An Adaptive Mesh Refinement (AMR) Method for Particle-Resolved Simulations of Multiphase Flows

by

İbrahim Nasuh YILDIRAN

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Mechanical Engineering

KOÇ UNIVERSITY

August 15, 2019

An Adaptive Mesh Refinement (AMR) Method for Particle-Resolved Simulations of Multiphase Flows

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

İbrahim Nasuh YILDIRAN

and have found that it is complete and satisfactory in all respects, and that any and all revisions required by the final examining committee have been made.

Committee Members:

[illegible]

Prof. Metin Muradoğlu

[illegible]

Asst. Prof. Didem Unat

[illegible]

Asst. Prof. Bayram Çelik

Date: 09/09/2019

"So, verily, with every difficulty, there is relief" 94:5

ABSTRACT

Increasing computational resources allow the fast development of numerical methods to simulate complex physical systems. However, numerical analysis of multiphase flows is still a challenging task due to wide disparity in length and time scales. Direct numerical simulation (DNS) methods, such as *front-tracking* method, provide valuable insights into multiphase flows by resolving all scales. But, in most of the DNS applications the quality of the grid resolution plays a crucial role in development of computational models of such flow systems due to presence of sharp interface between the phases. However the interface usually occupies a small portion of the overall domain, hence applying the resolution around the interface in the entire computational region is not an efficient way. Block-structured adaptive mesh refinement (AMR) method, which was first developed by Berger and Oliger [Berger and Oliger, 1984], offers a local and adaptive refinement of the grid. A properly-nested hierarchy of refinement levels increases the resolution around the region where the predetermined error criterion is exceeded. Almgren et al. [Almgren et al., 1998] combined block-structured AMR with a second-order projection method for incompressible Navier-Stokes equations and increased the efficiency by recursive time refinement algorithm. However, time refinement results in cumbersome synchronization operations to match coarse and fine levels after each time step. On the other hand, their projection method approximately satisfies the divergence-free constraint, hence, maintaining global conservation requires additional algorithmic complexity. Vanella et al. [Vanella et al., 2010] applied the structured AMR method to fluid-solid interaction problem by using a staggered grid arrangement and they ignored time refinement. Even though their multilevel multigrid solver simplified the synchronization step and satisfied divergence-free constraint exactly, using the same time step size in the

finest level in all other levels reduced the computational efficiency. The present study combines the subcycled block-structured AMR method with the three-dimensional finite-volume/front-tracking method developed by Unverdi and Tryggvason [Unverdi and Tryggvason, 1992]. The purpose is to accurately resolve multiphase flow by satisfying divergence-free constraint exactly and to gain the advantage of efficient AMR algorithm with time refinement at the same time. The algorithm presented here also avoids the complexity of synchronization step by using a fully-staggered grid arrangement first proposed by Harlow and Welch [Harlow and Welch, 1965]. The validation is performed by solving two benchmark problems: (1) The Hagen-Poiseuille problem with variable density/viscostiy and (2) mutiphase flow with a stationary bubble. Although the results to benchmark problems are very promising, the algorithm should be extended to simulate moving and deforming bubbles in the future studies. Moreover, to asses the efficiency of the algorithm, the performance analysis should be conducted when the algorithm is completed.

ÖZETÇE

Gün geçtikçe artan bilgisayarlı hesaplama kabiliyetleri, karmaşık fiziksel sistemleri simule etmeyi sağlayacak numerik metotların gelişmesini sağlamıştır. Ancak küçük boyut ve kısa zaman ölçeği sebebiyle çok fazlı akışların numerik modellenmesi halen zorluğunu korumaktadır. *Arayüz-izleme* gibi direkt numerik simulasyon (DNS) metotları tüm ölçekleri çözümleyerek çok fazlı akışlar ile ilgili değerli öngörüler elde edilmesini sağlamaktadır. Fakat, iki faz arasındaki keskin arayüz sebebiyle direkt numerik simulasyon modellerinin bir çoğunun geliştirilmesinde hücre çözünürlükleri çok büyük rol oynamaktadır. Toplam hesaplama alanı düşünüldüğünde çok küçük bir alanı kaplayan arayüz için tercih edilen çözünürlüğün tüm alana uygulanması ise verimli bir yöntem değildir.

İlk defa Berger ve Oliger [Berger and Oliger, 1984] tarafından hiperbolik sistemlerin çözümü için geliştirilen blok yapılı adaptif örgü artırma (AMR) yöntemi, bölgesel olarak ağ yapısının geliştirilmesini sağlamaktadır. Düzgün yuvalanma prensibine uygun olarak, önceden belirlenmiş bir hata kriterini aşan bölgeler artırma hiyerarşisine bağlı kalarak yeniden ayrıklaştırılır. Almgren et al. [Almgren et al., 1998] bu ayrıklaştırma yöntemini, sıkıştırılamaz Navier-Stokes denklemlerinin çözümü için de uygulanan ikinci-derece projeksiyon metodu ile birleştirmiş ve örgü seviyelerindeki zaman adımlarını da adaptif hale getirerek verimliliği artırmıştır. Ancak her bir zaman adımının bitiminde farklı çözünürlükteki örgü seviyelerinin senkronizasyonu kaçınılmaz hale gelmiştir. Öte yandan bu algoritma sıkıştırılamaz akışların simulasyonunda çok önemli olan kütle korunum kriterini yaklaşık olarak hesaplamaktadır. Bu sebeple, korunumu sağlayabilmek adına algoritma daha da kompleks hale gelmektedir. Vanella et al. [Vanella et al., 2010] aşamalı örgü düzeni kullanarak AMR metodunu katı-sıvı etkileşimi problemlerine uygulamıştır. Her ne kadar çok-seviyeli çok-örgülü Poisson çözücüsü örgü seviyeleri arasındaki senkronizasyon adımını basitleştirmiş ve kütle korunum kriterini tam olarak sağlamış olsa da, tek bir zaman adımını tüm seviyelere uygulayarak hesaplama verimini düşürmüştür. Bu tezde geliştirilen algoritma, Unverdi ve Tryggvason [Unverdi and Tryggvason, 1992] tarafından geliştirilen sonlu-hacim/arayüz-izleme metodu ile adaptif zaman adımlı örgü artırma yöntemini birleştirmektedir. Buradaki amaç çok fazlı akış denklemlerini çözerken korunum ilkelerini tam olarak sağlamak ve aynı zamanda örgü artırım uygulamasında adaptif zaman adımı yönteminin avantajlarından faydalanmaktır. Geliştirilen algoritma ilk kez Harlow ve Welch [Harlow and Welch, 1965] tarafından uygulanan aşamalı örgü düzenini kullanarak örgü seviyeleri arasındaki senkronizasyon adımını da basitleştirmektedir. Geliştirilen algoritmanın validasyonu iki problem üzerinde çalışılarak yapılmıştır: (1) Değişken özkütle/viskozite kullanarak çözülen Hagen-Poiseuille problemi ve (2) sabit baloncuk kullanılarak çözülen çok fazlı akış problemi. Her ne kadar sonuçlar başarılı olsa da, ilerdeki çalışmalarda algoritma hareketli ve deforme olabilen baloncukları da simule edebilecek şekilde geliştirilmelidir.

ACKNOWLEDGMENTS

People may not foresee how the road goes all the time. Experiences may not match expectations and there should be something strong that keeps you on the road. I would first like to thank God for endowing me the perseverance, knowledge and ability that help me to come to this point. Like many other people, I overcame lots of adversities to be able to be who I am right now. Without his blessings, I couldn't have achieved any of these. I may face tougher challenges in the rest of my life, however, I will hold no fears as long as I feel his existence.

My advisors are my true guides on this academic road. I am grateful to Assoc. Prof. Barbaros Çetin, who has provided me every academic ability to start this journey. Even though we have been estranged from each other a little bit, I will always owe you for everything we shared. I would also want to thank Assoc.Prof. Ilker Temizer for not only the research we have conducted together but also for his discipline and vision that will always enlighten my road. A very special thank goes to my current advisor Prof. Metin Muradoğlu, since his door was always open to me whenever I need help. I will never forget the day he accepted me to his group and I always want to collaborate with a CFD expert like him. I would also like to send my regards to all other professors at Bilkent University and Koç University who shared their valuable knowledge with me.

I am grateful to my dear parents Yüksel and Tülay and my dear sister Aybüke for their endless love and support. It is such a priceless feeling that they can understand and accept you although you unintentionally misbehave them, especially in stressful periods.

A very special gratitude goes out to my high-school friends Ahmet Bahaddin Ersöz, Murat Tokar, Oğuzhan Evsen, Murat Beycan, Ayşenur Öztürk, Çağatay Öztürk and

Ali Gharibdoust. We grew up together. We learned how to tackle adversities, how to champion each other whenever we need and how to have fun together. You deserve the biggest gratitudes, all the time.

Süleyman Doğan Öner... You will always be mentioned specially. The earth is like a hell for sensitive hearts, and my dearest Doğan, all struggles you have are direct results of being the most emotional, the cleverest and the most excited person I have ever known. It doesn't matter where you are or what you do, you will always be more than a friend, more than a brother to me.

I am also grateful to my friends from Bilkent University; Dilara Uslu, Umutcan Çalışkan, Okan Deniz Yılmaz, Şule Ayan, Karsu İpek Kılıç, Sedat Doğru and Ezgi Orhan. We not only support each other in MLRG and all other academic tasks but also we had enormous fun.

I also want to thank members of TDP (Social Awareness Projects) with whom we accomplished incredible social responsibility projects. I am always glad to meet you, glad to work with you.

I prepared some parts of the thesis at the hospital. I would like to thank Dr. Filiz Sadi Aykan for helping me patiently to get over this period as fast as possible. By witnessing her thoughtful approach to her patients, I understood better that compassion is above all else. Şükran!

And finally, last but by no means least, I send my deepest regards to my colleagues at Koç University. It is so great to know that I always have someone to discuss any question on my mind.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Nomenclature

δm	Infinitesimal mass
Δt	Time step size
δ	Dirac-δ function
κ	Curvature of the interface
λ	Second coefficient of viscosity
v	Infinitesimal volume
μ	Generic viscosity
μ_0	Viscosity of primary phase
μ_1	Viscosity of second phase
∇_S	Surface gradient
∇	Gradient operator
ω	Weight function
ϕ_f	Generic variable on Lagrangian grid
ϕ_g	Generic variable on Eulerian grid
ρ	Generic density
ρ_0	Density of primary phase

ρ_1	Density of second phase
σ	Surface tension coefficient
τ	Shear stress
$\mathbf{A}$	Advective term
$\mathbf{D}$	Viscous term
$\mathbf{f}$	Body force vector
$\mathbf{F}, \mathbf{G}, \mathbf{H}$	Fluxes in x, y and z directions
$\mathbf{f}_\sigma$	Surface tension force
$\mathbf{I}$	Unit tensor
$\mathbf{n}$	Normal vector to the area element
$\mathbf{P}$	Projection operator
$\mathbf{T}$	Symmetric stress tensor
$\mathbf{u}$	Velocity vector
$\mathbf{v}_f$	Velocity of front element
$\mathbf{x}_f$	Position of front element
C	Marker function (VOF method)
D	Material derivative
$d(r)$	Distance function

F	Level curve (Level-set method)
G	Gradient field of marker function
H	Heaviside function
h_x, h_y, h_z	Cell spacings in x,y and z directions
I	Marker function
i, j, k	Cell indices in x, y and z directions
l	Refinement level
L_x, L_y, L_z	Domain lengths in x, y and z directions
p	Thermodynamic pressure
t	Time
V	Interface velocity

Chapter 1

INTRODUCTION

Interactions of liquid-liquid and gas-liquid phases of fluids are encountered in many scientific and industrial applications. A significant number of examples can be given to demonstrate the importance of multiphase systems. Heat transfer by phase change in heat pipes is an efficient technique to remove extensive heat. Since they require minimum maintenance and offer significant reliability, they are extensively used to stabilize the thermal conditions in aerospace applications [Sundén and Fu, 2017]. The injectors in many internal combustion engines spray the fuel into the combustion chamber where the fuel interacts with the air. To increase the combustion efficiency, the fuel should spread evenly and it can be accomplished by understanding the gas-liquid interactions [Tryggvason et al., 2011]. Droplet-based studies are conducted to improve drug-delivery systems and bubble dynamics play a crucial role in these applications [Tryggvason et al., 2011]. In addition to these examples, multiphase systems are greatly involved in living organisms and advancements in this field can accelerate diagnosis and treatment operations in the medical field.

To make progress in gas-liquid multiphase systems, experimental and theoretical studies are conducted to understand the physics behind such systems. In both fields, there are certain adversities that challenge the studies. The experimental field requires improved equipment to resolve multiphase phenomena which involve wide range of length and time scales. From a mathematical perspective, together with the nonlinearity of governing equations, the complex behavior of the interface between two fluids constitutes significant difficulty during simulations. The computational results, therefore, cannot be compared with analytical results except for simple flow configurations such as Stokes flow. The *Marker-and-cell* (MAC) method is the primitive approach to resolve interface by using marker particles that identify each phase. The *Volume-of-fluid* method, similarly, uses a marker function to differentiate the phases [Scardovelli and Zaleski, 1999]. Even though they are very promising techniques, the geometric representation of the interface diminishes their accuracy. The level-set method first offered by Osher and Sethian [Osher and Sethian, 1988] and then improved by Sussman et al. [Sussman et al., 1994] to increase the accuracy by introducing *level-set* curves to compute the geometry of the interface. As an alternative method, the boundary-conforming grid is used by appropriate coordinate transformations to capture the interface. Ryskin and Leal [Ryskin and Leal, 1984] applied this method to simulate two-dimensional buoyancy driven flows, however, it is extremely difficult to implement this method to three-dimensional problems with complex geometries. The third method combines *Eulerian* and *Lagrangian* grids to separate the flow domain and the interface. Here, all phases present in the domain are represented by a single set of equations and point source functions, such as Dirac-δ function, are used to locate the interface. The method originates from front capturing and front tracking methods together. As Glimm et al. [Glimm et al., 2001] described in their landmark paper, in the front-tracking method a separated marker function can be used to follow the interface by adjusting the fixed grid in the vicinity of the interface. *One-fluid formulation* which uses a single set of equations, on the other hand, originates from the landmark paper of Peskin on immersed boundary [Peskin, 1977].

As a direct-numerical-simulation (DNS) tool, the one-fluid front tracking method is used in this study. DNS is a great approach to reveal the physical phenomena by fully resolving the simulation domain. All time and length scales are resolved to prevent misleading results caused by introducing excessive numerical approximations. However, DNS techniques bring their own challenges;

- The length and time scales may vary significantly within the simulation domain. For instance, a bubble occupies a relatively small region within the domain in multiphase flows. Hence the regions near to the bubble use different scales than regions away from it.

- Interface in multiphase flows introduces discontinuity and it is cumbersome to represent in DNS.

- The deformable interface requires frequent re-meshing.

The challenges described above influence discretization methods significantly. When different scales exist together, the accuracy of the simulation is maintained by arranging the mesh resolution according to the smallest length scale in flow. In one-fluid DNS algorithms for multiphase flows, re-meshing the region near the interface during the simulation is extremely inefficient because it continuously moves and undergoes large deformations often with topological changes, thus, the length scales change accordingly. In a nutshell, efficiency and accuracy introduce a two-sided problem. When desired resolution cannot be obtained, loss of global conservation contaminates the simulation and it prevents to capture actual physics accurately. Conversely, when the resolution increases due to accuracy concerns, the computational performance decreases significantly. For the sake of reasonable efficiency and accuracy, optimized algorithms are needed to take advantage of current computer power.

The main idea of adaptive mesh refinement (AMR) is to arrange the resolution of the computational grid according to the local accuracy requirements. With AMR implementation, different length scales are resolved separately to increase computational efficiency. The refinement regions can be spotted by tagging the cells according to different error estimation procedures. Extensive research has been done on implementing the AMR technique to different problems. AMR methods can be investigated roughly in two distinct categories: (1) Isotropic division of cells in an unstructured grid within hierarchical order as Bayyuk et al. [Bayyuk et al., 1993] suggested or using completely unstructured data structures as Ham et al. [Ham et al., 2002] pointed out. (2) Embedding block-structured refinement regions in a structured grid within a hierarchical order. In the former approach, maintaining the quality of the mesh can be problematic when complex geometries are involved. Therefore, accuracy and global conservation may not be sustained.

Block-structured AMR method was first proposed by Berger and Oliger [Berger and Oliger, 1984] to solve hyperbolic partial differential equations in the form;

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} + \frac{\partial \mathbf{H}}{\partial z} = 0 \tag{1.1}$$

where $\mathbf{F}$,$\mathbf{G}$ and $\mathbf{H}$ are the fluxes in x,y and z directions respectively.

They used a collocated grid arrangement and Richardson-type error estimation procedure. In their algorithm, the refinement regions can be rotated to be aligned with the sharp gradients or discontinuities. Even though this reduced the number of cells clustered into refinement patches, the coordinate transformation brought additional complications and inefficiency.

Then Berger and Colella [Berger and Colella, 1989] and Bell et al.[Bell et al., 1994] modified the algorithm and introduced a properly nested grid arrangement where the refinement patches are always aligned with the structured base grid. This increased the performance and simplified the refluxing operations which are required to synchronize fine and coarse level of refinements.

The level organization based on the algorithm proposed by Berger and Colella [Berger and Colella, 1989] can be seen in Figure 1.1. In studies that employ the block-structured AMR method, the same meshing strategy is also followed. Here cells painted in *grey* and *green* are ghost regions where the interlevel communications, *interpolation* and *restriction*, are performed. The grids in each level are properly nested and their boundaries are aligned with underlying coarser grids.

Although suggested algorithms provided a solid basis for the solution of hyperbolic equations, some problems such as incompressible flow and magnetohydrodynamics require to satisfy divergence-free constraint and it is not easy to satisfy in AMR methods. In the literature, the vast majority of incompressible flow solvers with AMR routines use second-order projection method first developed by Bell et al. [Bell et al., 1989] and Bell et al. [Bell et al., 1991]. This particular fractional step

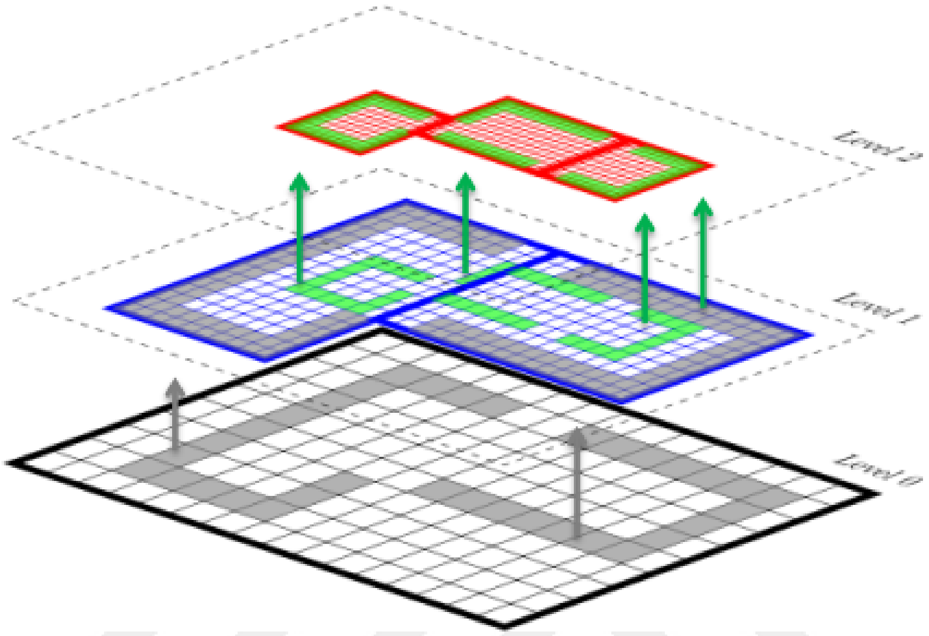


Figure 1.1: Grid organization in the block-structured AMR applications

scheme discretizes the viscous and advective terms by using the Crank-Nicholson scheme and nonlinear terms are treated by the second-order upwind method. Later, the projection step is employed according to the Hodge decomposition to generate a divergence-free velocity field. Bell and Marcus applied this algorithm to variable-density case [Bell and Marcus, 1992] and then Almgren et al. [Almgren et al., 1996] introduced the finite element based discretization scheme in the projection step to end up with a second-order approximate projection method. This formulation was then used to simulate incompressible Navier-Stokes equations with finite amplitude density variations in their landmark paper [Almgren et al., 1998]. This method provided here used a subcycled time-stepping algorithm for the grid hierarchy proposed by Berger and Colella [Berger and Colella, 1989]. Subcycling allows each level of refinement to advance with its own timestep. Although it increases the efficiency significantly, levels should be synchronized after each timestep to maintain global conservation. Due to the approximateness of the projection, two different obligatory synchronization steps emerge: (1) The first step updates the MAC velocities that are interpolated from cell centers to faces. This is a single level synchronization where additional Poisson equation is solved to spread correction concentrated at the interface to the entire level. (2) The second one is two-level synchronization where the multigrid-multilevel solver is used to calculate correction used to eliminate flux and pressure mismatch. The multigrid algorithm used in this step was first developed by Martin and Cartwright [Martin and Cartwright, 1996]. Sussman et al. [Sussman et al., 1999] combined the same adaptive approximate projection algorithm to multiphase flow analysis by using the level-set method and Martin et al. [Martin et al., 2008] extended the algorithm to three-dimensions. Even though Almgren's algorithm is very promising, it requires to solve additional elliptic and parabolic equations because interlevel operations satisfy Dirichlet type, $\phi = \phi_0$, boundary condition at interface of levels and fail to satisfy Neumann type, $\partial\phi/\partial n = 0$.

In addition to complications because of the subcycled time-stepping in Almgren's algorithm, a semi-staggered grid arrangement introduces an additional source of error and further corrections should be made for synchronization. In this arrangement, all variables except pressure are located at cell centers and pressure is located at nodes. To compute the nonlinear convective terms, the velocities are interpolated to faces and additional projection should be applied to make sure that face-centered MAC velocities satisfy the divergence-free constraint. The second projection, which approximately satisfies the continuity, is applied to update cell-centered velocities with appropriate pressure values.

In order to simplify the synchronization step, Minion [Minion, 1996] and Vanella et al. [Vanella et al., 2010] used the multigrid-multilevel solver for the entire refinement levels at once. This approach satisfies Dirichlet and Neumann type boundary conditions simultaneously, however, subcycled time-stepping cannot be performed and the timestep of the finest level should be used for all levels. Thus, simplification in algorithm resulted in a lack of computational efficiency.

Application of structured adaptive mesh refinement approach to fluid-solid interaction problems reinforced the development of AMR algorithms. The efforts started with Roma et al. [Roma et al., 1999] where they combined the immersed boundary method developed by Peskin [Peskin, 1977] and block-structured AMR algorithm proposed by Berger and Colella [Berger and Colella, 1989] to tackle with the two-dimensional incompressible viscous flow. They employed second-order discretization method developed by Bell et al. [Bell et al., 1989], but the algorithm couldn't offer significant improvement over using uniform mesh with the resolution of the finest level. Then Griffith et al. [Griffith et al., 2007] improved the algorithm by using the collocated grid arrangement.

As an alternative to block-structured AMR method, Dezeeuw and Powell [DeZeeuw and Powell, 1993] introduced the quad-tree approach to solve the steady Euler equations. This type of AMR algorithm is based on the idea of dividing individual parent cells into four cells in two-dimensions and eight for three-dimensions by bisecting each direction as seen in Figure 1.2. Popinet, [Popinet, 2003], extended quad-tree approach to Navier-Stokes equations and then worked on jet disintegration problem with the *Gerris* package [Popinet, 2009]. *Gerris* uses multilevel multigrid Poisson solver, however, grid management can be problematic due to newly generated neighboring stencils after refining the cells. Agresar et al. [Agresar et al., 1998] applied the quad-tree AMR approach to the front-tracking method developed by Unverdi and Tryggvason [Unverdi and Tryggvason, 1992] and worked on deformation and adhesion of circulating cells. One of the most important aspects of this study was to use a staggered grid arrangement.

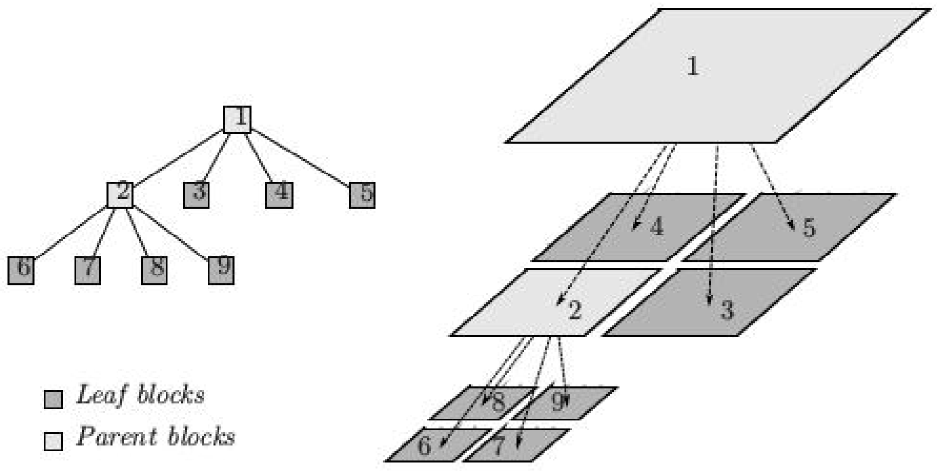


Figure 1.2: Quad-tree hieracrchy of the parent and the leaf blocks [Zuzio, 2011]

Vanella et al. [Vanella et al., 2010] demonstrated the application of block-structured AMR to the staggered grid by working on fluid-solid interaction problems. The grid hierarchy was managed by *Paramesh* package [MacNeice et al., 2000]. Although they successfully implemented structured adaptive mesh to the staggered arrangement by satisfying the divergence-free constraint, their algorithm employed a single time step and abandoned the subcycling approach. Moreover, their interpolation algorithm is restricted to the refinement ratio of two.

As can be seen in this brief literature overview, considerable effort has been spent to implement efficient AMR techniques to solve various flow problems. For incompressible flow problems, however, development is still required to obtain accurate, efficient and stable algorithms. In some of the landmark papers subcycled time-stepping was implemented on the semi-staggered arrangement with tedious synchronization efforts. On the other hand, some studies, such as [Minion, 1996] and [Vanella et al., 2010], ignored subcycling and simplified synchronization by using a multilevel Poisson solver. In the present study, the one-fluid front tracking method for multiphase flows is combined with the block-structured AMR algorithm. The subcyled timestepping approach is employed to increase the efficiency and divergence-free constraint of incompressible flow is satisfied with considerable accuracy since the computation is based on the staggered grid arrangement. The synchronization after each coarse level time step is expected to be simpler than Almgren et al. [Almgren et al., 1998] and more complicated than that of Minion [Minion, 1996]. The results are presented for the incompressible flows with variable density/viscosity and for multiphase flows with a stationary bubble. The grid hierarchy is managed by *AMReX* [Zhang et al., 2019] package, which is originated from the block-structured algorithm of Berger and Colella [Berger and Colella, 1989].

Chapter 2

PHYSICAL MODEL

All computational fluid dynamics (CFD) applications are a combination of various methods to deal with discretization of the governing equations. Although the set of equations can vary depending on the flow problem, there are two basic mathematical statements expressing the fundamental physical principles of fluid dynamics:

1. Conservation of mass

2. Conservation of momentum

In addition to these two basic conservation rules, additional rules, such as *conservation of energy* and *conservatin of spicies*, can be added to simulate more complex problems. Since energy and concentration effects are not considered in this study, this chapter focuses on the description of the mass and momentum conservation equations. Additionally, the incompressible flow assumption is made and discussed as a special case. In the upcoming chapters, the concept of multiphase flow is built on the basic principles explained in this chapter.

The governing equations can be derived in two different but interchangeable forms: *conservative* and *nonconservative*. The distinction of the form is especially important in the selection of numerical methods to discretize the terms in Navier-Stokes equations and it becomes clear as the governing equations are derived.

The *conservative* form of the governing equations is derived in this chapter. However, the nonconservative representations are also shown at the end of each section. Figure 2.1 shows the fixed control volume on which the equations are derived.

The differential form of the governing equations is going to be derived in this section. On the other hand, because the Finite Volume Method has been built

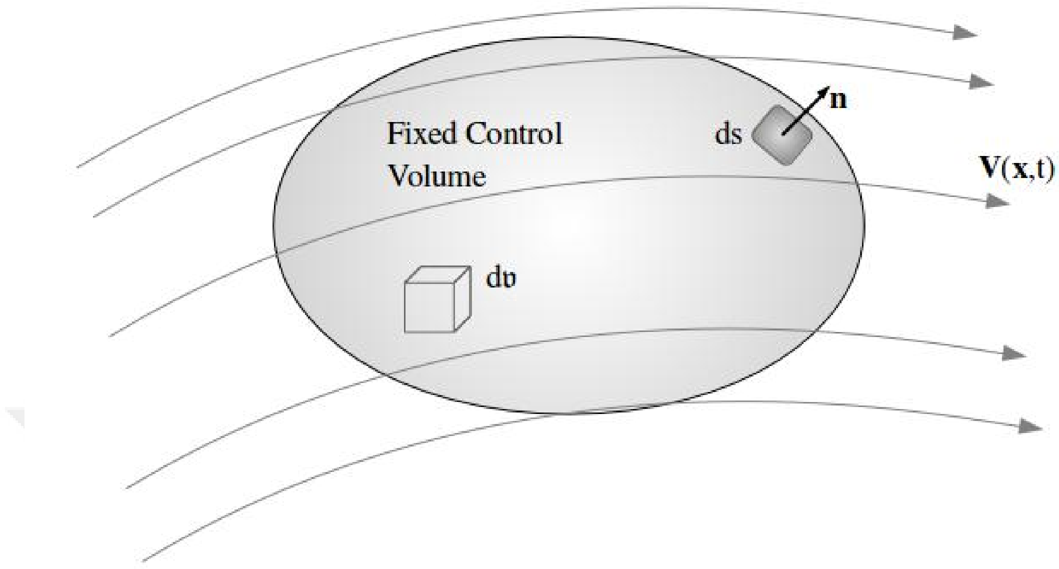


Figure 2.1: The representation of the fixed control volume in a fluid flow

upon the integral formulation, the importance of the integral equation is also stated briefly. The integral form of the governing equations has the flexibility to include discontinuities within the domain, while, the differential form expects continuous and differentiable functions.

