A Framework for Mesh Refinement Suitable for Finite-Volume and Discontinuous-Galerkin Schemes With Application to Multiphase Flow Prediction

Andrée-Anne Dion-Dallaire

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
M.A.Sc. degree in Mechanical Engineering

Department of Mechanical Engineering
Faculty of Engineering
University of Ottawa

Abstract

Modelling multiphase flow, more specifically particle-laden flow, poses multiple challenges. These difficulties are heightened when the particles are differentiated by a set of "internal" variables, such as size or temperature. Traditional treatments of such flows can be classified in two main categories, Lagrangian and Eulerian methods. The former approaches are highly accurate but can also lead to extremely expensive computations and challenges to load balancing on parallel machines. In contrast, the Eulerian models offer the promise of less expensive computations but often introduce modelling artifacts and can become more complicated and expensive when a large number of internal variables are treated. Recently, a new model was proposed to treat such situations. It extends the ten-moment Gaussian model for viscous gases to the treatment of a dilute particle phase with an arbitrary number of internal variables. In its initial application, the only internal variable chosen for the particle phase was the particle diameter. This new polydisperse Gaussian model (PGM) comprises 15 equations, has an eigensystem that can be expressed in closed form and also possesses a convex entropy. Previously, this model has been tested in one dimension [8]. The PGM was developed with the detonation of radiological dispersal devices (RDD) as an immediate application. The detonation of RDDs poses many numerical challenges, namely the wide range of spatial and temporal scales as well as the high computational costs to accurately resolve solutions. In order to address these issues, the goal of this current project is to develop a block-based adaptive mesh refinement (AMR) implementation that can be used in conjunction with a parallel computer. Another goal of this project is to obtain the first three-dimensional results for the PGM. In this thesis, the kinetic theory of gases underlying the development of the PGM is studied. Different numerical schemes and adaptive mesh refinement methods are described. The new block-based adaptive mesh refinement algorithm is presented. Finally, results for different flow problems using the new AMR algorithm are shown, as well as the first three-dimensional results for the PGM.

Acknowledgements

First of all, I would like to thank my wonderful adviser, Professor James M^{c}Donald. He has provided my with countless advice and words of encouragements. He was always available for an often not-so-quick chat, during which he provided guidance and gave me great ideas. I could not have had a better mentor for this project.

Along with my adviser, I would like to thank my thesis committee: Professor Catherine Mavriplis from the University of Ottawa and Professor Edgar A. Matida from Carleton University for their insightful comments.

My gratitude is also extended to Doctor Lucian Ivan from the Canadian Nuclear Laboratories, who provided me with great help in my research.

Finally, my sincere thanks goes to my fellow graduate colleagues, without whom the time spent at the University would have been less than pleasant, and to my friends and family for their endless support and encouragements.

Funding for the research presented in this thesis has been provided by the Brunsfield International Group and the Ontario Ministry of Training, Colleges and Universities via an Ontario Graduate Scholarship.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Multiphase flows can be used to model a variety of engineering and scientific situations, such as internal combustion engines, pollutant modelling and drug delivery. The type of multiphase flow of interest in this work comprises a disperse phase made up of relatively small, immiscible, solid particles that are suspended in a continuously connected carrier phase. The immediate application of interest is the dispersal of particles resulting from the detonation of a radiological dispersal device (RDD)[13], as this work was done in collaboration with the Canadian Nuclear Laboratories. However, the techniques described in this thesis are implemented within a general framework and can be used for a variety of applications.

Usually, an RDD comprises an explosive charge and a radioactive source. If such a device detonates, radioactive material would be dispersed through material fragmentation and thermal-gas-dynamics processes. The range of particle size and initial acceleration are determined through the explosive detonation, blast-wave propagation and fireball development, all of which happen on an extremely short time scale. However, following these events, the radioactive particles spread out over large distances on a time scale of minutes to hours, depending on meteorological conditions and terrain topography. These characteristics pose numerical challenges.

Traditional modelling of such flows can be classified in two main categories, La-

grangian and Eulerian methods. The former approaches are highly accurate but can also lead to extremely expensive computations and challenges to load balance on parallel machines. In contrast, the Eulerian models offer less expensive computations but often introduce modelling artifacts and can become more complicated and expensive when a large number of internal variables are treated. Recently, a new model, the Polydisperse Gaussian model (PGM), was proposed to treat such situations [8]. It takes its roots from the kinetic theory of gases, using a probabilistic treatment for particle behaviour and a technique called moment closures [11]. This method leads to a closed set of hyperbolic partial differential equations that describe the evolution of the statistics of particle velocities. The capabilities of the model were previously demonstrated in one dimension.

In order to address the numerical challenges, various techniques of adaptive mesh refinement (AMR) have been developed. These refinement techniques are usually classified as cell-based, patch-based and block-based. Berger [2, 1] initially developed cell-based AMR. In this method, cells that need refinement are first flagged and then refined individually, leading to the fewest number of total cells. Patch-based adaptive mesh refinement was then developed by Berger and Collela [3]. In this technique, one or more patches of refined cells are applied over the cells in need of refinement. Finally, block-based adaptive refinement was developed by Berger [4] and Quirk [19]. The initial grid is considered to be a parent block and sections of the parent block are replaced by more refined child blocks, the latter having the same number of cells as the parent. Although it often leads to over-refined sections of the grid, this is addressed by distributing the blocks across a parallel network. Hierarchical trees are often used to keep track of the connections between blocks as well as refinement events. However, these types of structures have some major drawbacks as described in Section 5.3.

1.2 Motivation

As mentioned above, the PGM has, so far, been tested in one dimension. One goal of this work is to implement, for the first time, a three-dimensional version of the PGM. Being able to visualize the dispersal of particles, especially in the case of the detonation of an RDD, is important, as it allows the modelling of urban environments. Furthermore, using this model in conjunction with a specialized discontinuous-Galerkin-Hancock scheme is another goal, as this scheme produces third-order accurate solutions and has the promise of scaling extremely well on parallel machines.

In order to efficiently and quickly obtain accurate results, a robust parallel framework as well as an efficient adaptive mesh refinement algorithm are needed. One goal of the work is to produce a block-based adaptive mesh refinement algorithm that does not use a hierarchical tree structure to store the block connectivity, as such trees can lead to issues when coarsening.

1.3 Contributions

This work contains two main contributions. Firstly, the PGM has been implemented with a three-dimensional solver. It is shown that the three-dimensional results obtained agree with the one-dimensional results previously obtained and greatly improves the solution accuracy. The model is used with a first-order accurate scheme.

Secondly, an efficient parallel implementation of a block-based adaptive mesh refinement algorithm that does not use a data tree structure is developed. This is done by implementing a C++ object that keeps track of the connections and is integral to the boundaries. Each of the boundaries that connects to another block, called a neighbour block, has one of these objects. The latter store information about the corresponding boundary faces on the neighbour block. Furthermore, the objects store Gaussian quadrature points and weights, that are used to compute fluxes through the boundaries.

1.4 Research Outline

The content of this thesis is structured as follows. Chapter 2 describes the multiphase flows used in the first part of this research. Chapter 3 introduces the distribution function used in kinetic theory of gases, as well as the moment closure techniques employed. In Section 3.3.3, an overview of the polydisperse Gaussian model is provided. Next, in Chapter 4, the finite-volume scheme as well as the discontinuous-Galerkin scheme used in the work are presented. Then, Chapter 5 provides an overview of the various adaptive mesh refinement schemes that have been developed, as well as some of their shortcomings. Chapter 6 extensively describes the three-dimensional block-based adaptive mesh refinement implementation developed as part of this thesis, as well as how the block connectivity is handled. Chapter 7 presents several sample problems of computational fluid dynamics as well as polydisperse multiphase problems, solved using a first-order finite-volume scheme and a third-order discontinuous-Galerkin scheme. Finally, Chapter 8 presents conclusions drawn from this research as well as some suggestions for future work.

Chapter 2

Multiphase Flow

Multiphase flows can be found in various situations, such as internal combustion engines, pollutant modelling and drug delivery. They are defined as flows exhibiting two or more thermodynamic phases. This research focuses on particle-laden flows, which are a subset of multiphase flows. This type of flow is made up of two phases, the particle phase and the carrier phase. The particle phase, also known as the dispersed phase, comprises small dilute particles that are un-mixable. The carrier phase, also known as the continuous phase, is assumed to be continuously connected. Particle-laden flows can either be monodisperse or polydisperse. In the former, the particles all have the same size, whereas in the latter, the particles have different sizes. The particles can be further differentiated by a set of "internal" variables, such as temperature or shape.

Particle-laden flows can be characterized by three different regimes, *i.e.* the continuum regime, the transition regime and the rarefied regime. The Knudsen number is a non-dimensional number that characterizes the regime of the flow. The Knudsen number, Kn, is defined as

$$\mathrm{Kn} = \frac{\lambda}{l}, \tag{2.1}$$

where λ is the mean-free path of the particles and l is a characteristic length of the problem. The mean-free path is defined as the average distance travelled by a particle between two collisions. In the continuum regime, the Knudsen number is small, $\mathrm{Kn} < 0.01$, meaning collisions between particles are numerous. Traditionally,

flows in this regime have been successfully described using continuous fields. In the transition regime, $0.01 < \mathrm{Kn} < 10$, the evolution of individual particles becomes increasingly important. Finally, in the rarefied regime, the Knudsen number is large, $\mathrm{Kn} > 10$, meaning the mean-free path is long relative to the problem and the collisions are few. This research focuses on particle-laden flows in rarefied regimes, wherein the particle collisions are so scarce that they can be ignored safely.

The behaviour of the particles, or how much the drag affects the particles, can be described using another non-dimensional number, the Stokes number, St, defined as

$$\mathrm{St} = \frac{\tau |V_i|}{l}, \tag{2.2}$$

where τ is the relaxation time due to drag forces, $|V_i|$ is the velocity of the background flow and l is a characteristic length of the particle. The relaxation time is defined as the rate at which the velocity of the particles settles to the velocity of the background fluid due to drag. For extremely low Reynolds number, the relaxation time, τ, is found to be

$$\tau = \frac{\rho_p d^2}{18 \mu_f}, \tag{2.3}$$

where ρ_p is the density of the material making up the particles, d is the particle diameter and μ_g is the dynamic viscosity of the background fluid. When the Stokes number is small, $\mathrm{St} \ll 1$, the drag caused by the carrier phase has a high impact on the particles, their velocity quickly settling to the velocity of the continuous phase. When the Stokes number is large, $\mathrm{St} \gg 1$, the drag caused by the carrier phase has little effect on the particles, their velocity slowly settles to the velocity of the continuous phase. In the middle, $\mathrm{St} \approx 1$, the particles are mildly affected by the drag of the background fluid.

In this thesis, the particles and the Reynolds number are assumed to be small enough to apply Stokes Law, defining the drag force between the particles and the background flow. The particles are also affected by the gravity and buoyancy forces.

The acceleration of the particles can therefore be described as

$$a_i = \frac{V_i - v_i}{\tau} + \phi_i, \tag{2.4}$$

where V_i is the background flow velocity, v_i is the particle velocity, τ is the relaxation time due to drag, defined in Eq. (2.3) and ϕ_i combines gravitational and buoyant effects and is defined as

$$\phi_i = \frac{\rho_p - \rho_f}{\rho_p} g_i, \tag{2.5}$$

where ρ_p is the density of the material making up the particle, ρ_f is the density of the carrier phase and g_i is the gravitational acceleration.

The particle phase of particle-laden flows can be modelled using either Lagrangian or Eulerian methods. Lagrangian methods are highly accurate, as they rely on the tracking of every single particle in the simulation. However, such methods can lead to extremely expensive computations when a large number of particles is treated and challenges in the computational load balancing on parallel machines [23]. Eulerian methods are computationally more affordable, but often introduce artifacts in the solution and become more complicated when a large number of internal variables is treated. Recently, a new model stemming from the kinetic theory of gases was proposed to treat such situations [8]. This model is further discussed in Section 3.3.3.

Chapter 3

Kinetic Theory of Gases and Multiphase Flows

The situation of a monodisperse particulate flow can be seen as similar to that of a gas made up of many identical atoms or molecules. It is therefore expected that powerful methods developed for gas flows could be adapted for multiphase flows.

Tracking every single particle in a gas flow system, even if the system is quite small, is beyond the capacity of any modern computer. Therefore, in order to overcome this issue, the kinetic theory of gases was developed. The kinetic theory of gases is usually used in the rarefied regime, where the mean-free path is much larger than the lengths of interest. In this field, a large number of particles constantly in motion describes gases and statistics of the particles as a group are treated, as opposed to each particle individually. The classical kinetic theory of gases is based on the following assumptions:

- Matter is composed of discrete particles.
- Particles of a pure gas are identical.
- Particles are points with no internal structure.
- Forces exerted by particles have influence on other particles only over a spherical domain, which is very small compared to the mean-free path.

- Only binary collisions happen, meaning that only two particles interact through attractive or repulsive force fields.
- Quantum effects are neglected.
- Colliding particles are statistically uncorrelated.

3.1 The Distribution Function

In this theory, particles are treated using a distribution function describing the behaviour of the particles with a probabilistic approach. This distribution function, denoted $\mathcal{F}(x_i, v_i, t)$, represents the number of particles in an infinitesimal cube between the points x_i and $x_i + \mathrm{d}x_i$, that have velocities between v_i and $v_i + \mathrm{d}v_i$, at time t. This distribution function exists in six dimensions, three space dimensions and three velocity dimensions, and traditionally does not take into account the fact that the particles may have different sizes.

3.1.1 Moments of the Distribution Function

Instead of needing the total of information contained in the distribution function, macroscopic, or "observable", properties are often desired. These macroscopic properties are usually velocity moments of the distribution function and are obtained by multiplying the distribution function by a velocity-dependent weight and integrating it over all velocity space. The zeroth moment of $\mathcal{F}$ yields the zeroth-order moment, or number density, denoted n, and is defined as

$$n = \iiint_{\infty} W(v_i)\mathcal{F}(x_i, v_i, t)\mathrm{d}v_i = \langle \mathcal{F} \rangle \,, \tag{3.1}$$

where the velocity-dependent weight, $W(v_i)$, is chosen as one and the angled brackets denote the integration over all velocity space. Multiplying the number density by the particle mass yields the mass density of the particle phase,

$$\rho = \langle m\mathcal{F} \rangle \,. \tag{3.2}$$

The first-order moment, or momentum density of the particles, denoted ρu_i, is obtained by taking the integration weight to be the particle momentum, mv_i, multiplying it by the distribution function and integrating the resulting expression over all velocity space, as

$$\rho u_i = \iiint_{\infty} W(v_i)\mathcal{F}(x_i, v_i, t)\mathrm{d}v_i = \langle mv_i\mathcal{F}\rangle \,. \tag{3.3}$$

where u_i is the average particle velocity. The average particle velocity can then be obtained by dividing Eq. (3.3) by the mass density of the particle phase, as

$$u_i = \frac{\langle mv_i\mathcal{F}\rangle}{\langle m\mathcal{F}\rangle}. \tag{3.4}$$

The average velocity is then used to define the random, or peculiar, velocity, as the difference between a particular particle's velocity and the average velocity,

$$c_i = v_i - u_i, \tag{3.5}$$

where c_i is the random velocity [10].

The second-order moment, or kinetic energy density of the particles, E, is obtained by taking the integration weight to be $\frac{1}{2}mv_iv_i$, multiplying it by the distribution function and integrating the resulting expression over all velocity space, as

$$E = \iiint_{\infty} W(v_i)\mathcal{F}(x_i, v_i, t)\mathrm{d}v_i = \left\langle \frac{1}{2}mv_iv_i\mathcal{F}\right\rangle . \tag{3.6}$$

Developing Eq. (3.6) by using the expression for the random velocity, Eq. (3.5), illustrates the two contributions of a system to the kinetic energy density of the

particles, as

$$\begin{aligned}
E &= \left\langle \frac{mv_iv_i}{2}\mathcal{F} \right\rangle \\
&= \left\langle \frac{m}{2}\left(u_i + c_i\right)\left(u_i + c_i\right)\mathcal{F} \right\rangle \\
&= \left\langle \frac{m}{2}\left(u_iu_i + 2u_ic_i + c_ic_i\right)\mathcal{F} \right\rangle \\
&= \frac{u_iu_i}{2}\left\langle m\mathcal{F}\right\rangle + u_i\left\langle mc_i\mathcal{F}\right\rangle + \left\langle \frac{m}{2}c_ic_i\mathcal{F} \right\rangle \\
&= \frac{\rho u_iu_i}{2} + u_i\left\langle m\left(v_i - u_i\right)\mathcal{F}\right\rangle + \left\langle \frac{m}{2}c_ic_i\mathcal{F} \right\rangle \\
&= \frac{\rho u_iu_i}{2} + u_i\left(\left\langle mv_i\mathcal{F}\right\rangle - \left\langle mu_i\mathcal{F}\right\rangle\right) + \left\langle \frac{m}{2}c_ic_i\mathcal{F} \right\rangle \\
&= \frac{\rho u_iu_i}{2} + u_i\left(\rho u_i - \rho u_i\right) + \left\langle \frac{m}{2}c_ic_i\mathcal{F} \right\rangle \\
&= \underbrace{\frac{\rho u_iu_i}{2}}_{\text{A}} + \underbrace{\left\langle \frac{m}{2}c_ic_i\mathcal{F} \right\rangle}_{\text{B}}.
\end{aligned} \tag{3.7}$$

Term A represents the macroscopic kinetic energy and term B represents the internal (thermal) energy density.

