. L17–L19. doi:10.1086/186361.

Strang, G. (1968). On the Construction and Comparison of Difference Schemes. *SIAM Journal on Numerical Analysis*, **5**, pp. 506–517. doi:10.1137/0705041.

Strickland, D. K. and T. M. Heckman (2007). Iron Line and Diffuse Hard X-Ray Emission from the Starburst Galaxy M82. *ApJ*, **658**, pp. 258–281. doi:10.1086/511174.

Strickland, D. K. and T. M. Heckman (2009). Supernova Feedback Efficiency and Mass Loading in the Starburst and Galactic Superwind Exemplar M82. *ApJ*, **697**, pp. 2030–2056. doi:10.1088/0004-637X/697/2/2030.

Sugimoto, D., Y. Chikada, J. Makino, T. Ito, T. Ebisuzaki, and M. Umemura (1990). A special-purpose computer for gravitational many-body problems. *Nature*, **345**, pp. 33–35. doi:10.1038/345033a0.

Teyssier, R. (2002). Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES. *A&A*, **385**, pp. 337–364. doi: 10.1051/0004-6361:20011817.

Teyssier, R. (2015). Grid-Based Hydrodynamics in Astrophysical Fluid Flows. *ARA&A*, **53**, pp. 325–364. doi:10.1146/annurev-astro-082214-122309.

Thompson, T. A., E. Quataert, D. Zhang, and D. H. Weinberg (2016). An origin for multiphase gas in galactic winds and haloes. *MNRAS*, **455**, pp. 1830–1844. doi:10.1093/mnras/stv2428.

Toro, E. F. (2009). *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer.

Toro, E. F., M. Spruce, and W. Speares (1994). Restoration of the contact surface in the HLL Riemann solver. *Shock Waves*, **4**, pp. 25–34.

Tripp, T. M., J. D. Meiring, J. X. Prochaska, C. N. A. Willmer, J. C. Howk, J. K. Werk, E. B. Jenkins, D. V. Bowen, N. Lehner, K. R. Sembach, C. Thom, and J. Tumlinson (2011). The Hidden Mass and Large Spatial Extent of a Post-Starburst Galaxy Outflow. *Science*, **334**, p. 952. doi:10.1126/science.1209850.

van Leer, B. (1977). Towards the Ultimate Conservative Difference Scheme. IV. A New Approach to Numerical Convection. *Journal of Computational Physics*, **23**, p. 276. doi:10.1016/0021-9991(77)90095-X.

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

Veilleux, S., G. Cecil, and J. Bland-Hawthorn (2005). Galactic Winds. *ARA&A*, **43**, pp. 769–826. doi:10.1146/annurev.astro.43.072103.150610.

Vogelsberger, M., S. Genel, V. Springel, P. Torrey, D. Sijacki, D. Xu, G. Snyder, S. Bird, D. Nelson, and L. Hernquist (2014). Properties of galaxies reproduced by a hydrodynamic simulation. *Nature*, **509**, pp. 177–182. doi:10.1038/nature13316.

Wang, B. (1995). Cooling gas outflows from galaxies. *ApJ*, **444**, pp. 590–609. doi:10.1086/175633.

Wang, P., T. Abel, and R. Kaehler (2010). Adaptive mesh fluid simulations on GPU. *New Astronomy*, **15**, pp. 581–589. doi:10.1016/j.newast.2009.10.002.

Weiner, B. J., A. L. Coil, J. X. Prochaska, J. A. Newman, M. C. Cooper, K. Bundy, C. J. Conselice, A. A. Dutton, S. M. Faber, D. C. Koo, J. M. Lotz, G. H. Rieke, and K. H. R. Rubin (2009). Ubiquitous Outflows in DEEP2 Spectra of Star-Forming Galaxies at z = 1.4. *ApJ*, **692**, pp. 187–211. doi:10.1088/0004-637X/692/1/187.

Westmoquette, M. S., L. J. Smith, J. S. Gallagher, and K. M. Exter (2009). Mapping the roots of the galactic outflow in NGC1569. *Ap&SS*, **324**, pp. 187–193. doi: 10.1007/s10509-009-0126-3.

Xu, J. and J. M. Stone (1995a). The Hydrodynamics of Shock-Cloud Interactions in Three Dimensions. *ApJ*, **454**, p. 172. doi:10.1086/176475.

Xu, J. and J. M. Stone (1995b). The Hydrodynamics of Shock-Cloud Interactions in Three Dimensions. *ApJ*, **454**, p. 172. doi:10.1086/176475.

Zeldovich, Y. B. and Y. P. Raizer (1966). *Elements of gasdynamics and the classical theory of shock waves.*

Zhang, D., T. A. Thompson, E. Quataert, and N. Murray (2015). Entrainment in Trouble: Cool Cloud Acceleration and Destruction in Hot Supernova-Driven Galactic Winds. *ArXiv e-prints.*