2.1 Conservation of Mass

The principle of mass conservation dictates that the mass is neither created nor destroyed as long as there is no chemical reaction within the domain. This physical principle is called *continuity* in the context of continuum mechanics and it is also a key concept in numerical approximations.

The fluid flow goes through the fixed control volume in Figure 2.1 across the control surface S. In words, the continuity equation tells that at any time interval the accumulation of mass inside the control volume should be equal to the net flow of mass into the control volume. This can be expressed mathematically as follows;

$$\frac{d}{dt}\int_{\mathcal{V}} \rho dv = -\oint_{S} \rho \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} ds \tag{2.1}$$

Eq. 2.1 is derived on the basis of mass conservation in the infinitesimal control volume *dv* depicted in Figure 2.1. The left-hand side of Eq.2.1 represents the time rate of change of mass within the control volume while the right-hand side shows the net mass flux crosses the boundary S. Since the control volume is fixed, we can take the time derivative into the volume integral. In addition, divergence (Gauss-Ostrogradsky) theorem states that the outward flux through a closed surface is always equal to the volume integral of the divergence of the same vector field. Hence if we apply the divergence theorem to the surface integral on the right-hand side we finally have a single volume integral as depicted in Eq.2.2.

$$\int_{\mathcal{V}} \left[\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) \right] dv = 0 \tag{2.2}$$

The equality holds for any arbitrary volume only when the expression between the brackets is zero provided it is continuous. Thus, the final form of the continuity equation in differential *conservative* form can be written as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{2.3}$$

The *nonconservative* form can be obtained by simply manipulating Eq.2.3 and it can be shown as follows;

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{u} = 0 \tag{2.4}$$

2.2 Conservation of Momentum

The momentum is another physical property of a fluid element that should be conserved. The equation of motion, eventually the momentum conservation equation, is obtained by equating the net momentum flux across the boundaries of the fluid element to momentum accumulation in the domain. Eq.2.5 represents the momentum balance in the integral form.

$$\frac{\partial}{\partial t}\int_{\mathcal{V}}\rho\mathbf{u}dv = -\oint_{S}\rho\mathbf{u}(\mathbf{u}\cdot\mathbf{n})ds + \int_{\mathcal{V}}\mathbf{f}dv + \oint_{S}\mathbf{n}\cdot\mathbf{T}ds \tag{2.5}$$

where the term on the left-hand side expresses the time rate of change of momentum within the control volume. The first and third terms on the right-hand side are the momentum flux through the surface and momentum contribution by the surface forces respectively. The symmetric tensor $\mathbf{T}$ encompasses both pressure and shear stress. Lastly, the second term on the right hand side is the body force, which is equal to $\rho\mathbf{g}$ when only gravitational forces are acting on the domain. In addition to these terms, the external forces, such as magnetic force and surface tension force, can have a contribution to the momentum balance.

As is the case with the conservation of mass, the divergence theorem can be applied to Eq.2.5. The resulting equation can be written as;

$$\frac{\partial\rho\mathbf{u}}{\partial t} = -\nabla\cdot(\rho\mathbf{u}\mathbf{u}) + \mathbf{f} + \nabla\cdot\mathbf{T} \tag{2.6}$$

For Newtonian fluids, the stress can be assumed to be the linear function of strain rate. Thus the tensor $\mathbf{T}$ can be expressed as;

$$\mathbf{T} = (-p + \lambda\nabla\cdot\mathbf{u})\mathbf{I} + 2\mu\mathbf{S} \tag{2.7}$$

where $\mathbf{I}$ refers to second-order identity tensor and $\mathbf{S} = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$ is the rate of strain tensor. λ is the second coefficient of viscosity which has the value of $-\frac{2}{3}\mu$ if the Stokes' assumption holds.

The Navier-Stokes equations in the *conservative* form can be obtained by introducing definition of $\mathbf{T}$ into Eq.2.6.

$$\frac{\partial\rho\mathbf{u}}{\partial t} + \nabla\cdot(\rho\mathbf{u}\mathbf{u}) = -\nabla p + \nabla(\lambda\nabla\cdot\mathbf{u}) + \nabla\cdot\left[\mu(\nabla\mathbf{u} + \nabla^T\mathbf{u})\right] + \mathbf{f} \tag{2.8}$$

The *nonconservative* form can be obtained by manipulating the dyadic product appears on the left hand side of Eq.2.8. The term $\nabla \cdot (\rho \mathbf{u}\mathbf{u})$ can be rewritten as;

$$\nabla \cdot (\rho \mathbf{u}\mathbf{u}) = \mathbf{u}\nabla \cdot (\rho \mathbf{u}) + (\rho \mathbf{u}) \cdot \nabla \mathbf{u} \tag{2.9}$$

Similarly;

$$\frac{\partial \rho \mathbf{u}}{\partial t} = \mathbf{u}\frac{\partial \rho}{\partial t} + \rho \frac{\partial \mathbf{u}}{\partial t} \tag{2.10}$$

If Eq.2.9 and Eq.2.10 are summed up, the summation of first terms on the right hand sides disappears because it gives directly the *conservative* continuity equation derived in Eq.2.3. Moreover, the summation of the second terms in the same equations gives the material derivative of $\mathbf{u}$ when they are taken into a common parenthesis of ρ. Therefore the remaining expression gives the Navier-Stokes equation in *nonconservative* form as it is shown in Eq.2.11

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla p + \nabla(\lambda \nabla \cdot \mathbf{u}) + \nabla \cdot \left[\mu(\nabla \mathbf{u} + \nabla^T \mathbf{u})\right] + \mathbf{f} \tag{2.11}$$

2.3 Incompressible Flow

The mass and momentum conservation equations depicted in Eq.2.3 and in Eq.2.8 are applicable in any type of Newtonian fluid flow. However, when the Mach number $M = {}^{V}\!/_{c}$, which compares the flow speed and the speed of sound, is roughly less than 0.3, the flow is assumed to be incompressible. Hence, the volumetric mass can be assumed to be constant over spatial dimensions and time. Even though density varies in multiphase flows, incompressibility still can be valid when the density of each phase is assumed to be constant in their respective regions. In this case, a discontinuous interface between two immiscible fluids should be expected.

Incompressibility assumption greatly simplifies the Navier-Stokes Equation. When density is assumed to be constant, the first term on the left-hand side in Eq.2.3 is going to disappear and the ρ can be taken out of the divergence operator. At the end, the continuity equation takes the form

$$\nabla \cdot \mathbf{u} = 0. \tag{2.12}$$

Eq.2.12 allows further simplification in Eq.2.8. The term with second coefficient of viscosity in Eq.2.8 also disappears when incompressiblity holds. Thus Eq.2.8 takes its final form as

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \nabla \cdot \left[\mu(\nabla \mathbf{u} + \nabla^T \mathbf{u})\right] + \mathbf{f}. \tag{2.13}$$

Chapter 3

MULTIPHASE FLOW

Understanding the mechanics of multiphase flow is a challenging task in both experimental and computational point of views. The limitations in experimental studies mainly result from the fact that the measurement devices are still not capable of resolving small length and short time scales.

The computational aspect of the multiphase flows is even more challenging. Highly nonlinear nature of governing equations and capturing the correct shape and position of the fluid-fluid interface constitute the most critical adversities. In addition, the analytical results are available only for simple flow configurations such as steady bubble flow in Stokes' regime [Tryggvason et al., 2011].

3.1 Interface Mechanics

Accurate representation of the shape and position of the interface is a major issue in multiphase flow analysis. There are several mathematical approaches to represent the interface and *parametrization* of the surface is one of the most convenient ways. Figure 3.1 shows how the parametrization can be performed in 3D. The normal and tangent vectors, which are necessary for calculating the curvature of the interface, are also depicted in the same figure.

The normal vector can be calculated by taking the cross product of tangent vectors, Eq.3.1, to the surface.

$$\mathbf{x}_u = \frac{\partial \mathbf{x}}{\partial u} \qquad \mathit{and} \qquad \mathbf{x}_v = \frac{\partial \mathbf{x}}{\partial v} \tag{3.1}$$

Thus the normal vector is defined as

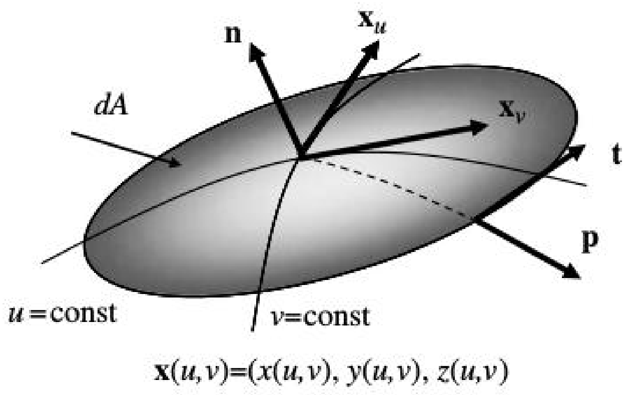


Figure 3.1: Parametrization of the surface of the interface with two independent coordinates u and v. The normal vector,$\mathbf{n}$, and the tangent vector, $\mathbf{t}$, are also depicted. [Tryggvason et al., 2011]

$$\mathbf{n} = \frac{\mathbf{x}_u \times \mathbf{x}_v}{|\mathbf{x}_u \times \mathbf{x}_v|} \tag{3.2}$$

When the normal field extends of the interface, the curvature of the interface can be calculated by taking the divergence as follows

$$\kappa = -\nabla \cdot \mathbf{n} \tag{3.3}$$

An alternative expression for the mean curvature of the interface can be derived in an integral form as well. This form is useful especially when the interface is discretized.

$$\kappa\mathbf{n} = \lim_{\delta A \to 0} \frac{1}{\delta A} \oint \mathbf{p} dl \tag{3.4}$$

where $\mathbf{p} = \mathbf{t} \times \mathbf{n}$.

It is a cumbersome process to calculate the position of each element on the interface. Alternatively, the marker functions can be used in the entire domain to label the different phases. The marker function may vary depending on which interface-tracking method is used. The level set method employs a special smooth function to locate the interface. On the other hand discontinuous step function, or Heaviside function, is used in the front-tracking method as shown in

$$H(x,y) = \begin{cases} 1, & \text{fluid phase 1} \\ 0, & \text{fluid phase 2} \end{cases} \tag{3.5}$$

As Eq.3.5 shows, the first fluid is marked as "1" and the other one is "0". At the interface, there is a discontinuous change between the values of the two phases. In order to represent the interface, that discontinuity should be resolved and it is going to be discussed especially in the section where front-tracking is explained.

3.2 Interfacial Fluid Mechanics

When the two phases in the domain are identified separately, the interfacial region forms between those phases. Since the phases are described by separate governing equations, the continuity at the interface should be somehow established. The interface is a region of zero thickness and it separates the phases present in the domain. Unless the one-fluid formulation is used, the interface brings discontinuity either in physical parameters themselves or in their derivatives. Thus, this *jump condition* should be managed to satisfy global conservation. At this point, mass and momentum conservations should be revisited to incorporate multiphase aspect. After the modifications to the conservation laws due to jumping conditions are presented, *one-fluid formulation* is described as an alternative. The front-tracking method employed in this study is also based on the one-fluid formulation.

3.2.1 Conservation of Mass

The continuity condition for single phase bulk flow was discussed in previous chapter. When another phase is introduced to the domain, the interface inherently violates the continuity. Therefore the jump conditions should be defined to maintain the mass conservation.

In Figure 3.2 $\mathbf{u}_1$ shows the flow velocity for the fluid on the right hand side and similarly $\mathbf{u}_2$ is velocity for fluid on the left hand side. $\mathbf{V}$ is the velocity of the interface

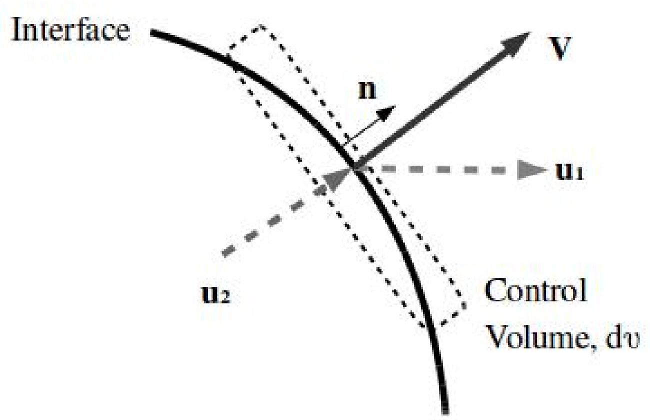


Figure 3.2: Representation of the control volume around the portion of the interface

and it should be taken into account to calculate the relative mass flow. Since the thickness of the interface is assumed to be zero, there is no mass accumulated within the control volume. Therefore the first term on Eq.2.1 vanishes and the mass balance is maintained by equating the incoming and outgoing mass fluxes. The mathematical representation of this situation can be given as;

$$\rho_1(\mathbf{u}_1 \cdot \mathbf{n} - V) = \rho_2(\mathbf{u}_2 \cdot \mathbf{n} - V) = 0 \tag{3.6}$$

This equation shows that the velocity of the interface should be defined as

$$V = \mathbf{u}_1 \cdot \mathbf{n} = \mathbf{u}_2 \cdot \mathbf{n} \tag{3.7}$$

The relation between the velocities of two phases at the interface, on the other hand, comes from the fact that the equality of shear stress at the interface. Thus under the assumption of incompressibility and absence of phase change the velocities of the phases should be equal at the interface, $\mathbf{u}_1 = \mathbf{u}_2$.

3.2.2 Surface Tension and Consevation of Momentum

As is the case with conservation of mass, the momentum balance is analyzed for the control volume shown in Fig.3.2.

Since there is no momentum accumulated within the control volume due to zero thickness Eq.2.5 reduces to

$$\oint_S \rho \mathbf{u}(\mathbf{u} \cdot \mathbf{n} - V) ds + \oint_S \mathbf{n} \cdot \mathbf{T} ds + \int_S \mathbf{f}_\sigma ds = 0 \tag{3.8}$$

Eq.3.7 derived for conservation of mass requires that the first term on the left hand side is zero. Now the last term which indicates the contribution of surface tension force should be defined.

At the end of a clear derivation performed by Tryggvason and Zaleski [Tryggvason et al., 2011] the surface tension force is calculated as in Eq.3.9.

$$f_\sigma = \sigma \kappa \mathbf{n} + \nabla_s \sigma \tag{3.9}$$

where $\nabla_s = \nabla - \mathbf{n}(\mathbf{n} \cdot \nabla)$ and the last term in Eq.3.9 is obviously zero for constant surface tension coefficient.

When Eq.3.8, Eq.3.9 and Eq.3.7 are combined, the interfacial jump condition for momentum can be obtained as;

$$-(\mathbf{T}_2 - \mathbf{T}_1) \cdot \mathbf{n} = \sigma \kappa \mathbf{n} + \nabla_s \sigma \tag{3.10}$$

3.2.3 One Fluid Formulation

The previous section presented the coupling of two independent governing equations by using jump conditions at the interface. As an alternative, single set of equations can be used to model entire flow field occupied by different phases. Thus, cumbersome procedure of implementing jump conditions is avoided. Different phases are represented in this one set of equations through variable material properties which have sudden change at the interface. This alternative approach is called *one fluid formulation* and it is the focus of multiphase analiyisis in this study.

Even though one fluid formulation automatically satisfies the jump conditions, the interface and the forces generated due to the presence of it should be included into the governing equations. For this purpose, the Dirac-δ functions are used.

Entire derivation procedure is the same with single phase case with an addition of surface tension as a body force to momentum equation. Unless chemical reactions are present, the mass conservation equation is going to be the same with the single phase.

As it is pointed out in the derivation of the momentum equation for the single phase case, different type of external forces can be included along with the body force. Therefore, after manipulating Eq.3.9 by Dirac-δ function, $\mathbf{f}_\sigma$ is added to the Eq.2.8. Before proceeding with implementation, it is useful to define Dirac-δ function concentrated over the interface, $\delta_S(\mathbf{x})$, by performing appropriate coordinate manipulation [Onural, 2006]. After defining $\delta_S = \delta_S(\mathbf{x} - \mathbf{x}_s)$, the *sifting* property of Dirac-δ function for interface can be stated as;

$$\int_{\mathcal{V}} \delta_S(\mathbf{x}) f(\mathbf{x}) dv = \int_S f(\mathbf{x}) ds \tag{3.11}$$

For a control volume which contains the interface, the surface tension force can be calculated by taking the integral of $\mathbf{f}_\sigma$ over the interface surface that covers the control volume. Then if the principle shown in Eq.3.11 is applied to that surface integral, surface tension force can be described over the volume

$$\int_S \mathbf{f}_\sigma ds = \int_{\mathcal{V}} \mathbf{f}_\sigma \delta_S dv \tag{3.12}$$

The last step of derivation of one fluid formulation is to add the term on the right hand side of Eq.3.12 to Eq.2.5. The resulting integral equation is;

$$\frac{\partial}{\partial t}\int_{\mathcal{V}} \rho \mathbf{u} dv = -\oint_S \rho \mathbf{u}(\mathbf{u}\cdot\mathbf{n}) ds + \int_{\mathcal{V}} \mathbf{f} dv + \int_{\mathcal{V}} \mathbf{f}_\sigma \delta_S dv + \oint_S \mathbf{n}\cdot\mathbf{T} ds \tag{3.13}$$

Applying the divergence theorem to the integral equation, Eq.3.13, results in the Navier-stokes equation equations for incompressibe flows with the interface

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \mathbf{f} + \mathbf{f}_\sigma \delta_S \tag{3.14}$$

For the sake of completeness, the surface tension force on the right-hand side of Eq.3.14 can be rearranged by using the relation in Eq.3.9 for constant surface tension coefficient;

$$\mathbf{f}_\sigma \delta_S = \int \mathbf{f}_\sigma \delta(\mathbf{x} - \mathbf{x}_s) dA \quad \text{where} \quad f_\sigma = \sigma \kappa \mathbf{n}$$

$$\text{hence} \tag{3.15}$$

$$\mathbf{f}_\sigma \delta_S = \int \sigma \kappa \mathbf{n} \delta(\mathbf{x} - \mathbf{x}_s) dA$$

Therefore incompressible Navier-Stokes equations with surface tension force can be obtained as;

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \mathbf{f} + \int \sigma \kappa \mathbf{n} \delta(\mathbf{x} - \mathbf{x}_s) dA \tag{3.16}$$

3.3 Front-Tracking Method for Advecting the Interface

As discussed in the previous chapter, according to the *one-fluid* formulation, all phases are represented by a single set of equations with variable material properties. The discontinuity due to the interface between the phases enters the Navier-Stokes equation via the integral form of Dirac-δ function. Now the next task is computing the location and orientation of the interface. Front-tracking method is explained briefly in this section.

The front-tracking method utilizes the marker points to define the interface between two phases. It is a very powerful implementation of one-fluid formulation in which the phases are updated according to the solution of a single set of equations. The front-tracking method is essentialy based on the immersed boundary method, [Peskin, 1977]. The history of the front-tracking method goes back to Daly et al. [Daly, 1969] who used the marker points to calculate the surface tension in MAC simulations. Later, Richtmayer and Morton [Richtmyer and Morton, 1967] proposed to use marker points to capture the shock-waves but they didn't provide any formulation. Glimm and McBryan. [Glimm and McBryan, 1985] proposed the first front-tracking algorithm. For moderate Reynolds number regimes of multiphase flows, Unverdi and Tryggvason [Unverdi and Tryggvason, 1992] developed a front-tracking algorithm where marker points were used to represent the interface.

As Unverdi and Tryggvason described in detail [Unverdi and Tryggvason, 1992], in order to represent the interface between the phases in the domain, the marker points are used to form a *Lagrangian* grid. At each time step, the velocity is interpolated from the Eulerian grid onto marker points to move them to their new position for updating the shape of the interface. The Lagrangian grid is also called as *front* in this method and each front element is created by linking two (or three in 3D) marker points with any interpolation scheme.

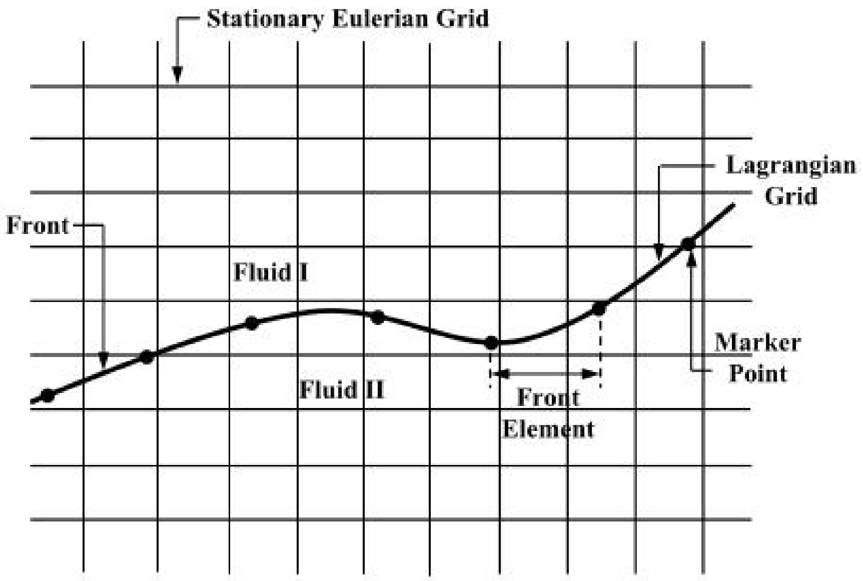


Figure 3.3: Representation of the fixed Eulerian (flow domain) and Lagrangian (interface) grids [Muradoglu and Tryggvason, 2008]

A one-set of equations is solved for the Eulerian grid in Figure 3.3, however, as Eq.3.8 demonstrated that the surface tension force is required to achieve complete solution. In addition, the material properties in each phase must be updated according to the position of the front. Thus, the Eulerian grid should get this information from the Lagrangian. Moreover, at the end of each time step, Lagrangian grid points move to their new positions with the velocity information coming from the Eulerian grid. The *smoothing* operation realizes all these necessary transformations from the fixed grid to the front or from the front to the fixed Eulerian grids.

Discretization and solution procedures of the single set of governing flow equations are not described here. Instead they are described in the next chapter where these procedures are explained for the single grid. In this section, the calculations for advecting the front to update the material properties are going to be discussed.

In a nutshell, there are four fundamental front operations and each of them is presented in the given order:

1. Smoothing the front properties onto the fixed grid

2. Calculation of indicator function

3. Advecting the front

4. Restructuring the front

Before proceeding with the process of indicator function generation, Eq. 3.17 tells that each material property is updated by using corresponding value of that function.

$$\rho(x) = \rho_0 + (\rho_b - \rho_0)I(x) \tag{3.17}$$

$$\mu(x) = \mu_0 + (\mu_b - \mu_0)I(x) \tag{3.18}$$

In Eq.3.17, $I(x)$ denotes the indicator function for the fixed grid.

As is the case with the volume-of-fluid method, an indicator function is introduced for the front-tracking method.

The material properties of different fluids change abruptly across the interface and Dirac-δ function is implemented in the governing equations to model this sudden change. Within the context of the front-tracking method, it is reasonable to represent each phase with the Heaviside function, Eq.3.5. The interface, on the other hand, is labeled by the non-zero gradient of the Heaviside (step) function. Therefore the Dirac-δ function and the gradient of the step function should be correlated. To show the correlation, it is better to start with comparing Dirac-δ function and Heaviside function in Eq.3.19.

$$H(x,y) = \begin{cases} 1, & \text{fluid phase 1} \\ 0, & \text{fluid phase 2} \end{cases} \qquad \int_{-\infty}^{x} \delta(x-x')dx' = \begin{cases} 1, & \text{x' < x} \\ 0, & \text{x' > x} \end{cases} \tag{3.19}$$

Eq.3.19 showed that the Heaviside function is equal to integral of the Dirac-δ function as is shown in Eq.3.20 for 1D case. The dimension can be increased easily by adding δ function in other directions.

$$H(x) = \int_{-\infty}^{x} \delta(x-x')dx' \tag{3.20}$$

If the gradient operator is applied to both sides, the operator can go into the integral on the right-hand side of Eq.3.20 because the operator is defined with respect to the unprimed variables. Moreover, the operator can relate to primed variables inside integral because δ function is antisymmetric in this context. Hence the gradient of the step function is calculated as;

$$\nabla H = -\int_{A} \nabla' \left[\delta(x-x')\delta(y-y')\right] da' \tag{3.21}$$

Here the '-' in Eq.3.21 comes from the conversion from ∇ to ∇'. Eventually area integral in Eq.3.21 can be transformed to line (surface in 3D) integral as;

$$\nabla H = -\oint \delta(x-x')\delta(y-y')\mathbf{n}'ds' \tag{3.22}$$

This gradient field is zero, except the finite region in the vicinity of the interface. The next step is to spread the gradient field onto the fixed grids around the interface by *smoothing* operation. There are several alternative methods to accomplish the *smoothing* the front properties, but every method should maintain the conservation of transferred quantity. For the method explained here the variables ϕ_g and ϕ_f are going to denote the generic variables located on the Eulerian gird and on the front respectively. The conservation principle states that, after each transformation Eq.3.23 must hold.

$$\int_{\Delta s} \phi_f(s)ds = \int_{\Delta v} \phi_g(\mathbf{x})dv \tag{3.23}$$

The approximation to Eq.3.23 for a three-dimensional case can be performed by

$$\phi_{g,ijk} = \sum_l \phi_{f,l}\omega^l_{ijk}\frac{\Delta s_l}{h^3} \tag{3.24}$$

where $\phi_{g,ijk}$ is the Eulerian grid property ϕ_g on the cell with indices i, j, k, which show the computational location of the cell in each direction. The summation on the right hand side is performed over front elements, l. Similar to grid properies, the front property at element l is denoted by $\phi_{f,l}$. ω^l_{ijk} is the weight of grid point i, j, k with respect to the front element l and it decides what proportion of $\phi_{f,l}$ is transferred to $\phi_{g,ijk}$. Finally, h is the cell spacing of the grid and Δs_l is the area of the front element l.

There are many different expressions for weight function in the literature, but in this study, the method suggested by Peskin [Peskin, 1977] is used. The weight function must satisfy the property in Eq.3.25.

$$\sum_{ijk} \omega^l_{ijk} = 1 \tag{3.25}$$

Let $\mathbf{x}_P = (x_p, y_p, z_p)$ be the coordinates of front element point from which smoothing is to be done. The weight function for grid point at i, j, k can be expressed as [Tryggvason et al., 2001]

$$\omega_{ijk}(\mathbf{x}_P) = d(x_p - ih)d(y_p - jh)d(z_p - kh) \tag{3.26}$$

where x_p, y_p and z_p denote the location of the Lagrangian point. The Eulerian grid spacing, h, is assumed to be constant in each direction. To calculate the weight function accurately, the distance function $d(r)$ should be defined properly. In this study we used Peskin's cosine function defined by

$$d(r) = \begin{cases} (1/4h)(1 + cos(\pi r/2h)), & |r|<h \\ 0, & |r|>=2h \end{cases} \tag{3.27}$$

To calculate the indicator function on the fixed grid and then update the material properties, the gradient of the indicator function should be distributed over the fixed grid by the smoothing procedure described above. When gradient of indicator function at the interface, Eq.3.21, is smoothed onto the grid, $G(x, y)$ will be obtained as in Eq.3.28 [Unverdi and Tryggvason, 1992]. Here notice that the generic variable ϕ_g is replaced with G_{ijk}.

$$G_{ijk} = \sum_l \omega^l_{ijk}\mathbf{n}^l \Delta s^l = (\nabla I)_{ijk} \tag{3.28}$$

When Eq.3.28 is applied to all grid points, the gradient field of indicator function, ∇I, is established for fixed grid points. The summation in Eq.3.28 is an approximation to the integral of Dirac-δ function.

However, as Eq.3.17 indicates that, values of indicator function itself, $I(x, y)$ are required in the fixed grid. To obtain the function itself, the divergence operator is applied to both sides of Eq.3.29 to obtain Eq.3.30.

$$\mathbf{G}(x,y) = \nabla I \tag{3.29}$$

$$\nabla \cdot \mathbf{G} = \nabla^2 I \tag{3.30}$$

Eq.3.30 is the Poisson equation whose solution gives the indicator function, $I(x,y)$, on the fixed grid points. It can directly be used to update the material property fields as given by Eq.3.17. Solution of the Poisson equation, 3.30 will be discussed in next chapter.

The surface tension force is another property calculated on the front and the same smoothing operation can be applied to transfer it from the front to the fixed grid. The generic variable ϕ used to explain the smoothing procedure above can be replaced with the surface tension force, $\mathbf{f}_\sigma$.

After the surface tension force is transferred from the front to the fixed grid, coupled system Eq.3.16 and Eq.2.12 are solved for the velocity and pressure fields on the fixed grid. The velocity of the front elements are decided according to the velocity field calculated on the fixed grid. Hence interpolation operation is performed to send the data from the fixed grids to the front elements as;

$$\phi_f = \sum_{ijk} \omega_{ijk} \phi_{ijk} \tag{3.31}$$

The weight function in Eq.3.31 is used to determine what proportion of the grid information at location $i.j.k$ is sent to the front element, l. The summation is over the fixed grids in the vicinity of the front element, l. Even though the interpolation is the reverse operation of smoothing, the same weight function can be used in both applications.

As is the case with the momentum equation on the fixed grid, the first-order Euler method is employed to advect the marker points as;

$$\mathbf{x}_f^{n+1} = \mathbf{x}_f^n + \mathbf{v}_f^n \Delta t \tag{3.32}$$

where $\mathbf{x}_f^{n+1}$ and $\mathbf{x}_f^n$ are the new and old positions of the marker points respectively. $\mathbf{v}_f^n$ is, on the other hand, is the velocity interpolated from the fixed grid.