Additional higher-order moments of the distribution function can be obtained by using different integration weights and are defined as

$$\begin{aligned}
U_{ij} &= \left\langle mv_iv_j\mathcal{F}\right\rangle , & P_{ij} &= \left\langle mc_ic_j\mathcal{F}\right\rangle , \\
U_{ijk} &= \left\langle mv_iv_jv_k\mathcal{F}\right\rangle , & Q_{ijk} &= \left\langle mc_ic_jc_k\mathcal{F}\right\rangle , \\
U_{ijkl} &= \left\langle mv_iv_jv_kv_l\mathcal{F}\right\rangle , & R_{ijkl} &= \left\langle mc_ic_jc_kc_l\mathcal{F}\right\rangle , \\
U_{ijklm} &= \left\langle mv_iv_jv_kv_lv_m\mathcal{F}\right\rangle , & S_{ijklm} &= \left\langle mc_ic_jc_kc_lc_m\mathcal{F}\right\rangle .
\end{aligned} \tag{3.8}$$

This can continue and moments of arbitrary high order can be computed, however, their physical significance quickly becomes less obvious.

3.1.2 The Maxwell-Boltzmann Distribution

The Maxwell-Boltzmann distribution describes a gas in local thermal equilibrium. In other words, it describes a gas left to settle at rest with no external influences. The following assumptions were made to develop the Maxwell-Boltzmann distribution

[17]:

- The gas is in equilibrium.
- The statistics are isotropic.
- The orthogonal components of the velocities of the particles are independent of each other.

The resulting distribution function, also called Maxwellian $\mathcal{M}$, is

$$\mathcal{M} = n\left(\frac{\rho}{2\pi p}\right)^{\frac{3}{2}} \exp\left(-\frac{\rho v_i v_i}{2p}\right) = n\left(\frac{m}{2\pi kT}\right)^{\frac{3}{2}} \exp\left(-\frac{m v_i v_i}{2kT}\right), \tag{3.9}$$

where p is the thermodynamic pressure, $k = 1.38065 \times 10^{-23}$J/K is the Boltzmann constant and T is the temperature.

3.2 The Boltzmann Equation

The evolution of the distribution function is described by the Boltzmann equation,

$$\underbrace{\frac{\partial \mathcal{F}}{\partial t} + v_i \frac{\partial \mathcal{F}}{\partial x_i} + \frac{\partial}{\partial v_i}(a_i \mathcal{F})}_{\text{A}} = \underbrace{\frac{\delta \mathcal{F}}{\delta t}}_{\text{B}}, \tag{3.10}$$

where a_i is the acceleration of particles due to external forces, term A describes the particles moving through phase space and term B is the collision operator. The collision operator is assumed to be zero for the current research, $\frac{\delta \mathcal{F}}{\delta t} = 0$, as the particle phase is assumed to be dispersed. Consequently, Eq. (3.10) becomes the collisionless kinetic equation,

$$\frac{\partial \mathcal{F}}{\partial t} + v_i \frac{\partial \mathcal{F}}{\partial x_i} + \frac{\partial}{\partial v_i}(a_i \mathcal{F}) = 0. \tag{3.11}$$

Eq. (3.11) is mathematically simple, but high dimensional, as it has three space dimensions, three velocity dimensions and one time dimension. This makes its numerical solution too costly for any particle problem.

3.3 Moment Methods

Several techniques have been developed to solve the Boltzmann equation. The first technique is molecular dynamics. It involves directly solving for the evolution of each particle. However, it is extremely computationally expensive. The second group of approaches are the Monte Carlo methods, the most popular of which is the direct simulation Monte Carlo (DSMC) method [5]. It involves randomly choosing a subset of particles and evolving them based on statistical approximations. While Monte Carlo methods are simple to implement, they are very slow to converge and are inherently $\frac{1}{2}$ order accurate. The third technique is the direct discretization of the Boltzmann equation. It involves discretizing the three-dimensional position space and the three-dimensional velocity space directly. However, this approach is very computationally expensive due to the high dimensionality of the Boltzmann equation. The fourth technique is the Chapman-Enskog method [6]. It involves extending the Navier-Stokes equations by adding higher-order terms in an effort to extend their validity beyond continuum flows. Such methods have had limited success, as the resulting PDEs are linearly unstable and ill-posed. Finally, there are the moment methods. These involve trading the high-dimensionality of the Boltzmann equation for an expanded solution vector in three dimensions, which always yields first-order hyperbolic balance laws.

The Boltzmann equation provides a large amount of information, most of which is not useful. The important properties are usually macroscopic variables, obtained by taking moments of the distribution function, as described in Section 3.1. Evolution laws of these moments are obtained by taking moments of Eq. (3.11), which leads to

$$\left\langle mW\left(\frac{\partial \mathcal{F}}{\partial t}+v_i\frac{\partial \mathcal{F}}{\partial x_i}+\frac{\partial}{\partial v_i}(a_i\mathcal{F})\right)\right\rangle = 0$$

$$\left\langle mW\frac{\partial \mathcal{F}}{\partial t}\right\rangle + \left\langle mWv_i\frac{\partial \mathcal{F}}{\partial x_i}\right\rangle + \left\langle mW\frac{\partial}{\partial v_i}(a_i\mathcal{F})\right\rangle = 0.$$

Because v_i and t are independent and x_i and v_i are also independent, the above

equation can be simplified to the following equation,

$$\frac{\partial}{\partial t}\left\langle mW\mathcal{F}\right\rangle + \frac{\partial}{\partial x_i}\left\langle mWv_i\mathcal{F}\right\rangle + \underbrace{\left\langle mW\frac{\partial}{\partial v_i}(a_i\mathcal{F})\right\rangle}_{\text{acceleration term}} = 0. \tag{3.12}$$

This is known as the collisionless Maxwell's equation of change. Using the product rule, the acceleration term can be simplified to

$$\left\langle mW\frac{\partial}{\partial v_i}(a_i\mathcal{F})\right\rangle = \left\langle \frac{\partial}{\partial v_i}\left(mWa_i\mathcal{F}\right)\right\rangle - \left\langle ma_i\mathcal{F}\frac{\partial W}{\partial v_i}\right\rangle. \tag{3.13}$$

As the velocity $v_i \to \infty$, the distribution function $\mathcal{F} \to 0$. Therefore, the first term on the right hand-side of Eq. (3.13) can be neglected.

3.3.1 Moment Closure

Unfortunately, Eq. (3.12) does not naturally lead to a closed system of equations. The time evolution of a moment always depends on the spatial divergence of a moment that is one order higher. An infinite number of moments would be required to describe the evolution of the system fully.

In order to obtain a closed system, the form of the distribution function is prescribed in terms of a finite number of free coefficients, $\boldsymbol{\alpha}$. These coefficients are chosen so that the distribution function is consistent with a finite set of moments of interest, and satisfy the moments of Eqs. (3.1), (3.4), (3.6) and (3.8). The number of coefficients used is equal to the number of moments of interest. The moments of interest are collected in a solution vector, $\boldsymbol{U}$, such that

$$\boldsymbol{U} = \left\langle m\boldsymbol{W}\mathcal{F}\right\rangle, \tag{3.14}$$

where $\boldsymbol{W}$ is a vector of velocity weights. Once the exact form of the distribution function is specified, the higher-order moments needed to close the system can then be integrated, thus, closing the system.

3.3.2 The Grad Moment Hierarchy

In 1949, Harold Grad proposed the first moment closure technique [11]. His idea was to write a general non-equilibrium distribution function as a Hermite expansion around equilibrium,

$$\mathcal{F}_{Grad} = \mathcal{M}(\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{H}), \tag{3.15}$$

where $\mathcal{M}$ is a Maxwellian, $\boldsymbol{\alpha}$ are coefficients chosen to satisfy the moment relations and $\boldsymbol{H}$ is a vector containing Hermite polynomials. The Hermite polynomials form an orthogonal basis for the function space with $\mathcal{M}$ as a weight. The approximation of $\mathcal{F}_{Grad}$ is formed by truncating the coefficients at a chosen order.

The resulting partial differential equations are a closed-form set of first-order balance laws. In order for the system to be well-posed, the eigenvalues of the flux Jacobian must be real. However, in this specific moment closure, this is not always the case, and the system can become ill-posed for physically realistic cases. For many years, this limited the further development of the technique of the moment closure.

3.3.3 Maximum-Entropy Moment Closures

As an alternative, a moment closure hierarchy that approximates the distribution function such that the entropy is maximized while still ensuring $\mathcal{F}$ is consistent with the known moments was developed [15]. The resulting distribution function is the least biased distribution, as the maximum-entropy state is the most likely state under the moment constraints. If a maximum-entropy state exists for a moment vector $\boldsymbol{U}$, the resulting partial differential equations can be proven to be hyperbolic and well-posed. All distribution functions in this hierarchy are positive and have the form

$$\mathcal{F}_{M.E.} = \exp\left(-\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{W}\right), \tag{3.16}$$

where $\boldsymbol{\alpha}$ is the vector of free parameters and $\boldsymbol{W}$ is the vector of generating weights.

The first member of the maximum-entropy closure hierarchy is the 5-moment closure, also called the Euler closure. The generating weights vector $\boldsymbol{W} = [1, v_i, \frac{1}{2}v_i v_i]^{\mathrm{T}}$

results in the following distribution function

$$\mathcal{F}_E = \frac{\rho}{m}\left(\frac{\rho}{2\pi p}\right)^{\frac{3}{2}} \exp\left(-\frac{\rho}{2p} c_i c_i\right). \tag{3.17}$$

The distribution function has the shape of a normal distribution function in all three directions. The five moment equations generated by the distribution function of Eq. (3.17) are

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i) = 0, \tag{3.18}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}(\rho u_i u_j + \delta_{ij} p) = 0, \tag{3.19}$$

$$\frac{\partial}{\partial t}\left(\frac{3p}{2} + \frac{\rho u_i u_i}{2}\right) + \frac{\partial}{\partial x_j} u_j \left(\frac{5p}{2} + \frac{\rho u_i u_i}{2}\right) = 0, \tag{3.20}$$

where δ_{ij} is the Kronecker delta. The left-hand side terms of Eqs. (3.18), (3.19) and (3.20) are the same as the terms in the traditional Euler equations. Eq. (3.18) represents the continuity equation and has a total of one equation, Eq. (3.19) represents the momentum equation and has a total of three equations and Eq. (3.20) represents the energy equation and has a total of one equation.

The second member of the maximum-entropy closure hierarchy is the 10-moment closure, also called the Gaussian closure [16]. The generating weights vector $\boldsymbol{W} = [1, v_i, \frac{1}{2} v_i v_j]^{\mathrm{T}}$ results in the following distribution function

$$\mathcal{F}_G = \frac{\rho}{m(2\pi)^{\frac{3}{2}} (\det \Theta_{ij})^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\Theta_{ij}^{-1} c_i c_j\right), \tag{3.21}$$

where ρ is the mass density, m is the particle mass, c_i is the random velocity, as shown in Eq. (3.5), $\Theta_{ij} = \frac{P_{ij}}{\rho}$ and P_{ij} is the anisotropic pressure tensor. The distribution function is, again, a normal distribution function, but it is stretched in one direction. The ten moment equations generated by the distribution function of Eq. (3.21) are

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_k}(\rho u_k) = 0, \tag{3.22}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_k}(\rho u_i u_k + P_{ik}) = 0, \tag{3.23}$$

$$\frac{\partial}{\partial t}\left(\rho u_i u_j + P_{ij}\right) + \frac{\partial}{\partial x_K}\left(\rho u_i u_j u_k + u_i P_{jk} + u_j P_{ik} + u_k P_{ij}\right) = 0, \tag{3.24}$$

where P_{ij} is the anisotropic pressure tensor. Eq. (3.22) represents the continuity equation and has a total of one equation, Eq. (3.23) represents the momentum equation and has a total of three equations and Eq. (3.24) has a total of six equations for the anisotropic pressure tensor.

There is also the 14-moment closure, where the generating weights vector is $\boldsymbol{W} = [1, v_i, v_i v_j, v_i v_j v_j, v_i v_i v_j v_j]^{\mathrm{T}}$. It is the simplest member of the maximum-entropy moment closure hierarchy that treats heat transfer [18]. This closure is more difficult, as the resulting distribution function cannot be integrated in closed-form. Though a closed-form approximation is available for gases, work on this closure is ongoing and its possible application to multiphase flow is beyond the scope of this thesis.

Extension to Multiphase Flows

The maximum-entropy moment closures described in Section 3.3.3 can be also be used for multiphase flows. In particular, for Stokes flow, using Eq. (2.4), the acceleration term becomes

$$\left\langle mW \frac{\partial}{\partial v_i}(a_i \mathcal{F}) \right\rangle = \frac{1}{\tau} \left(V_i \left\langle m\mathcal{F} \frac{\partial W}{\partial v_i} \right\rangle - \left\langle m v_i \mathcal{F} \frac{\partial W}{\partial v_i} \right\rangle \right), \tag{3.25}$$

where V_i is the velocity of the background flow. Therefore, the Maxwell equation of change with Stokes drag as the acceleration term becomes

$$\frac{\partial}{\partial t} \langle mW\mathcal{F} \rangle + \frac{\partial}{\partial x_i} \langle mW v_i \mathcal{F} \rangle = -\frac{1}{\tau} \left(V_i \left\langle m\mathcal{F} \frac{\partial W}{\partial v_i} \right\rangle - \left\langle m v_i \mathcal{F} \frac{\partial W}{\partial v_i} \right\rangle \right). \tag{3.26}$$

The ten moments equations generated by Eq. (3.21) in the case of multiphase flows are thus

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_k}(\rho u_k) = 0, \tag{3.27}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_k}(\rho u_i u_k + P_{ik}) = \frac{\rho}{\tau}(V_k - u_k), \tag{3.28}$$

$$\frac{\partial}{\partial t}(\rho u_i u_j + P_{ij}) + \frac{\partial}{\partial x_k}(\rho u_i u_j u_k + u_i P_{jk} + u_j Pik + u_k P_{ij}) = \frac{\rho u_i (V_j - u_j) + \rho u_j (V_i - u_i) - 2P_{ij}}{\tau}. \tag{3.29}$$

Polydisperse Gaussian Model

In order to solve polydisperse flows, as is the case when a radiological dispersal device (RDD) explodes, the traditional Gaussian closure, as described in Section 3.3.3, was previously extended to include information about the size of the particles [8]. The new distribution function, denoted $\mathcal{F}(x_i, v_i, t, d)$, represents the number of particles in an infinitesimal cube between the points x_i and $x_i + \mathrm{d}x_i$, that have velocities between v_i and $v_i + \mathrm{d}v_i$, and have diameters between d and $d + \mathrm{d}d$, at time t. As the size of the particles is assumed to stay constant during the evolution of the system and as collisions are neglected, Eq. (3.11) still applies to the evolution of the current system. However, the acceleration is now a function of the size of each individual particle.