In most of the cases the conservation cannot be held for the Lagrangian grid. Inaccurate advection results in changing the mass of the enclosed multiphase region as simulation proceeds. The order of accuracy of advection and interpolation are the main sources of the lack of conservation. Using higher order schemes instead of first order Euler method for front advection is relatively easy to implement. Changing the interpolation scheme, on the other hand, may result in a complicated pressure equation [Tryggvason et al., 2001].

Advection of the marker points, Eq.3.32, completes the third step among four important steps given as a list at the beginning of the section. The last step is the restructuring of the front. This is a very important operation since the accuracy and global conservation of numerical setup depends on the quality of the grids. As the front moves within another phase, it changes its shape and orientation. Therefore some parts of the front become excessively refined and some parts remain critically coarse. This situation arises due to the advection of the front elements with different velocities. The regriding procedure checks the Lagrangian grid once in a few time-steps to maintain the quality of front elements at a reasonable level. To decide if regridding is required, each front element is compared with the predefined values of the size of the front elements.

Chapter 4

SINGLE GRID SOLVER

In this chapter, the numerical methods are described to discretize and numerically solve the flow equations. First the grid arrangements are presented. Then the projection method, the strategies for elliptic solvers and appropriate boundary conditions are described. In later chapters the adaptive mesh refinement technique will be built upon numerical implementations described in this chapter.

4.1 Grid Arrangements

Before studying the numerical methods used to approximate the governing PDEs in discrete form, it is required to decide where the flow variables and material properties are stored in a computational cell. Different *grid arrangements* can be used depending on the issues, such as numerical stability and accuracy. For some numerical methods, such as second-order projection method [Bell et al., 1989], more than one grid arrangement can be used. Here the standard Staggered grid arrangement is employed.

Even though there are various possible grid arrangements that suit different numerical methods, two main arrangements are mostly preferred in CFD studies. These two techniques can be distinguished by the locations of material properties and flow quantities on the computational cell.

- Collocated grid arrangement

- Stagerred grid arrangement

4.1.1 Collocated Grid Arrangement

In the collocated grid arrangement, all fluid/flow quantities are stored at the center or nodes of the computational cell. The cell-centered arrangement is usually preferred in application. This type of arrangement greatly simplifies the implementation of the numerical method and data management becomes straightforward. The collocated grid allows to use the same interlevel communication operators for each variable in multigrid solvers. Similarly in adaptive mesh refinement (AMR) applications, for all variables, the same interpolation schemes can be used to update the refinement levels. In most of the AMR packages including the *AMReX* [Zhang et al., 2019], there is a large number of options to perform cell-centered interpolations. The accuracy of the numerical approximation of the nonlinear terms in the advective term in Eq.2.13 significantly increases when all velocity components are stored at the same location.

In spite of the advantages of collocated grid arrangement, lack of velocity-pressure coupling constitutes an important disadvantage. In most of the cases, inaccurate coupling results in nonphysical oscillatory behaviors in pressure fields. The error-prone coupling in the collocated grid for the two-dimensional case is shown in Figure 4.1.

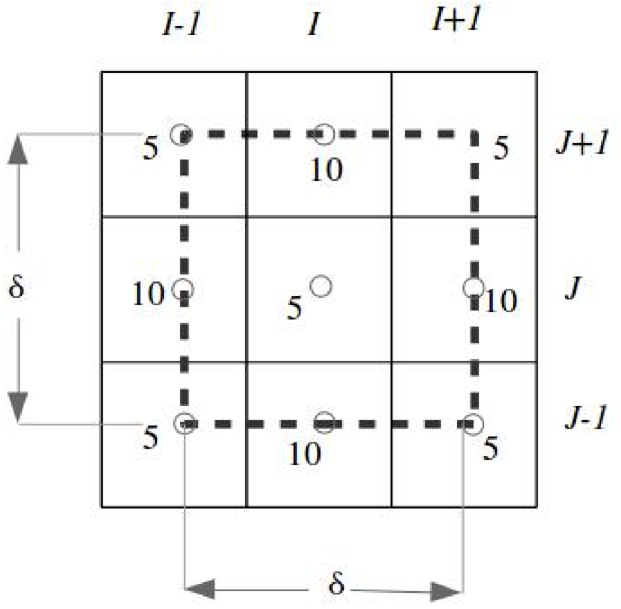


Figure 4.1: Pressure values stored in the collocated grid to represent the checker-board problem

As seen in Figure 4.1 the checker-board problem occurs when there is a highly irregular pressure field within the domain such that it wiggles, e.g from *5* to *10* from one grid location to another. The indices I and J show the grid in x and y directions respectively. Referring to Figure 4.1, it can be seen that the pressure gradient calculated at location *I,J* by using central differences becomes zero, e.g

$$\frac{\partial P}{\partial x} = \frac{P_{I+1,J}}{P_{I-1,J}} = \frac{10-10}{\delta} = 0, \tag{4.1}$$

$$\frac{\partial P}{\partial y} = \frac{P_{I,J+1}}{P_{I,J-1}} = \frac{10-10}{\delta} = 0. \tag{4.2}$$

Figure 4.1 and Eq.4.1-4.2 clearly show that the collocated grid produces zero pressure field within the domain, even though it is highly oscillatory. This means that the collocated grid arrangement allows to have nonphysical pressure oscillations when central differences are used. Although Rhie and Chow [Rhie and Chow, 1983] developed an interpolation scheme to eliminate the checker-board problem from the collocated grid, oscillatory behavior still persists when nonlinearity increases.

4.1.2 Staggered Grid Arrangement

As Figure 4.1 and Eq.4.1-4.2 suggested, the effect of pressure gradient on velocities can be lost if all variables are located at the same position within the cell. To solve this problem, Harlow and Welch [Harlow and Welch, 1965] proposed the *staggered grid* arrangement within the context of the Marker and Cell (MAC) method. In this arrangement, regardless of the numerical method pressure is used to enforce the continuity equation, Eq.2.12, for the incompressible flows. To compute the mass flux through the boundaries of pressure control volume accurately, vectorial quantities, such as velocity and surface tension force, are located at the face of the computational cell in the staggered grid. Conversely, the scalar variables, such as material properties and pressure, are stored at the cell centers. To take the advantage of simplicity of the collocated grid in programming, in the staggered grid arrangement, different control volumes are used to represent variables as shown in Figure 4.2. In order to put the u and v velocities at the center, the velocity control volumes are separated from scalar control volume and they shift by half cell spacing to right and up respectively. However, it should be noted that implementation is not as easy as that in the collocated grid even with this modification.

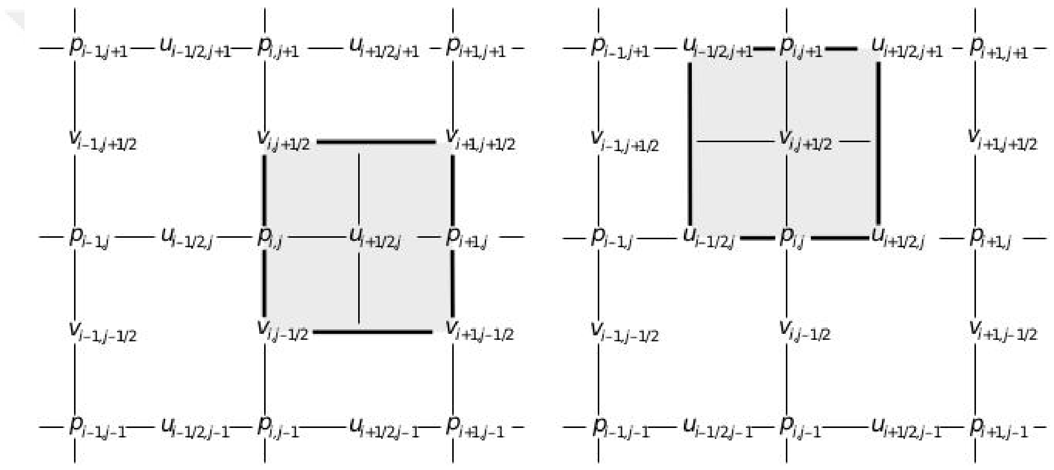


Figure 4.2: Different control volumes in the staggered grid arrangement for u- and v-components of the velocity field [Tryggvason et al., 2011]

As is the case with the collocated grid, indices *i,j* show the center coordinates of the cell. *i+1/2,j* and *i,j+1/2*, on the other hand, show the center of u and v velocity control volumes. The staggered grid offers great advantages over the collocated grid. It provides direct coupling between the velocity and pressure fields, and this eliminates the checker-board problem. In addition, when the rectangular domain is used, the inflow boundary condition can be specified at the boundary explicitly. The other advantage is that it provides a divergence-free velocity field with great accuracy because the velocity components that are used to compute derivatives in $\nabla \cdot \mathbf{u}$ are already available in cell faces. However, the location of velocities can create a problem when the advective term in the Navier-Stokes equation is discretized because the cross derivatives require the averaging of the velocity components.

4.2 Numerical Solution to the Navier-Stokes Equations

The Navier-Stokes equations for single phase flow were derived in the previous chapter. The one-fluid formulation for multiphase flows requires the numerical approximation to conservation equations as if there is a single phase in the domain. The presence of another phase is introduced by advecting the material properties according to the position of the interface. In addition, the surface tension force is included in the Navier-stokes equation. In this section, numerical methods used to discretize and solve the governing equations are presented. The important steps of numerical procedures can be given in the following order:

- Setting the grid arrangement
- Deciding time integration scheme for the momentum equations
- Spatial discretization of the Navier-Stokes equations
- Imposing the boundary conditions
- Solving the pressure equation
- Correcting the unprojected velocity to satisfy mass conservation.

This section focuses on the single grid time advancement, however, the same approach will be taken in each refinement level when the AMR is implemented. In this study, numerical methods are implemented on the *staggered grid* arrangement as explained in the previous section.

4.2.1 Time Integration Scheme: Projection Method

Since the pressure field is used to satisfy the continuity equation for the incompressible flows, the predictor-corrector approach, such as *projection method*, is preferred as a time integration scheme.

The complication of the projection method may vary depending on the grid arrangements and where the variables are located on the cell. The staggered grid offers simplicity and accuracy although it introduces additional approximations while discretizing the advective and viscous terms.

The projection method consists of two steps: (1) In the predictor step the temporary velocity field is obtained by excluding the pressure gradient. This velocity field does not satisfy the continuity equation in most of the cases. (2) The corrector step projects the temporary velocity field onto the divergence-free vector space by using the pressure gradient term. The concept of the projection method was initially proposed by Chorin [Chorin, 1968] and Yanenko et al. [Yanenko, 1971].

For the simplicity, the projection method is combined with a first order *Euler* scheme for time integration. However, higher-order time integration schemes can also be used. Before proceeding, it is better to define some specific notations used by Zaleski et al. [Tryggvason et al., 2011]. Their formulation is followed for explaining the single grid discretization of the governing equations. The non-conservative Navier-Stokes equation for incompressible flow given in Eq.2.13 is expressed in compact form as;

$$\rho(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{A}) = -\nabla p + \mathbf{D} + \mathbf{f} \tag{4.3}$$

In Eq.4.3 the advective and viscous terms are denoted by $\mathbf{A} = \nabla \cdot (\mathbf{uu})$ and $\mathbf{D} = \nabla \cdot \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)$. Finally $\mathbf{f}$ denotes all forces acting on the fluid element, i.e both body and the surface tension forces.

An explicit forward Euler method is employed for time integration and we obtain;

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \mathbf{A}_h^n = \frac{1}{\rho^n}(-\nabla_h p + \mathbf{D}_h^n + \mathbf{f}^n) \tag{4.4}$$

where Δt denotes the time step size and the superscript n denotes the current time step. The subscript that h, appears in the terms $\mathbf{A}_h, \mathbf{D}_h$ and $\nabla_h p$, indicates the spatial discretization of the corresponding terms. It should not be confused with the cell spacing h.

After discretization of the Navier-Stokes, the incompressible continuity equation can be approximated using central differences as;

$$\nabla_h \cdot \mathbf{u}^{n+1} = 0 \tag{4.5}$$

The velocity field at the new time step obtained from Eq.4.4 should also satisfy Eq.4.20 by using an appropriate pressure gradient. However, there is no explicit equation to find the pressure gradient. The projection method finds solutions to all velocity components and pressure appearing in Eq.4.4 in two steps by using the *Hodge decomposition* method. It is useful to explain this method before proceeding with the projection method.

Hodge Decomposition

According to the *Hodge decomposition*, a vector field ψ can be decomposed for simply connected domain and smooth boundary [Minion, 1996] as;

$$\psi = \psi^D + \nabla\phi \tag{4.6}$$

where ϕ is a scalar field. ψ^D is a divergence-free component of ψ and it satisfies the following conditions given.

$$\nabla \cdot \psi^D = 0 \quad \text{in} \quad \Omega, \quad \psi^D \cdot \mathbf{n} = 0 \quad \text{on} \quad \partial\Omega \tag{4.7}$$

where Ω and $\partial\Omega$ denote the simply connected domain and smooth boundary of the domain, respectively.

Since the divergence of the vector field ψ^D is zero, Eq.4.8 also must hold when divergence operator is applied to the both sides of Eq.4.6, e.g

$$\nabla \cdot (\nabla \phi) = \nabla \cdot \psi \tag{4.8}$$

The information on the boundary can be obtained as in Eq.4.9 whenever Eq.4.8 satisfies the requirements of the divergence theorem.

$$\frac{\partial \phi}{\partial n} = \psi \cdot \mathbf{n} \quad on \quad \partial\Omega \tag{4.9}$$

Eq.4.8 - 4.9 forms together a *Neumann problem.* The projection method extracts the divergence free vector field, ψ^D, from the field itself, ψ, by solving a Poisson problem stated in Eq.4.8 with the Neumann boundary conditions for ϕ. The projection on ψ can be defined as

$$\mathbf{P}(\psi) = \psi^D, \tag{4.10}$$

where $\mathbf{P}()$ in Eq.4.10 indicates the projection operation. We then obtain from Eq.4.10 and Eq.4.6 that

$$(\mathbf{I} - \mathbf{P})\psi = \nabla \phi. \tag{4.11}$$

The Hodge decomposition and projection operation are used in the derivation of the time integration scheme for the Navier-Stokes equations in this chapter. Before completing the discussion on the Hodge decomposition, it is important to note that $\psi^D \cdot \mathbf{n}$ does not necessarily be equal to zero. It can take a finite value such that

$$\int_{\partial\Omega} g = 0 \quad \text{where} \quad \psi^D \cdot \mathbf{n} = g \tag{4.12}$$

After completing the brief discussion on the Hodge decomposition, the application of the projection method to the Navier-Stokes equations can start with computing the temporary velocity field by ignoring the pressure gradient

$$\frac{\mathbf{u}^\star - \mathbf{u}^n}{\Delta t} = -\mathbf{A}_h^n + \frac{1}{\rho^n}(\mathbf{D}_h^n + \mathbf{f}^n). \tag{4.13}$$

Hence, the temporary velocity field can be obtained by rearranging Eq.4.13 as

$$\mathbf{u}^\star = \mathbf{u}^n + \Delta t(-\mathbf{A}_h^n + \frac{1}{\rho^n}(\mathbf{D}_h^n + \mathbf{f}^n)). \tag{4.14}$$

In the second step, the pressure gradient is used to correct $\mathbf{u}^\star$ as

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^\star}{\Delta t} = -\frac{1}{\rho^n}\nabla_h p. \tag{4.15}$$

It is useful to establish analogy between predictor-corrector scheme given in Eq.4.13-4.15 and the Hodge decomposition in Eq.4.6. If $\mathbf{u}^\star$ is decomposed and the projection operator, $\mathbf{P}()$, is applied on $\mathbf{u}^\star$

$$\mathbf{P}(\mathbf{u}^\star) = \mathbf{u}^{n+1} \quad \text{hence} \quad \mathbf{u}^\star = \mathbf{u}^{n+1} + \frac{1}{\rho^n}\nabla_h p \tag{4.16}$$

Here $\mathbf{u}^\star$, $\mathbf{u}^{n+1}$ and $\nabla_h p$ are analogous to ψ, ψ^D and $\nabla\phi$, in Eq.4.6 respectively.

To find the pressure field, divergence operator is applied to the both sides of Eq.4.16 to eliminate the divergence-free velocity field $\mathbf{u}^{n+1}$. The resulting equation is called the *Poisson equation* for the pressure field.

$$\nabla_h \cdot \left(\frac{1}{\rho^n}\nabla_h p\right) = \frac{1}{\Delta t}\nabla_h \cdot \mathbf{u}^\star \tag{4.17}$$

After the pressure field is calculated by solving Eq.4.17, the divergence-free velocity field for the new time step can be evaluated by using Eq.4.15.

4.2.2 Discretization of Advection and Viscous Terms

To compute $(\nabla \cdot \mathbf{u}^\star)$, Eq.4.13 is discretized using a *finite volume method* (FVM) where each term in the governing conservation equations are converted into volume integrals.

Since each phase within the domain is represented by a single set of equations, the interface introduces a sharp transition between the phases. The integral form of FVM eliminates the disadvantage of having discontinuous regions within the domain, hence it is very compatible with the one-fluid formulation.

FVM starts with applying divergence theorem to volume average of the terms in Eq.4.13 over control volume $\mathcal{V}$, i.e

$$\begin{aligned}
\mathbf{A}_c &= \frac{1}{\mathcal{V}}\int_{\mathcal{V}} \nabla\cdot(\mathbf{uu})\ dv = \frac{1}{\mathcal{V}}\oint_S \mathbf{u}(\mathbf{u}\cdot\mathbf{n})\ ds,\\
\mathbf{D}_c &= \frac{1}{\mathcal{V}}\int_{\mathcal{V}} \nabla\cdot(\mathbf{T}^v)\ dv = \frac{1}{\mathcal{V}}\oint_S \mathbf{T}^v\cdot\mathbf{n}\ ds,\\
(\nabla p)_c &= \frac{1}{\mathcal{V}}\int_{\mathcal{V}} \nabla p\ dv = \frac{1}{\mathcal{V}}\oint_S p\cdot\mathbf{n}\ ds,\\
\mathbf{f}_c &= \frac{1}{\mathcal{V}}\int_{\mathcal{V}} \mathbf{f}(x)\ dv,
\end{aligned} \tag{4.18}$$

where the subscript c denotes the integral representation, $\mathbf{T}^v$ is stress tensor and S is the boundary of the control volume $\mathcal{V}$.

Similarly, the continuity equation can be represented by volume average as

$$\frac{1}{\mathcal{V}}\int_{\mathcal{V}} \nabla\cdot\mathbf{u}^{n+1} = \oint_S \mathbf{u}^{n+1}\cdot\mathbf{n}\ ds = 0 \tag{4.19}$$

A second-order central differencing scheme is used to approximate the integral equations given in Eq.4.18 and 4.19. However, difference schemes should be used carefully depending on the flow configuration. For instance, high order upwind schemes can eliminate the stability problem that may arise for inviscid flows, but it may deteriorates the numerical accuracy for the viscous flows.

For simplicity, the derivations are performed for the two-dimensional case by employing central differencing in this chapter. However, the formulation can easily be extended to three-dimensions.

As a grid arrangement, the staggered grid is used to take advantage of accurate representations of derivatives. As depicted in Figure 4.2 the u velocity control volume is shifted by half grid spacing from the scalar control volume i, j to $i + 1/2, j$. Similarly the v velocity control volume is shifted to $i, j + 1/2$. In this arrangement the continuity equation is discretized using central differences as

$$\frac{1}{\Delta x \Delta y}\left[(u^{n+1}_{i+1/2,j} - u^{n+1}_{i-1/2,j})\Delta y + (v^{n+1}_{i,j+1/2} - v^{n+1}_{i,j-1/2})\Delta x\right] = 0 \tag{4.20}$$

which can be rearranged as

$$\frac{u^{n+1}_{i+1/2,j} - u^{n+1}_{i-1/2,j}}{\Delta x} + \frac{v^{n+1}_{i,j+1/2} - v^{n+1}_{i,j-1/2}}{\Delta y} = 0. \tag{4.21}$$

The momentum equation is also approximated in the same manner. The temporary velocity in the x and y directions are obtained from Eq.4.22 and Eq.4.23, respectively.

$$\begin{aligned} u^{\star}_{i+1/2,j} = u^{n}_{i+1/2,j} + \Delta t\{(-A_x)^{n}_{i+1/2,j} + (f_x)^{n}_{i+1/2,j} \\ + \frac{1}{1/2(\rho^{n}_{i+1,j} + \rho^{n}_{i,j})}\left[(D_x)^{n}_{i+1/2,j} + (f_x)^{n}_{i+1/2,j}\right]\}, \end{aligned} \tag{4.22}$$

$$\begin{aligned} v^{\star}_{i,j+1/2} = v^{n}_{i,j+1/2} + \Delta t\{(-A_y)^{n}_{i,j+1/2} + (f_y)^{n}_{i,j+1/2} \\ + \frac{1}{1/2(\rho^{n}_{i,j+1} + \rho^{n}_{i,j})}\left[(D_y)^{n}_{i,j+1/2} + (f_y)^{n}_{i,j+1/2}\right]\}. \end{aligned} \tag{4.23}$$

Similarly, the projection $\mathbf{P}(\mathbf{u}^{\star})$ in Eq.4.16 is approximated as

$$u^{n+1}_{i+1/2,j} = u^{\star}_{i+1/2,j} - \frac{\Delta t}{1/2(\rho^{n}_{i+1,j} + \rho^{n}_{i,j})}\frac{p_{i+1,j} - p_{i,j}}{\Delta x}, \tag{4.24}$$

$$v^{n+1}_{i,j+1/2} = v^{\star}_{i,j+1/2} - \frac{\Delta t}{1/2(\rho^{n}_{i,j+1} + \rho^{n}_{i,j})}\frac{p_{i,j+1} - p_{i,j}}{\Delta y}. \tag{4.25}$$

To complete the discretization the advective terms $(A_x)_{i+1/2,j}$, $(A_y)_{i,j+1/2}$ and viscous terms $(D_x)_{i+1/2,j}$,$(D_y)_{i,j+1/2}$ should also be approximated.

Advective Terms

The volume integral of the advective terms is

$$\mathbf{A}_c = \oint_S \mathbf{u}(\mathbf{u}\cdot\mathbf{n})\; ds \quad \text{where} \quad \mathbf{u} = u\hat{\imath} + v\hat{\jmath} \tag{4.26}$$

The x componment of the advective term is approximated by using the central differences;

$$(A_x)_{i+1/2,j} = \frac{1}{\Delta x \Delta y}\{[(uu)_{i+1,j} - (uu)_{i,j}]\,\Delta y + \\ \left[(uv)_{i+1/2,j+1/2} - (uu)_{i+1/2,j-1/2}\right]\Delta x\} \tag{4.27}$$

Similar approximation is also applied to the y-component to get

$$(A_y)_{i,j+1/2} = \frac{1}{\Delta x \Delta y}\{\left[(uv)_{i+1,j+1/2} - (uv)_{i-1/2,j+1/2}\right]\Delta y + \\ [(vv)_{i,j} - (vv)_{i,j}]\,\Delta x\} \tag{4.28}$$

In Eq.4.27 and 4.28 velocity fluxes are located at the faces of the velocity control volumes which coincide with the center of the scalar control volumes. Since u and v velocities are not explicitly defined at cell centers, they are approximated by using the alternative schemes. The decision on the scheme should be made by taking the physics into account. For demonstration purposes, the linear interpolation scheme is used in this section, but alternative schemes are also discussed. When fluxes are approximated x and y components of the advection term become

$$\begin{aligned}(A_x)^n_{i+1/2,j} =& \frac{1}{\Delta x}\left[\left(\frac{u^n_{i+3/2,j} + u^n_{i+1/2,j}}{2}\right)^2 - \left(\frac{u^n_{i+1/2,j} + u^n_{i-1/2,j}}{2}\right)^2\right] \\ &+ \frac{1}{\Delta y}\left[\left(\frac{u^n_{i+1/2,j+1} + u^n_{i+1/2,j}}{2}\right)\left(\frac{v^n_{i+1,j+1/2} + v^n_{i,j+1/2}}{2}\right)\right] \\ &- \frac{1}{\Delta y}\left[\left(\frac{u^n_{i+1/2,j} + u^n_{i+1/2,j-1}}{2}\right)\left(\frac{v^n_{i+1,j-1/2} + v^n_{i,j-1/2}}{2}\right)\right]\end{aligned} \tag{4.29}$$

$$
\begin{aligned}
(A_y)^n_{i,j+1/2} = & \frac{1}{\Delta x}\left[\left(\frac{u^n_{i+1/2,j}+u^n_{i+1/2,j+1}}{2}\right)\left(\frac{v^n_{i,j+1/2}+v^n_{i+1,j+1/2}}{2}\right)\right] \\
& - \frac{1}{\Delta x}\left[\left(\frac{u^n_{i-1/2,j+1}+u^n_{i-1/2,j}}{2}\right)\left(\frac{v^n_{i,j+1/2}+v^n_{i-1,j+1/2}}{2}\right)\right] \\
& + \frac{1}{\Delta y}\left[\left(\frac{v^n_{i,j+3/2}+v^n_{i,j+1/2}}{2}\right)^2 - \left(\frac{v^n_{i,j+1/2}+u^n_{i,j-1/2}}{2}\right)^2\right]
\end{aligned}
\tag{4.30}
$$

The central differencing scheme is second-order accurate, hence it is advantageous over other non-centered schemes. However, it is sensitive in some cases. It can give non-physical oscillatory results when discontinuities are not fully resolved. On the other hand, forward-in-time centered-in-space (FTCS) schemes are unconditionally unstable for inviscid flows. Therefore artificial viscosity should be added to provide stability, however, in that case, non-optimized viscosities can smear out the problem.

There are alternative and more stable schemes to be used especially for inviscid flows. The *upwinding scheme* approximates the values at the edge of velocity control volumes and for $u_{i,j}$;

$$
u_{i,j} = \begin{cases} u_{i-1/2,j}, & \text{if} \quad {}^1\!/_2\left(u_{i-1/2,j}+u_{i+1/2,j}\right) > 0 \\ u_{i+1/2,j}, & \text{if} \quad {}^1\!/_2\left(u_{i-1/2,j}+u_{i+1/2,j}\right) < 0 \end{cases}
\tag{4.31}
$$

The same relation is valid for the v velocity as well. Even though Eq.4.31 is stable, the accuracy is its biggest disadvantage. Since it is first-order accurate, numerical diffusion can produce non-physical results. The remedy to the accuracy problem is to use higher-order upwind schemes. One option is to use the *QUICK* (quadratic upstream interpolation for convective kinematics) scheme;

$$
u_{i,j} = \begin{cases} {}^1\!/_8(3u_{i+1/2,j}+6u_{i-1/2,j}-u_{i-3/2,j}), & \text{if} \quad {}^1\!/_2\left(u_{i-1/2,j}+u_{i+1/2,j}\right) > 0 \\ {}^1\!/_8(3u_{i-1/2,j}+6u_{i+1/2,j}-u_{i+3/2,j}), & \text{if} \quad {}^1\!/_2\left(u_{i-1/2,j}+u_{i+1/2,j}\right) < 0 \end{cases}
\tag{4.32}
$$

Since the problem is viscous, both centered differencing and QUICK schemes are implemented.

Viscous Terms

Viscous terms can be approximated using central differences

$$(D_x)^n_{i+1/2,j} = \frac{T^{v,xx}_{i+1,j} - T^{v,xx}_{i,j}}{\Delta x} + \frac{T^{v,xy}_{i+1/2,j+1/2} - T^{v,xy}_{i+1/2,j-1/2}}{\Delta y} \tag{4.33}$$

and

$$(D_y)^n_{i,j+1/2} = \frac{T^{v,xy}_{i+1/2,j+1/2} - T^{v,xy}_{i-1/2,j+1/2}}{\Delta x} + \frac{T^{v,yy}_{i,j+1} - T^{v,yy}_{i,j}}{\Delta y} \tag{4.34}$$

In Eq.4.33 and 4.34 $\mathbf{T}^v$ denotes the viscous fluxes located at the faces of the velocity control volumes. Fluxes in the normal direction is approximated as

$$T^{v,xx}_{i,j} = \left[2\mu(\frac{\partial u}{\partial x})\right]_h = 2\mu_{i,j}\frac{u^n_{i+1/2,j} - u^n_{i-1/2,j}}{\Delta x} \tag{4.35}$$

$$T^{v,yy}_{i,j} = \left[2\mu(\frac{\partial v}{\partial y})\right]_h = 2\mu_{i,j}\frac{v^n_{i,j+1/2} - v^n_{i,j-1/2}}{\Delta y} \tag{4.36}$$

and in tangential direction

$$\begin{aligned} T^{v,xy}_{i+1/2,j+1/2} &= \left[\mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right]_h = \\ \mu^n_{i+1/2,j+1/2}&\left(\frac{u^n_{i+1/2,j+1} - u^n_{i+1/2,j}}{\Delta y} + \frac{v^n_{i+1,j+1/2} - v^n_{i,j+1/2}}{\Delta x}\right) \end{aligned} \tag{4.37}$$

It is assumed that the viscosity is constant within each phase, but can change across the interface. Therefore the indicator function calculated by solving the Poisson equation in Eq.3.29 is used to update the viscosity as;

$$\mu(\mathbf{x}) = \mu_1 H(\mathbf{x}) + \mu_2[1 - H(\mathbf{x})] \tag{4.38}$$

In the staggered grid arrangement, viscosity is kept at the center of the cell. However, in Eq.4.37 viscosity is required at the node of the scalar control volume.

Therefore it is approximated by taking an average of four viscosities surrounding that particular node as

$$\mu_{i+1/2,j+1/2} = \frac{1}{4}\left(\mu_{i,j} + \mu_{i+1,j} + \mu_{i.j+1} + \mu_{i+1,j+1}\right) \tag{4.39}$$

Eq.4.35 and 4.37 complete the required discretizations for the temporary velocity field calculation in Eq.4.14

4.2.3 Time Step Restrictions

For explicit time integration methods, such as the forward-Euler method, obtaining stable solutions depends on the relation between temporal and spatial resolutions. Both advective and diffusive terms bring restriction on the time step. The CFL condition for the advective term requires

$$\frac{u_{max}\Delta t}{h}\leq 1 \tag{4.40}$$

where u_{max} is the maximum velocity in the flow domain and it is calculated as $u_{max} = \sqrt{u^2 + v^2 + w^2}$. The CFL condition requires that the time step is sufficiently small so that the information travels not more than single cell size in a single time-step. The viscous term also brings a constraint on the time step size. Since it is discretized by central differencing the restriction can be given by

$$\frac{\mu\Delta t}{\rho h^2}\leq\frac{1}{6} \tag{4.41}$$

where h is the smallest cell spacing in each direction. For two-dimensional case the right hand side becomes $\frac{1}{4}$.