As has been previously shown, in the event of a radiological dispersal device explosion, the size of the particles is well approximated by a log-normal distribution [12]. With this information in mind, the solution vector $\boldsymbol{U}$ of the traditional Gaussian closure is augmented with additional moments carrying information about the local average logarithm of the diameter of particles, μ, the local variance of the logarithm of the diameter of particles, Ψ_{dd}, and the covariance of the logarithm of the diameter of particles with each component of the velocity of the particle, Ψ_{id}. The resulting distribution function has a very similar form to Eq. (3.21) and is expressed as

$$\mathcal{F}_{PGM} = \frac{n}{d(2\pi)^2(\det \Psi_{ij})^{\frac{1}{2}}} \exp\left(-\frac{1}{2} \det \Psi_{ij}^{-1} \tilde{c}_i \tilde{c}_j\right), \tag{3.30}$$

where $\tilde{c}_i$ is the random velocity vector augmented with the difference between the natural logarithm of the particle diameter and the local average logarithm of the diameter of particles. It is defined as

$$\tilde{c}_i = \begin{bmatrix} v_x - u_x \\ v_y - u_y \\ v_z - u_z \\ \ln(d) - \mu \end{bmatrix}, \tag{3.31}$$

and Ψ_{ij} is a second-order tensor defined as

$$\Psi_{ij} = \begin{bmatrix} \Theta_{xx} & \Theta_{xy} & \Theta_{xz} & \Psi_{xd} \\ \Theta_{xy} & \Theta_{yy} & \Theta_{yz} & \Psi_{yd} \\ \Theta_{xz} & \Theta_{yz} & \Theta_{zz} & \Psi_{zd} \\ \Psi_{xd} & \Psi_{yd} & \Psi_{zd} & \Psi_{dd} \end{bmatrix}. \tag{3.32}$$

The covariance between the logarithm of the diameter of particles and each component of the velocity of the particle is expressed as

$$\Psi_{id} = \frac{1}{n} \left\langle (v_i - u_i)(\ln d - \mu)\mathcal{F}) \right\rangle, \tag{3.33}$$

and gives information as to whether particles are more or less likely to have larger velocities in a given direction. The local variance of the logarithm of the diameter of particles is expressed as

$$\Psi_{dd} = \frac{1}{n} \left\langle (\ln d - \mu)^2 \mathcal{F}) \right\rangle. \tag{3.34}$$

The extension of the Gaussian closure has the same 10 equations as Eqs. (3.22), (3.23) and (3.24), with an additional five equations that account for statistics about the size of the particles. The model is extended by tracking the statistics describing the logarithm of the particle diameter. The distribution function of Eq. (3.30) is used along with the generating weights vector is $\boldsymbol{W} = [1, v_i, v_i v_j, \ln d, \ln d v_i, (\ln d)^2]^{\mathrm{T}}$. The new system of 15 first-order hyperbolic partial differential equations has the form

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{F}_x}{\partial x} + \frac{\partial \boldsymbol{F}_y}{\partial y} + \frac{\partial \boldsymbol{F}_z}{\partial z} = \boldsymbol{S}_1 + \boldsymbol{S}_2, \tag{3.35}$$

where the solution vector $\boldsymbol{U}$ can be expressed as

$$\boldsymbol{U} = n \begin{bmatrix} 1 \\ u_x \\ u_y \\ u_z \\ (u_x^2 + \Theta_{xx}) \\ (u_x u_y + \Theta_{xy}) \\ (u_x u_z + \Theta_{xz}) \\ (u_y^2 + \Theta_{yy}) \\ (u_y u_z + \Theta_{yz}) \\ (u_z^2 + \Theta_{zz}) \\ \mu \\ (\mu u_x + \Psi_{xd}) \\ (\mu u_y + \Psi_{yd}) \\ (\mu u_z + \Psi_{zd}) \\ (\mu^2 + \Psi_{dd}) \end{bmatrix} \tag{3.36}$$

and the flux in the x-direction, $\boldsymbol{F}_x$ can be expressed as

$$\boldsymbol{F}_x = n \begin{bmatrix} u_x \\ (u_x^2 + \Theta_{xx}) \\ (u_x u_y + \Theta_{xy}) \\ (u_x u_z + \Theta_{xz}) \\ (u_x^3 + 3u_x \Theta_{xx}) \\ (u_x^2 u_y + 2u_x \Theta_{xy} + u_y \Theta_{xx}) \\ (u_x^2 u_z + 2u_x \Theta_{xz} + u_z \Theta_{xx}) \\ (u_x u_y^2 + u_x \Theta_{yy} + 2u_y \Theta_{xy}) \\ (u_x u_y u_z + u_x \Theta_{yz} + u_y \Theta_{xz} + u_z \Theta_{xy}) \\ (u_x u_z^2 + u_x \Theta_{zz} + 2u_z \Theta_{xz}) \\ (\mu u_x + \Psi_{xd}) \\ (\mu u_x^2 + 2u_x \Psi_{xd} + \mu \Theta_{xx}) \\ (\mu u_x u_y + u_x \Psi_{yd} + u_y \Psi_{xd} + \mu \Theta_{xy}) \\ (\mu u_x u_z + u_x \Psi_{zd} + u_z \Psi_{xd} + \mu \Theta_{xz}) \\ (\mu^2 u_x + 2\mu \Psi_{xd} + u_x \Psi_{dd}) \end{bmatrix} \tag{3.37}$$

The fluxes in the y and z-directions are symmetrical to Eq. (3.37). The source

vector that takes into account Stokes drag, $\boldsymbol{S}_1$ can be expressed as

$$\boldsymbol{S}_1 = \frac{n}{\tau_G}\begin{bmatrix} 0 \\ V_x - (u_x - 2\Psi_{xd}) \\ V_y - (u_y - 2\Psi_{yd}) \\ V_z - (u_z - 2\Psi_{zd}) \\ 2\left(V_x(u_x - 2\Psi_{xd}) - (u_x^2 - 4u_x\Psi_{xd} + 4\Psi_{xd}^2 + \Theta_{xx})\right) \\ V_x(u_y - 2\Psi_{yd}) + V_y(u_x - 2\Psi_{xd}) - 2(u_xu_y - 2u_x\Psi_{yd} - 2u_y\Psi_{xd} + \\ \qquad 4\Psi_{xd}\Psi_{yd} + \Theta_{xy}) \\ V_x(u_z - 2\Psi_{zd}) + V_z(u_x - 2\Psi_{xd}) - 2(u_xu_z - 2u_x\Psi_{zd} - 2u_z\Psi_{xd} + \\ \qquad 4\Psi_{xd}\Psi_{zd} + \Theta_{xz}) \\ 2\left(V_y(u_y - 2\Psi_{yd}) - (u_y^2 - 4u_y\Psi_{yd} + 4\Psi_{yd}^2 + \Theta_{yy})\right) \\ V_y(u_z - 2\Psi_{zd}) + V_z(u_y - 2\Psi_{yd}) - 2(u_yu_z - 2u_y\Psi_{zd} - 2u_z\Psi_{yd} + \\ \qquad 4\Psi_{yd}\Psi_{zd} + \Theta_{yz}) \\ 2\left(V_z(u_z - 2\Psi_{zd}) - (u_z^2 - 4u_z\Psi_{zd} + 4\Psi_{zd}^2 + \Theta_{zz})\right) \\ 0 \\ V_x(\mu - 2\Psi_{dd}) - (\mu u_x - 2\mu\Psi_{xd} - 2u_x\Psi_{dd} + 4\Psi_{dd}\Psi_{xd} + \Psi_{xd}) \\ V_y(\mu - 2\Psi_{dd}) - (\mu u_y - 2\mu\Psi_{yd} - 2u_y\Psi_{dd} + 4\Psi_{dd}\Psi_{yd} + \Psi_{yd}) \\ V_z(\mu - 2\Psi_{dd}) - (\mu u_z - 2\mu\Psi_{zd} - 2u_z\Psi_{dd} + 4\Psi_{dd}\Psi_{zd} + \Psi_{zd}) \\ 0 \end{bmatrix} \tag{3.38}$$

and the source term that accounts for gravity and buoyancy forces can be expressed as

$$\boldsymbol{S}_2 = n\begin{bmatrix} 0 \\ \phi_x \\ \phi_y \\ \phi_z \\ 2u_x\phi_x \\ u_x\phi_y + u_y\phi_x \\ u_x\phi_z + u_z\phi_x \\ 2u_y\phi_y \\ u_y\phi_z + u_z\phi_y \\ 2u_z\phi_z \\ 0 \\ \mu\phi_x \\ \mu\phi_y \\ \mu\phi_z \\ 0 \end{bmatrix} \tag{3.39}$$

where

$$\tau_G = \frac{\rho_p}{18\mu_f}\exp\left(2\mu - 2\Psi_{dd}\right), \tag{3.40}$$

and ϕ_i is defined in Eq. (2.5).

The partial differential equations of the 15-equation system can also be expressed

in its primitive form as

$$\frac{\partial n}{\partial t} + u_k \frac{\partial n}{\partial x_k} + n \frac{\partial u_k}{\partial x_k} = 0 \tag{3.41}$$

$$\frac{\partial u_i}{\partial t} + u_k \frac{\partial u_i}{\partial x_k} + \frac{\Theta_{ik}}{n} \frac{\partial n}{\partial x_k} + \frac{\partial \Theta_{ik}}{\partial x_k} = \frac{1}{\tau_G} \left(V_i - (u_i - 2\Psi_{id}) \right) + \phi_i \tag{3.42}$$

$$\begin{aligned} &\frac{\partial \Theta_{ij}}{\partial t} + u_k \frac{\partial \Theta_{ij}}{\partial x_k} + \Theta_{jk} \frac{\partial u_i}{\partial x_k} + \Theta_{ik} \frac{\partial u_j}{\partial x_k} \\ &\qquad = \frac{1}{\tau_G} \left(-2V_i \Psi_{jd} - 2V_j \Psi_{id} + 2u_i \Psi_{jd} + 2u_j \Psi_{id} - 8\Psi_{id}\Psi_{jd} - 2\Theta_{ij} \right) \end{aligned} \tag{3.43}$$

$$\frac{\partial \mu}{\partial t} + u_k \frac{\partial \mu}{\partial x_k} + \frac{\Psi_{kd}}{n} \frac{\partial n}{\partial x_k} + \frac{\partial \Psi_{kd}}{\partial x_k} = 0 \tag{3.44}$$

$$\frac{\partial \Psi_{id}}{\partial t} + u_k \frac{\partial \Psi_{id}}{\partial x_k} + \Psi_{kd} \frac{\partial u_i}{\partial x_k} + \Theta_{ik} \frac{\partial \mu}{\partial x_k} = \frac{1}{\tau_G} \left(-2V_i \Psi_{dd} + 2u_i \Psi_{dd} - 4\Psi_{id}\Psi_{dd} - \Psi_{id} \right) \tag{3.45}$$

$$\frac{\partial \Psi_{dd}}{\partial t} + u_k \frac{\partial \Psi_{dd}}{\partial x_k} + 2\Psi_{kd} \frac{\partial \mu}{\partial x_k} = 0 \,. \tag{3.46}$$

Although this new model was not developed as part of this thesis, it had previously only been applied to simple one-dimensional problems. A major goal of this thesis is to produce the first multidimensional solutions to this model.

Chapter 4

Numerical Methods

In general, numerical methods are tools used to approximately solve problems when it is difficult or impossible to obtain exact solutions. These techniques are especially useful in fluid dynamics, when an approximation of a flow solution is desired. In this chapter, a finite-volume scheme and a discontinuous-Galerkin-Hancock scheme are described. Results using these two techniques are shown in Chapter 7.

4.1 Finite-Volume Scheme

The finite-volume method is a discretization scheme for a system of partial differential equations that describe the balance of one or more quantities. A mesh is built over the domain, the mesh being a collection of nodes, the nodes delimiting the cells. Each cell is a control volume, over which partial differential equations are integrated, resulting in balance equations. The set of balance equations is then discretized in time and space.

4.1.1 Domain Discretization

Although most of the work in this thesis is in three dimensions, the discretization of space and time will be illustrated in one dimension, for the sake of simplicity. As demonstrated in Figure 4.1, the domain of interest in the x-direction is $a < x < b$.

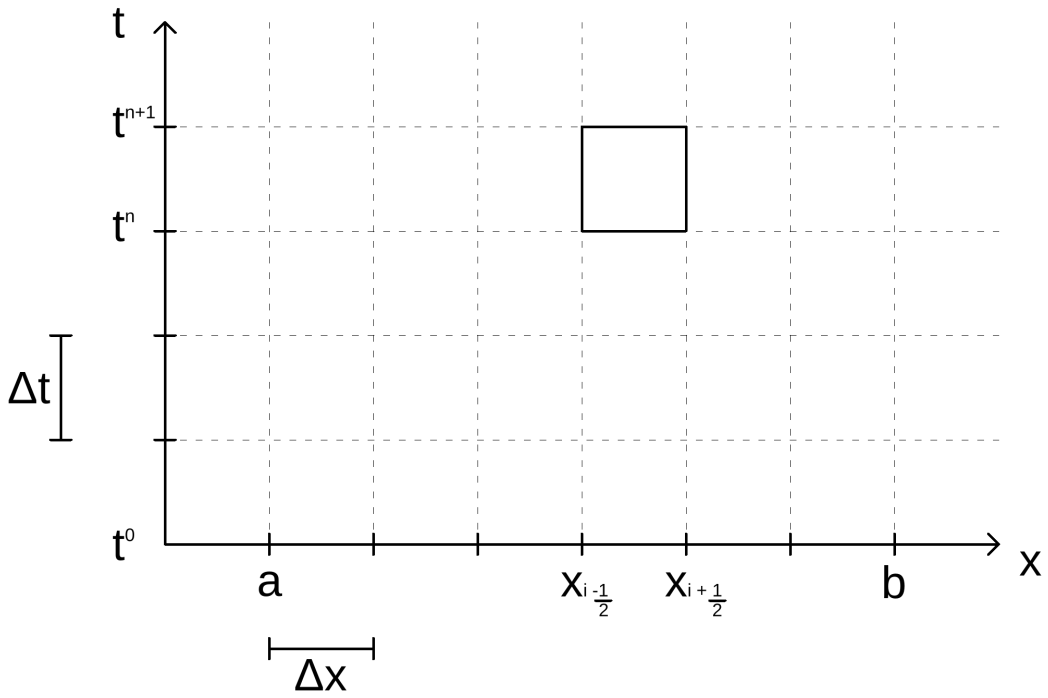


Figure 4.1: Finite-volume discretization of the space-time domain.

The solution vector typically contains the cell average value of various densities in each cell. The coordinates of the cell centroid, x_i, is given by

$$x_i = a + \left(i + \frac{1}{2}\right)\Delta x, \tag{4.1}$$

where i is the index of the cell, Δx is the distance between two nodes in the x-direction and a is the origin of the domain. In three dimensions, the coordinates of the centroid of a cell, x_i, x_j and x_k are given by

$$\begin{bmatrix} x_i \\ x_j \\ x_k \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + \left(\begin{bmatrix} i + \frac{1}{2} \\ j + \frac{1}{2} \\ k + \frac{1}{2} \end{bmatrix} \right) \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{4.2}$$

4.1.2 Balance Law Discretization

In order to discretize the balance laws, the first step is to express the balance in its differential form, as

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \boldsymbol{F} = \boldsymbol{S}, \tag{4.3}$$

where U is the solution vector, F is the flux coming in and going out of the cell and S is the source vector. In three dimensions, each cell has a volume, and therefore, a volume integral of Eq. (4.3) is taken over each cell

$$\iiint_{V_i} \frac{\partial U}{\partial t} \mathrm{d}V + \iiint_{V_i} \nabla \cdot F \mathrm{d}V = \iiint_{V_i} S \mathrm{d}V, \tag{4.4}$$

where V_i is the volume of cell i. It is possible to apply the divergence theorem over Eq. (4.4) to obtain

$$\iiint_{V_i} \frac{\partial U}{\partial t} \mathrm{d}V + \oiint_{A_i} F \cdot \hat{n} \mathrm{d}A = \iiint_{V_i} S \mathrm{d}V, \tag{4.5}$$

where $\hat{n}$ is the outwards facing unit normal and A_i is the surface of cell i. In order to obtain the average density, $\bar{U}_i$, the first term of Eq. (4.5) can be integrated, yielding

$$\bar{U}_i = \frac{1}{V_i} \iiint_{V_i} U_i \mathrm{d}V. \tag{4.6}$$

In this thesis, the volume of a cell is assumed to be constant and surrounded by a finite number of possibly curved faces, indexed by j, where each face has an arbitrary number of quadrature points, k. Therefore, the balance law of Eq. (4.3) can be approximated as

$$\frac{\partial \bar{U}_i}{\partial t} + \frac{1}{V_i} \sum_{j=0}^{\#faces} \sum_{k=0}^{\#points} F_{jk} \cdot \hat{n}_{jk} A_{jk} = \iiint_{V_i} S \mathrm{d}V. \tag{4.7}$$

In order to obtain the rate of change of $\bar{U}_i$, the flux is added or subtracted at each cell boundary and the effect of the source term is taken into account as well using quadrature points. In this thesis, to advance the balance law in time, an explicit-Euler time-marching scheme with a first-order finite-volume scheme, is implemented as follows

$$\bar{U}_i^{n+1} = \bar{U}_i^{n} - \frac{\Delta t}{V_i} \sum_{j=0}^{\#faces} F_j \cdot \hat{n}_j A_j + \Delta t \bar{S}^{n+1}, \tag{4.8}$$

where $\bar{U}_i^{n+1}$ is the average density in cell i at the next time step and $\bar{U}_i^{n}$ is the average density in cell i at the current time step.