4.2.4 Pressure Equation

After $\mathbf{u}^\star$ is computed, the pressure field is evaluated by Eq.4.17 in order to calculate the divergence-free velocity field at the new time step. The velocity components at the new time step is given by;

$$u^{n+1}_{i+1/2,j} = u^{\star}_{i+1/2,j} - \frac{\Delta t}{{}^{1}\!/\!{}_{2}(\rho_{i+1,j} + \rho_{i,j})} \frac{p_{i+1,j} - p_{i,j}}{\Delta x} \tag{4.42}$$

and

$$v^{n+1}_{i,j+1/2} = v^{\star}_{i,j+1/2} - \frac{\Delta t}{{}^{1}\!/\!{}_{2}(\rho_{i,j+1} + \rho_{i,j})} \frac{p_{i,j+1} - p_{i,j}}{\Delta y} \tag{4.43}$$

To obtain the discretized form of Eq.4.15, Eq.4.42 and 4.43 are put into Eq.2.12 to get

$$\begin{aligned} &\frac{1}{\Delta x^2}\left(\frac{p_{i+1,j} - p_{i,j}}{\rho^n_{i+1,j} - \rho^n_{i,j}} - \frac{p_{i,j} - p_{i-1,j}}{\rho^n_{i,j} - \rho^n_{i-1,j}}\right) + \frac{1}{\Delta y^2}\left(\frac{p_{i,j+1} - p_{i,j}}{\rho^n_{i,j+1} - \rho^n_{i,j}} - \frac{p_{i,j} - p_{i,j-1}}{\rho^n_{i,j} - \rho^n_{i,j-1}}\right) \\ &= \frac{1}{2\Delta t}\left(\frac{u^{\star}_{i+1/2,j} - u^{\star}_{i-1/2,j}}{\Delta x} + \frac{v^{\star}_{i,j+1/2} - v^{\star}_{i,j-1/2}}{\Delta y}\right). \end{aligned} \tag{4.44}$$

Eq.4.44 is called the *pressure equation* which is discretized version of the Poisson equation, Eq.4.17. There are many advanced and well-developed elliptic solvers to tackle with Eq.4.44 such as relaxation and multigrid solvers. As the solver gets complicated, the efficiency generally increases but implementation also gets harder. Multigrid methods have been used in many multiphase flow studies because they provide computational efficiency without loss of accuracy.

The accuracy is increased as the resolution of the mesh gets finer. However, the time step size is also restricted by stability conditions given in Eq.4.40 and 4.41. Thus, computational efficiency decreases dramatically as the cell size gets smaller, because the low-frequency errors are eliminated much later than the high-frequency errors. Multigrid methods offer a solution procedure where low and high-frequency errors are treated in different levels. The iteration starts with the finest grid and continues until the high-frequency wavenumbers are removed. Then the approximate solution is passed to the coarser grid where the lower wavenumber errors can be eliminated. After several iterations in the coarser grid, the correction is added to the fine grid solution. The transfer between the grids is realized by special operations. To send the data from the fine to the coarse grid, the restriction algorithm is used where the approximate solution in the fine grid is averaged down. The operation from the coarse grid to the fine grid, on the other hand, is called interpolation and different schemes are used according to where the data is stored in the computational cell. After coarse grid data is supplied to the fine grid, it is iterated a few more times to eliminate errors due to intergrid operations. HYPRE package, which is used in this study, also uses multigrid method as main solver and the details of that package is provided in the next section. In the adaptive mesh refinement applications the same operations, restriction and interpolation, are used to set communication between the refinement levels.

Even though the multigrid method provides high efficiency, it has convergence issues for high-density ratio applications and implementation is not straightforward. The *relaxation methods*, alternatively, are very easy to implement, although they are far less efficient than the multigrid method. Relaxation methods can be preferred to gain insight about the problem quickly and then high-efficiency solvers can be implemented. In this study, *successive over-relaxation (SOR)* is used to obtain preliminary results. The SOR method computes the value of pressure at new timestep after the relative error between two successive iterations becomes less than a tolerance value. For a new iteration of $p_{i,j}^{\alpha+1}$ at location *i,j* the SOR method takes the weighted average of $p_{i,j}^{\alpha}$ and the current updates for neighbouring grids. Hence $p_{i,j}^{\alpha+1}$ can be computed as;

$$C_\rho = \left[\frac{1}{\Delta x^2}\left(\frac{1}{\rho_{i+1,j}^n+\rho_{i,j}^n}+\frac{1}{\rho_{i,j}^n+\rho_{i-1,j}^n}\right)+\frac{1}{\Delta y^2}\left(\frac{1}{\rho_{i,j+1}^n+\rho_{i,j}^n}+\frac{1}{\rho_{i,j}^n+\rho_{i,j-1}^n}\right)\right]^{-1}$$

$$C_p = \left[\frac{1}{\Delta x^2}\left(\frac{p_{i+1,j}^{\alpha}}{\rho_{i+1,j}^n+\rho_{i,j}^n}+\frac{p_{i-1,j}^{\alpha+1}}{\rho_{i,j}^n+\rho_{i-1,j}^n}\right)+\frac{1}{\Delta y^2}\left(\frac{p_{i,j+1}^{\alpha}}{\rho_{i,j+1}^n+\rho_{i,j}^n}+\frac{p_{i,j-1}^{\alpha+1}}{\rho_{i,j}^n+\rho_{i,j-1}^n}\right)\right]$$

$$C_{\mathbf{u}^\star} = \frac{1}{2\Delta t}\left(\frac{u_{i+1/2,j}^\star-u_{i-1/2,j}^\star}{\Delta x}+\frac{v_{i,j+1/2}^\star-v_{i,j-1/2}^\star}{\Delta y}\right)$$

$$p_{i,j}^{\alpha+1} = \beta\left[C_\rho\right]^{-1}\times\left[C_p+C_{\mathbf{u}^\star}\right]+(1-\beta)p_{i,j}^{\alpha}$$

(4.45)

where β is the relaxation factor. For over relaxation schemes it should be greater than 1 and due to consistency concerns it cannot exceed 2. There is an expression for the optimum value of β, but around $\beta = 1.5$ usually gives good results. As stated before, implementation simplicity is the biggest advantage of the SOR method.

4.3 Boundary Conditions

The boundary conditions are needed to complete the differential equation system, hence they are critical to simulate actual physical solutions. In this section, several boundary conditions implemented in our benchmark problems are discussed.

4.3.1 Periodic Boundary Conditions

The periodicity condition represents the infinite domain configuration in the direction it is implemented. The domain is treated as a unit cell and the solution is copied from one side of the domain to the other. Even though it is easy to implement, in multiphase simulations the interface should be located carefully. For flow simulations in two-dimensional configuration, the implementation of periodicity can be seen as;

$$\begin{matrix} u_{nx,j} = u_{1,j} & v_{nx,j} = u_{1,j} \\ u_{nx+1,j} = u_{i+1,j} & v_{nx+1,j} = v_{i+1,j} \end{matrix} \quad \textit{and} \quad p_{nx,j} = p_{1,j} \tag{4.46}$$

The periodic boundary condition is preferred for preliminary runs of the simulation because the solution is easy to converge when preiodicity is applied.

4.3.2 No-Slip Boundary Condition

In many cases, implementing normal velocity boundary conditions in the staggered grid arrangement is very straightforward, because the boundary of the velocity control volume coincides with the physical domain boundary. However, the tangential velocity requires a little more attention. The *no-slip* or *wall* boundary condition requires that the velocity should be zero. Figure 4.3 shows a typical case in two-dimensions

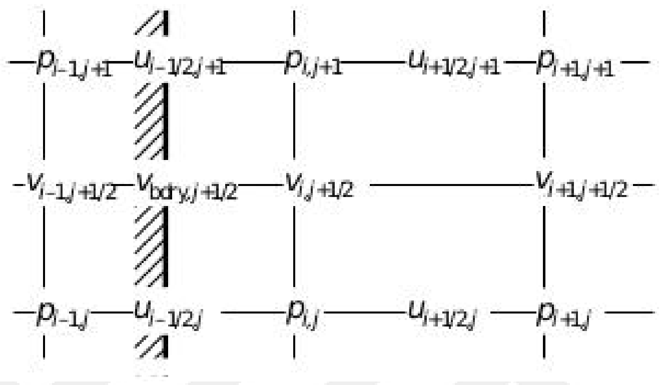


Figure 4.3: Noslip boundary condition in x direction can be implmented by using the ghost cells [Tryggvason et al., 2011]

The no-slip boundary condition is applied to v velocity for grid in Figure 4.3. Since v velocity is not available on the wall in x direction, the *ghost cells* are used to impose the boundary condition. These *ghost cells* lie outside of the computational domain and their values can be obtained by interpolation. The linear interpolation suggests that

$$v_{bdry,j+1/2} = \frac{v_{i,j+1/2} + v_{i-1,j+1/2}}{2} \tag{4.47}$$

In Eq.4.47, $v_{bdry,j+1/2}$ is the tangential velocity located on the wall. No-slip condition requires that it should be equal to zero, therefore the ghost cell, $v_{i-1,j+1/2}$, can be obtained as;

$$v_{i-1,j+1/2} = -v_{i,j+1/2} \tag{4.48}$$

Eq.4.48 can also be applied to the slip boundary conditions. In that case, $v_{bdry,j+1/2}$ is going to take a finite value, therefore the ghost cell should be updated as;

$$\begin{aligned} u_{i-1/2,j} &= u_{bdry,j} \\ v_{i-1,j+1/2} &= 2v_{bdry,j+1/2} - v_{i,j+1/2} \end{aligned} \tag{4.49}$$

4.3.3 Inflow Boundary Condition

Inflow boundary condition sets the mass flow rate with prescribed velocity profile at the inlet of the domain. For the two-dimensional domain where the mass flow is introduced at the location $x = 0$, the velocity and the pressure conditions can be stated as;

$$\begin{aligned} &\mathbf{n} \cdot \mathbf{u}^{\star}|_{x=0} = u_{in}(y), \\ &\mathbf{n} \cdot \mathbf{u}^{n+1}|_{x=0} = u_{in}(y), \\ &\mathbf{n} \cdot \nabla p^{n+1}|_{x=0} = 0. \end{aligned} \tag{4.50}$$

Inflow condition is very easy to implement in the staggered grid arrangement because the velocity control volume coincides with the physical domain. Therefore the inlet velocity, which is normal to the boundary, can be directly set to the initial node. Depending on the numerical method, the ghost cells may still be used. In that case, the ghost cells should be arranged to give the velocity value on the boundary.

4.3.4 Outflow Boundary Condition

In most of the cases it is desired to simulate a part of the physical domain where the physical phenomenon takes place. It is very efficient in terms of computation since it is not required to discretize the entire domain. The *outflow boundary condition* implies that the fluid continues to flow physically, although the computational grid is ended.

For two-dimensional setting, implementation of outflow condition in x direction starts with considering constant pressure and satisfying continuity at the last computational cell;

$$\begin{aligned} &\nabla \cdot \mathbf{u}^{n+1}|_{x=L} = 0, \\ &p^{n+1}|_{x=L} = const. \end{aligned} \tag{4.51}$$

If it is assumed that the streamlines are perpendicular to the boundary in the x direction, thus the tangential velocity becomes exactly zero, $v|_{x=L} = 0$.

Therefore Eq.4.51 takes the form;

$$\frac{\partial u}{\partial x}|_{x=L} = 0 \tag{4.52}$$

To impose outflow conditions, ghost cells are required to be filled like no-slip condition. Implementation of Eq.4.52 in discrete form is;

$$u_{nx+1/2,j} = u_{nx-1/2,j} \tag{4.53}$$

where nx denotes the total number of computational cells in x direction.

Even though implementation is straightforward, outflow condition is very restrictive. Physically it is important to state that the downstream flow should have no effect on the upstream. Implementing outflow condition can create unphysical solutions and convergence problems when it is applied to cells where the flow pattern changes significantly.

4.4 Hypre *Solver*

Discretization of the pressure Poisson equation with appropriate boundary conditions results in a linear system of equations, in the form $Ax = b$. The technique used to solve this equation has a big impact on the efficiency and convergence of the overall simulation. In this study *Hypre* is used as a main solver and this section is devoted to introducing the working principles of the package.

Hypre is a package of the high-performance solver for a large, sparse linear system of equations. It encompasses the multigrid solvers for both structured and unstructured grid. Since the adaptive mesh refinement algorithm in *AMReX* is based on a structured grid system, this section focuses on the fundamental steps to solve the linear systems in structured grids. Constructing the grid and solving the equation system, $Ax = b$ involve four basic steps:

1. Setting up the grid.

2. Specifying the stencil.

3. Constructing the matrix and vectors.

4. Solving the linear system of equations iteratively.

These four steps are accomplished by using built-in functions coded in the *Hypre* package. Initially, the computational grid where the governing equations are discretized should be introduced to the *Hypre* by setting the *index space*. The most basic tool of grid generation operations is the *Box* and it includes cell-centered computational cells by default. The idea of a *Box* is the same as the one used in *AMReX* package as will be discussed later. *Box*es divide the computational domain according to their *lower* and *upper* indices and they are distributed over the parallel processes. Figure 4.4 shows a sample domain the domain divided into two boxes and each box is sent to a different process. The integers inside brackets indicate the *lower* and *upper* indices of the boxes.

The grid creation in *Hypre* uses the following functions in the given order.

1. `HYPRE_StructGridCreate(MPI_COMM_WORLD,ndim, &grid)` : It creates the grid object `&grid`

2. `HYPRE_StructGridSetExtents(grid,ilower[0],iupper[0])` : The *lower* and *upper* coordinates of the first box is introduced and the box is reshaped accordingly.

3. `HYPRE_StructGridSetExtents(grid,ilower[1],iupper[1])` : The second box is respahed in the same way with the first one.

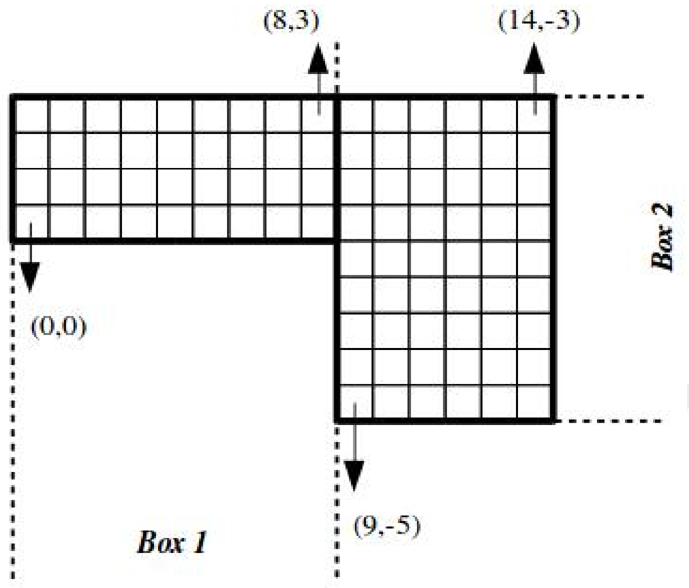


Figure 4.4: Example computational domain consists of two boxes whose indices can be both positive and negative integers

4. `HYPRE_StructGridAssemble(grid)` : The computational grid is assembled.

After the grid is created, the stencil system that identifies the cells used to discretize the equations is introduced to the grid object. For instance, the central differencing approximation uses 5 cells in its stencil system for a two-dimensional Poisson equation and 7 points for a three-dimensional one. The stencil system is represented by array of indices whose components show the offsets from the gridpoint at the center.

Entries of the stencil array usually start with the cell for which the equations are discretized. To represent the neighboring cells, offset values are used. Thus, the zeroth entry of the array is (0,0) and it indicates the central cell. The first neighboring cell has offset in negative x direction, therefore its stencil entry is (-1,0). The next entry belongs to the cell in positive x direction, hence its value is (1,0). The stencil array is completed in this manner until all neighboring cells are represented. Figure 4.5 shows the order of entries. The stencil array should be the same for all boxes in all processes.

Following *Hypre* functions are used to create the stencil system.

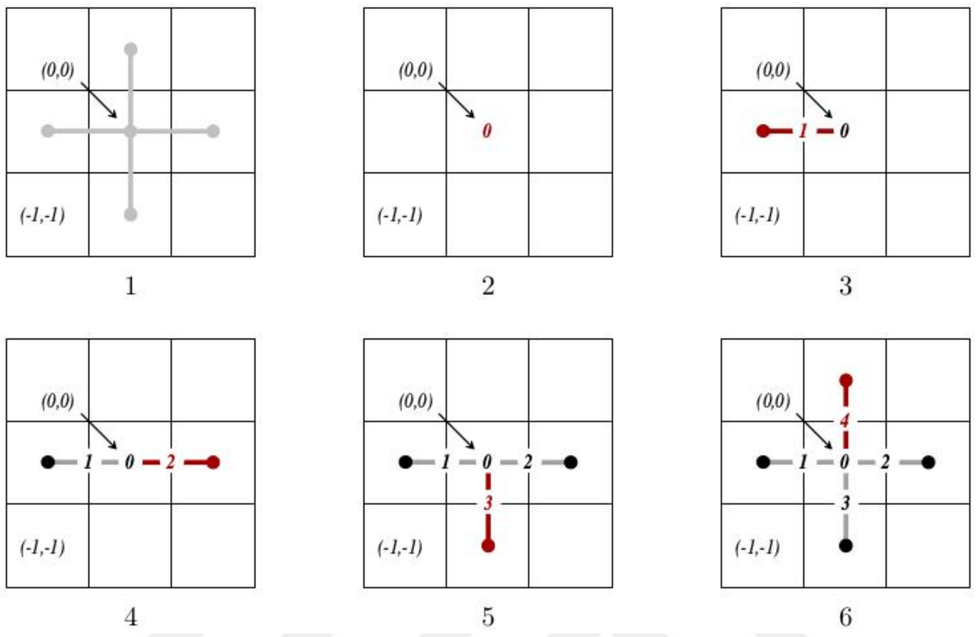


Figure 4.5: 5-pt stencil system for the two-dimensional problem

1. `HYPRE_StructStencilCreate(ndim,size,&stencil)` : This function takes the dimension of the problem and size of the stencil system (5-pt, 7-pt etc.) to create the stencil object.

2. `HYPRE_StructStencilSetElement(stencil,entry,offsets[entry])` : Each `entry` represents a gridpoint in Figure 4.5 and `offsets[entry]` sets the offset of that gridpoint from central grid.

This completes the second step in the overall procedure and now the matrix A in equation $Ax = b$ is ready to be filled according to the stencil system introduced above. The temporary vector with a length of $n_{stencil} \times N$ is used to fill the matrix object. $n_{stencil}$ is the number of stencils that each cell is associated with, and N is the total number of computational cells in the box. The values of the vector, i.e., the values of entries for matrix A, are determined by the numerical method and the boundary conditions. The data is stored in this temporary vector starting from the bottom left corner and by sweeping the cells to the right. There should be exactly $n_{stencil}$ entries for each cell in the vector. Following functions are used to update the matrix A;

1. `HYPRE_StructMatrixCreate(MPI_COMM_WORLD,grid,stencil,&A)` : The matrix object `A` is created

2. `HYPRE_StructMatrixInitialize(A)` : The matrix is initialized

3. `HYPRE_StructMatrixSetBoxValues(A,ilower[0],iupper[0],n_stencil, stencil_indices,values)` : `values` is the vector that will feed the matrix A and `stencil_indices` is the offset values associated with each `stencil`

4. `HYPRE_StructMatrixSetBoxValues(A,ilower[1],iupper[1],n_stencil, stencil_indices,values)` : Same procedure is repeated for the second box.

5. `HYPRE_StructMatrixAssemble(A)` : Assembles the matrix

After the matrix is created, the values can be manipulated to account for the boundary conditions. For instance, the value of the left stencil for cells adjacent to the physical boundary at $x = 0$ should be exactly zero.

Finally, the vector components, x and b, are filled. Vector b consists of *known values* of the linear system and vector x is composed of initial values of the *unknowns*.

As is the case with the matrix A, the vector objects are filled by using temporary vectors which store the data in the same orderly fashion. The *Hypre* functions used to generate vectors x and b are very similar to the functions used for matrix A. The sequence will be given only for vector b since it is the same for vector x.

1. `HYPRE_StructVectorCreate(MPI_COMM_WORLD,grid,stencil,&b)` : The vector object `b` is created

2. `HYPRE_StructVectorInitialize(b)` : The vector is initialized

3. `HYPRE_StructVectorSetBoxValues(b,ilower[0],iupper[0],,values)` : `values` is the vector that will feed b for the first box.

4. `HYPRE_StructVectorSetBoxValues(b,ilower[0],iupper[0],,values)` : The same procedure in previous step is repeated for the second box.

5. `HYPRE_StructVectorAssemble(A)` : Assembles the vector.

After matrix A and vectors b and x are assigned, the solver is initiated. There are several alternative solvers contained in *Hypre* and they should be picked according to requirements of the problem. *SMG* solver is a parallel multigrid solver and uses semi-coarsening principle to handle any anisotropy of the problem. It is a robust solver, however, per V-cycle it is less efficient than another solver *PFMG*. The main difference between these two solvers is the smoothing technique. *SMG* uses a plane-smoothing method and it increases the robustness. *PFMG*, on the other hand, uses pointwise smoothing which makes the solver more efficient but less robust [Falgout and Jones, 2000]. There are also built-in functions designed to set up the solver with appropriate *iteration* and *tolerance* limits. In this study, the *PFMG* solver is preferred and the sequence of built-in functions to activate this solver can be given as;

1. `HYPRE_StructPFMGCreate(MPI_COMM_WORLD,&solver)` : Creates the solver object.

2. `HYPRE_StructPFMGSetMaxIter(solver,maxiter)` : Sets the maximum number of iterations.

3. `HYPRE_StructPFMGSetTol(solver,tol)` : Sets the tolerance for relaxation method.

4. `HYPRE_StructPFMGSetRelaxType(solver,`*`method #`*`)` : Determines the relaxation method (SOR, RBGS etc.).

5. `HYPRE_StructPFMGSetup(solver,A,b,x)` : Setup the solver by bringing matrix and vector objects.

6. `HYPRE_StructPFMGSolve(solver,A,b,x)` : Solves $Ax = b$.

The implementation of *Hypre* to AMR applications is going to be discussed in details in following chapters.

Chapter 5

ADAPTIVE MESH REFINEMENT (AMR)

5.1 Introduction

Numerical simulations require significant computational resources as the physics of the problem gets complicated. In many contemporary computational studies, researchers try to formulate robust numerical tools to attack the physical phenomena. To capture the critical physical features in a reliable computational setup, the discretization should satisfy some quality criteria. In multiphase flows, the interface between two immiscible fluids is the most critical part of the problem and, therefore, the discretization around it must ensure that the physics of the problem is represented accurately. In further analysis, some surface agents can also be added to the interface. The concentration difference creates significant discontinuity. Hence, the accuracy around the interface becomes much more critical.

There is a computational trade-off between accuracy and computational resources in terms of both computational time and memory storage requirements. The first option for improving the accuracy of the problem is obviously to reduce the cell size. Even though it remedies the problem to some extent, additional problems arise simultaneously. Refining the entire domain increases both the computational time and the memory requirement significantly. However, as mentioned before, usually only some specific parts of the domain need higher resolution. On the other hand, the cell size has a negative effect on the time-step size. To maintain the numerical stability, the time-step size must be reduced to satisfy the stability requirements. Since the simulations must be performed to resolve the flow time scale, many time steps are required to obtain reasonable solutions.

Adaptive Mesh Refinement (AMR) emerges as a valuable technique that focuses on a specific region of interest without reducing the computational efficiency dramatically. In a nutshell, AMR is a well-developed tool to create a mesh in an adaptive manner so that it can be refined and coarsened both in space and time depending on the requirements of the physics of the problem.

The idea of using adaptive mesh can be implemented in various ways depending on the physics of the problem as well as available computational resources. The AMR strategies can be investigated in two main categories. The widespread strategy is so-called *h-refinement* where the resolution of the mesh is increased by dividing the cells recursively. In *h-refinement*, computational resources determine the ratio of the refinement between two consecutive cells and the number of refinement levels used throughout the simulation. Implementation of this strategy for unstructured grids offer significant improvement for CFD simulations in which complex geometries are involved. However, difficulties in maintaining the grid quality, load balancing for parallel architectures and managing the mesh connectivity create unignorable drawbacks. The second strategy is the *r-refinement* where the total number of cells and nodes remain constant, however, the positions of these cells rearranged according to resolution requirements. In addition to h and r refinements, where the focus is on the spatial resolution, the *p-refinement* can also be implemented to change the resolution by adjusting the order of numerical method for the same mesh. A simulation setup can involve both *h-refinement* and *r-refinement* simultaneously to form so-called *complete AMR*. Even though this coupling creates complexity during implementation, especially moving boundary problems need it to maintain sufficient grid quality [Vanella et al., 2010].

In this study, we selected *AMReX* package to manage grid generation and organizations in AMR applications. It has been developed upon a hierarchical block-structured method first proposed by Berger and Oliger [Berger and Oliger, 1984] as briefly discussed in the introduction of the thesis. This method uses successive refined levels to increase the spatial accuracy in specific region of interests. Refined levels are created by putting patches on coarser level. The resolution of the level is increased by a factor of refinement ratio which can be limited by some interlevel operations such as prolongation. Unless it is destroyed during the simulation, each level is in communication with the consecutive coarser and finer levels if they exist. We denote a single AMR level with the lower case l in the present thesis.

The communication between levels resembles the operations performed in multigrid simulations. The prolongation and restriction operations in multigrid methods are adopted in AMR algorithms. However, multigrid methods are not included among AMR techniques, because the multigrid methods do not create refined levels on top of the existing base level. Prolongation and restriction operations are performed in multigrid methods between base level and coarser levels which are used to eliminate persistent low-frequency error components. Moreover, in the majority of AMR algorithms multigrid solvers are used as a base solver to deal with PDEs. The levels used in multigrid solvers are not the same levels created in AMR grid generation routines. The multigrid levels are temporarily available during the solution process, conversely, AMR levels retain until the resolution is required.

In the following sections, we directly focus on introducing block-structured AMR routines to set a theoretical basis.

5.2 Block-structured Adaptive Mesh Refinement

5.2.1 Overall Algorithm

AMReX package has been built upon hierarchical block-structured AMR method, therefore it is necessary to investigate the development of the algorithm.

The block-structured algorithm has been developed at two distinctive steps. In the first step, Berger and Oliger [Berger and Oliger, 1984] applied the primitive form of the hierarchical algorithm on two-dimensional hyperbolic partial differential equations by using the finite difference technique. The hyperbolic PDEs are advantageous in the sense that they can easily be represented by a wide range of numerical methods. However, they may encompass certain critical physical phenomena such as shock waves and very steep gradients [Berger and Oliger, 1984]. Those features require special attention and they proved that AMR offers significant improvement. In this first algorithm, they created rectangular-shaped patches with arbitrary orientations due to computational efficiency concerns. In the coarse level, the cells are tagged according to the Richardson-type error estimation procedure and the rectangular patches are created around those cells. The arbitrary orientation of the patches in the fine levels allows the algorithm to align the coordinate system with the feature we are interested in. Thus, the patches can be created with a minimum amount of cells that surrounds the discontinuity. Although arbitrary orientation decreased the cost of clustering operations, mapping the finite difference equations into the new coordinate system brought additional computational burden. In addition, maintaining the overall conservation became problematic when the patches were not aligned with the coordinate system of the base level.

The major claim of the algorithm is that the grids are independent so that different numerical methods can be applied in each of them depending on stability concerns. For instance, higher-order schemes can be preferred in the grids away from the discontinuity, while, lower-order schemes are beneficial in the vicinity of the discontinuity. This independency also makes the parallelization easier.

In the second important step of the development of the block-structured method, Berger and Colella [Berger and Colella, 1989] improved the algorithm in the first step. The modifications implemented in this step has formed the basis for the algorithms in the upcoming studies. Different than Berger and Oliger [Berger and Oliger, 1984], here they used the nested refinement levels whose boundaries coincide with the grid lines of the coarser levels to maintain the global conservation.

Starting from these very first algorithms, the finer levels are created by employing a refinement ratio that sets the proportionality relation between cell sizes of two consecutive levels. The same spatial ratio can also be applied to time-step size as well. Thus, the small time-step sizes due to the CFL condition in finer levels are not dictated on coarser levels. Even though this adaptation is implemented in the following algorithms and it seems very advantageous in terms of computational efficiency, the drawback emerges during the selection of base solver. Even though multilevel multigrid base solvers are very advantageous during flux correction operations, they cannot be employed while time-step adaptation is used. Berger and Colella [Berger and Colella, 1989], Almgren et al. [Almgren et al., 1998] and Sussman et al. [Sussman et al., 1999] have followed the time step adaptation especially in incompressible Navier-Stokes algorithms due to efficiency concerns, but they had to implement complicated level synchronization and refluxing procedures. Conversely, Minion [Minion, 1996] and Vanella et al. [Vanella et al., 2010] have selected to use a fixed time-step to take advantage of multilevel solvers. In the present study, we will try to combine time-step adaptation with relatively easier synchronization operations.

As can be seen in the overall solution algorithm in Algorithm 1, each level is created in a properly nested manner and time-step adaptation is applied by using the spatial refinement ratio.

In the next sections the grid generation, time advancement, ghost cell organization, and synchronization procedures are discussed in detail in the framework of the *AMReX* package.

5.2.2 Grid Generation

In block-structured AMR method, the grid generation procedure follows some certain rules and restrictions in order to increase the computational efficiency and to make implementation easier. Tagging the cells which need refinement and clustering to form rectangular patches determines the efficiency of in-level integration and communication between refinement levels. We can list the attributes of the grid generation procedure as follows;

- Whenever a level is created, it takes its position in the hierarchical tree. The grids in that level can have multiple interactions with grids in other levels depending on its position in the tree. In the usual notation the grids in the coarse level, *l*, are called *parent*, finer level grids, *l+1*, are called *children* and

Algorithm 1: Pseudo-code for the AMR algorithm

```
1 Initialization : Initialize and define the base level
2 if Max Level > 0 then
      do
      Create and initialize finer grids
      while Level < Max Level
3 end
4 Evolve
    do
    Compute time step size based on CFL number
    Advance the solution for base level
      if Level < Finest Level then
        do
        Advance the solution for finer levels
        while r < Ref. Ratio
        Store the fluxes for later correction
      end
      Correct the fluxes and update coarse level
    while time < solution time
```

finally the grids located at the highest level of refinement are called *leaf.* The grids in the same level of refinement are named as *siblings.*

- The cells in a level l can be tagged by either using a truncation error estimator function or using a specific user-defined function that tracks the evolution of the physical parameters. For each level, different criteria can be set for the tracked variables.

- The same refinement ratio is applied in all grids in a level, however, different refinement ratios can be selected for different levels. In the case of time synchronization, the same ratio is used for time-step size.

- The grids are properly nested so that each fine grid, *l+1*, starts and ends within the borders of the coarser level, *l*. Unless the finer level abuts the physical domain, there is at least one coarser level cell in each direction to surround the level *l+1*. This additional region called *buffer region* and it can be extended to make sure that the high error regions are contained within a refinement level. As the buffer region gets larger the re-gridding requirement decreases, but the computational cost increases at the same time.

- There should always be a level *l* between level *l+1* and *l-1*. This statement ensures that each refined level can be contained in only a single coarser level. For instance, a grid in level *l+1* cannot be included in both level *l* and *l-1* at the same time.

- A grid in a level *l+1* can be contained in two different grids in a coarser level *l*.

- *AMReX* takes two parameters called *max_ grid_ size* and *blocking_ factor* to regulate the generated grids. *max_ grid_ size* is used to chop the grids if any of them in a level has the number of cells higher than this specific parameter. For instance, if there are 32 cells in each direction in a level and the *max_ grid_ size* is 16, then the grid generation algorithm chops up the single grid into two. Additionally, the number of cells in each direction should be proportional to the *blocking_ factor*. For example, we cannot use 25 cells in a direction if the *blocking_ factor* is 2. *AMReX* has the flexibility to set different *blocking_ factor*s in different directions.

- The re-gridding frequency is set by the user. At the end of the re-gridding operations, a level can be created, destroyed or remains still. After level *l* is re-gridded, levels from *l+1* to l_{max} are re-gridded to sustain the proper nesting. The base level, *l=0* remains the same throughout the simulation.

The grids are created and re-gridded according to these basic rules and restrictions. Figure 5.1 shows every possible situation that may arise during grid generation.

5.2.3 Time Advancement

Time-step adaptation is one of the most important features of the hierarchical block-structured AMR method. It is also implemented in the *AMReX* package. Due to the CFL constraint, the time step size is inversely proportional to the cell size. As the simulation creates finer levels, the cell size decreases according to the refinement ratio. Thus, time-step size decreases dramatically in finer levels and, inevitably, simulation becomes inefficient. The inefficiency results from the necessity to use the same time-step size determined at the finest level in all other coarser levels.

A time-step adaptation algorithm, or subcycling-in-time, integrates each level independently with its own time step size. Figure.5.2 represents the advancement of the base level for a single time step. The numbers on arrows indicate the operation. The subcycled time-stepping procedure for a single time step can be described as follows:

1. Advance Level-0 over Δt

   Then fill the ghost cells of Level-1

2. Advance Level-1 over $\Delta t/2$

   Then fill the ghost cells of Level-2

3. Advance Level-2 over $\Delta t/4$

   Then update the ghost cells of Level-2

4. Advance Level-2 over $\Delta t/4$

   Level-2 is advanced over $\Delta t/2$, now synchronize it with Level-1

   Then update the ghost cells of Level-1

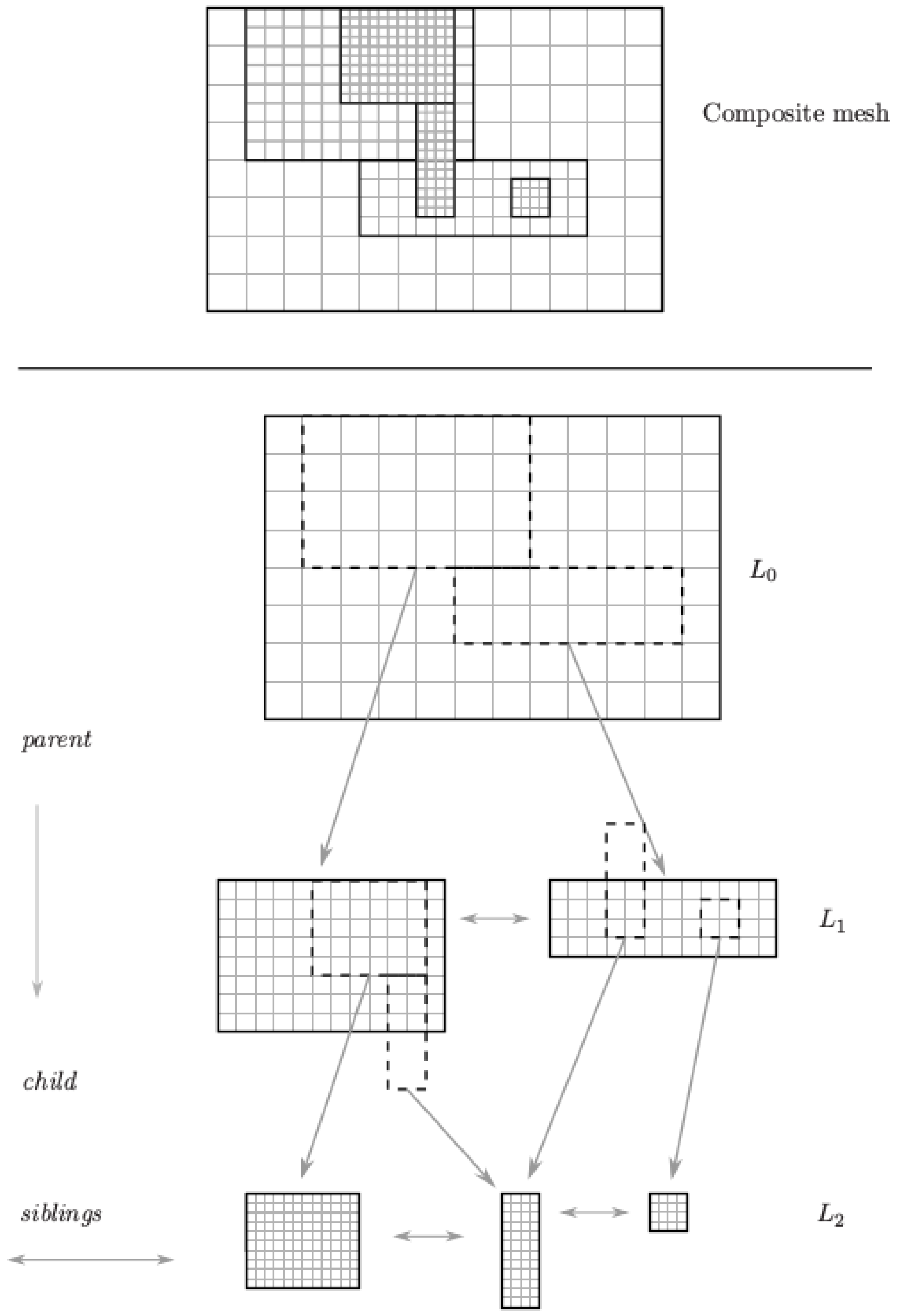


Figure 5.1: Representation of the properly nested grid hierarchy [Zuzio, 2011]

5. Advance Level-1 over $\Delta t/2$

   Then fill the ghost cells of Level-2

6. Advance Level-2 over $\Delta t/4$

   Then update the ghost cells of Level-2

7. Advance Level-2 over $\Delta t/4$

   Level-2 is advanced over Δt, now synchronize it with Level-1

   Level-1 is advanced over Δt, now synchronize it with Level-0

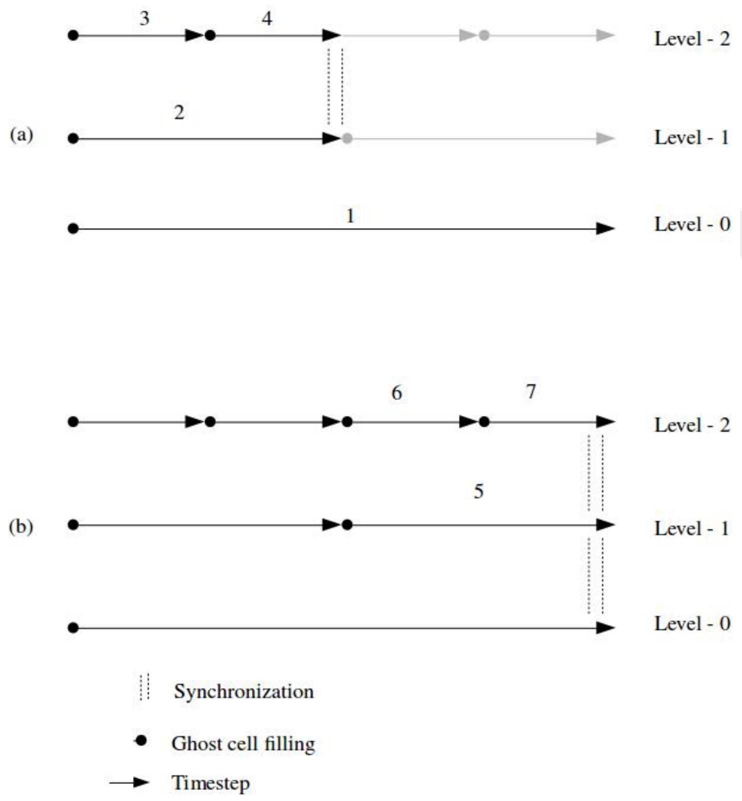


Figure 5.2: Schematic representation of the time-step adaptation: (a) Advancement in the fine level (b) Advancement in the base level

Since the flow equations are solved in each level independently, after the advancement of each level the mismatch between two consecutive levels may arise. The synchronization algorithm is devised to correct this mismatch to maintain global conservation. Even though the synchronization is easier in multilevel solvers, it is difficult to implement the time-step adaptation in these solvers easily.

5.2.4 Ghost Cell Organization

The ghost cells are additional computational cells used to impose the physical boundary conditions and/or maintain communication between the boundary cells in the same level or from next coarser level. Therefore managing the ghost cells appropriately is a vital task for the simulation. As depicted in Figure.5.3, there are three fundamental types of ghost cells located in three different boundaries:

- *Boundary between siblings*: The ghost cells in this boundary maintain communication between two siblings. In *AMReX* package, these boundaries are generated especially when the grids are chopped up. Therefore they are immediately filled by the information from the neighboring sibling.

- *Physical boundary*: The physical boundary conditions are applied using the ghost cells in these boundaries

- *Boundary between levels*: The ghost cells in these boundaries are used to exchange the information between coarse level and the next finer level. Depending on the type of the variable, different interpolation schemes are used to transfer the data. For instance, scalar variables are stored in the cell centers in the staggered grid arrangement, hence, the interpolation scheme which is compatible with the cell-centered data type should be selected.

The data in the ghost cells have a direct impact on global accuracy, on the convergence of the problem and also on the simulation results. Each type of the boundary condition has a different type of implementation and they should be treated carefully through the ghost cells.

5.2.5 Flux Correction (Refluxing)

Maintaining global conservation is of crucial importance in many physical problems such as magnetohydrodynamics (MHD) and multiphase flows. In the subcycled time-stepping algorithm the coarse level is advanced first as if there is no fine level, then

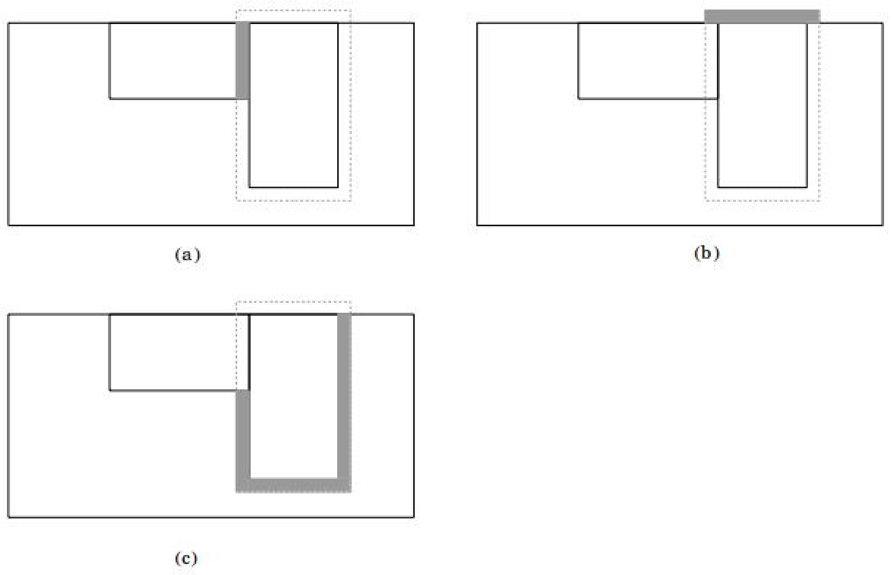


Figure 5.3: Schematic representation of the three different boundary: (a) Boundary between the siblings (b) Physical boundary (c) Boundary between the levels

the fine level is advanced by using the interpolated coarse level data in the ghost cells. However, when the simulation is completed and all levels reach the same simulation time, the flux comes into or goes out of the fine cell in the coarse/fine interface doesn't necessarily be equal to the flux goes out of or comes into the coarse cell. A typical flux matching condition is depicted in Figure.5.4.

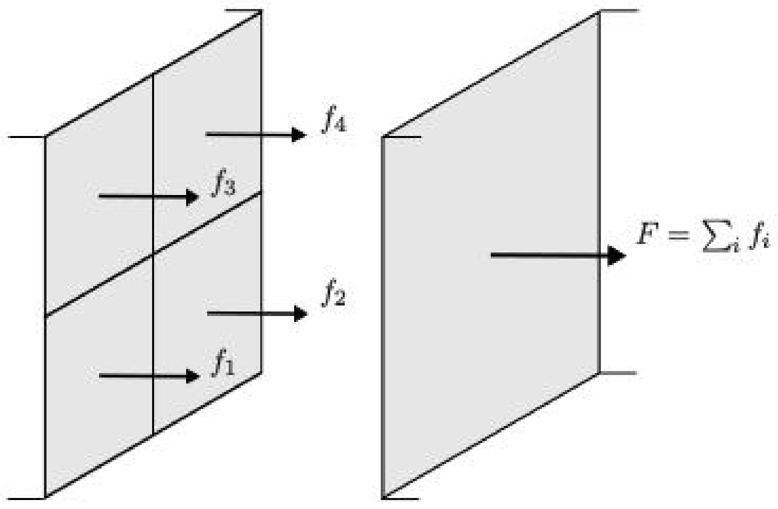


Figure 5.4: Fluxes at the coarse/fine interface. They should match in order to preserve global conservation of physical quantities.

The flux correction algorithm first calculates the difference between coarse and fine level fluxes at the interface and then synchronizes the levels by updating the coarse level. We can consider the hyperbolic advection equation, Eq.5.1, as an example problem.

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot (\phi \mathbf{U}). \tag{5.1}$$

After the coarse level is advanced the flux is computed and stored by multiplying with a minus sign as

$$\delta F = -F^{l} \tag{5.2}$$

where $F = \phi\mathbf{U}$ is the flux. Then the fine level is integrated and the computed flux at the interface is added to the stored coarse level flux as

$$\delta F = \delta F + \frac{1}{r}\sum_{k=1}^{r}\sum_{faces} F^{l+1,k} \tag{5.3}$$

Here we compute fine level flux by two summation operations. The left summation is for the time refinement and the right one is for taking the contribution of all fine faces corresponding to the coarse face. The δF on the right hand side is equal to the coarse level flux and the one on the left hand side is the calculated flux mismatch when the fine level contribution is added. After the mismatch is computed, the coarse level, l, is updated depending on the type of the governing PDE. For hyperbolic PDEs, the information is seen only by neighbors of the active computational cell. Therefore it is sufficient to update the coarse cells in the vicinity of the interface and we can use the flux difference to update the coarse level data as

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \delta \mathbf{F}, \tag{5.4}$$

where the entries of $\delta\mathbf{F}$ are all zero everywhere except for the region around the interface. Therefore only the scalar variable ϕ around the interface is updated.

Conversely, for elliptic problems, the entire coarse domain should be updated to spread the effect of mismatch, because the information at a point in the domain travels at an infinite speed. For instance, if the Poisson equation governs the problem, additional Poisson equation with $-\nabla\cdot\delta\mathbf{F}$ on the right-hand side is required to update the coarse level. This is discussed in detail in upcoming chapters.

5.3 The AMReX *Package*

The *AMReX* [Zhang et al., 2019] is an open-source software developed for constructing block-structured AMR applications. It is a successor of BoxLib [Bell et al., 2012] software and constitutes a basis for a large number of applications (for hyperbolic conservation laws [Bell et al., 1994], for combustion applications [Pember et al., 1998], for astrophysical applications [Zingale et al., 2018] etc.). *AMReX* is a very flexible framework in terms of solution strategies. Depending on the problem and the numerical method, a user can select a level-by-level strategy with a time-step adaptation, multilevel approach with a fixed time step and any combination of these two. Interoperability with other packages such as *Hypre* extends the alternatives for base solvers. Kernels for integration, error estimation, interpolation and other problem-dependent operations can be written either in *C++* or in *Fortran*. The binding routine between these two languages increases the flexibility of the solver. Although *AMReX* is very advantages for multiphase flow applications, the structure of the framework is suitable for collocated or semi-staggered grid arrangements. Hence, important features should be modified to use *AMReX* with a staggered grid arrangement. The main features of the framework can be listed as follows:

- It offers both *C++* and *Fortran* interfaces.

- 1-, 2- and 3-dimensional problems can be simulated.

- There are available solution strategies for parabolic, elliptic and hyperbolic PDEs.
- Time refinement approach is implemented.
- It supports particle tracing algorithms

In this chapter, we go through the basic features of the *AMReX* framework and the key operations that are vital for the AMR applications.

5.3.1 Building Blocks of AMReX

Box and BoxArray

Including the base level, each level of refinement is divided into grids that have a specific number of computational cells. Within the context of *AMReX*, each grid is represented by a class named *Box*. The collection of the *Box*es in a refinement level is represented by *BoxArray* class. Attributes of these two classes reflect the basic features of the grids in a level. Figure 5.5 shows the boxes located in two refinement levels on top of a base level.

To create a *Box* the index type and computational indices for lower and upper corners of the box should be defined. A *Box* can be extended in each direction to cover the ghost cells.

The index type reflects where the variables are stored in a computational cell and a *Box* can be created by using only one index type. *AMReX* supports three different index types. A *Box* can be cell-centered, face-centered or node-based. If it is not stated, the default setting is the cell-centered configuration. Boxes can survive in positive and negative indexing spaces. Each *Box* with a specific index type can be converted to other index types easily. The grids in the refined levels are created depending on the index type because different interpolation operations should be picked for different index types. For instance, cell-centered bilinear interpolation does not give accurate results for the face-centered setting. Since the interpolation scheme

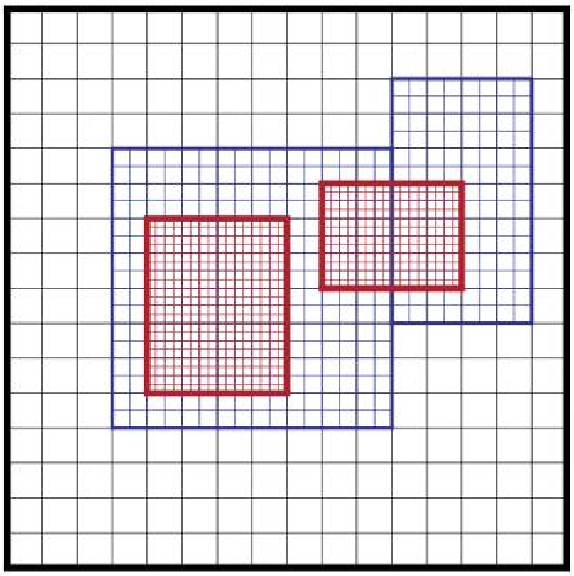


Figure 5.5: Representation of the boxes in 2 different refinement levels. The white cells belong to the base level. The Red and the blue boxes are in the first and the second refinement levels respectively. [Zhang et al., 2019]

is very critical in ghost cell filling, the whole simulation is affected when a wrong type is selected for a *Box*. The index type of a *BoxArray* should be the same as the *Box*es contained in it.

FArrayBox, FabArray and MultiFab

An organized data structure is an important advantage of block-structured AMR algorithms. *AMReX* uses *FArrayBox*, called as FAB, to store the data. It is a derived class from a base class template called *BaseFab*. *FArrayBox* is a multidimensional data storage unit and it is defined on a single *Box* in a level. Therefore it has the same index type as the *Box*. Several components of variables can be described on a single *FArrayBox*. For instance, in the staggered grid arrangement every scalar data such as density, pressure, and viscosity are stored in the cell center. Therefore all scalar variables can be attached to the same *FArrayBox* since they share the same index type. This reduces the complexity of the algorithm significantly.

In *AMReX*, there is a binding routine that allows sharing the data in an *FArrayBox* with Fortran and other C++ kernels. The user-defined Fortran functions can only take

FArrayBox as an input to manipulate the data. The member functions of *FArrayBox* class give freedom to copy, manipulate or combine the data in them.

FabArray is a class template which combines every FAB in a single level. While FABs are defined on the *Box* objects, *FabArrays* are defined on the *BoxArrays*. Therefore it is required to provide a valid *BoxArray* to create a *FabArray*.

FabArrays are parallel data structures and their load is distributed among the processors. The *DistributionMapping* class deals with management of the *FabArray* parallelization. Thus, the *DistributionMapping* object should also be defined before creating a *FabArray*.

Referring to Figure 5.5, the red boxes are located in the second refinement level, while the blue boxes are in the first level. For each box, different FAB is created, but for each level, there is a single *FabArray* and the corresponding *DistributionMapping*.

MultiFab is a special class derived from *FabArray* class to manage the data in a *BoxArray* in a level. Throughout this study, we employ *MultiFabs* as data storage units. Even though *MultiFabs* are collections of FABs in a level, the concept of ghost cell is only applicable to *MultiFabs*. Hence, a *Box* used to define a FAB may not be the same as the *Box* contained in the *BoxArray* that defines the corresponding *MultiFab*. That discrepancy occurs during the initialization of *MultiFab*. If ghost cells are introduced, *BoxArray* and each *Box* contained in it is modified to cover these additional cells.

Except for the *Box*es with nodal index type, the *Boxes* in a *BoxArray* do not overlap. However, when ghost cells are introduced to the *MultiFab*, overlapping occurs and the `FillBoundary` function fills the ghost cells with valid data in the context of parallel communication.

As long as the *MultiFabs* are defined on the same *BoxArray* with the same *DistributionMapping*, they can do the arithmetic operations such as adding, subtracting and copying.

MFIter

MFIter is a widely used *MultiFab* functionality which allows the user to operate on it. It is a specific iterator like the basic loops in common languages such as C++ and Fortran. *MFIter* divides the *MultiFab* into FABs and loops over them. This iteration can be performed only over the *Box*es owned by the corresponding processor. For instance if a processor owns two FABs from total of five FABs in a *MultiFab*, then *MFIter* iteration is realized over those 2 FABs only. In a single loop, more than one *MultiFab* with different *BoxArray*s can be iterated, as long as they have the same *DistributionMapping* objects.

Ghost Cells

As we discussed before, there are three different type of boundaries defined in the context of *AMReX*. Depending on its location each boundary is used to provide the information from the physical boundary, previous level of refinement or from the sibling grid. These data are stored in the ghost cells of the grids.

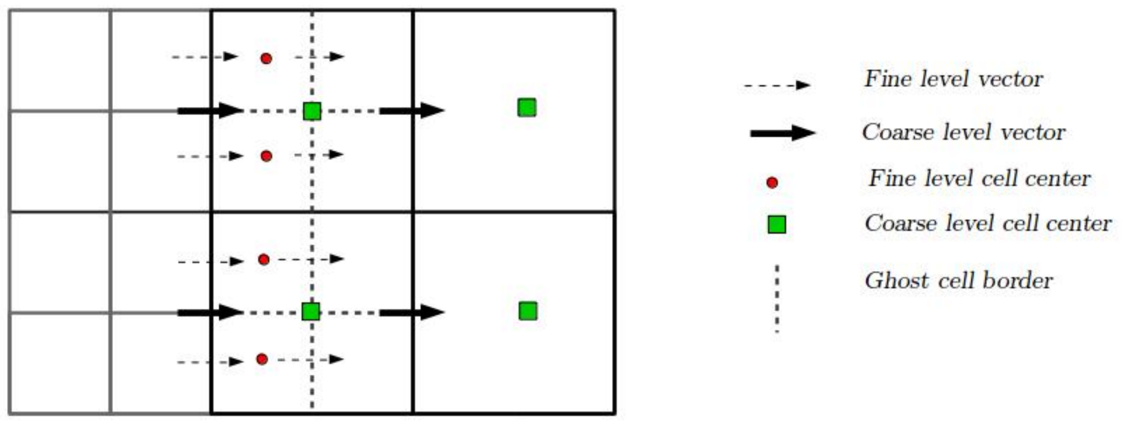


Figure 5.6: Ghost cell region at the coarse/fine interface.

Figure 5.6 shows the ghost region at the coarse/fine interface. Here *red dots* show the center of the ghost cells for the fine region and they provide the information from the underlying coarse level whose cell centers are shown by *green dots*. When the fine level is created, the ghost cells are added around the fine level grids to establish the communication with the underlying coarse grid.

In general, the ghost cells can be located in three different regions as depicted in Figure 5.3, however, the configuration in Figure 5.6 is applicable to all three scenarios. The information is coming from different sources for ghost cells in different locations. Therefore the *AMReX* routines used to fill these cells may differ. The following list briefly describes the location of the ghost cells and the source of information.

1. Ghost cells between the siblings - Figure 5.3 -(a): The information is provided by the neighboring grids. This is the easiest ghost cell filling operation. Since it is interior boundary, `FillBoundary()` can handle this operation.

2. Ghost cells at the physical boundaries - Figure 5.3 - (b): The source of information is the physical boundary conditions set by the user. If a fine level abuts the physical domain, the ghost cells are filled directly by the physical boundary conditions. If the boundary condition in the corresponding edge is periodic, then the `FillBoundary()` routine in the *AMReX* can fill those cells.

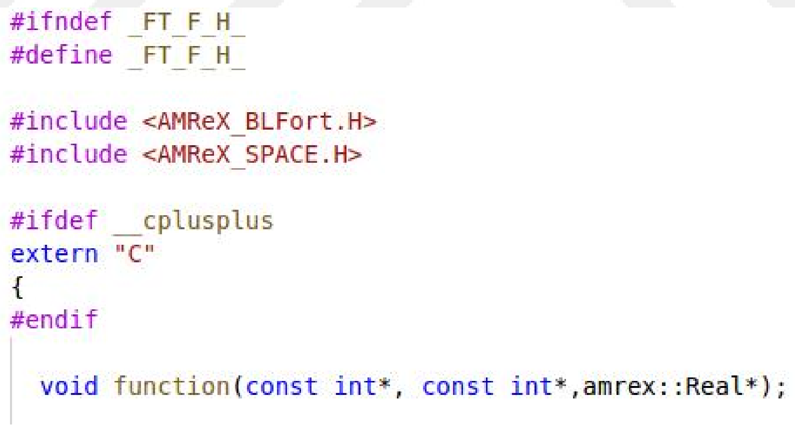
3. Ghost cells at the coarse/fine interface - Figure 5.3 - (c): The information for these ghost cells comes from the underlying coarse level grids. The interpolation schemes are used to perform this operation. The `FillPatch()` operations are used within the *AMReX* package.

Here we mentioned `FillBoundary()` and `FillPatch()` routines. As their names imply, these are the principle algorithms to fill the ghost cells and we will explain them in the upcoming sections.

The ghost cell operations for the vectorial quantities depicted in Figure 5.6 require more attention due to the global conservation concerns. In many CFD applications the staggered grid arrangement is preferred because it results in higher accuracy than the collocated and other grid arrangements. However, it brings complexity especially to the interpolation operations because the data itself lives at the interface. The special treatment for the staggered grid is going to be discussed in the further sections.

Fortran Kernels

Many critical operations in the AMR algorithms, such as error estimation, time-step calculation and integration of a level, can be accomplished by the Fortran kernels which are binded to the main C++ routine through special protocols. Because the Fortran language gives support to work on multi-dimensional arrays, these kernels are more convenient than the C/C++ functions. In Fortran 2003 the function names in the Fortran kernels can be accessed within C++ by `bind(c)` function. The important step here is to create a C++ header file with the extension `*_F.H`. Figure 5.7 shows how a Fortran kernel is introduced to the C++ in the header file.



```
#ifndef _FT_F_H_
#define _FT_F_H_

#include <AMReX_BLFort.H>
#include <AMReX_SPACE.H>

#ifdef __cplusplus
extern "C"
{
#endif

  void function(const int*, const int*,amrex::Real*);
```

Figure 5.7: The C++ header file which introduces the Fortran kernels.

To be able to work on a data stored within the context of *AMReX*, the *MultiFab* should be iterated under *MFIter* to reach the FABs in it. The Fortran kernels can take a FAB as an input regardless of how many components included in that FAB.

To send the FAB from the C++ to the Fortran, `BL_TO_FORTRAN_ANYD` macro function is used. It takes the FAB as an input and it returns 3 pointers. The first one is the data pointer in the type of `amrex::Real*` and the other two are integer type pointers which indicate the lower and the upper bounds of the *Box* in which the FAB is stored.

5.3.2 Error Estimation

Error estimation routine is the starting point of level creation and re-gridding. The cells which need to be refined are tagged depending on the result of the specific error function.