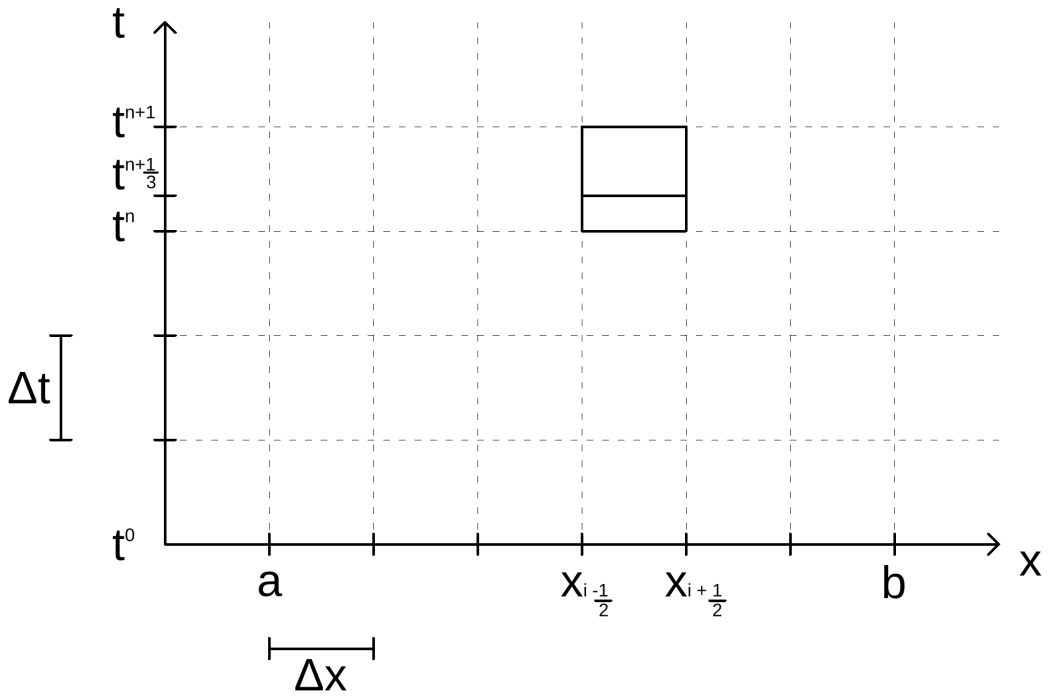


Figure 4.2: Discontinuous-Galerkin-Hancock discretization of the space-time domain.

4.2 Discontinuous-Galerkin Scheme

While a first-order finite-volume scheme is quite robust, it lacks accuracy. Higher-order schemes are desirable as they can be more accurate without being necessarily computationally expensive. Discontinuous-Galerkin methods are a category of higher-order numerical methods used to solve differential equations that are particularly well-suited for first-order balance laws. They are easily used with complex geometries and have high parallel efficiency. The first formulation of the method was introduced by Reed and Hill [20] to solve the neutron transport equation. LeSaint and Raviart [14] produced the first analysis of the method. One highly popular model in this class is the Runge-Kutta discontinuous-Galerkin method, developed extensively by Cockburn and Shu [7]. It uses a piecewise linear discontinuous-Galerkin method for the space discretization and an explicit total variation diminishing second-order Runge-Kutta scheme for the time discretization.

The discontinuous-Galerkin-Hancock scheme, developed by Suzuki and van Leer [21, 22], was developed specifically for hyperbolic partial differential equations that

result from moment closures. This scheme obtains weak solutions to the governing partial differential equation, Eq. (4.3), such that the projection of tests function with the PDE are satisfied. The test functions are chosen such that they are non-zero in only one cell and a finite number of them are used. Because the local source terms resulting from moment closures are stiff, they have to be treated implicitly. Suzuki used the Radau IIA method, which is a third-order accurate implicit two-step time-marching algorithm [21]. As shown in Figure 4.2, an intermediate solution is added at time step $n + \frac{1}{3}$. Figure 4.3 shows the quadrature points used in a discontinuous-Galerkin-Hancock scheme. The green face represents a two-dimensional cell at time step n, the orange face represents the same cell at time step $n + \frac{1}{3}$ and the blue face represents the same face at time step $n + 1$. First, surface flux integrals, defined with circles in Figure 4.3, are evaluated on all four vertical colourless faces. These surface flux integrals are evaluated at time steps $n + \frac{1}{6}$ and $n + \frac{1}{2}$. Then, the source volume integrals, defined with squares in Figure 4.3, are evaluated at time steps $n + \frac{1}{3}$ and $n + 1$. Finally, the volume flux integrals, defined with triangles in Figure 4.3, are evaluated at time steps n, $n + \frac{1}{3}$ and $n + 1$. The "x" represent a two-point quadrature rule and make up the Hancock predictor. In order to ensure that the discontinuous solutions are stable, slope limiters are used. The slope limiters are updated once the cell-averaged solutions at time steps $n + \frac{1}{3}$ and $n + 1$ are known.

The full description of the discontinuous-Galerkin-Hancock scheme can be found in Suzuki's thesis [21]. Although the implementation of this numerical scheme on one block was done by another student, the parallelization and AMR use the framework described in Chapter 6.

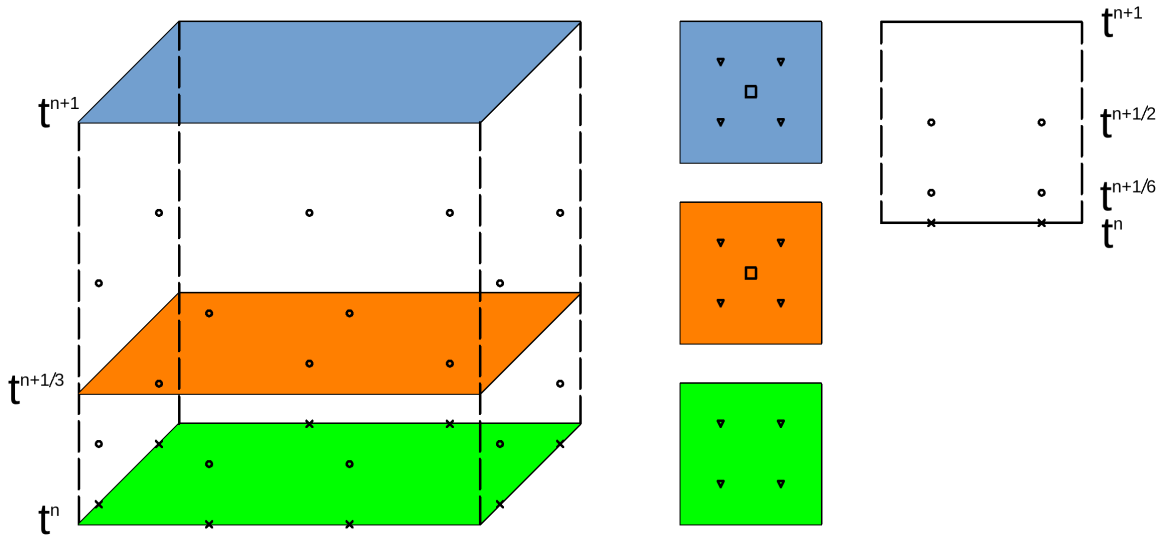


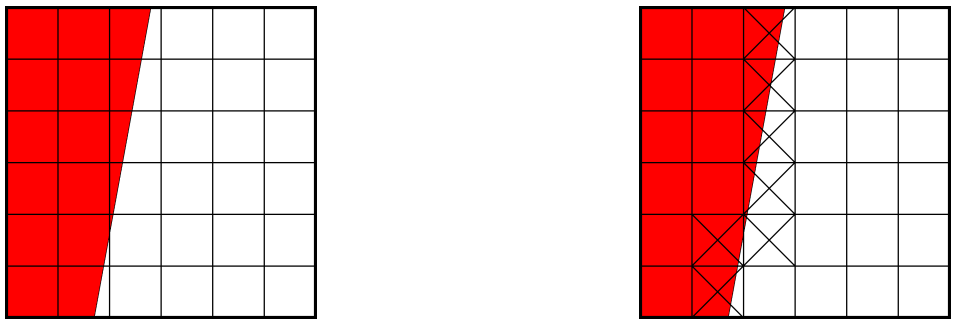
Figure 4.3: Discontinuous-Galerkin-Hancock quadrature rules.

Chapter 5

History of Adaptive Mesh Refinement

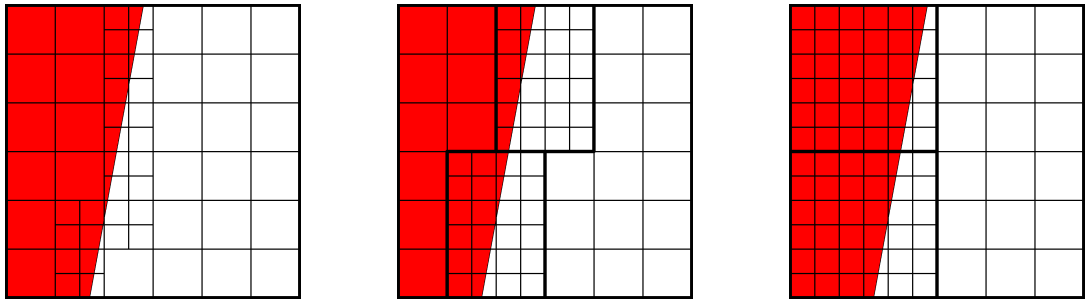
Adaptive mesh refinement (AMR) is a technique used in numerical analysis to adapt the precision of a solution while the solution is being computed. Having a uniform grid in terms of cell density can limit the accuracy of the solution, as having more cells in the mesh leads to more expensive computations. However, many numerical simulations do not require a high resolution mesh across the whole domain, only in certain areas. Sometimes, certain areas of the domain are of interest, while others remain more or less unchanged during the computation. Adaptive mesh refinement provides the dynamic framework necessary to increase the resolution of certain areas of the mesh, while leaving other areas at lower resolution. In this chapter, cell-based, patch-based and block-based adaptive mesh refinement are reviewed and their advantages and caveats are put forward.

It often happens that computations simulate a shock wave propagating through an initially coarse mesh. There is one state that is relatively constant in front of the shock wave and another relatively constant state behind the shock wave. The area of interest is mainly in the vicinity of the shock wave, where more cells should be added. Figure 5.1a shows a solution where a discontinuity, for example a shock wave, is propagating in the domain. The initial grid is coarse and will lead to an inaccurate solution. During computations, the cells where the discontinuity is detected are flagged for refinement. Over the years, different methods have been developed to refine grids



(a) Initial grid. (b) Cells flagged for refinement.

Figure 5.1: Discontinuity in a domain.



(a) Cell-based refinement. (b) Patch-based refinement. (c) Block-based refinement.

Figure 5.2: Three different types of AMR.

during computations. Cell-based refinement, patch-based refinement as well as block-based refinement are be described in this chapter.

5.1 Cell-Based Adaptive Mesh Refinement

Cell-based adaptive mesh refinement was first developed by Berger and Leveque [1]. This method refines individual cells where there is a discontinuity or another phenomena of interest. Each cell flagged in Figure 5.1b, denoted with an "x", is refined, in order to capture more accurately the discontinuity in the solution, as shown in Figure 5.2a. This technique refines strictly the cells that have been flagged for refinement, which results into a smaller number of total cells than other refinement techniques. However, cell-based AMR often loses the advantages of structured meshes and can lead to more complex meshes that results in larger connectivity data storage and domains that are harder to parallelize.

5.2 Patch-Based Adaptive Mesh Refinement

Patch-based adaptive mesh refinement was first developed by Berger and Collela [3]. This method applies a patch of refined cells over a discontinuity or other phenomena of interest. One or more patches of refined cells is applied over the cells flagged for refinement in Figure 5.1b. As shown in Figure 5.2b, two patches of refined cells replaced the flagged cells, in order to better capture the phenomena of interest. Although patch-based AMR preserves many of the advantages of structured meshes, since the generated patches do not all necessarily have the same sizes, efficient parallelization poses a challenge.

5.3 Block-Based Adaptive Mesh Refinement

Block-based adaptive mesh refinement was developed by Quirk [19] and Berger [4]. It combines elements of both cell-based and patch-based refinement, yielding a relatively simple algorithm. The initial grid, as seen in Figure 5.1a is considered the root or parent block. Each new refined block has the same number of cells as the parent block. As shown in Figure 5.2c, two new child blocks replace the coarse cells covering the discontinuity.

Although some sections of the grid may be over-refined, notably the left-most section of Figure 5.2c, this flaw is addressed by distributing blocks on a parallel computational network. However, this technique preserves the advantages of structured meshes. In order to obtain solutions using a parallel framework, the connectivity between the refined blocks has to be stored. This is traditionally done using a hierarchical binary tree, which keeps track of the refinement history of the blocks [24]. These data structures are often simpler than that of other types of AMR.

As shown in Figure 5.3, a single block is refined into two blocks, A and B, which represent the first two branches of the tree. Then, block B is refined into two more blocks, C and D, and two more branches are added to the tree. Finally, block A refines into two blocks, E and F, which represents two more branches in the tree.

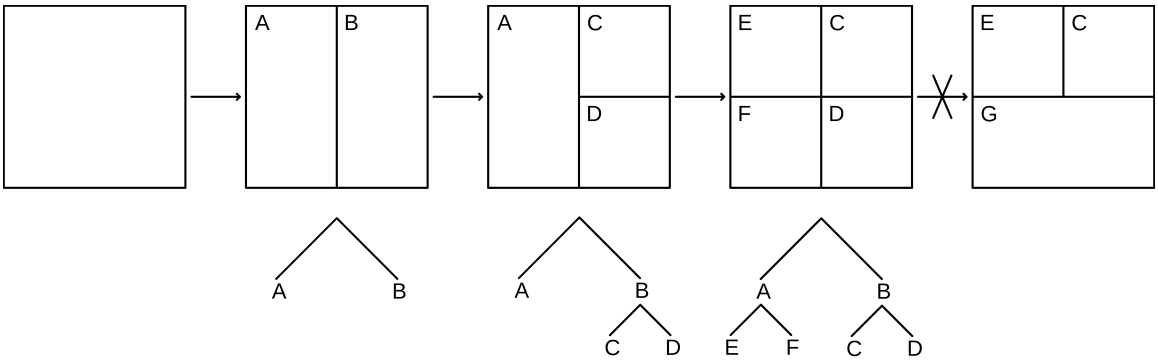


Figure 5.3: Block refinement and the corresponding binary tree.

Although binary trees are a simple way to keep track of the refinements, problems arise when anisotropic refinement is used and blocks are flagged for coarsening. The only easy way to coarsen blocks is to undo a refinement that has been done, meaning that they have to be on the same side of the tree. For example, in Figure 5.3, blocks D and F cannot be coarsened together, since they are on different branches of the tree. This limitation is common in many current block-based AMR implementations [24]. A goal of this work is to avoid this constraint in a simple elegant way. A block-based adaptive mesh refinement that does not use a hierarchical tree structure is presented in Chapter 6.

Chapter 6

The Multi-Dimensional Framework

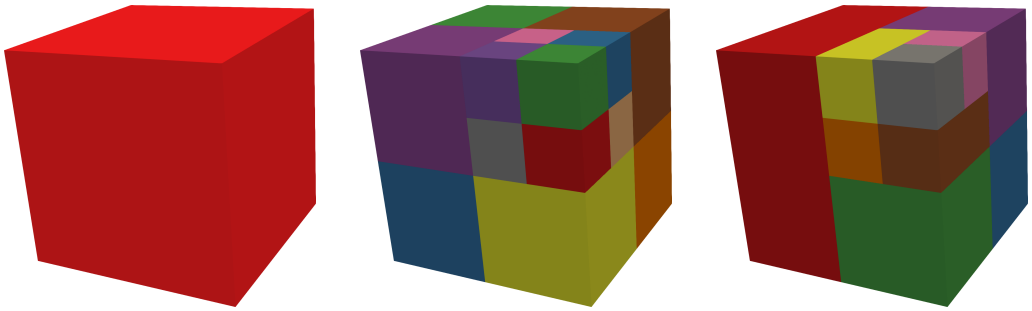
The goals of designing a new adaptive mesh refinement scheme is to develop an algorithm that

- does not rely on a hierarchical tree to keep track of a block's refinement history;
- can be used with one, two and three dimensional blocks;
- can be used with a parallel network; and
- can easily be used with different numerical methods and different partial differential equations.

The new developed adaptive mesh refinement scheme is implemented within a framework for solving partial differential equations. Structured meshes are made up of at least one block, all having the same number of cells. Once a block is flagged for refinement, it can be divided in two, four or eight children, each child having the same number of cells as the parent. In this chapter, the new adaptive mesh refinement algorithm is described in terms of the block representation, block connectivity and CPU communication, in order to clearly demonstrate how the algorithm works.

6.1 Three Dimensional Block Representation

Each block has a number of nodes and cells, which are described using i, j and k indexing. When a three-dimensional block is constructed, the domain dimension in



(a) Initial block. (b) Isotropic refinement. (c) Anisotropic refinement.

Figure 6.1: Types of block refinement.

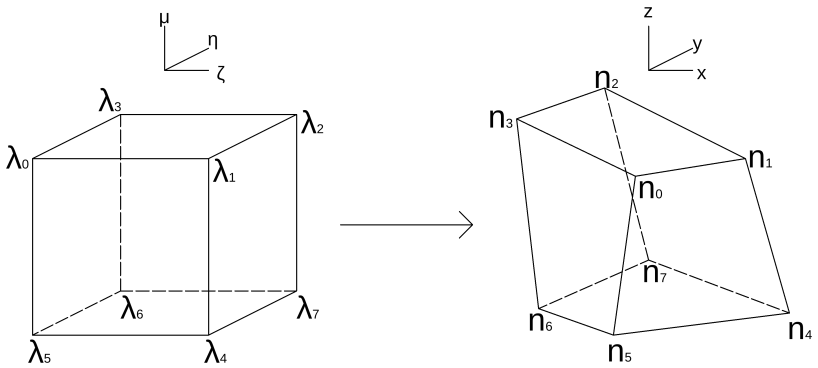


Figure 6.2: Trilinear mapping.

the x, y and z-directions are specified, as well as the number of nodes desired in the i, j and k-directions. In this research, block-based adaptive mesh refinement is used, meaning that during computations, blocks can be refined in the i, the j, or the k direction, or any combination thereof, resulting in two, four or eight children. Those children can then be distributed among available CPUs to accelerate computations. As shown in Figure 6.1, a single block, Figure 6.1a, can be divided either in an isotropic fashion, Figure 6.1b, or in an anisotropic fashion, Figure 6.1c. Isotropic refinement means that the blocks are refined in all three directions every refinement and anisotropic refinement means that the block can be refined in one direction, two directions or all three directions, as needed.

In order to compute fluxes, quadrature points are needed. Every cell in a three dimensional block has a trilinear mapping that is used to compute the volume of the cell, the centroid of the cell as well as any quadrature points needed [21].

Figure 6.2 shows how a cube in a reference space is related to a cell in the domain space, the cell can be skewed or a rectangular prism. The coordinates of the reference cube nodes are

$$\begin{aligned} \lambda_0 &= (-1,-1,1)\,, & \lambda_4 &= (1,-1,-1)\,, \\ \lambda_1 &= (1,-1,1)\,, & \lambda_5 &= (-1,-1,-1)\,, \\ \lambda_2 &= (1,1,1)\,, & \lambda_6 &= (-1,1,-1)\,, \\ \lambda_3 &= (-1,1,1)\,, & \lambda_7 &= (1,1,-1)\,, \end{aligned} \tag{6.1}$$

where node λ_0 maps to n_0, node λ_1 maps to n_1, node λ_2 maps to n_2, and so on. It is possible to map any point in the reference cube $\boldsymbol{\zeta} = (\zeta, \eta, \mu)$ to its corresponding point in a cell $\boldsymbol{x} = (x, y, z)$ with

$$\begin{aligned} \boldsymbol{x}(\boldsymbol{\zeta}) = \frac{(1-\zeta)(1-\eta)(1+\mu)}{8} n_0 + \frac{(1+\zeta)(1-\eta)(1+\mu)}{8} n_1 + \frac{(1+\zeta)(1+\eta)(1+\mu)}{8} n_2 \\ + \frac{(1-\zeta)(1+\eta)(1+\mu)}{8} n_3 + \frac{(1+\zeta)(1-\eta)(1-\mu)}{8} n_4 + \frac{(1-\zeta)(1-\eta)(1-\mu)}{8} n_5 \\ + \frac{(1-\zeta)(1+\eta)(1-\mu)}{8} n_6 + \frac{(1+\zeta)(1+\eta)(1-\mu)}{8} n_7, \end{aligned} \tag{6.2}$$

where $\boldsymbol{x}(\boldsymbol{\zeta})$ is the coordinate of the mapped point in the cell, $\zeta, \eta, \mu \in [-1, 1]$ are the coordinates of the point in the reference cube and n_0, n_1, n_2, n_3, n_4, n_5, n_6 and n_7 are the coordinates of the eight vertices of the cell. A Jacobian matrix for the transformation is defined as

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial x}{\partial \zeta} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \mu} \\ \frac{\partial y}{\partial \zeta} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \mu} \\ \frac{\partial z}{\partial \zeta} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \mu} \end{bmatrix}. \tag{6.3}$$

The Jacobian determinant is used to compute geometric properties of the cell, using the following coordinate transformation identity

$$\mathrm{d}x\mathrm{d}y\mathrm{d}z = |\boldsymbol{J}(\zeta, \eta, \mu)|\,\mathrm{d}\zeta\mathrm{d}\eta\mathrm{d}\mu. \tag{6.4}$$

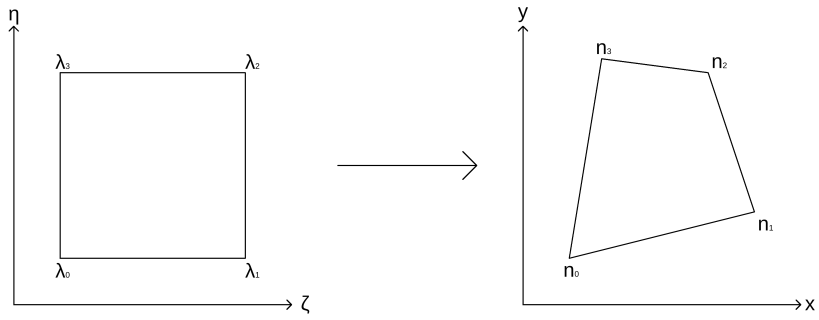


Figure 6.3: Bilinear mapping.

The volume of a cell is computed by using

$$V = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} |\boldsymbol{J}(\zeta,\eta,\mu)|\,\mathrm{d}\zeta\mathrm{d}\eta\mathrm{d}\mu. \tag{6.5}$$

The centroid of a cell is computed by using

$$\bar{\boldsymbol{x}} = \frac{1}{V}\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \boldsymbol{x}(\zeta,\eta,\mu)\,|\boldsymbol{J}(\zeta,\eta,\mu)|\,\mathrm{d}\zeta\mathrm{d}\eta\mathrm{d}\mu. \tag{6.6}$$

Every face of a three-dimensional cell is represented by a bilinear mapping, shown in Figure 6.3 that follows the same principle as the trilinear mapping.

In order to compute fluxes through the cell boundaries, quadrature points are used. Depending the type of numerical scheme, a different number of quadrature points per face is required. The coordinates of Gaussian quadrature points in the reference space are mapped to their coordinates in the domain space using Eq. (6.2).

6.2 Block Connectivity

Once blocks have been distributed across the parallel network using the Message Passing Interface (MPI) and Open Multi-Processing (OpenMP), a way to know how the blocks connect to each other is needed, as fluxes leaving one block need to be added to the neighbouring block. In order to achieve proper communication, different components are necessary. In the present implementation, this is done using block signatures, boundary signatures and inter-block faces.

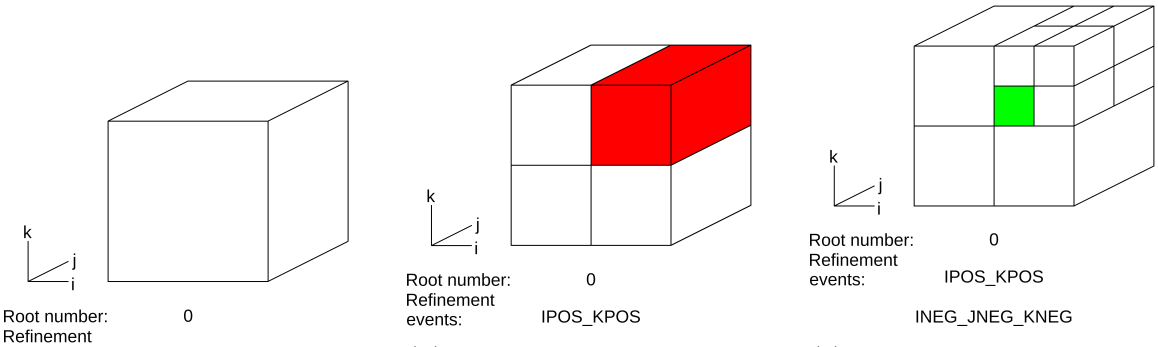
6.2.1 Open Multi-Processing and Message Passing Interface

Open Multi-Processing (OpenMP) is an application programming interface built into the compiler that parallelizes through local multi-threading. It has compiler directives, library functions and environment variables. A thread is defined as the smallest unit of instructions that can be executed in parallel by the scheduler. Once the parallel directive has been given, multiple thread are spawned and each thread executes the section of code that follows the directive. The resulting threads all have access to the same memory, but each of them has its own stack, used to keep track of function calls. Threads are independent from one another and are not automatically synchronized in any way, which means that nothing can be assumed about the order in which things happen. If a certain order is necessary, one can use synchronization directives. In the case of the present code, OpenMP is used during time-marching to parallelize the computation of fluxes.

The Message Passing Interface (MPI) is a library that simplifies the passing of messages between processors on the same machine or through a network. It defines a standard that is implemented by library developers. MPI defines communicators that connect groups of processes that can communicate directly, where a process is the instance of a program including its used memory and can be a member of multiple communicators. The number of processes corresponds to the number of CPUs available. Every communicator gives each of its processes a unique, integer identifier, called rank, when the process is initialized. The rank of a CPU is used to specify the source and destination of messages. Unlike OpenMP, the MPI processes do not share memory and messages have to be sent. In the present code, MPI is used to distribute blocks across the network of large-scale modern distributed computer.

6.2.2 The Block Signature

The block signatures are used to differentiate one block from another and to keep track of the refinement events the blocks have been through. As coarse blocks are initially created, they are given a number. This number is a positive integer that



(a) Initial block.

(b) Block after one refinement event.

(c) Block after two refinement events.

Figure 6.4: Update of block signature after refinement.

becomes part of the block's signature. The first entry in the block signature is always the root number, which can be any arbitrary integer. The refinement events are made up of the direction in which the block was refined as well as whether the block is located on the positive side or negative side of the refinement. A block is deemed to be on the negative side of a refinement if it is on the side where the index i, j or k is minimum and a block is located on the positive side of a refinement if it is located on the side where the index is maximum.

As shown in Figure 6.4, the initial block, on the left, is given an arbitrary root number 0. Then, the block is refined in the i-direction and the k-direction at the same time, yielding four children. The red block has the same root number as its parent and has one refinement event in its block signature. The refinement event is determined based on the position of the child with respect to the split. In the i-direction, the red block is on the positive side of the split, as i is minimum on the left side and maximum on the right side of the block. In the k-direction, the red block is also on the positive side of the split, as the k index is minimum at the bottom and maximum at the top. Since the refinement in i and k happened simultaneously, the block signature is given the refinement event "IPOS_KPOS". Finally, the red block is refined in the i, j and k-directions at the same time, yielding eight children. The domain is now covered by a total of 11 blocks. The green block has the same root number and first refinement event as its parent, the red block, but has one more refinement event added to its block signature. The green block is on the negative side

of the i split, as i minimum on the left side. The green block is on the negative side of the j split, as j is minimum at the front of the block and maximum at the back of the block. Finally, in the k-direction, the green block is on the negative side of the split, as k is minimum at the bottom of the block. Therefore, because the block was refined in all three directions at the same time, the refinement event "INEG_JNEG_KNEG" is added to the block signature.

In order to keep track of the blocks in the same way on different CPUs, the block signatures have to be ordered in a logical way. The blocks are first placed in ascending order based on their root number. Then, if two blocks have the same root number, the refinement events are lexicographically ordered. The first non-matching element of the refinement history defines which one of the two signatures being compared will come before the other. They are compared using lexicographical order. For example, if "IPOS" and "JNEG" are being compared, "IPOS" will come first. If "IPOS" and "INEG" are being compared, "INEG" will come first.

6.2.3 The Boundary Signature

Once the domain is represented using multiple blocks, they have to be connected to account for the fluxes that go from one block to the other through the block boundaries. Boundary signatures, existing only on cell boundaries that connect to a cell on another block, contain information about the two blocks on either side of the boundary as well as the boundary side index of both connected boundaries. The collection of information ensures that every boundary between cells on different blocks has a unique signature.

As shown in Figure 6.5, the blue boundary of block 1 is connected to the orange boundary of block 2. Blocks 1 and 2 both have four cells in each direction, defined using dashed lines. Every boundary is defined to have both a side A and a side B. In Figure 6.5, block 1 is deemed to be on side A of the boundary and block 2 is on side B of the boundary.

Figure 6.6 represents the two connecting faces of Figure 6.5 in two dimensions.

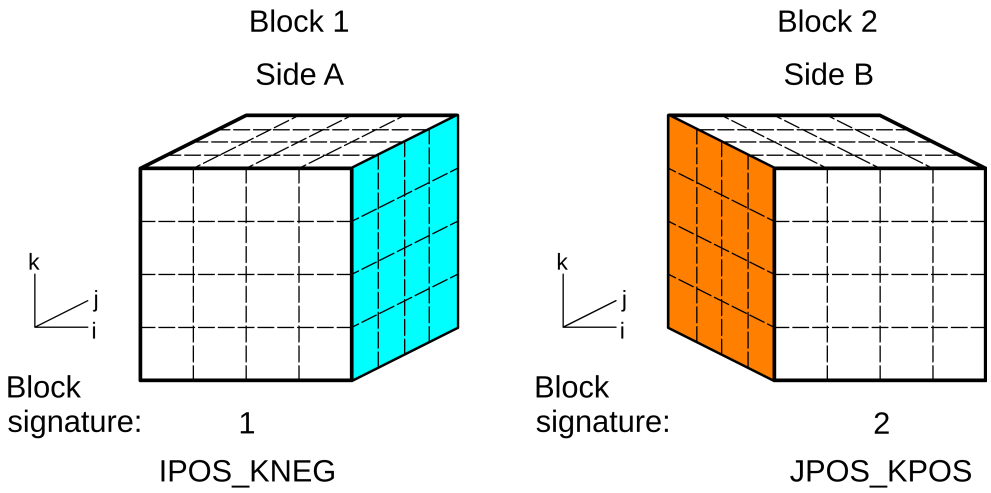


Figure 6.5: Two blocks sharing a boundary.

The blue face slides directly on top of the orange face. All sixteen boundary faces will have unique boundary signatures, due to the boundary side indices included in every signature. Every cell boundary faces of a block is placed into a vector, regardless of if they are connected to another block or not. The boundary side index represents the position of the boundary face in the vector. The red faces in Figure 6.6 will have the same boundary signature because, even though they are on two different blocks, they are in reality only one boundary through which fluxes will pass.

Following the same idea as with block signatures, it is also possible to order boundary signatures. When comparing two boundary signatures, both block signatures A are compared to determine which one comes first. In the case where they are equal, both block signatures B are compared. If they are equal, both side indices A are easily compared, as they are integer. Finally, if they are the same, both side indices B are compared.