Error estimation can be performed mainly in two ways. The first one is keeping track of the truncation error. Berger and Oliger [Berger and Oliger, 1984] studied the hyperbolic type of PDEs where the propagation speed of information was very important to decide on the cell size. They picked the Richardson extrapolation to calculate the truncation error and they claimed that it gave satisfactory results even for nonsmooth solutions. The second method is the user-defined function in which the evolution of a specific parameter is tracked. Vanella et al. [Vanella et al., 2010] studied a fluid-solid interaction problem, hence the interface between the two phases should always be included in the refined region. Therefore they picked interfacial parameters as error estimators to refine and de-refine the domain. Even though tracking the values of some physical parameters allows the algorithm to create the refinement levels accurately, there can be overhead during the clustering operation by including unnecessary cells in the patch. Thus, the user-defined function should be arranged carefully to avoid unnecessary refinements.

We studied the implementation of adaptive mesh refinement algorithms to multiphase flow simulations. As is the case with Sussman et al. [Sussman et al., 1999] and Vanella et al. [Vanella et al., 2010], it is significant to capture the interface accurately. Hence we also pick the interfacial properties to decide the position of the refinement level.

In the *AMReX* package tagging the cells through user-defined functions is performed in the `ErrorEst()` member function of the `AmrCore` class. The object of the `TagBox` class is used to store the tagged cells for refinement and finally the `Cluster` class creates the rectangular fine grid.

5.3.3 FillPatch Operations

As we stated before, *MultiFabs* have the concept of the ghost cell. In the *AMReX* package the `FillPatch` operations have the duty to fill both the ghost and the computational cells. In a nutshell, the `FillPatch` functions accomplish the interpolation and the ghost cell operations.

Since there are three different types of boundaries, `FillPatch` operations are capable of filling all three different type of ghost cells and the computational cells during the level creation. For the base level, the computational cells are updated by integration, however, the ghost cells on the boundaries between siblings (interior boundary) and on the physical boundary should be filled using boundary conditions. When a refinement level is created, the computational cells are filled either from the readily available valid data calculated at the previous time-step at the same refinement level or from the interpolated coarser level data. The `FillPatch` operations can interpolate the data both in space and time in the case of the time-step adaptation. The data for the ghost cells in the refined levels can come from the physical boundary conditions, the neighboring sibling cells or directly from the coarser level.

There are two specialized routines within the context of `FillPatch`:

1. `FillPatchSingleLevel()` : This function operates on the single level of refinement. It has the capability of interpolating in time. The physical boundary conditions are also managed in this function through the `FillBoundary` routines.

2. `FillPatchTwoLevels()` : If there is any underlying coarser level, this function fills the ghost and the computational cells. The Interpolation operations in space and time are performed in this function. As is the case with the single level operations, the `FillPatchSingleLevel()` function interpolates in time and interpolation in space can be performed by any scheme suitable for the index type of the *MultiFab*.

Since the `FillPatch` operations are very critical for the rest of the simulation, the sequence of the member functions used to fill the ghost and computational cells are discussed in the following algorithms. Algorithm 2, 3 and 4 show the three consecutive steps of a single time step advancement. It is assumed that there is a single level of refinement and the refinement ratio is 2. The algorithm starts with the initialization.

Algorithm 2: FillPatch operations for the base level

```
1 initData()
      Update new_data
2 advance() - Level = 0
      Switch new_data$^0$ and old_data$^0$
      Increment new_time$^0$ by dt
      old_time$^0$ = 0,time = 0
      FillPatch()
         FillFromLevel0()
            getData()
               Call one time and store only old_data$^0$ in state$^0$ object
         FillPatchSingleLevel(state$^0$)
3 Finalize time step for Level = 0
```

The Algorithm 2 describes the ghost/computational cell filling operations for the base level. The **new_data** and **old_data** are the objects responsible to store the new time-step and the previous time-step data related with a variable. The subscript *0* refers to the base level and **time** is the simulation time. The *state*0 object stores the data which will be used to fill the ghost/valid cells. Since the **old_time**^{0} and the **time** is equal, only available data is the **old_data**^{0}. `FillPatchSingleLevel()` directly copies the **old_data**^{0} without any time interpolation.

The Algorithm 3 shows the first cycle of the fine time step. Here the FillPatch routines change because there is an underlying coarse level. At the beginning of this time step, the **new_time**^{1} incremented from 0 to **dt**/2 due to the time-refinement. Different than the base level advancement, the `FillFromTwoLevels()` function is

Algorithm 3: FillPatch operation for the refined level (first cycle)

```
1 advance() - Level = 1, Cycle = 1
      Switch new_data$^1$ and old_data$^1$
      Increment new_time$^1$ by dt/2
      old_time$^1$ = 0,time = 0 FillPatch()
         FillFromTwoLevels()
            getData()
               Call for Level-0
               Store only old_data$^0$ in state$^0$
            getData()
               Call for Level-1
               Store only old_data$^1$ in state$^1$
         FillPatchTwoLevels(state$^0$,state$^1$)
2 Finalize time step for Level = 1. Cycle = 1
```

called to be able to use the coarse level data as well. In this step we have the coarse level data. The first `getData()` function stores the **old_data**^{0} of the coarse level because the **time** is still equal to the **old_time**^{1}. In the second call of the `getData()`, the fine level data is going to be stored in the *state*^{1}. In this first cycle of the fine level time-step, the **time** is equal to the **old_time**^{1}. Therefore only the **old_data**^{1} is stored in the *state*^{1}.

After the `getData()` operations, *state*^{0} and *state*^{1} are sent to the `FillPatchTwoLevels()` function. Together with the valid data, the ghost cells of the rectangular fine patch are filled in this function. According to the index type of the target *MultiFab*, a specific interpolator is used to fill the ghost cells by the data coming from the underlying coarse level. The *state*^{0} that was filled by the `getData()` function is used here to provide the data from the coarse level. The interpolation operations, which are mainly run by member functions of the `AMReX_Interpolator.cpp/H` class, are hidden from the users. The available interpolators in the *AMReX* package are mainly for cell-centered and node-based *MultiFabs*. The Interpolator for the face-centered *MultiFabs* is going to be discussed in the following chapter.

At the end of the `FillPatchTwoLevels()` function, the `FillPatchSingleLevel()` is called to fill the valid cell region. During the interpolation step above, the data from the coarse level are interpolated not only to the ghost region but also to the computational region of the fine level. However, it is not desired to lose the fine level data. Thus, the `FillPatchSingleLevel()` copies the simulation data to the computational cells of the fine level.

Algorithm 4: FillPatch operation for the refined level (second cycle)

```
1 advance() - Level = 1, Cycle = 2
      Switch new_data$^1$ and old_data$^1$
      Increment new_time$^1$ by dt/2
      old_time$^1$ = 0,time = dt/2 FillPatch()
          FillFromTwoLevels()
              getData()
                  Call for Level-0
                  Store old_data$^0$ and new_data$^0$
              getData()
                  Call for Level-1
                  Store only old_data$^1$
          FillPatchTwoLevels(state$^0$,state$^1$)
2 Finalize time step for Level = 1. Cycle = 2
```

The Algorithm 4 explains the final part of the single coarse time step advancement. Since the refinement ratio is 2, there should be 2 fine level iterations per single coarse level iteration. The second cycle starts exactly the same as the first cycle. The only difference is the simulation time is not 0 anymore. After the half coarse time step advancement of fine level, the simulation time has increased by **dt**/2. We already advanced the coarse level by a coarse level time step. Therefore the **new_data**^{0} was occupied for **time** = **dt**. But the actual simulation time is **dt**/2, hence we need the coarse level data for that simulation time. Unfortunately, we only have the coarse level data for **time** = **dt**, but we need the data at **dt**/2. The time interpolation comes into play at this stage. When the `getData()` function is called for the coarse level, we store both the **old_data**^{0} and the **new_data**^{0} in *state*^{0}. On the other hand, we store only the **old_data**^{1} for the fine level because **time** is equal to **old_time**^{1}. It should be noted that the **old_time**^{0} and the **old_time**^{1} are two different objects and their values do not necessarily be the same.

As is the case with the first cycle, the `FillPatchSingleLevel()` is called just before the interpolation and the coarse level data are interpolated in time to have the data at **time=dt/2**. These interpolated data are used to fill the ghost cells of the refined level. When interpolation is completed, the `FillPatchSingleLevel()` is called once more to fill the computational region of the fine level with the simulation data of the first cycle.

5.3.4 Synchronization Operations

As we briefly described in the Flux Correction (Refluxing) section, the synchronization of the coarse and the fine levels is crucial for preserving global conservation. Refluxing and averaging are the two basic operations to maintain synchronization between the levels. The first one remedies the problem that occurs as shown in Figure 5.4, but the implementation details can change according to the index type of the *MultiFabs*. The second one is to update the coarse level region overlaid by the fine region. Without averaging, the coarse level cannot have the accurate fine level solution, thus the adaptive mesh refinement becomes completely redundant.

The fluxes coming from the coarse and the fine region at the interface and the difference between them are accumulated by the `FluxRegister.cpp/H` member functions, `CrseInit()` and `FineAdd()`. The first function performs the operation in Eq.5.2, while the second function performs the operation in Eq.5.3.

The flux calculated at an intermediate level, *l*, has two meanings. It is a "finer level flux" for the level, *l-1* and it is going to be a "coarse level flux" when we advance the level, *l+1*. Therefore after we advance the level *l*, we should save the same flux to two different `FluxRegister.cpp/H` objects. Following two statements show how the difference between fluxes is calculated.

```
flux_reg[l]->FineAdd(flux,...)
flux_reg[l+1]->CrseInit(flux,...)
```

The first statement is used to calculate flux mismatch between levels *l* and *l-1*. The second statement, on the other hand, is used to calculate the mismatch between level *l* and *l+1*. After the difference is calculated, The `Reflux()` function updates the coarse level data.

Even though the averaging, or also called restriction, operation varies with different index types, it is a much simpler operation as compared with the refluxing. The restriction for cell centered data simply takes the average of four fine level cells and replaces the data in the underlying coarse level cell with that average. This operation does not violate the symmetry and it is very easy to implement. In the *AMReX* the `average_down()` member function contained in the `MultiFab.cpp/H` class is responsible for the cell-centered restriction.

After explaining the refluxing and the averaging briefly, the Algorithm 5 shows how the synchronization operations are performed in the *AMReX* package. For this algorithm, it is assumed that there are two levels of refinements and the refinement ratio is 2. These values are picked to demonstrate the sequence clearly.

Algorithm 5: Reflux algorithm

```
advance() - Level = L
    Store the fluxes in flux_reg[L+1]->CrseInit()
        advance() - Level = L+1,Cycle = 1
            Store the flux in flux_reg[L]->FineAdd() and
             flux_reg[L+2]->CrseInit()
                advance() - Level = L+2,Cycle = 1
                    Store the flux in flux_reg[L+1]->FineAdd()
                advance() - Level = L+2,Cycle = 2
                    Store the flux in flux_reg[L+1]->FineAdd()
        Reflux between Level 1 and Level 2
        advance() - Level = L+1,Cycle = 2
            Store the flux in flux_reg[L]->FineAdd() and
             flux_reg[L+2]->CrseInit()
                advance() - Level = L+2,Cycle = 1
                    Store the flux in flux_reg[L+1]->FineAdd()
                advance() - Level = L+2,Cycle = 2
                    Store the flux in flux_reg[L+1]->FineAdd()
        Reflux between Level 1 and Level 2
Reflux between Level 0 and Level 1
Average down the values from coarse to fine level
```

Chapter 6

AMREX WITH THE FRONT-TRACKING METHOD

In the numerical multiphase flow studies, the accurate representation of the physical phenomena requires the discretization of some part of the computational domain with a higher resolution than the rest of the domain. However, there is no sharp transition between the high and the regular-resolution regions in many cases. More importantly, the high-resolution region can vary in the course of a simulation. For instance, the interface of two immiscible fluids travels in space as time marches, therefore the high-resolution region which encloses the interface changes accordingly. A direct solution to this problem is to discretize the entire domain with the highest resolution. But, obviously, this is not applicable in terms of computational resources. Alternatively, curvilinear grids, which are obtained by distorting the rectangular grids, can offer a significant improvement in terms of both the efficiency and accuracy as Muradoglu and Kayaalp [Muradoglu and Kayaalp, 2006] suggested. But using a fixed number of grid points does not allow the mesh to be fully adaptable.

To overcome this difficulty, the adaptive mesh refinement offers a great advantage by refining and coarsening the regions as the simulation proceeds. Application of the AMR to the multiphase flows has been practiced in many studies. Popinet ([Popinet, 2003] and [Popinet, 2009]) have combined the octree method with the volume-of-fluid scheme in the *Gerris* package to simulate some important multiphase problems such as the fluid-jet breakup and the rising bubble.
Hua and Lou [Hua and Lou, 2007] applied the adaptive mesh method to the interface elements in the front-tracking method. Similarly, Sussman and Almgren [Sussman et al., 1999] have established the block-structured AMR algorithm for two-phase flows by using the level-set method.

In this study, the *AMReX* package is selected to manage the adaptive grid hierarchy and the *front-tracking method* developed by Unverdi and Tryggvason et al. [Unverdi and Tryggvason, 1992] is used to solve the multiphase flow part. The combination of these two algorithms is advantageous in the following aspects:

- Block-structured method used in the *AMReX* maintains global conservation with sufficient accuracy.
- Time-step adaptation allows us to integrate the refinement levels independently with their own timestep size. Therefore the overall efficiency increases.
- The connection between the Eulerian and the Lagrangian grids in the front-tracking method is relatively easy.

Besides the advantageous, there are some challenges in the combination of these algorithms. The major adversity is that the front-tracking algorithm has been developed on the staggered grid arrangement where the vectorial properties are kept at the cell faces, however, the *AMReX* supports the cell-centered and the node-based approximations especially for the restriction and the prolongation operations. As a result of the grid arrangement mismatch, interpolation and refluxing operations should be reconsidered in order to make the algorithms run coherently. The following sections discuss the time-stepping procedures with important modifications, the divergence-free Balsara interpolation for staggered grid and details of the Poisson solver to incorporate the boundary conditions in *Hypre*.

6.1 Time-Stepping for the Staggered Grid

There are two common ways to solve the governing equations in the AMR applications. The first approach is to solve all levels simultaneously with the same time-step size as Minion [Minion, 1996] and Vanella et al. [Vanella et al., 2010] have done. The second one is to advance each level independently with its own timestep, as Almgren et al.[Almgren et al., 1998], Sussman et al. [Sussman et al., 1999] and Roma et al.[Roma et al., 1999] suggested. In this approach, the interlevel communication is established by passing the Dirichlet condition to finer levels if they exist. This study focuses on the latter approach.

The subcycled time-stepping is a recursive algorithm where the advancement of each level is completed by following the steps:

- The level, l is advanced with its own time-step. If $l > 0$ then the boundary conditions can be provided either from the level l-1 or from the physical boundaries.

- The level $l+1$ is advanced multiple times depending on the *refinement ratio, r*. For instance, if r is two, then the level $l+1$ is advanced two times to be syncronized in time with the level l. Therefore the relation between time-steps of each level l and $l+1$ can be stated as

$$\Delta t^{l+1} = ({}^{1}\!/\!{}_{r})\Delta t^{l}. \tag{6.1}$$

  The boundary conditions for the level $l+1$ is again provided either by the level l or the physical boundaries.

- The mismatch at the interface of the level l and $l+1$ is calculated and is eliminated by the synchronization operations.

Since each level is advanced as if it is the only level, the synchronization is the most critical part of the sequence given above. There are different sources of the mismatch at the interface of levels and they need to be eliminated to maintain overall conservation. When the level l and $l+1$ are advanced to the same simulation time, the synchronization operations should deal with three major sources of mismatch:

1. *Interior mismatch*: The level l and $l+1$ data are not synchronized.

2. *Velocity mismatch*: The normal velocity flux from the level l and area weighted average over the level l face of the level *l+1* velocity fluxes do not agree at the *l/l+1* interface. For the two-dimensional setting in Figure 6.1 , this mismatch can be shown mathematically by [Vanella et al., 2010];

$$u^{l}_{i+1/2,j}\Delta y^{l} = u^{l+1}_{i+1/2,j-1/4}\Delta y^{l+1} + u^{l+1}_{i+1/2,j+1/4}\Delta y^{l+1}. \tag{6.2}$$

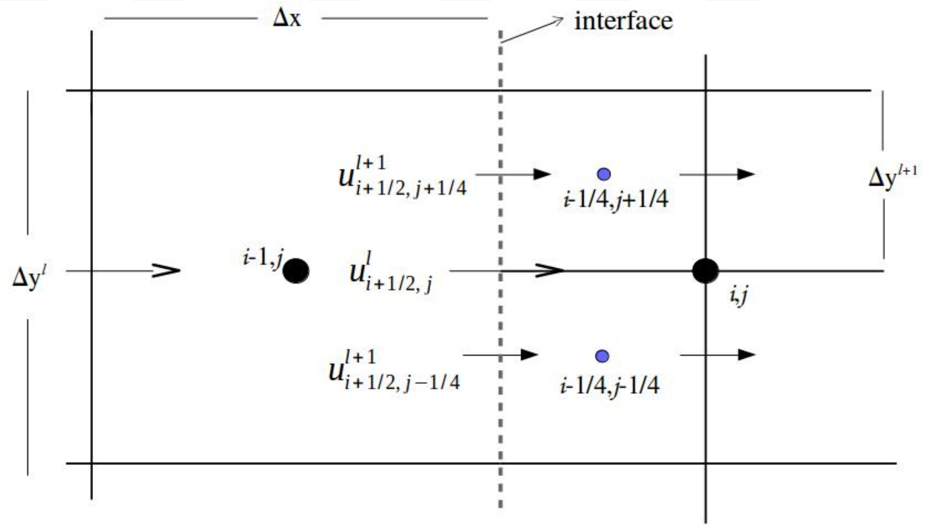


Figure 6.1: Velocity mismatch between the velocities at the level l and *l+1*

3. *Pressure mismatch*: In the projection method, the pressure field is calculated by the elliptic Poisson equation, Eq.4.17, to enforce the continuity equation in the domain. Since the pressure is cell-centered in the staggered grid arrangement, discretization of the elliptic operator $\nabla \cdot \frac{1}{\rho}\nabla p$ requires to take the difference of pressure gradient, $\frac{1}{\rho}\nabla p$, located at the faces of the cells. This pressure gradient term is called *pressure flux* and, similar to the *velocity mismatch*, there is a disagreement at the *l/l+1* interface between the pressure fluxes coming from the coarse and fine levels.

The purpose of the synchronization operations is to eliminate the mismatches listed above. Before introducing the operations, it is important to note that the refluxing strategies proposed by Berger and Colella [Berger and Colella, 1989] for the hyperbolic problems cannot be used in this study. Since the information propagates with infinite speed in elliptic problems, an additional Poisson equation should be solved to update all grids in the level even though the mismatch is localized at the interface.

The velocity for the new timestep at the level l and $l+1$ can be written as:

$$\mathbf{u}^{n+1,l} = \mathbf{u}^{\star,l} - {}^{1}\!/_{\rho^l}\nabla p^l, \tag{6.3}$$

$$\mathbf{u}^{n+1,l+1} = \mathbf{u}^{\star,l+1} - {}^{1}\!/_{\rho^{l+1}}\nabla p^{l+1}. \tag{6.4}$$

Therefore the *velocity mismatch* between $\mathbf{u}^{\star,l}$ and $\mathbf{u}^{\star,l+1}$ and the *pressure mismatch* between ${}^{1}\!/_{\rho^l}\nabla p^l$ and ${}^{1}\!/_{\rho^{l+1}}\nabla p^{l+1}$ results in the total mismatch between $\mathbf{u}^{n+1,l}$ and $\mathbf{u}^{n+1,l+1}$. The solutions to these mismatches can be listed as follows:

1. *Interior synchronization*: The solution data at the level $l+1$ are assumed to be more accurate than the underlying coarse level data. Hence the interior cells of level l should be updated. The regular restriction operation performs this synchronization by averaging the solution from fine to coarse level.

2. *Velocity and Pressure synchronization*: At the end of each time-step important quantities are stored in the registers described in section 5.3.4. These quantities are the mismatches calculated at the $l/l+1$ interface. As Almgren et al. [Almgren et al., 1998] suggested the velocity register accumulates the area-weighted difference between $\mathbf{u}^{\star,l}$ and $\mathbf{u}^{\star,l+1}$, i.e,

$$\delta\mathbf{u}^{\star,l} = -\mathbf{u}^{\star,l} + \frac{1}{r}\sum_{k=1}^{r}\sum_{faces}\left(A^{l+1}\mathbf{u}^{\star,k,l+1}\right), \tag{6.5}$$

where A^l and A^{l+1} denote the areas of the faces of the coarse and fine level cells respectively, at the $l/l+1$ interface. The outer summation in Eq.6.5 is up to the refinement ratio due to the subcycling and the inner summation is over the faces of fine level cells which abut the coarse level cell at the interface. Pressure flux mismatch can be accumulated in the pressure registers in the same way

$$\delta({}^{1}/_{\rho^l}\nabla p)^l = -A^l({}^{1}/_{\rho^l}\nabla p)^l + \frac{1}{r}\sum_{k=1}^{r}\sum_{faces}\left(A^{l+1}({}^{1}/_{\rho^{l+1}}\nabla p)^{l+1}\right), \tag{6.6}$$

where $\delta\mathbf{u}^{\star,l}$ and $\delta({}^{1}/_{\rho^l}\nabla p)^l$ define the right-hand side of the additional Poisson equation, which determines the correction velocity field to synchronize the level l and l*+1* as

$$\nabla\cdot\left(\frac{A^l}{\rho^l}\nabla(\delta e^l)\right) = \nabla\cdot\left[(\delta\mathbf{u}^{\star,l}) + (\delta({}^{1}/_{\rho^l}\nabla p)^l)\right]. \tag{6.7}$$

The Neumann boundary condition, ${}^{\partial(\delta e)^l}/_{\partial n} = 0$ can be set when the outflow is defined on the physical boundaries. If it is inflow, then the Dirichlet condition, $(\delta e)^l = 0$, works fine. When $l >$ *0*, then prolongation from the coarser level can supply the interior boundary conditions.

After the solution is obtained for $(\delta e)^l$, the velocity in the level l can be updated as

$$\mathbf{u}^{n+1,l} = \mathbf{u}^{n+1,l} - \frac{1}{\rho^l}\nabla(\delta e)^l. \tag{6.8}$$

Equations Eq.6.3 - 6.8 use density as average of two adjacent cells since it is stored at the center of the cells.

6.2 Interpolation Scheme for the Staggered Grid

As briefly discussed before, the ghost cells are used at the coarse/fine interface to supply the information from coarser to the finer grid by *prolongation* operation.

Similarly, the fine level solution is used to update the underlying coarse level data by the *restriction* operation.

Depending on the problem and the numerical grid arrangements, the nature of these two important operations changes significantly. Berger and Oliger [Berger and Oliger, 1984] developed the adaptive mesh method on hyperbolic conservation equations, Eq.6.9.

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} + \frac{\partial \mathbf{H}}{\partial z} = 0 \tag{6.9}$$

where $\mathbf{U}$ denotes the conserved quantity and $\mathbf{F}, \mathbf{G}$ and $\mathbf{H}$ are the flux vectors in x, y and z directions respectively. The integral form of the formulation in Eq.6.9 allows the physical discontinuity present within the flow domain. To account for the discontinuities, Colella et al.[Colella, 1990] have developed a conservative, higher-order Godunov method. However, when global conservation is not satisfied during AMR operations, the solution may contain unphysical features. Hence, Berger and Oliger [Berger and Colella, 1989] incorporated bilinear interpolation scheme to interpolate the coarse level data to the finer level and similarly at the end of each timestep fine level data are restricted onto the coarse level. But, they noticed that restriction violates conservation at the coarse-fine interface, hence, to remedy this deficiency they computed the flux mismatch at the interface and update the coarse level data accordingly.

As a part of the Navier-Stokes equations, formulations for the incompressible flows require to satisfy the divergence-free velocity, $\nabla \cdot \mathbf{u}$, condition to properly represent the flow domain. Almgren et al.[Almgren et al., 1998], Sussman et al.[Sussman et al., 1999] and Minion [Minion, 1996] applied the approximate projection method on the semi-staggered grid arrangement and they used bilinear interpolation scheme for the prolongation. Because they approximately satisfied the continuity, they solved an additional Poisson equation to preserve global conservation. Martin et al.[Martin et al., 2008], on the other hand, developed the divergence cleaning algorithm by solving an additional Poisson equation. For fluid-structure interaction problems in the adaptive structured-grid hierarchy, Vanella et al. [Vanella et al., 2010] preferred to use a divergence-free prolongation operator utilizing high order polynomials. Although it satisfied conservation constraints successfully, the method is valid only when the refinement ratio between levels is two.

Magnetohydrodynamics (MHD) equations are very similar to incompressible flow equations in terms of the updating solution to preserve continuity. As the divergence of velocity should be kept zero throughout the incompressible flow simulations, the magnetic field needs to satisfy the same condition, $\nabla \cdot \mathbf{B} = 0$, for MHD problems. Balsara [Balsara, 2001] has developed a divergence-free second-order accurate interpolation scheme for the MHD in three-dimensions. He used the staggered grid arrangement to store the magnetic field. In this study, his reconstruction approach is used to interpolate the data from the coarse to the fine grid. The main feature of Balsara prolongation is that it is built upon the TVD principle to maintain stability.

First the reconstruction sequence is introduced for two-dimensional case for the sake of simplicity, Then the three-dimensional version is given. It is useful to define the notation before proceeding with the sequence. As shown in Figure 6.2 let $u^{l,-} = u^l_{i-1/2,j}$ and $u^{l,+} = u^l_{i+1/2,j}$ denote the u-velocity at lower and upper faces in x direction respectively in coarse level, l. Similary $v^{l,-} = v^l_{i,j-1/2}$ and $v^{l,+} = v^l_{i,j+1/2}$ are v velocity components at lower and upper faces in y direction, in coarse level, l. Throughout this analysis the origin is located at the center of the control volume i, j, hence, at i, j $x = 0$ and $y = 0$. Δx and Δy are the edge sizes in x and y directions as shown in Figure 6.2. Since the origin is at i, j, the interval of the domain can be stated as $[-\Delta x/2, \Delta x/2] \times [-\Delta y/2, \Delta y/2]$. Balsara's prolongation scheme can be summarized as follows:

1. The reconstruction starts with fitting piecewise linear profiles for velocities at lower and upper faces in both x and y directions in level l. There are several ways to obtain these slopes. If there is any discontinuity present in the domain, slope limiters, such as `minmod()`, can be prefered. Since there is no physical discontinuity, the slopes can be computed using central differences as

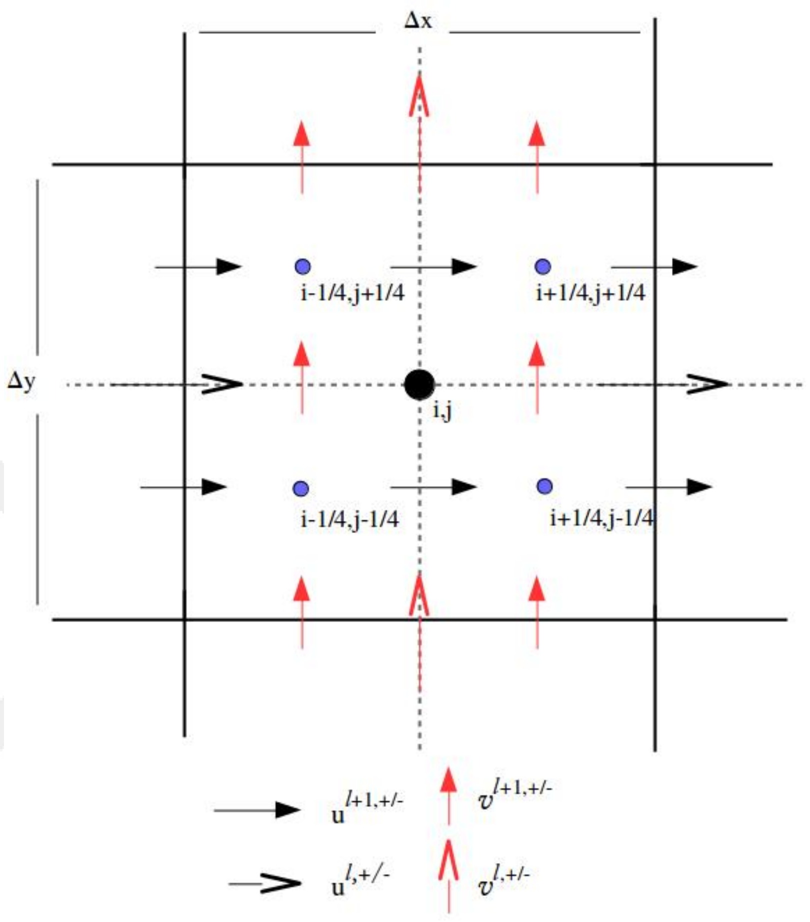


Figure 6.2: Representation of the grid used to explain the Balsara reconstruction.

$$\Delta_y u^{l,\pm} = \frac{u^{l,i\pm1/2,j+1} - u^{l,i\pm1/2,j-1}}{2}, \tag{6.10}$$

$$\Delta_x v^{l,\pm} = \frac{v^{l,i+1,j\pm1/2} - u^{l,i-1,j\pm1/2}}{2}. \tag{6.11}$$

Therefore u and v velocities vary at lower and upper faces in x and y directions as

$$u^l(x = \pm\Delta x/2, y) = u^{l,\pm} + \frac{\Delta_y u^{l,\pm}}{\Delta y} y, \tag{6.12}$$

$$v^l(x, y = \pm\Delta y/2) = v^{l,\pm} + \frac{\Delta_x v^{l,\pm}}{\Delta x} x. \tag{6.13}$$

2. After linear profiles are determined for faces where the velocities are located, polynomial functions are fitted inside the control volume in x and y directions seperately. The coefficients of the polynomials are computed so that they satisfy the continuity in every point inside the control volume. Then these polynomials can be used to calculate velocities in refined levels by prolongation and divergence-free property always holds because the coefficients in coarse level are picked to satisfy the continuity. The polynomials can be constructed as

$$u(x, y) = a_0 + a_x x + a_y y + a_{xx} x^2 + a_{xy} xy + a_{yy} y^2, \tag{6.14}$$

$$v(x, y) = b_0 + b_x x + b_y y + b_{xx} x^2 + b_{xy} xy + b_{yy} y^2. \tag{6.15}$$

3. In this last step, the coefficients are determined to satisfy the continuity. Since linear profiles are fitted at faces, the coefficient of second order terms should be zero, i.e,

$$a_{yy} = b_{xx} = 0. \tag{6.16}$$

When $\nabla \cdot \mathbf{u}$ is computed by the polynomials in Eq.6.14, three additional constraints appear as;

$$a_x + b_y = 0; \quad 2a_{xx} + b_{xy} = 0; \quad a_{xy} + 2b_{yy} = 0. \tag{6.17}$$

Eq.6.16 and 6.17 guarantee that polynomials in Eq.6.14 construct divergence-free velocity field within the control volume.