For example, comparing boundary signatures 1 and 2 from Table 6.1, block signature A of boundary signature 1 is a prefix of block signature A of boundary signature 2. Therefore, boundary signature 1 comes before boundary signature 2. Comparing boundary signature 1 and 3 from Table 6.1, both block signatures A are the same, therefore, the block signatures B will be compared next. Again, they are both the same, so side indices A will be compared. As they are both the same, as a final step,

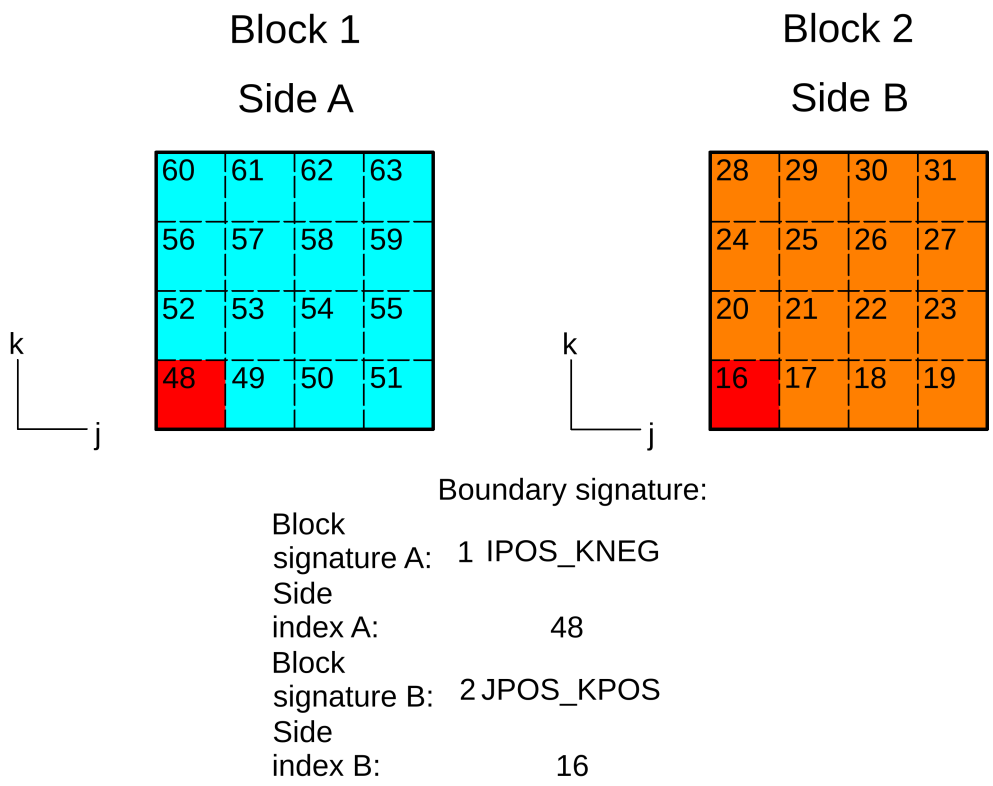


Figure 6.6: Boundary connecting two blocks.

side indices B will be compared. Since the side index B of boundary signature 1 is smaller than the side index B of boundary signature 3, boundary signature 1 comes before boundary signature 3. Finally, comparing boundary signatures 2 and 3 from Table 6.1, block signature A of boundary signature 3 is a prefix of block signature A of boundary signature 2, so boundary signature 3 comes before boundary signature 2. The final order of the boundary signatures is boundary signature 1, boundary signature 3 and boundary signature 2.

Table 6.1: Comparing boundary signatures.

	Boundary signature 1	**Boundary signature 2**	**Boundary signature 3**
Block signature A	0_IPOS	0_IPOS_KPOS	0_IPOS
Side index A	14	16	14
Block signature B	0_JNEG_KPOS	0_IPOS_KPOS	0_JNEG_KPOS
Side index B	3	6	12

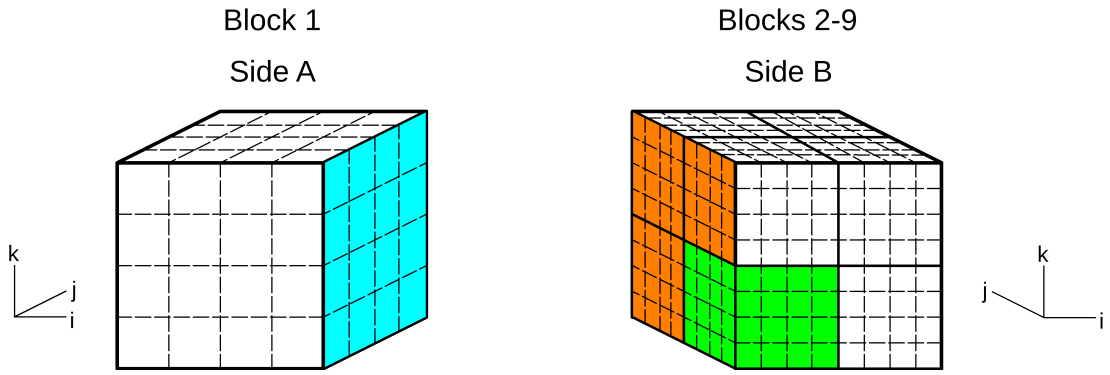


Figure 6.7: One coarse block and one refined block sharing a boundary.

6.2.4 Composite Interblock Face and Interblock Face

In the case where one block on the side of the boundary refines, and the other keeps the same resolution, the boundary on the coarser side will have to communicate with multiples boundaries on the more refined side. That is where composite interblock faces and interblock faces come into play. Composite interblock faces represent the actual boundary of the cell, which holds one or more interblock faces. The interblock faces are a representation of the single or multiple faces on the other side of the boundary.

For example, say the block 2 in Figure 6.5 refines in all three directions, as shown in Figure 6.7. Every blue boundary face communicates with four orange or green boundary faces. Therefore, looking at block 1, each boundary face on the blue side will be represented by a single composite interblock face containing four interblock faces. From the perspective of the green block, each boundary communicating with block 1 will be represented by a single composite interblock face containing a single interblock face.

Each interblock face stores the location of its four corner nodes, the boundary signature of the boundary, the rank of the CPU holding its neighbour as well as pointers. The pointers stored within the interblock faces point to information about the solution vector, the CPU rank as well as the refinement information of the cell and of its neighbour. The refinement information tells the composite interblock face how to refine and how its neighbour refines, whether it be in one, two or three directions.

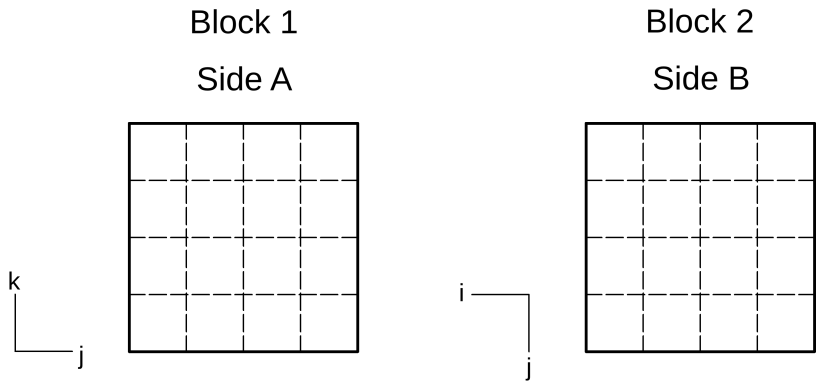


Figure 6.8: Two connected face with different cardinal directions.

Each composite interblock face stores all of the interblock faces representing the faces on the other of the boundary, the side of the face, either A or B, as well as the cardinal directions of the face and of its neighbour. The cardinal directions represent the directions, i, j or k, the face is defined in as well as whether the index, i, j or k, increases or decreases. For example, in Figure 6.6, the first cardinal direction of side A would be "JPOS", as the horizontal axis is defined in the j direction and grows from front to back. The second cardinal direction of side A would be "KPOS", as the vertical axis is defined in the k direction and grows from top to bottom. The cardinal directions of its neighbour, side B, would be the same as the cardinal directions of the face on side A. Although, in the previous example, the cardinal directions of the face and of its neighbour are the same, they do not have to be. Rotation of one block with respect to the other would cause the cardinal directions to be different.

As shown in Figure 6.8, the first cardinal direction of side A would be "JPOS" and the second cardinal direction of side A would be "KPOS". The first cardinal direction of side B would be "INEG", as the horizontal axis is defined in the i direction and decreases from left to right. The second cardinal direction of side B would be "JNEG", as the vertical axis is defined in the j direction and decreases from top to bottom.

For example, one can focus on one boundary face of block 1 and its corresponding neighbour faces on the green block, or block 2, of Figure 6.7. One cell boundary on block 1 has to share fluxes with four cell boundaries on block 2, as shown in Figure 6.9. Therefore, one composite interblock face is created to represent the boundary face of

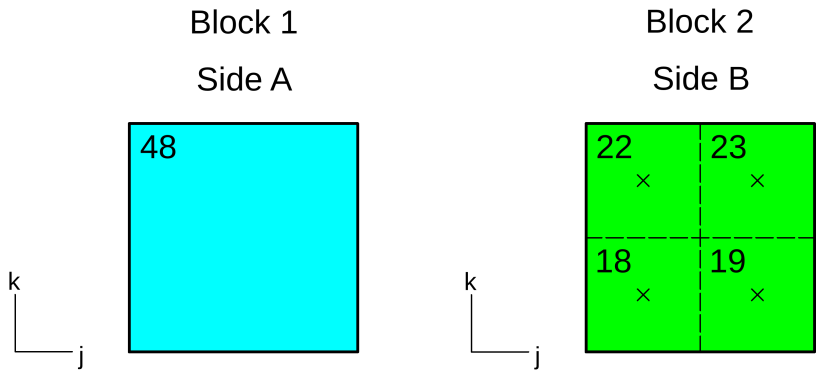


Figure 6.9: Composite interblock faces and its corresponding interblock faces.

block 1, this composite interblock face containing four interblock faces that represent the boundary faces of block 2. The Gaussian quadrature points, denoted by "x", are used to compute fluxes through the boundaries. When the flux calculations through the boundaries are done, the block asks each one of its composite interblock faces a list for its quadrature points, weights, as well as the pointers to the solution vectors, CPU ranks and refinement information of both sides of the boundary. The composite interblock faces get a list of quadrature points, weights and pointers from going through each one of its interblock faces. For example, in Figure 6.9, the composite face of block one will return a list of four quadrature points and weights, and a total of twenty-four pointers, three per side per interblock faces.

6.3 The Comm Hub

Once the blocks have been distributed between available CPUs, they need to communicate in order for the blocks to get information about their neighbours. To do so, each process contains an object called a comm hub. Every time a block is created or moved to a different CPU, it registers its boundary faces with the comm hub by giving it a list of its boundaries and the CPU responsible for its neighbour. Once every block has registered its boundaries, the comm hub separates them into list based on their neighbours MPI ranks and orders them according to the procedure described in Section 6.2.3. Two neighbouring CPUs will generate exactly the same lists of shared boundaries, as it is possible to perfectly sort boundary signatures.

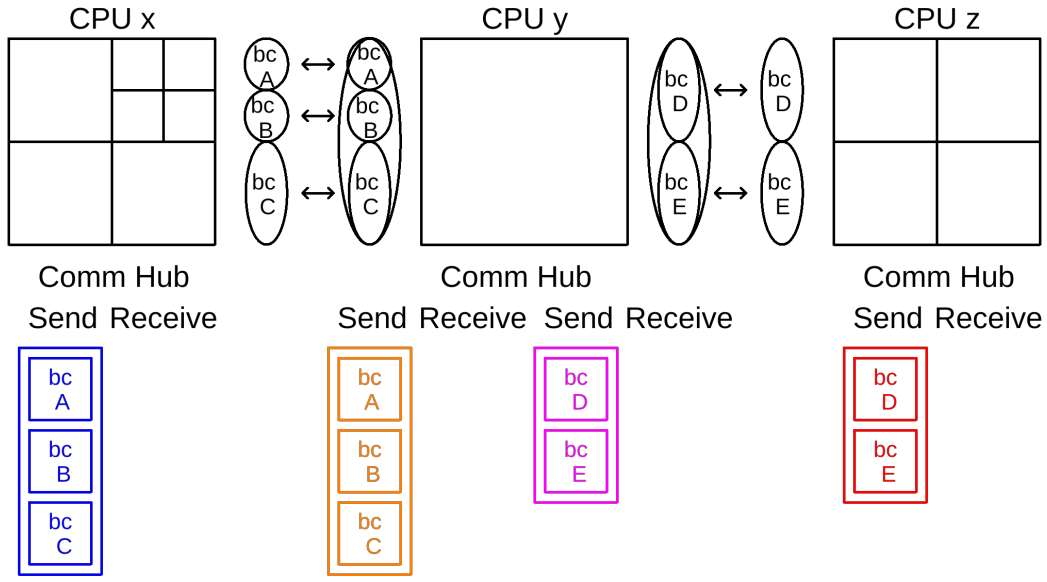


Figure 6.10: Representation of the comm hub before the neighbouring boundaries are linked.

As shown in Figure 6.10, each CPU will register their boundaries into the send buffer, so named because it will send its boundary information to its neighbour CPU. The receive buffer will be filled with information about neighbour cells. CPU x depicts seven two dimensional cells, three of them having boundary edges communicating with CPU y and has three composite interblock faces, each of them containing one interblock face. CPU y depicts one two dimensional cell having one boundary edge communicating with CPU x with one composite interblock face containing three interblock faces and one boundary edge communicating with CPU z with one composite interblock face containing two interblock faces. CPU z depicts four two dimensional cells, two of them having boundary edges communicating with CPU y and has two composite interblock faces, each of them containing one interblock face. Since CPU y has to communicate with two other CPUs, it has two sets of send and receive buffers in its comm hub. Based on its boundary signature, each interblock face is given pointers into its send buffer as well as into its receive buffer. Once the buffer space is allocated, the boundaries are linked, meaning that the receive buffers are filled with the neighbour cells information, as shown in Figure 6.11. All of the comm hubs and buffers are set up once after each refinement events and are subsequently

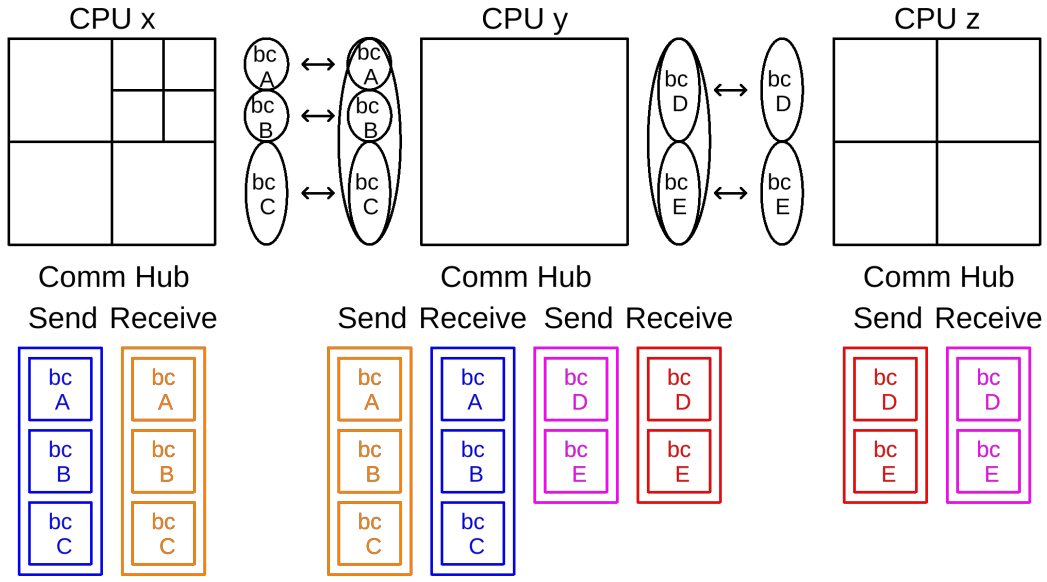


Figure 6.11: Representation of the comm hub after the neighbouring boundaries are linked.

reused every iterations.

MPI messages containing the send buffer information are sent as early as possible, and are used as late as possible to reduce the chance of having to wait. While the fluxes through internal boundaries are computed, the messages are sent through the network. The communication between two blocks is organized in such a way that at most one message per iteration is exchanged. For local time stepping, i.e. advancing a block in time according to its locally defined time step, no global communication is used. Furthermore, no part of the implementation grows with the global size of the problem, meaning that the memory used by one process with a given number of cells is the same regardless of the global problem size.

Chapter 7

Results

In this chapter, numerical simulations are presented. The shown cases were chosen in order to demonstrate the capabilities of the AMR algorithm with a finite-volume scheme and a discontinuous-Galerkin-Hancock scheme. The Euler equations as well as the PGM equations are used in the presented cases.

7.1 Shock Cube

The first case considered is a shock cube, in which two different initial states are separated by discontinuities [9]. It was chosen as the first case to test the AMR algorithm without any resolution changes and make sure that the fluxes between blocks on different CPUs are handled properly. The domain considered is a cube of 1 m side, where $-0.5\,\mathrm{m} < x < 0.5\,\mathrm{m}$, $-0.5\,\mathrm{m} < y < 0.5\,\mathrm{m}$ and $-0.5\,\mathrm{m} < z < 0.5\,\mathrm{m}$. A discontinuity is introduced, where the state on one side of the discontinuity has a lower density and pressure than the state on the other side of the discontinuity. The initial conditions are

$$U_1 : \rho = 1.225\,\mathrm{kg/m^3},\ u_x = u_y = u_z = 0\,\mathrm{m/s},\ P = 101325\,\mathrm{Pa}$$

$$U_2 : \rho = 0.30625\,\mathrm{kg/m^3},\ u_x = u_y = u_z = 0\,\mathrm{m/s},\ P = 25331.25\,\mathrm{Pa}$$

$$U = \begin{cases} U_2 \text{ if } x < 0,\ y < 0 \text{ and } z < 0; \\ U_1 \text{ otherwise;} \end{cases} \tag{7.1}$$

The partial differential equations used for this first demonstration are the compressible Euler equations, found using the Maxwell-Boltzmann distribution, where the gas is assumed to be in equilibrium. The continuity equation is given by

$$\frac{\partial}{\partial t}\rho + \frac{\partial}{\partial x_k}(\rho u_k) = 0, \tag{7.2}$$

the momentum equation is given by

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_k}(\rho u_i u_k + p\delta_{ik}) = 0, \tag{7.3}$$

and the energy equation is given by

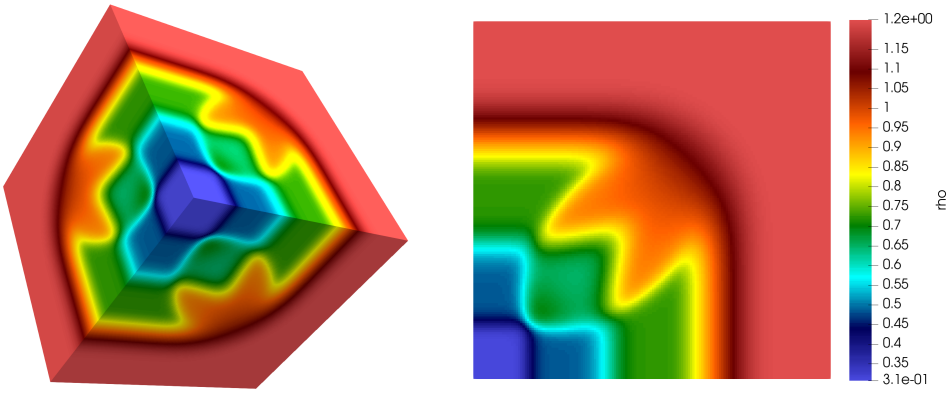
$$\frac{\partial}{\partial t}\left(\frac{\rho u_i u_i}{2} + \frac{p}{\gamma - 1}\right) + \frac{\partial}{\partial x_k}\left(u_k\left(\frac{\rho u_i u_i}{2} + \frac{\gamma p}{\gamma - 1}\right)\right) = 0, \tag{7.4}$$

where $\gamma = 1.4$. The solution and flux vectors in three dimensions are given by

$$\boldsymbol{U} = \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho u_z \\ \frac{p}{\gamma-1} + \frac{1}{2}\rho(u_x^2 + u_y^2 + u_z^2) \end{bmatrix}, \tag{7.5}$$

$$\boldsymbol{F}_x = \begin{bmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_x u_y \\ \rho u_x u_z \\ u_x\left(\frac{\gamma p}{\gamma-1} + \frac{1}{2}\rho(u_x^2 + u_y^2 + u_z^2)\right) \end{bmatrix}, \tag{7.6}$$

$$\boldsymbol{F}_y = \begin{bmatrix} \rho u_y \\ \rho u_x u_y \\ \rho u_y^2 + p \\ \rho u_y u_z \\ u_y\left(\frac{\gamma p}{\gamma-1} + \frac{1}{2}\rho(u_x^2 + u_y^2 + u_z^2)\right) \end{bmatrix}, \tag{7.7}$$



(a) Three-dimensional view.

(b) View of a face of the cube.

Figure 7.1: Shock cube after 0.00075 seconds using three levels of refinement.

$$\boldsymbol{F}_z = \begin{bmatrix} \rho u_z \\ \rho u_x u_z \\ \rho u_y u_z \\ \rho u_z^2 + p \\ u_z \left(\frac{\gamma p}{\gamma - 1} + \frac{1}{2}\rho(u_x^2 + u_y^2 + u_z^2) \right) \end{bmatrix}. \tag{7.8}$$

The initial mesh is a single block of $20 \times 20 \times 20$ cells. The mesh is refined in all three directions three times, for a total of 512 blocks, each having $20 \times 20 \times 20$ cells, for a total of 4 096 000 computational cells covering the domain. A CFL number of 0.4 was used.

As shown in Figure 7.1, after 0.00075 seconds, a plurality of shock, contact and rarefaction waves are generated in the domain. As shown in Figure 7.1a, the solution is the same on the top three faces of the cube, as is expected, since the initial conditions are the same on said faces. The results also agree with those obtained by Freret et al. [9]. Figure 7.1b shows one face of the cube, for a better view of the solution.

7.2 Supersonic Inviscid Flow

The second case considered is a supersonic inviscid flow over a bump. This case was chosen to test the AMR algorithm. The scheme is supposed to refine blocks where there is a pre-determined density change in neighbouring cells and the strong shocks

created by a supersonic inviscid flow over a bump offer a good opportunity to test the implementation. The domain considered is a channel in both two dimensions and three dimensions. In two dimensions, the channel is 5 meters long ($-1.0\,\mathrm{m} < x < 4.0\,\mathrm{m}$) and 1 meter high ($0.0\,\mathrm{m} < y < 1.0\,\mathrm{m}$). In three dimensions, the channel has the same dimensions in x and y and is one meter deep in the z-direction ($0.0\,\mathrm{m} < z < 1.0\,\mathrm{m}$). A circular arc extends 10% into an otherwise straight-edged channel, between $x = 0.0\,\mathrm{m}$ and $x = 1.0\,\mathrm{m}$. The top and bottom boundaries are reflections, the left boundary is held fixed with an incoming supersonic flow with a Mach number of 2 and the right boundary is a zero-gradient boundary condition.

The partial differential equations used for this case are again the compressible Euler equations, seen in Eqs. (7.2)-(7.4).

In two dimensions, the solution and flux vectors are given by

$$\boldsymbol{U} = \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \frac{p}{\gamma-1} + \frac{1}{2}\rho(u_x^2 + u_y^2) \end{bmatrix}, \tag{7.9}$$

$$\boldsymbol{F}_x = \begin{bmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_x u_y \\ u_x \left(\frac{\gamma p}{\gamma-1} + \frac{1}{2}\rho(u_x^2 + u_y^2) \right) \end{bmatrix}, \tag{7.10}$$

$$\boldsymbol{F}_y = \begin{bmatrix} \rho u_y \\ \rho u_x u_y \\ \rho u_y^2 + p \\ u_y \left(\frac{\gamma p}{\gamma-1} + \frac{1}{2}\rho(u_x^2 + u_y^2) \right) \end{bmatrix}, \tag{7.11}$$

In three dimensions, the solution and flux vectors are give by Equations (7.5)-(7.8).

For this problem, the gas is air with density $\rho = 1\,\mathrm{kg/m^3}$, pressure $p = 100\,\mathrm{kPa}$, heat capacity ratio $\gamma = 1.4$ and an incoming Mach number of 2. In both the two dimensional and three dimensional cases, a CFL number of 0.4 was used.

The initial mesh is made up of five blocks, each block having ten computational cells in each direction. In order to decide which blocks need refinement, the density between two neighbouring cells is checked. If the density in the ith cell is 1.2 times

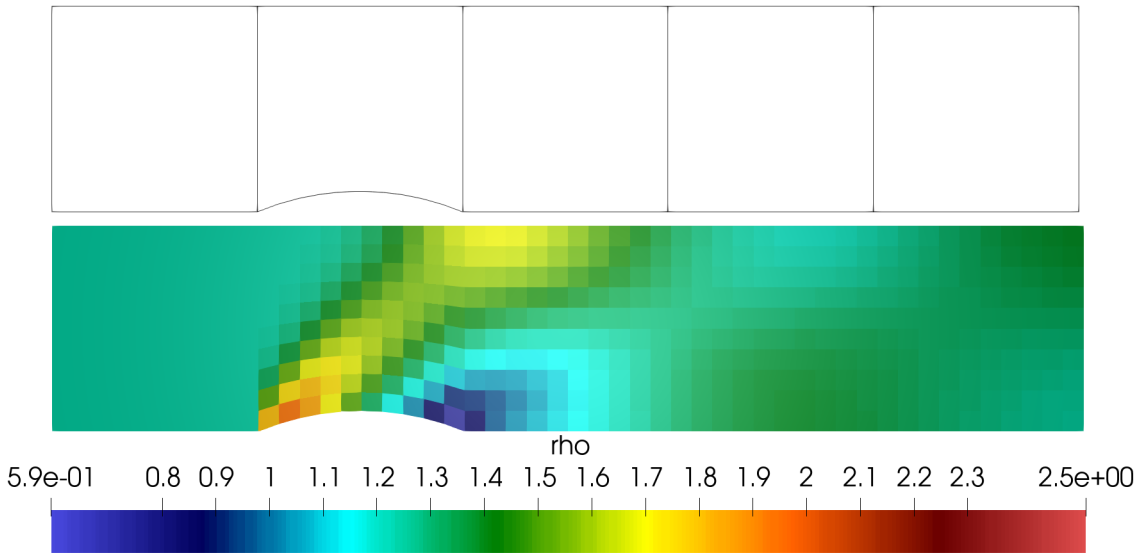


Figure 7.2: Two-Dimensional Flow with a Mach number of 2 over a bump with zero level of refinement using a finite-volume scheme.

higher than the density in the ith $+ 1$ cell, the block is flagged for refinement. This criteria was chosen in order to capture the shock waves. When steady state is reached, the mesh is refined where needed and the simulation continues running until a new steady state is achieved. This process is repeated four times for the two dimensional case and three times for the three dimensional case.

7.2.1 Results Obtained with the Finite-Volume Scheme

The final two-dimensional mesh has 125 blocks and 12 500 cells. Clearly, as is shown in Figure 7.2, the initial unrefined mesh does a poor job of capturing the shock waves. The solution goes through multiple stages of refinement, demonstrated in Figures 7.3-7.5. As the mesh is refined, the shock waves appear more clearly, as shown in Figure 7.6. The flexibility of the adaptive mesh refinement scheme is clearly demonstrated in Figure 7.6, where the left-most block remained unrefined while its neighbour refined multiple times, resulting in a single block having to communicate with eight neighbour blocks.

The final three-dimensional mesh has 502 blocks and 502 000 cells. Once again, as is shown in Figure 7.7, the initial unrefined mesh does a poor job of capturing the

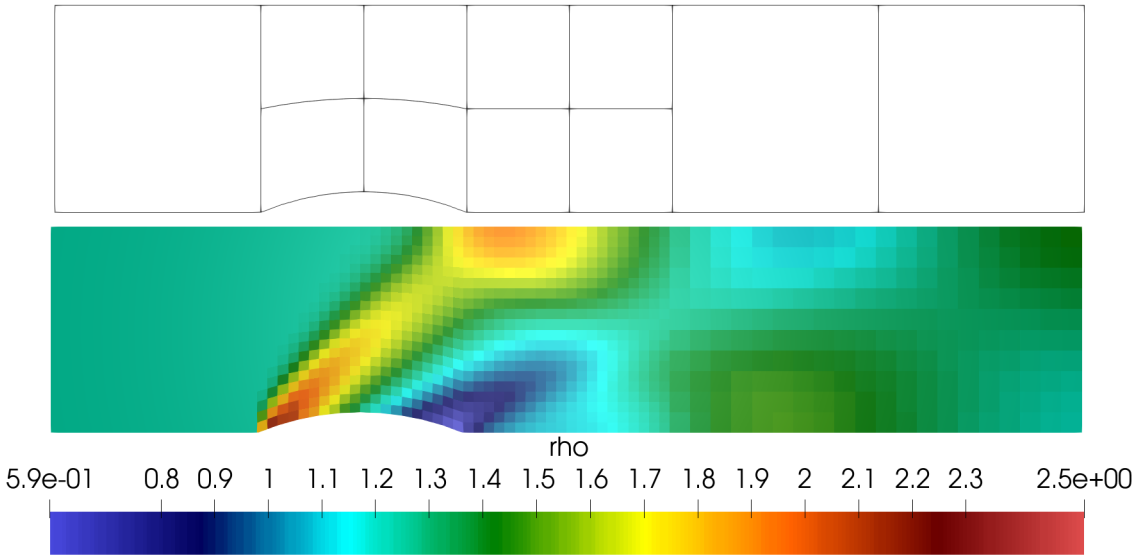


Figure 7.3: Two-Dimensional Flow with a Mach number of 2 over a bump with one level of refinement using a finite-volume scheme.

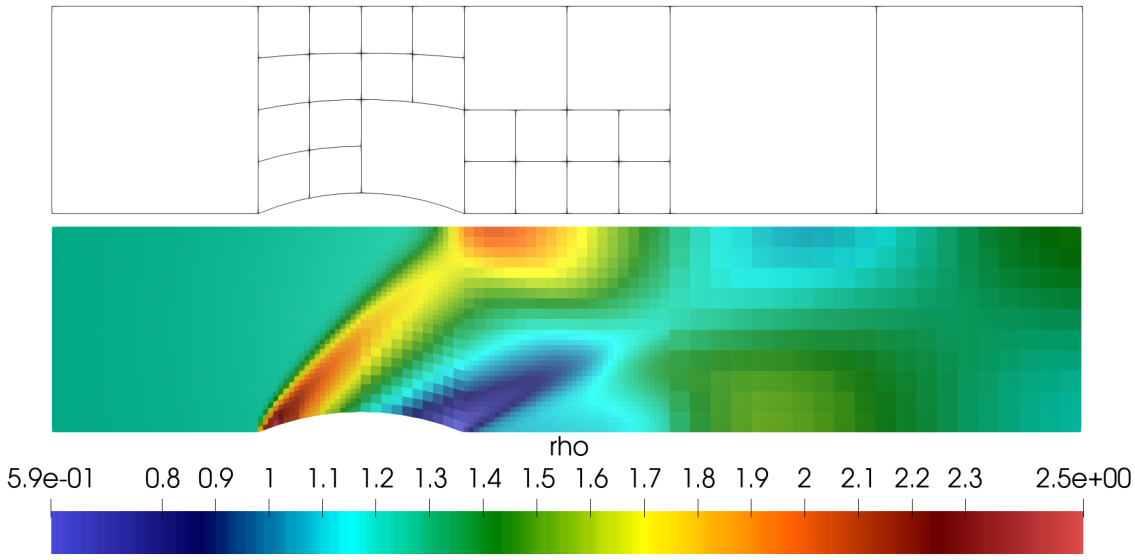


Figure 7.4: Two-Dimensional Flow with a Mach number of 2 over a bump with two levels of refinement using a finite-volume scheme.