Now there are seven independent coefficients to determine in Eq.6.14. Two additional constraints provide relations for those coefficients. The first constraint is that the discrete version of continuity, Eq.6.18 , should be satisfied, i.e,

$$(u^{l,+} - u^{l,-})\Delta y + (v^{l,+} - v^{l,-})\Delta x = 0 \tag{6.18}$$

The second condition comes from the fact that polynomials in Eq.6.14 should match the linear profiles given in Eq.6.10 at the lower and upper faces of the control volume in each direction. Equating linear profiles and polynomials at faces gives seven equations to determine seven independent coefficients. Even though eight coefficients are used to represent the variation of polynomials at faces, Eq.6.18 reduces this number to seven.

After some algebraic manipulations, those seven independent coefficients are obtained as

$$a_x = -b_y = \frac{(u^{l,+} - u^{l,-})}{\Delta x} = -\frac{(v^{l,+} - v^{l,-})}{\Delta y}, \tag{6.19}$$

$$a_y = \frac{1}{2}\left(\frac{\Delta y u^{l,+}}{\Delta y} + \frac{\Delta y u^{l,-}}{\Delta y}\right), \tag{6.20}$$

$$b_x = \frac{1}{2}\left(\frac{\Delta x v^{l,+}}{\Delta x} + \frac{\Delta x v^{l,-}}{\Delta x}\right), \tag{6.21}$$

$$a_{xy} = -2b_{yy} = \frac{1}{\Delta x}\left(\frac{\Delta y u^{l,+}}{\Delta y} - \frac{\Delta y u^{l,-}}{\Delta y}\right), \tag{6.22}$$

$$b_{xy} = -2a_{xx} = \frac{1}{\Delta y}\left(\frac{\Delta x v^{l,+}}{\Delta x} - \frac{\Delta x v^{l,-}}{\Delta x}\right), \tag{6.23}$$

$$a_0 = \frac{u^{l,+} + u^{l,-}}{2} - a_{xx}\frac{\Delta x^2}{4}, \tag{6.24}$$

$$b_0 = \frac{v^{l,+} + v_{l,-}}{2} - b_{yy}\frac{\Delta y^2}{4}. \tag{6.25}$$

These coefficients fully determine reconstruction polynomials which satisfy the continuity at every point within the domain.

4. The final step in prolongation is to calculate the velocities at fine level as shown in Figure 6.2. It is very straightforward since the polynomials are fully known.

The locations of fine level velocities can be determined by taking center of the coarse cell as the origin. Then polynomials in Eq.6.14 can be used to calculate the velocities.

The same principles can be applied to the three-dimensional case. The sequence is entirely analogous to the two-dimensional reconstruction with an addition of third direction. The slopes of u, v and w velocities at each face are given by

$$\Delta_y u^{l,\pm} = \frac{u^{l,i\pm1/2,j+1,k} - u^{l,i\pm1/2,j-1,k}}{2}, \tag{6.26}$$

$$\Delta_z u^{l,\pm} = \frac{u^{l,i\pm1/2,j,k+1} - u^{l,i\pm1/2,j,k-1}}{2}, \tag{6.27}$$

$$\Delta_x v^{l,\pm} = \frac{v^{l,i+1,j\pm1/2,k} - u^{l,i-1,j\pm1/2,k}}{2}, \tag{6.28}$$

$$\Delta_z v^{l,\pm} = \frac{v^{l,i,j\pm1/2,k+1} - u^{l,i,j\pm1/2,k-1}}{2}, \tag{6.29}$$

$$\Delta_x w^{l,\pm} = \frac{w^{l,i+1,j,k\pm1/2} - w^{l,i-1,j,k\pm1/2}}{2}, \tag{6.30}$$

$$\Delta_y w^{l,\pm} = \frac{w^{l,i,j+1,k\pm1/2} - w^{l,i,j-1,k\pm1/2}}{2}. \tag{6.31}$$

The piecewise linear profile in three-dimensions then becomes

$$u^l(x = \pm\Delta x/2, y, z) = u^{l,\pm} + \frac{\Delta_y u^{l,\pm}}{\Delta y} y + \frac{\Delta_z u^{l,\pm}}{\Delta z} z, \tag{6.32}$$

$$v^l(x, y = \pm\Delta y/2, z) = v^{l,\pm} + \frac{\Delta_x v^{l,\pm}}{\Delta x} x + \frac{\Delta_z v^{l,\pm}}{\Delta z} z, \tag{6.33}$$

$$w^l(x, y, z = \pm\Delta z/2) = w^{l,\pm} + \frac{\Delta_x w^{l,\pm}}{\Delta x} x + \frac{\Delta_y w^{l,\pm}}{\Delta y} y. \tag{6.34}$$

Like in the two-dimensions, the reconstruction polynomials in the three-dimensions are given as

$$u(x,y,z) = a_0 + a_x x + a_y y + a_z z + a_{xx}x^2 + a_{xy}xy + a_{xz}xz, \tag{6.35}$$

$$v(x,y,z) = b_0 + b_x x + b_y y + b_z z + b_{xy}xy + b_{yy}y^2 + b_{yz}yz, \tag{6.36}$$

$$w(x,y,z) = c_0 + c_x x + c_y y + c_z z + c_{xz}xz + c_{yz}yz + c_{zz}z^2. \tag{6.37}$$

Divergence-free condition requires

$$a_x+b_y+c_z = 0; \quad 2a_{xx}+b_{xy}+c_{xz} = 0; \quad a_{xy}+2b_{yy}+c_{yz} = 0; \quad a_{xz}+b_{yz}+2c_{zz} = 0. \tag{6.38}$$

The discrete form of the continuity in the three-dimensions reduces 18 independent coefficients to 17, i.e,

$$(u^{l,+} - u^{l,-})\Delta y \Delta z + (v^{l,+} - v^{l,-})\Delta x \Delta z + (w^{l,+} - w^{l,-})\Delta x \Delta y = 0 \tag{6.39}$$

Algebraic computation results in the coefficients for u veloctiy as

$$a_x = \frac{(u^{l,+} - u^{l,-})}{\Delta x}, \tag{6.40}$$

$$a_y = \frac{1}{2}\left(\frac{\Delta y u^{l,+}}{\Delta y} + \frac{\Delta y u^{l,-}}{\Delta y}\right), \tag{6.41}$$

$$a_z = \frac{1}{2}\left(\frac{\Delta z u^{l,+}}{\Delta z} + \frac{\Delta z u^{l,-}}{\Delta z}\right), \tag{6.42}$$

$$a_{xy} = \frac{1}{\Delta x}\left(\frac{\Delta y u^{l,+}}{\Delta y} - \frac{\Delta y u^{l,-}}{\Delta y},\right) \tag{6.43}$$

$$a_{xz} = \frac{1}{\Delta x}\left(\frac{\Delta z u^{l,+}}{\Delta z} - \frac{\Delta z u^{l,-}}{\Delta z},\right) \tag{6.44}$$

$$a_{xx} = -\frac{1}{2}(b_{xy} + c_{xz}), \tag{6.45}$$

$$a_0 = \frac{u^{l,+} + u^{l,-}}{2} - a_{xx}\frac{\Delta x^2}{4}. \tag{6.46}$$

Coefficients of polynomials for v and w velocities can be obtained by applying the transformation to coefficients in Eq.6.40-6.46. To calculate $v(x,y,z)$, we make replacement of $a \to b$, $b \to c$, $c \to a$, $x \to y$, $y \to z$, $x \to z$. Similarly for $w(x, y, z)$ we make replacements of $a \to c$, $b \to a$, $c \to b$, $a \to b$, $x \to z$, $y \to x$, $z \to y$.

6.3 Details of the Poisson Solver

Time advancement of flow domain for single grid is described in Section 4 and it can be roughly divided into two main steps: (1) Temporary velocity calculation. (2) Projection step to calculate divergence-free velocity field at new time level. Advancing the front in multiphase simulations also involves two distinct steps; (1) Calculating gradient of marker function on Lagrangian grid, (2) Solving Poisson equation in Eulerian grid to compute indicator function. Advancing both flow and front domains requires to solve Poisson equation which can be written in generic form as;

$$\nabla \cdot \beta \nabla \phi = RHS, \tag{6.47}$$

where β is a scalar field and RHS is the known source term. In the AMR applications levels can be composed of more than one grid depending on the maximum number of cells a single grid is allowed to contain. The first of two steps for advancing the flow and the front domains can be performed one grid at a time. However, the elliptic Poisson equation, Eq.6.47 should be solved for all grids at the same time for a single level.

Boundary condition implementation also changes when there are multiple grids at a level. During the first step of advancing flow and front domains, the boundary conditions are supplied either from the underlying coarse level, from the other fine grids at the same level or from the physical boundaries. In the second step, all grids are combined and the boundary conditions are implemented as if there is a single grid at the level.

The Poisson equation is solved using the *Hypre* package. To incorporate the boundary conditions in *Hypre* for the second steps of time advancement, the following linear algebra manipulation is performed:

- After discretization of the Poisson equation, *Hypre* solves the equation in the form $Ax = b$. In general case, the matrix A and the vectors b and x are composed of discretization stencil values of cells in *white* region in Figure 6.3.

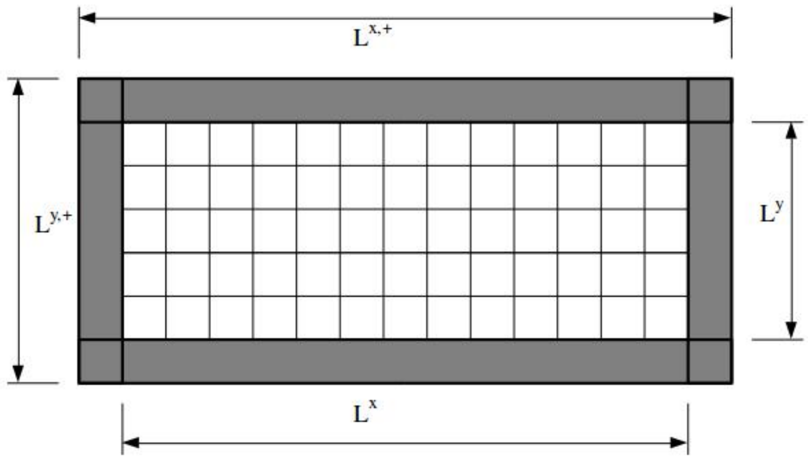


Figure 6.3: The computational grid is extended to include the boundary region for the *Hypre* solver

The *grey* regions with thick black borders are the ghost cells used to impose the boundary conditions.

- For *Hypre* solver, the computational domain is extended from $[L^x, L^y]$ to $[L^{x,+}, L^{y,+}]$ in order to include the boundary region. Then the matrix A is splitted as

$$A = \begin{bmatrix} A_{ii} & A_{ib} \\ A_{bi} & A_{bb} \end{bmatrix} \tag{6.48}$$

Here $[A_{ii}]$ refers to the interior cells, $[A_{ib}]$ and $[A_{bi}]$ are submatrices for cells adjacent to the boundary and finally $[A_{bb}]$ is for the boundary cells. Since the values at the boundaries are known, the equation $Ax = b$ can be rearranged as

$$\begin{bmatrix} A_i & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}_i \\ \mathbf{x}_b \end{bmatrix} = \begin{bmatrix} b_i - A_{ib}\mathbf{u}_0 \\ \mathbf{u}_0 \end{bmatrix} \tag{6.49}$$

When Eq.6.49 is solved, $[\mathbf{x}_i]$ gives the solution values for interior cells and $\mathbf{x}_b$ remains unchanged since it is already determined by the boundary conditions. An important remark is that the manipulation above works fine when *Dirichlet* and *Neumann* boundary conditions are applied on the physical domain. It is also useful when the refinement level is contained entirely in underlying coarse level. If periodicity is applied in the domain, then the manipulation may results in wrong results.

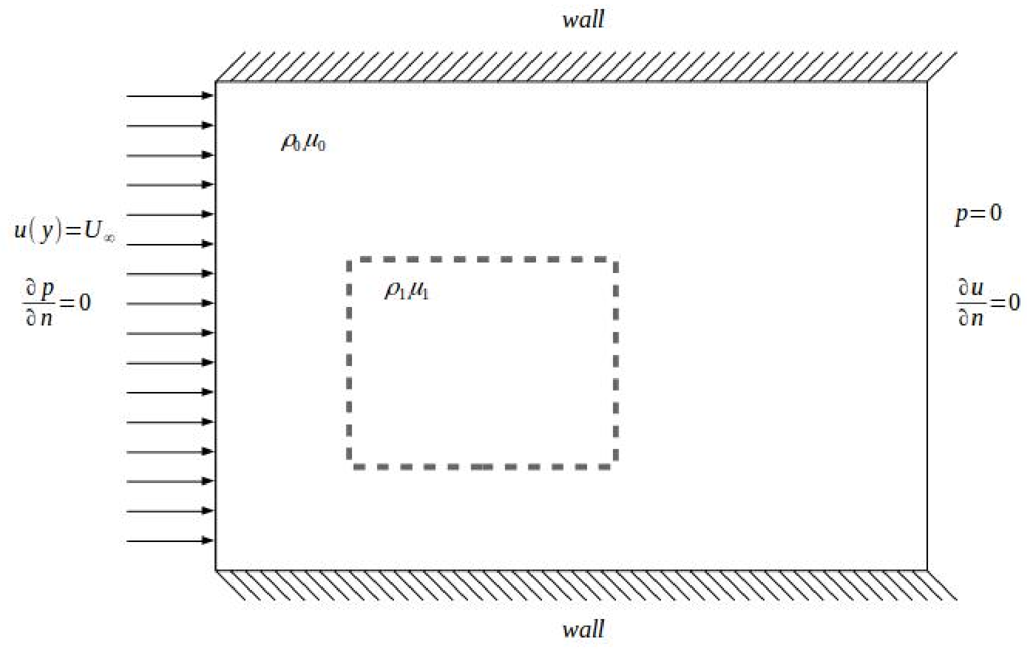
6.4 Benchmark Results: Variable Density/Viscosity Incompressible Flow

The one-fluid front tracking algorithm with the block-structured AMR implementation is validated using two basic benchmark problems. The first problem is incompressible viscous flow in a square duct with variable density/viscosity. The schematic representation of the problem is shown in Figure 6.4. Here ρ_0 and μ_0 denote the density and viscosity of primary flow domain while ρ_1 and μ_1 stand for the small region within the domain. The horizontal and vertical lengths of the domain are equal to 1. The boundary conditions can be given as

- $x = 0$: Inflow condition where $u(y, z) = U_0 = 0.01$ and $\partial p/\partial n = 0$.
- $x = L_x$: Outflow condition where $p = 0$ and $\partial u/\partial n = 0$.
- $y = 0$ and $y = L_y$: No-slip condition where $\mathbf{u} \cdot \mathbf{n} = 0$.
- $z = 0$ and $z = L_y$: Periodicity.

Since the domain is periodic in z direction, the number of dimensions reduces by one and this is why the problem is illustrated in two-dimensions in Figure 6.4.

The computational grid for problem in Figure 6.4 is shown in Figure 6.5. The length of the domain in each direction is $L_x = L_y = L_z = 1$ and there are 32 elements in the base grid. The refinement ratio between the coarse and single fine grid is two, therefore there are 64 grids per direction in the refined region.



wall
$\rho_0 \mu_0$
$u(y)=U_\infty$
$\frac{\partial p}{\partial n}=0$
$\rho_1 \mu_1$
$p=0$
$\frac{\partial u}{\partial n}=0$
wall

Figure 6.4: Schematic representation of the flow in a square duct with the variable density/viscosity.

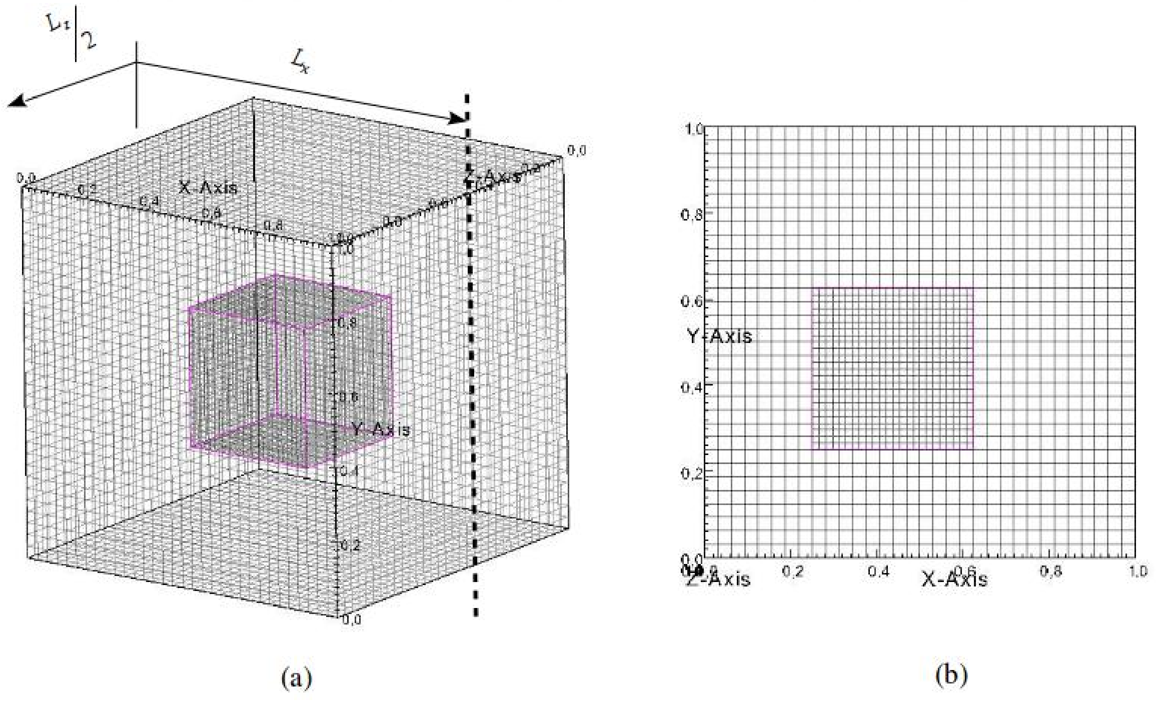


Figure 6.5: Computational domain for the problem depicted in Figure 6.4. There are 32 elements in each direction for the base grid.

The purpose of the simulation is to show the accuracy of the AMR method for single phase *Hagen-Poiseuille* flow. This can be accomplished by establishing the communication between levels correctly and performing the synchronization after each time step to satisfy the divergence-free constraint.

The first part of this benchmark problem involves variable density in the domain. The density in each region is constant therefore the incompressibility is not violated. The level is created by tagging the cells where density is different than the primary domain. Instead of error estimation procedures which calculates truncation error in each cell as Almgren et al.[Almgren et al., 1998] used, here the user-defined function is used to tag the cells. The region enclosed by the dashed lines in Figure 6.4 shows the possible refinement region. To determine the flow regime the Reynolds number can be calculated for the given domain parameters as

$$Re = \frac{\rho U L_c}{\mu} \tag{6.50}$$

where U is the characteristic velocity selected as the average inflow velocity in this case and L_c is the characteristic length, $L_c = 1$. If density and viscosity in primary domain are $\rho_0 = 1.0$ and $\mu_0 = 0.1$, then Reynolds number becomes $Re = 1$, i.e, the flow is *Laminar*.

The analytical solution of the *Hagen-Poiseuille* type laminar flow for symmetric rectangular domain can be given for two-dimensions as;

$$u(y) = u_{max}\left[1 - \left(\frac{y^2}{L_y^2}\right)\right] \quad \text{where} \quad u_{max} = -\frac{1}{8\mu}\frac{dp}{dx}L_y^2 \tag{6.51}$$

Before proceeding with the solution, a few remarks on flow configuration should be stated:

1. No-slip boundary condition in upper and lower faces in vertical direction dictates that the maximum velocity is obtained at the center of the domain.

2. Shear stress varies from 0 at the center line to maximum at the wall.

3. In the steady state the wall shear is related to the maximum velocity which is independent of density.

4. The body force term in the Navier-Stokes equation, $\rho\mathbf{g}$, is ignored in this case. Therefore the density is used just to create the refined region.

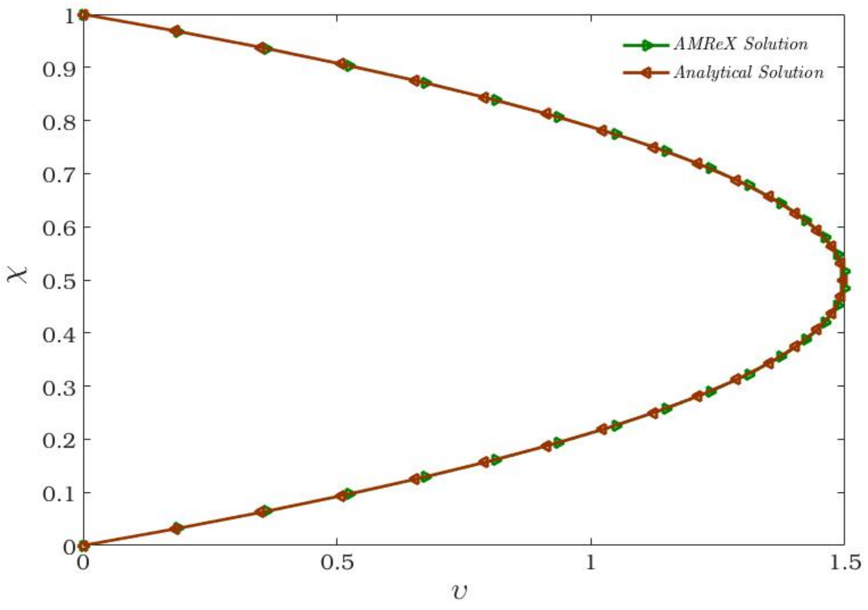


Figure 6.6: Results for the variable density *Hagen-Poiseuille* flow. Green line indicates the *AMReX* solution

Figure 6.6 compares the analytical solution with the *AMReX* solution. Here both x and y axes are nondimensionalized by $\upsilon = {}^{u}/_{U_0}$ and $\chi = {}^{y}/_{L_y}$ respectively. The ratio of density between ρ_0 and ρ_1 is $\bar{\rho} = {}^{\rho_1}/_{\rho_0} = 0.75$. The solution is obtained from the dashed line located vertically at $[L_x, L_z/2]$ and depicted in Figure 6.5. The analytical and the *AMReX* results perfectly match. The numerical error with respect to the analytical solution is quantified as

$$\|err_o\| = \sqrt{\sum_{i,j} h^2 e_{i,j}^2} \tag{6.52}$$

where h is the cell spacing and $e_{i,j}$ is the velocity difference between the analytical solution and the *AMReX* result for the cell located at i, j. v and w velocities are

$\|err_0\|$	
n = 16	n = 32
5.16e-5	7.85e-5

Table 6.1: $\|err_0\|$ for the variable density Hagen-Poiseuille flow

zero in the analytical solution. For the *AMReX* result it is calculated as $V = \sqrt{u^2 + v^2 + w^2}$

Table 6.4 shows two calculations for two different resolutions. The error is calculated in the region of coarse grid where refinement level is placed. Even 16 elements provides considerable accuracy and it indicates that interlevel communications are work properly.

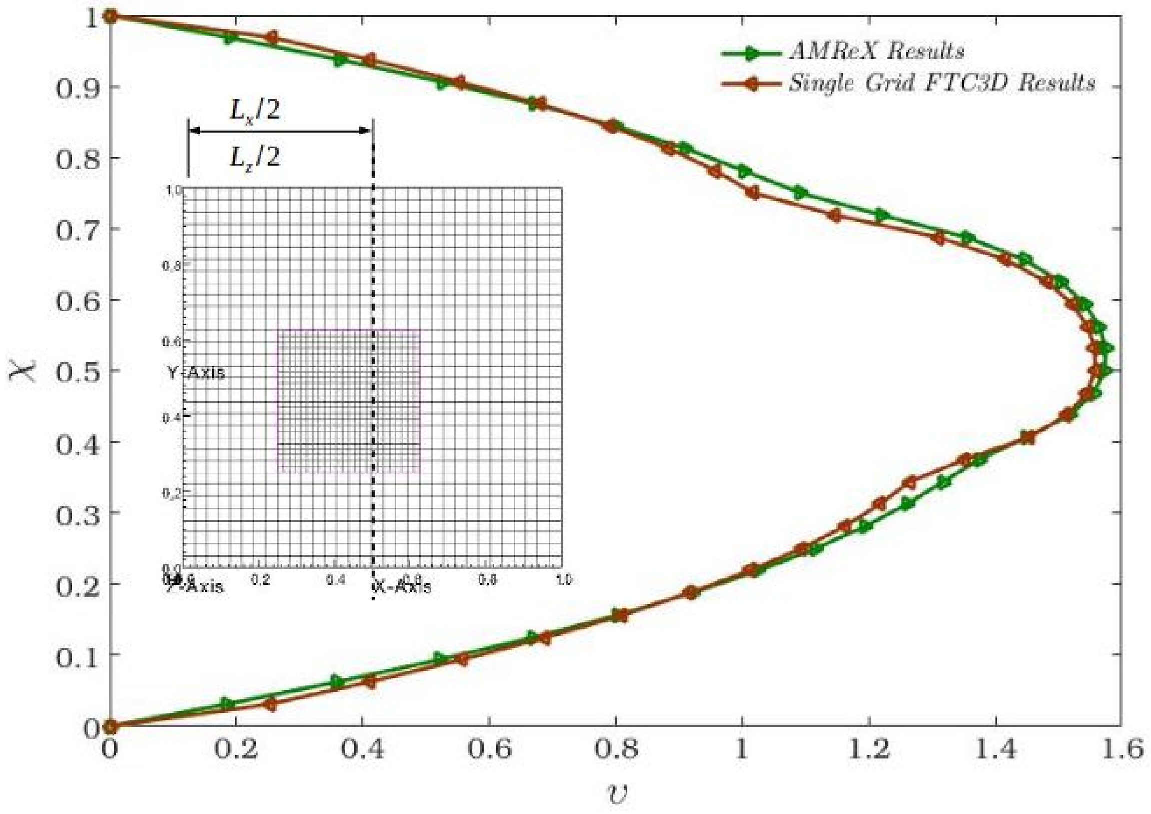


Figure 6.7: Comparison of the velocity profiles calculated in the single grid and the *AMReX* solvers

The second test case deals with the incompressible flow with variable viscosity. Change in the viscosity has direct effect on the simulation results as shown in Figure 6.7 which shows the results of the variable viscosity simulation where 32 elements are used in each direction. Since the viscosity ratio is $\bar{\mu} = {}^{\mu_1}/_{\mu_0} = 0.1$, the velocity increases significantly along the center line. There is a small discrepancy between the single grid and the *AMReX* solutions as shown in Figure 6.7. The data are taken along the vertical dashed line at $x = L_x/2, z = L_z/2$, hence, it goes through the refined region. The difference between the solutions results from the errors in the synchronization step because the divergence-free constraint cannot be satisfied perfectly along the coarse/fine interface and thus the global conservation is violated.

Figure 6.8 shows the situation of divergence-free constraint, $\nabla \cdot \mathbf{u}$, calculated on cross-sectional plane at $z = L_z/2$. Since it is an incompressible flow, the continuity should be satisfied in the entire domain. As Figure 6.8 indicates, only the cells in the vicinity of the interface produce non-zero value for $\nabla \cdot \mathbf{u}$. Even though the non-zero divergence results in loss of conservation, the error does not exceed $1e-5$ even around the interface. This error tends to decrease when more effort is put in the sychronization step. In the current algorithm, temporary velocity and pressure gradient mismatch at the interface are tried to be eliminated by single synchronization interation. If more iterations are done, the error values are expected to decrease as much as desired. However, there is always a trade-off between the computational efficiency and the accuracy when the synchronization is implemented. While the accuracy increases with the higher number of refluxing iterations, the simulation cost also increases.

Interlevel communications bring another source of error around the refined region. Although Balsara interpolation is designed to satsify the divergence-free constraint across the levels, the data on the coarse grid cannot be conveyed to the ghost cells of the finer grid perfectly because the piecewise linear profiles are implemented on each faces. Using more stencils to increase the accuracy of the interpolation requires more computational effort. In addtion to the interpolation, the restriction operation causes

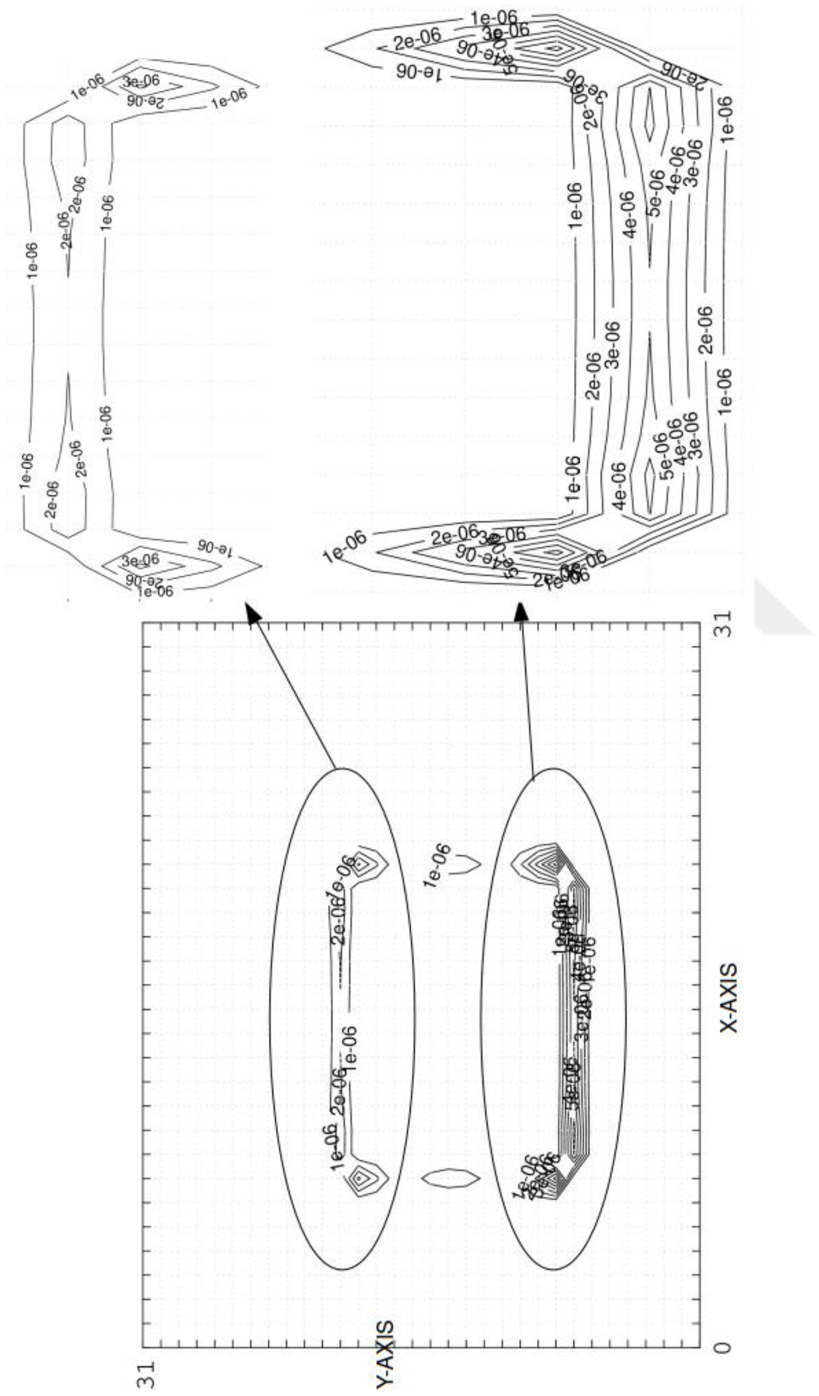


Figure 6.8: Constant contours of $\nabla \cdot \mathbf{u}$. The divergence condition is satisfied except for the coarse/fine interface.

loss of conservation due to the simple averaging of the fine level data.

The results of these benchmark problems indicates that the synchronization and the interlevel communication operations work with acceptable error margins and the overall behavior of the solution is not effected by this deficiency.

6.5 Benchmark Results: Incompressible Multiphase Flow with the Stationary Bubble

In the previous section, the variabe density/viscosity problem demonstrated that the algorithm provides proper grid generation as well as the accurate implementation of interpolation and scynchronization schemes. In this section the results for multiphase part is presented for a stationary bubble. Essentially the same analysis are performed to validate the algorithm.

Figure 6.9 shows the properly-nested two-level adaptive grid with refinement ratio of two. The light green and the red lines indicate the base (or coarse) and the fine levels, respectively. The red sphere in the middle of the domain represents the stationary bubble. The two colors, red and white, inside and outside of the bubble show the constant density contours in both phases. As in the case with the previous problem, ρ_0 and ρ_1 denote the densities in the primary flow domain and in the bubble region, respectively. The density ratio is: $\bar{\rho} = {}^{\rho_1}/_{\rho_0} = 0.8$.

Since the maximum number of cells allowed in a single grid is less then the total number of cells in each level, multiple grids are created in both the coarse and the fine levels as shown in Figure 6.9. This is an important feature in terms of the solution strategies, therefore a few remark should be stated:

- Using the multiple grids allows the algorithm to take advantage of the parallelization capabilities of the *AMReX*.
- The boundary conditions between the subgrids are directly applied by the *AMReX*, therefore no additional user-defined boundary condition routine is required.

- Interpolation operation fills the ghost cells of the outer grids at the fine level.

- Calculation of the temporary velocity field, which requires discretization of the advective and the viscous terms, can be done in each subgrid independently. However the elliptic Poisson equation should be solved in all grids at once. As it is explained in the previous sections, this is a direct result of the nature of the elliptic equation where the information travels with an infinite speed.

- Smoothing the surface tension and the gradient of indicator function from the front to the flow grid may require the communication among multiple grids. When more than one processor is used, the specialized MPI routines can handle the communication.

- There are two ways to solve all grids at once with *Hypre* package. In the first way, the matrix and the vector objects are inititated for the first grid by using the appropriate computational domain indices for that first grid. Then for the next grids the extent of those objects are enlarged by updating the indices and the values are stored in the same objects. At the end of this operation, each object contains the information from all grids. The second way is to convert multiple grids into a single grid temporarily. The latter approach is easier in terms of the implementation, however, it is highly susceptible to be case dependent.

Instead of the truncation error, the gradient of indicator function, which is smoothed from the front to the grid, is used to tag the cells that constitute the refinement level. Even though there are many other alternatives, smoothed variables are the most appropriate candidates to locate the exact position of the front, because the values of these variables are non-zero only in the near region around the interface. The value of indicator function can also be used, but after solving the Poisson equation, $\nabla \cdot (\nabla I) = \nabla \cdot G$, the values of the indicator function are updated in almost every cell in the domain especially when small number of cells are used. Therefore tagging the cells which should enclose the bubble becomes a difficult task. When appropriate

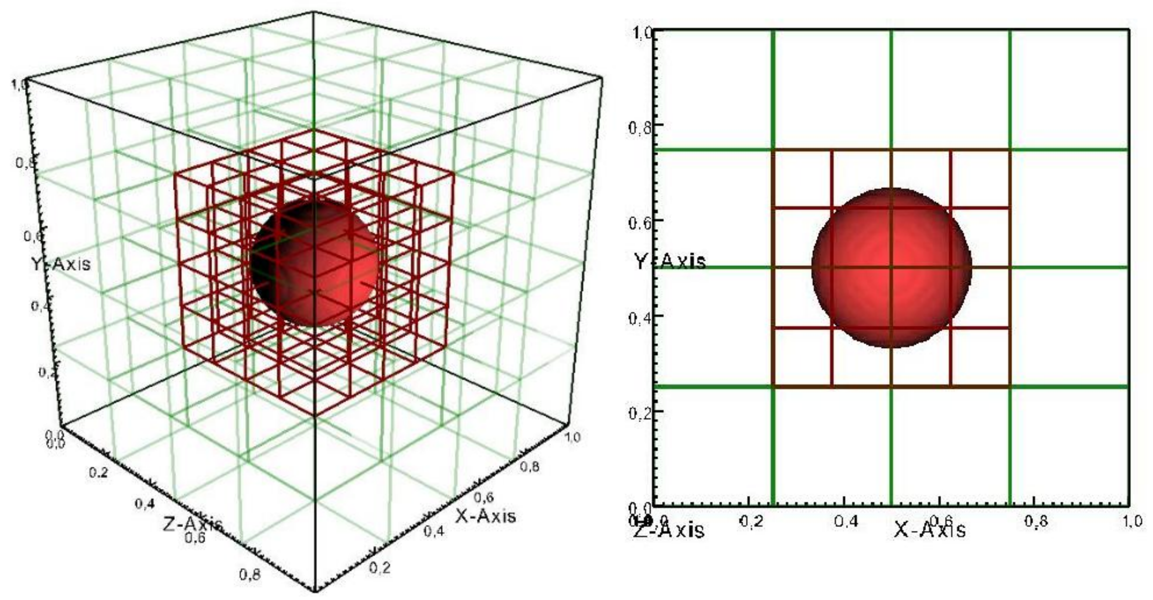


Figure 6.9: Properly-nested grid generation for the multiphase flow problem

error function is not used, the excessive number of cells are clustered in the refinement level, which reduces the computational efficiency dramatically.

Figure 6.10 shows the borders of the refinement level and the gradient of density in both phases simultaneously. As it demonstrates clearly, the refinement level is created just around the bubble to avoid loss of efficiency.

The results of multiphase flow simulation with the stationary bubble are presented in Figure 6.11.

Figure 6.11 -(a) shows the distribution of the velocity magnitude, $V = \sqrt{u^2 + v^2 + w^2}$, at constant $z = L_z/2$ plane. The same boundary conditions with the previous section are applied here. The the flow is developed due to the *no-slip* condition at both the upper and the lower faces. The domain is again periodic in z direction and the simulation run up to $t = 1$, which is sufficient to obtain a fully-developed flow. The velocity takes its maximum value in the refined region due to presence of the bubble. The black dashed-rectangle indicates the location of refinement region. The distribution of the surface tension force, that Figure 6.11 -(b) shows for x direction, influences the flow regime. As Figure 6.11 -(c) clearly shows that the results obtained

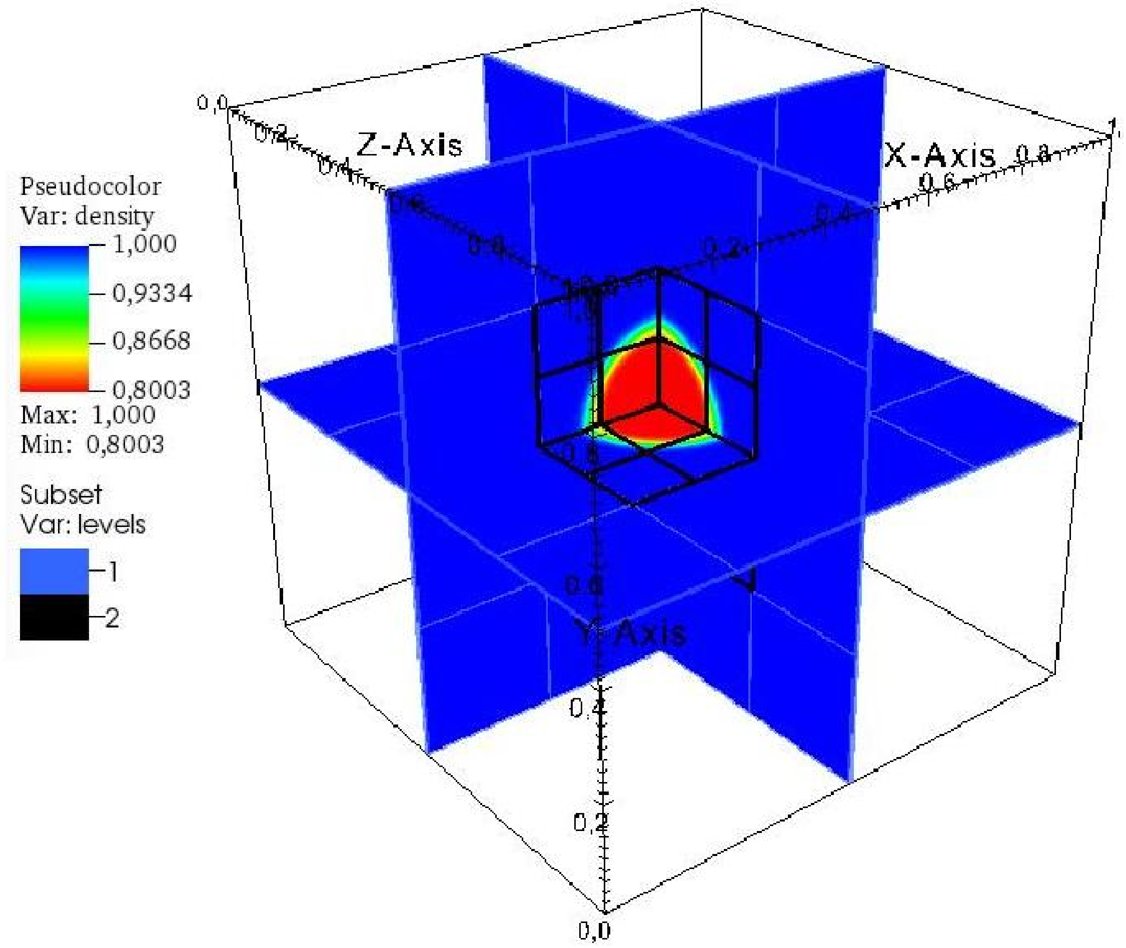


Figure 6.10: Gradient of density in both phases

using the single grid and using the *AMReX* solvers compare very well.

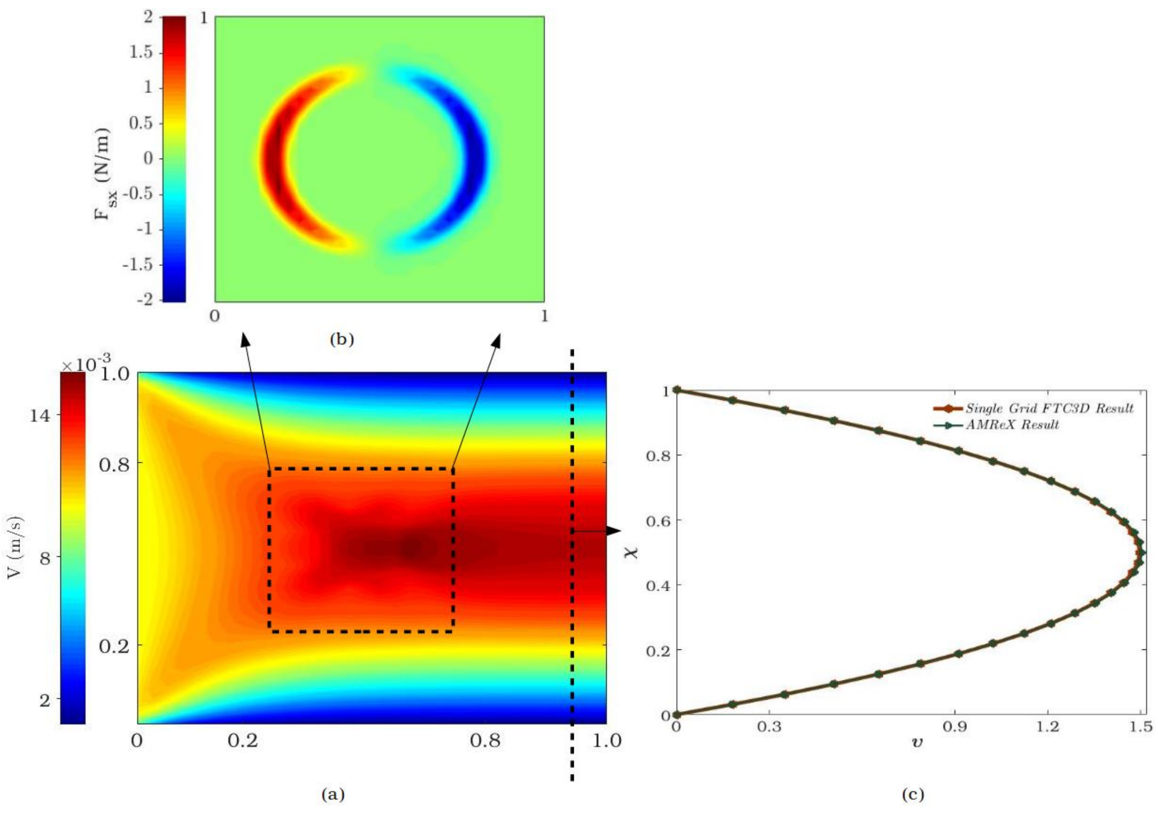


Figure 6.11: The results of multiphase flow simulation. (a) The distribution of the u velocity field at constant $z = L_z/2$ plane. (b) The surface tension force in x direction in the refined level at constant $z = L_z/2$ plane. (c) Single grid and the *AMReX* results for the u velocity at the end of the domain.

Chapter 7

CONCLUSION

The present study combines the finite-volume/front-tracking algorithm on the staggered grid arrangement with the block-structured and subcycled adaptive mesh refinement method. The algorithm developed here aims two main objectives: (1) Applying the structured AMR method on the staggered grid to satisfy the divergence-free constraint in an exact manner, (2) utilizing the subcycled timestepping algorithm in order to make the algorithm as efficient as possible. Simplified synchronization operations after each time step is the natural outcome of the succesful combination of these two steps. The grid management is handled by the *AMReX* package, since the recursive algorithm for subcycled timestepping is inherently implemented in this package.

The thesis consists of three main parts. The first part concantrates on the mathematical derivation of the governing equations and the front-tracking method used to simulate the multiphase flow. In this part, solution methods for the Poisson equation are also discussed. Finally the *Hypre* package used to solve the resulting linear system of equation $Ax = b$ is described by providing the necessary built-in functions.

The second part presents the main idea of the adaptive mesh refinement by explaining the key operations such as flux correction. After the fundamentals of the AMR are established, following sections are devoted to investigate how the key operations and the data management are handeled in the *AMReX* package.

A certain numer of adversities have occured during combination of the front-tracking method and the AMR procedures described in the first and the second parts. The third and last part of the thesis has been devoted to attack those problems:

- The interlevel communication operations, i.e, restriction and interpolation, are mainly developed for cell-centered or node-based schemes in the *AMReX* package. Since the fully-staggered arrangement keeps the vectorial quantities at the cell faces, the special interpolation routines should have been implemented. For this purpose the Balsara interpolation has been selected and implemented to fill the fine level ghost cells. Balsara has formulated this routine originally for the magnetohydrodynamics (MHD). Since both MHD and the incompressible flow equations require to satisfy similar divergence-free constraint, his interpolation scheme appears as a good choice.

- *AMReX* is highly compatible with a second-order approximate projection method described by Almgren et al.[Almgren et al., 1996]. In this scheme the primary variables other than pressure are stored at the cell centers. Therefore following cumbersome synchronization operations are required: (1) Interpolated normal velocities from cell centers to faces should be corrected at coarse/fine interface. (2) Refluxing should be performed to eliminate mismatch due to viscous and advective fluxes at the interface. (3) Node-based pressure residuals due to approximate projection method should be calculated and included in the synchronization.

  Since the staggered grid arrangement is preferred, only the normal velocity and the pressure gradient fluxes are the sources of mismatch at the interface. An additional elliptic Poisson equation is solved to calculate the correction field. This is a much simpler synchronization operation than the one presented in Almgren et al. [Almgren et al., 1998].

Two benchmark problems are used to validate the present algorithm. The first one is a basic Hagen-Poiseuille flow problem with the variable density/viscosity. In the case of variable density, the body force term is neglected for the sake of simplicity and the density variation is used only for creating the refinement levels. Since the resulting velocity profile perfectly matches the analytical solution, it is deduced that the divergence-free interpolation works properly. Variable viscosity, on the other hand, directly influences the solution. After validating the interpolation scheme, the variable viscosity case was used to monitor the effectiveness of the synchronization step. As Figure 6.8 demonstrated, $\nabla \cdot \mathbf{u}$ constraint has been violated around the interface by the order of 10^{-6} which is deemed to be acceptable in this case.

The mutiphase flow problem with the stationary bubble showed that the *Hypre* works perfectly when there are multiple grids at each level. The gradient of indicator function is used as error estimator and the location of the refinement level proved that it is a good candidate to tag the cells. The results showed that the interpolation and the synchronization operations work properly for the multiphase regime as well.

Even though the results of benchmark problems are very promising, further developments on the algorithm are required. Important points for the future studies can be briefly listed as follows:

- For multiphase flows, the algorithm has been validated only for stationary bubble case. It should be extended to simulate both moving and deforming bubbles. When bubble changes its position and/or its orientation the regridding should be performed properly.

- The algorithm should be designed to benefit parallelization capabilities of the *AMReX* package.

- Synchronization operations can be improved by implementing two-level multigrid solvers.

- When the algorithm is completed, the performance analysis should be conducted to compare its efficiency with the single grid solvers.

BIBLIOGRAPHY

[Agresar et al., 1998] Agresar, G., Linderman, J., Tryggvason, G., and Powell, K. (1998). An adaptive, cartesian, front-tracking method for the motion, deformation and adhesion of circulating cells. *Journal of Computational Physics*, 143(2):346 – 380.

[Almgren et al., 1996] Almgren, A., Bell, J., and Szymczak, W. (1996). A numerical method for the incompressible navier-stokes equations based on an approximate projection. *SIAM Journal on Scientific Computing*, 17(2):358–369.

[Almgren et al., 1998] Almgren, A. S., Bell, J. B., Colella, P., Howell, L. H., and Welcome, M. L. (1998). A conservative adaptive projection method for the variable density incompressible navier–stokes equations. *Journal of Computational Physics*, 142:1–46.

[Balsara, 2001] Balsara, D. (2001). Divergence-free adaptive mesh refinement for magnetohydrodynamics. *Journal of Computational Physics*, 174:614–648.

[Bayyuk et al., 1993] Bayyuk, S., Powell, K., and Va, B. (1993). A simulation technique for 2-d unsteady inviscid flows around arbitrarily moving and deforming bodies of arbitrary geometry. *AIAA-93-3391-CP*, pages 1013–1024.

[Bell et al., 2012] Bell, J., Almgren, A., Beckner, V., Day, M., Lijewski, M., Nonaka, A., and Zhang, W. (2012). Boxlib user's guide. *github. com/BoxLib-Codes/BoxLib*.

[Bell et al., 1994] Bell, J., Berger, M., Saltzman, J., and Welcome, M. (1994). Three-dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM Journal on Scientific Computing*, 15(1):127–138.

[Bell et al., 1991] Bell, J., H. Howell, L., and Colella, P. (1991). An efficient second-order projection method for viscous incompressible flow. *Proc. 10th AIAA Computational Fluid Dynamics Conference.*

[Bell et al., 1989] Bell, J. B., Colella, P., and Glaz, H. M. (1989). A second-order projection method for the incompressible navier-stokes equations. *Journal of Computational Physics*, 85(2):257 – 283.

[Bell and Marcus, 1992] Bell, J. B. and Marcus, D. L. (1992). A second-order projection method for variable-density flows. *Journal of Computational Physics*, 101(2):334 – 348.

[Berger and Colella, 1989] Berger, M. and Colella, P. (1989). Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84.

[Berger and Oliger, 1984] Berger, M. and Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512.

[Chorin, 1968] Chorin, A. (1968). Numerical solutions of the navier–stokes equations. *Mathematics of computation*, 22:745–762.

[Colella, 1990] Colella, P. (1990). Multidimensional upwind methods for hyperbolic conservation laws. *Journal of Computational Physics*, 87(1):171 – 200.

[Daly, 1969] Daly, B. J. (1969). Numerical study of the effect of surface tension on interface instability. *The Physics of Fluids*, 12(7):1340–1354.

[DeZeeuw and Powell, 1993] DeZeeuw, D. and Powell, K. G. (1993). An adaptively refined cartesian mesh solver for the euler equations. *Journal of Computational Physics*, 104(1):56 – 68.

[Falgout and Jones, 2000] Falgout, R. D. and Jones, J. E. (2000). Multigrid on massively parallel architectures. *Lecture Notes in Computational Science and Engineering*, 14:101–107.

[Glimm et al., 2001] Glimm, J., Grove, J., Li, X., Oh, W., and Sharp, D. (2001). A critical analysis of rayleigh–taylor growth rates. *Journal of Computational Physics*, 169(2):652 – 677.

[Glimm and McBryan, 1985] Glimm, J. and McBryan, O. A. (1985). A computational model for interfaces. *Advances in Applied Mathematics*, 6(4):422 – 435.

[Griffith et al., 2007] Griffith, B. E., Hornung, R. D., McQueen, D. M., and Peskin, C. S. (2007). An adaptive, formally second order accurate version of the immersed boundary method. *Journal of Computational Physics*, 223(1):10 – 49.

[Ham et al., 2002] Ham, F., Lien, F., and Strong, A. (2002). A cartesian grid method with transient anisotropic adaptation. *Journal of Computational Physics*, 179(2):469 – 494.

[Harlow and Welch, 1965] Harlow, F. H. and Welch, J. E. (1965). Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189.

[Hua and Lou, 2007] Hua, J. and Lou, J. (2007). Numerical simulation of bubble rising in viscous liquid. *Journal of Computational Physics*, 222(2):769 – 795.

[MacNeice et al., 2000] MacNeice, P., Olson, K. M., Mobarry, C., de Fainchtein, R., and Packer, C. (2000). Paramesh: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126(3):330 – 354.

[Martin and Cartwright, 1996] Martin, D. and Cartwright, K. (1996). Solving poisson's equation using adaptive mesh refinement. *LBL Research Report*.

[Martin et al., 2008] Martin, D. F., Colella, P., and Graves, D. (2008). A cell-centered adaptive projection method for the incompressible navier–stokes equations in three dimensions. *Journal of Computational Physics*, 227(3):1863 – 1886.

[Minion, 1996] Minion, M. L. (1996). A projection method for locally refined grids. *Journal of Computational Physics*, 127:158–178.

[Muradoglu and Kayaalp, 2006] Muradoglu, M. and Kayaalp, A. D. (2006). An auxiliary grid method for computations of multiphase flows in complex geometries. *Journal of Computational Physics*, 214(2):858 – 877.

[Muradoglu and Tryggvason, 2008] Muradoglu, M. and Tryggvason, G. (2008). A front-tracking method for computation of interfacial flows with soluble surfactants. *Journal of Computational Physics*, 227(4):2238 – 2262.

[Onural, 2006] Onural, L. (2006). Impulse functions over curves and surfaces and their applications to diffraction. *Journal of mathematical analysis and applications*, 322(1):18–27.

[Osher and Sethian, 1988] Osher, S. and Sethian, J. A. (1988). Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12 – 49.

[Pember et al., 1998] Pember, R. B., Howell, L. H., Bell, J. B., Colella, P., Crutchfield, W. Y., Fiveland, W. A., and Jessee, J. P. (1998). An adaptive projection method for unsteady, low-mach number combustion. *Combustion Science and Technology*, 140(1-6):123–168.

[Peskin, 1977] Peskin, C. S. (1977). Numerical analysis of blood flow in the heart. *Journal of Computational Physics*, 25:220.

[Popinet, 2003] Popinet, S. (2003). Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *Journal of Computational Physics*, 190(2):572 – 600.

[Popinet, 2009] Popinet, S. (2009). An accurate adaptive solver for surface-tension-driven interfacial flows. *Journal of Computational Physics*, 228(16):5838 – 5866.

[Rhie and Chow, 1983] Rhie, C. M. and Chow, W. L. (1983). Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21(11):1525–1532.

[Richtmyer and Morton, 1967] Richtmyer, R. D. and Morton, K. W. (1967). *Testing Statistical Hypotheses of Equivalence*. Wiley-Interscience, New York, 2nd edition.

[Roma et al., 1999] Roma, A. M., Peskin, C. S., and Berger, M. J. (1999). An adaptive version of the immersed boundary method. *Journal of Computational Physics*, 153:509–534.

[Ryskin and Leal, 1984] Ryskin, G. and Leal, L. G. (1984). Numerical solution of free-boundary problems in fluid mechanics. part 2. buoyancy-driven motion of a gas bubble through a quiescent liquid. *Journal of Fluid Mechanics*, 148:19–35.

[Scardovelli and Zaleski, 1999] Scardovelli, R. and Zaleski, S. (1999). Direct numerical simulation of free-surface and interfacial flow. *Annual Review of Fluid Mechanics*, 31(1):567–603.

[Sundén and Fu, 2017] Sundén, B. and Fu, J. (2017). Chapter 1 - introduction. In Sundén, B. and Fu, J., editors, *Heat Transfer in Aerospace Applications*, pages 1 – 17. Academic Press.

[Sussman et al., 1999] Sussman, M., Almgren, A. S., Bell, J. B., Colella, P., Howell, L. H., and Welcome, M. L. (1999). An adaptive vevel set approach for incompressible two-phase flows. *Journal of Computational Physics*, 148:81–124.

[Sussman et al., 1994] Sussman, M., Smereka, P., and Osher, S. (1994). A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114(1):146 – 159.

[Tryggvason et al., 2001] Tryggvason, G., Bunner, B., Esmaeeli, A., Juric, D., Al-Rawahi, N., Tauber, W., Han, J., Nas, S., and Jan, Y.-J. (2001). A front-tracking method for the computations of multiphase flow. *Journal of Computational Physics*, 169(2):708 – 759.

[Tryggvason et al., 2011] Tryggvason, G., Scardovelli, R., and Zaleski, S. (2011). Direct numerical simulations of gas-liquid multiphase flows. *Cambridge University Press.*

[Unverdi and Tryggvason, 1992] Unverdi, S. O. and Tryggvason, G. (1992). A front-tracking method for viscous, incompressible, multi-fluid flows. *Journal of Computational Physics*, 100(1):25 – 37.

[Vanella et al., 2010] Vanella, M., Rabenold, P., and Balaras, E. (2010). A direct-forcing embedded-boundary method with adaptive mesh refinement for fluid-structure interaction problems. *Journal of Computational Physics*, 229:6427–6449.

[Yanenko, 1971] Yanenko, N. (1971). *The Method of Fractional Steps for Solving Multidimensional Problems of Mathematical Physics in Several Variables.* Springer-Verlag, Berlin.

[Zhang et al., 2019] Zhang, W., Almgren, A., Beckner, V., Bell, J., Blaschke, J., Chan, C., Day, M., Friesen, B., Gott, K., Graves, D., Katz, M., Myers, A., Nguyen, T., Nonaka, A., Rosso, M., Williams, S., and Zingale, M. (2019). AMReX: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software*, 4(37):1370.

[Zingale et al., 2018] Zingale, M., Almgren, A. S., Sazo, M. G. B., Beckner, V. E., Bell, J. B., Friesen, B., Jacobs, A. M., Katz, M. P., Malone, C. M., Nonaka, A. J., Willcox, D. E., and Zhang, W. (2018). Meeting the challenges of modeling astrophysical thermonuclear explosions: Castro, maestro, and the AMReX astrophysics suite. *Journal of Physics: Conference Series*, 1031:012024.

[Zuzio, 2011] Zuzio, D. (2011). Direct numerical simulation of two phase flows with adaptive mesh refinement. *PhD Thesis, University of Toulouse.*