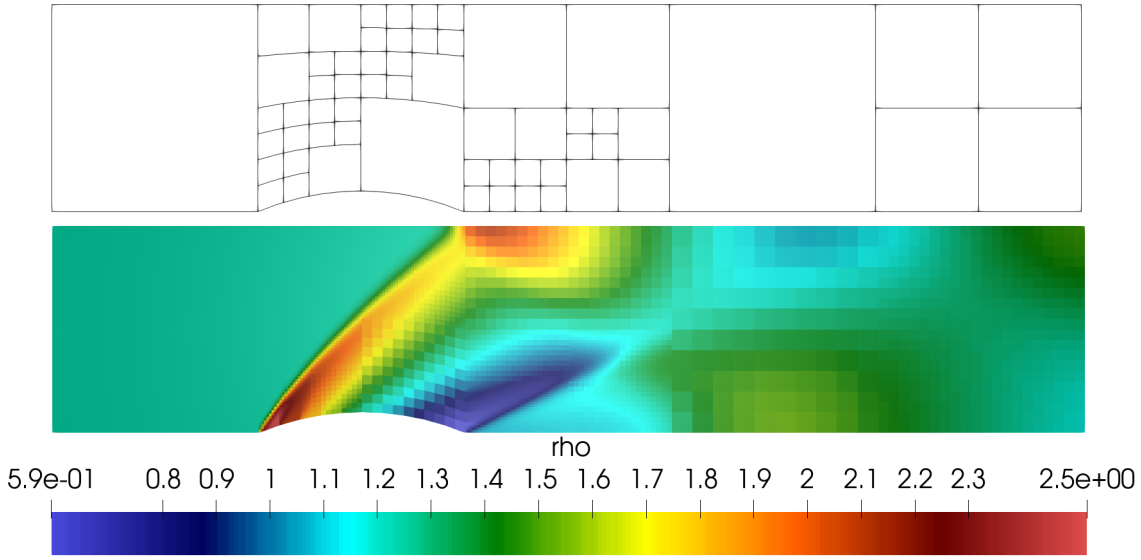


Figure 7.5: Two-Dimensional Flow with a Mach number of 2 over a bump with three levels of refinement using a finite-volume scheme.

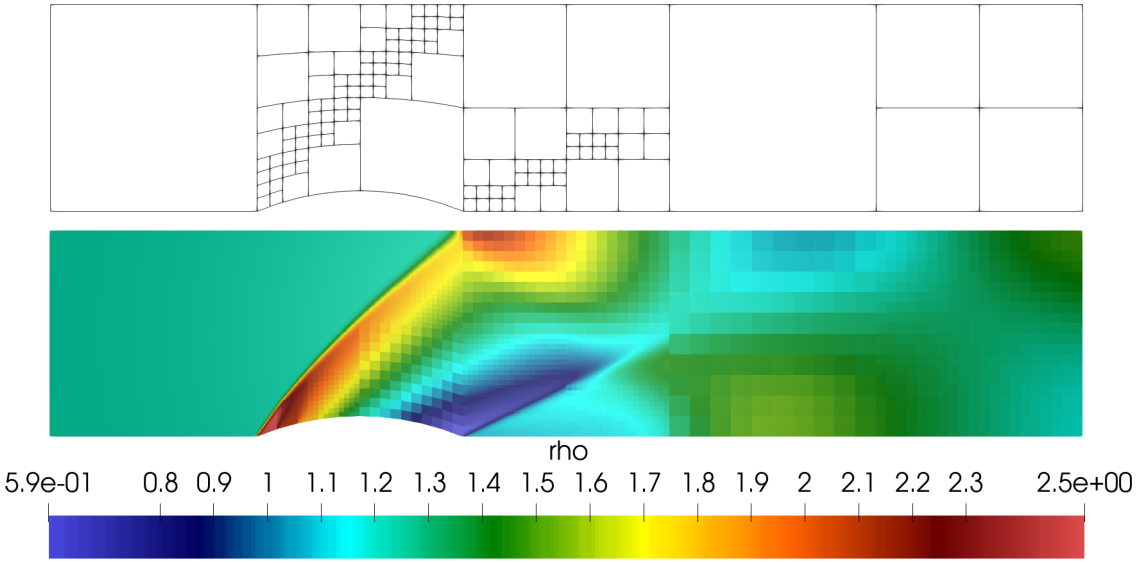


Figure 7.6: Two-Dimensional Flow with a Mach number of 2 over a bump with four levels of refinement using a finite-volume scheme.

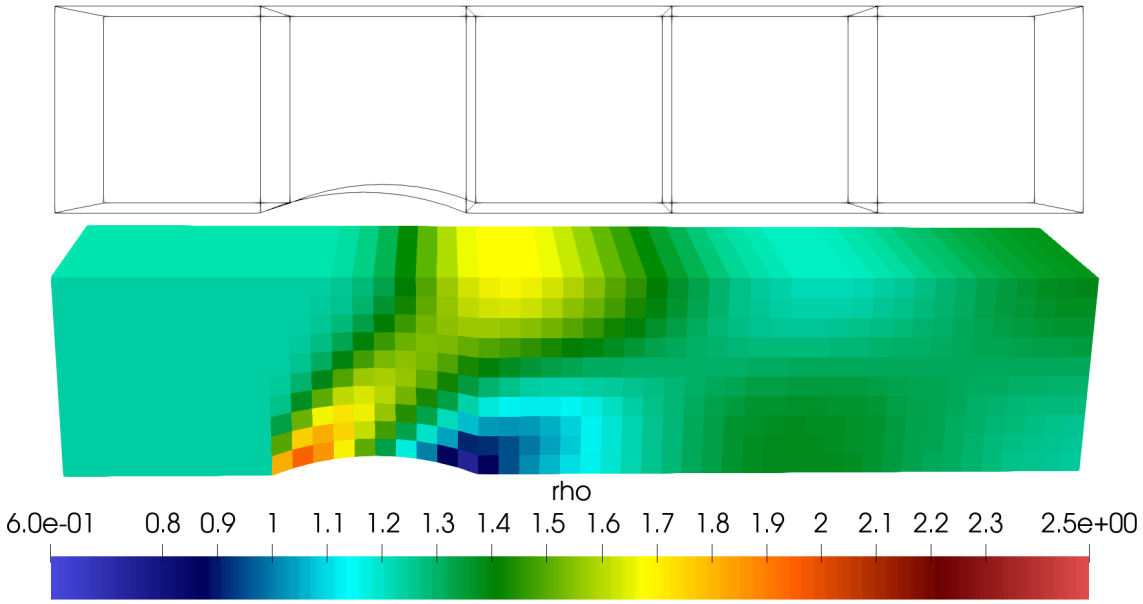


Figure 7.7: Three-Dimensional Flow with a Mach number of 2 over a bump with zero level of refinement using a finite-volume scheme.

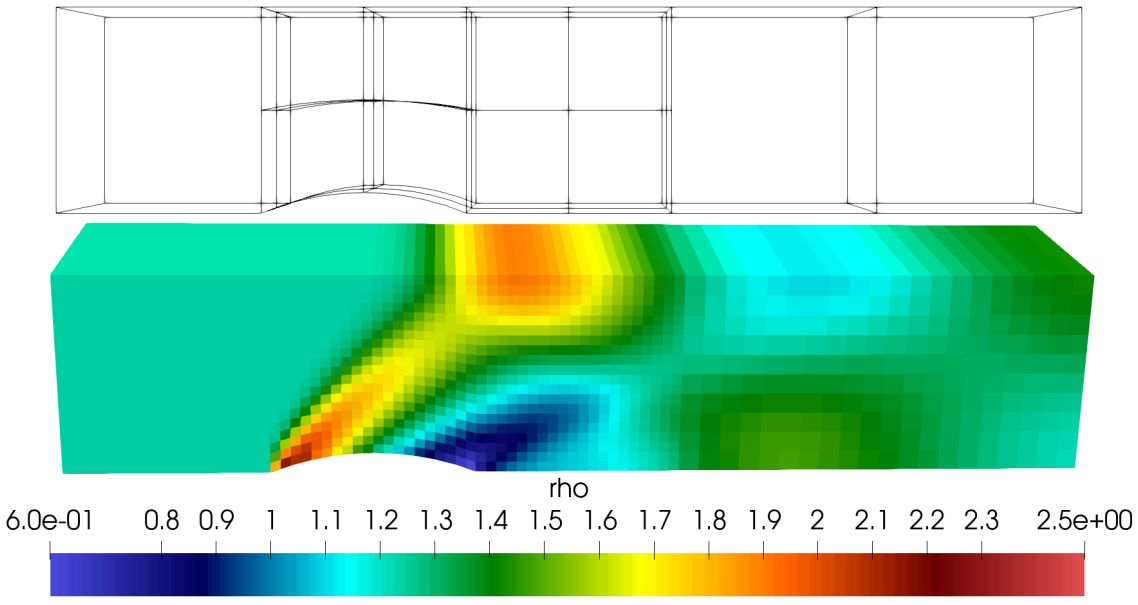


Figure 7.8: Three-Dimensional Flow with a Mach number of 2 over a bump with one level of refinement using a finite-volume scheme.

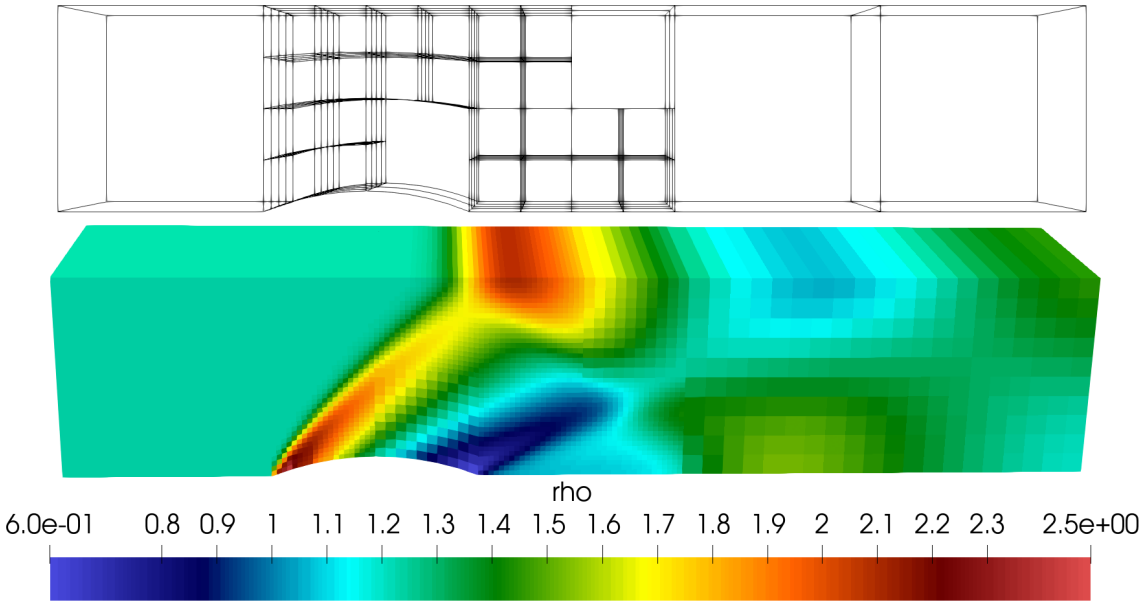


Figure 7.9: Three-Dimensional Flow with a Mach number of 2 over a bump with two levels of refinement using a finite-volume scheme.

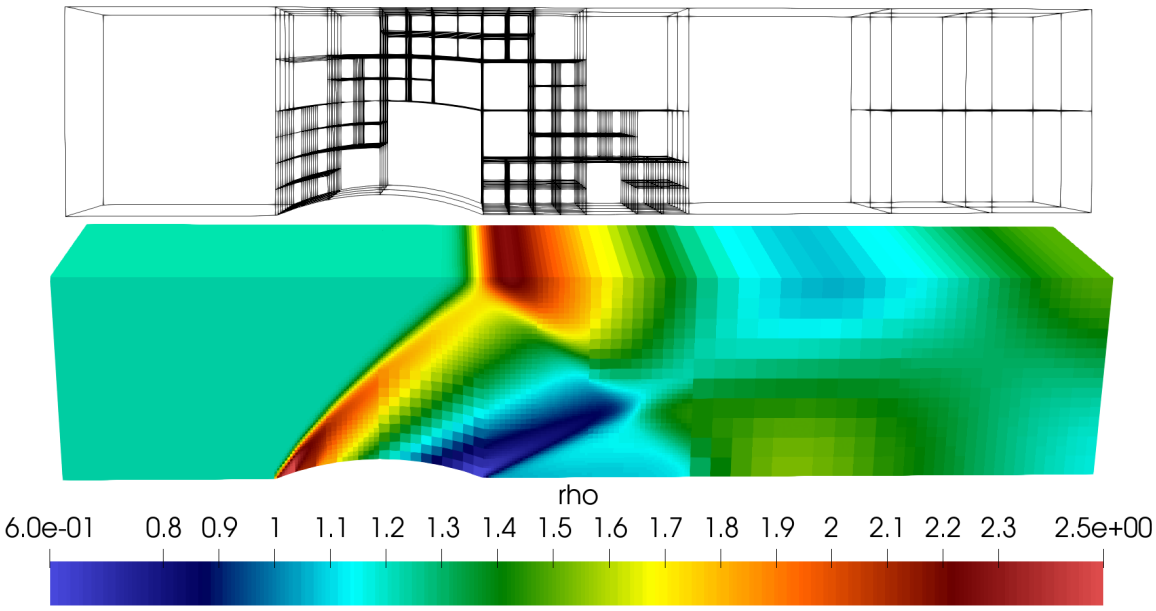


Figure 7.10: Three-Dimensional Flow with a Mach number of 2 over a bump with three levels of refinement using a finite-volume scheme.

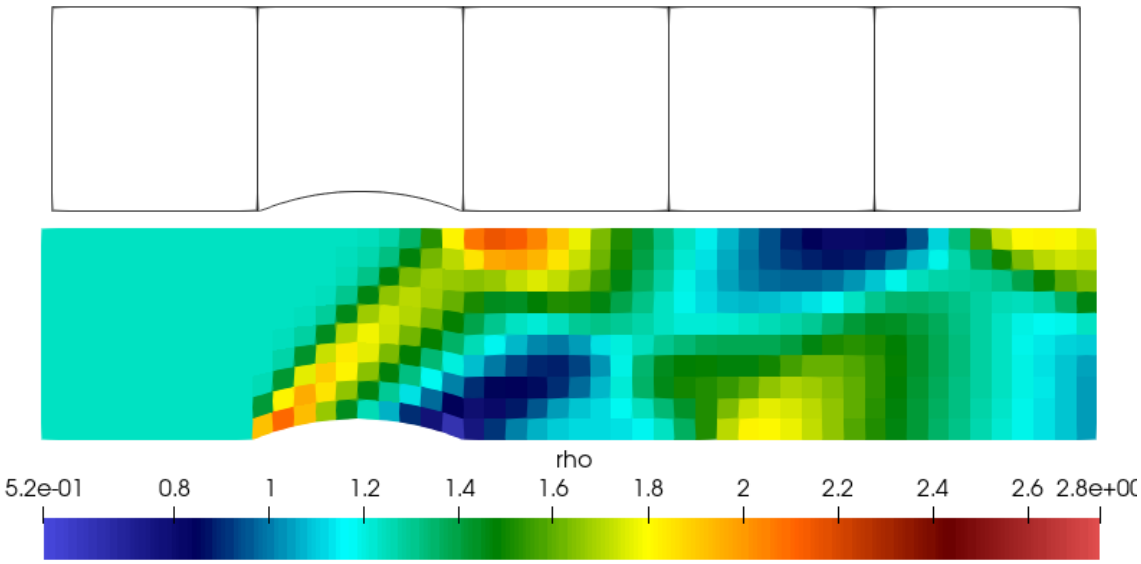


Figure 7.11: Two-Dimensional Flow with a Mach number of 2 over a bump with zero level of refinement using a discontinuous-Galerkin Hancock scheme.

shock waves, but after going through multiple stages of refinement shown in Figures 7.8 and 7.9, the final mesh, shown in 7.10, captures the shock waves much better. Once again, the flexibility of the mesh refinement scheme is demonstrated in Figure 7.10, as the left-most block communicates with 40 neighbours.

7.2.2 Results Obtained with the Discontinuous-Galerkin Hancock Scheme

While the finite-volume scheme has captured the left half of the domain relatively well, the right half of the domain, where discontinuities are still present, lacks definition. The results obtained with the discontinuous-Galerkin-Hancock scheme using the same refinement criterion as defined in Section 7.2.1 capture the discontinuities across the whole domain. The final two-dimensional mesh has 260 blocks and 260 000 cells. Again, as shown in Figures 7.11-7.15, the solution becomes more accurate as the mesh is refined. Although the refinement criteria used may not be the best to capture all of the discontinuities, this shows the versatility of the adaptive mesh refinement framework. It is completely independent from the numerical scheme and the partial differential equation used. Because this algorithm is part of an academic code, this

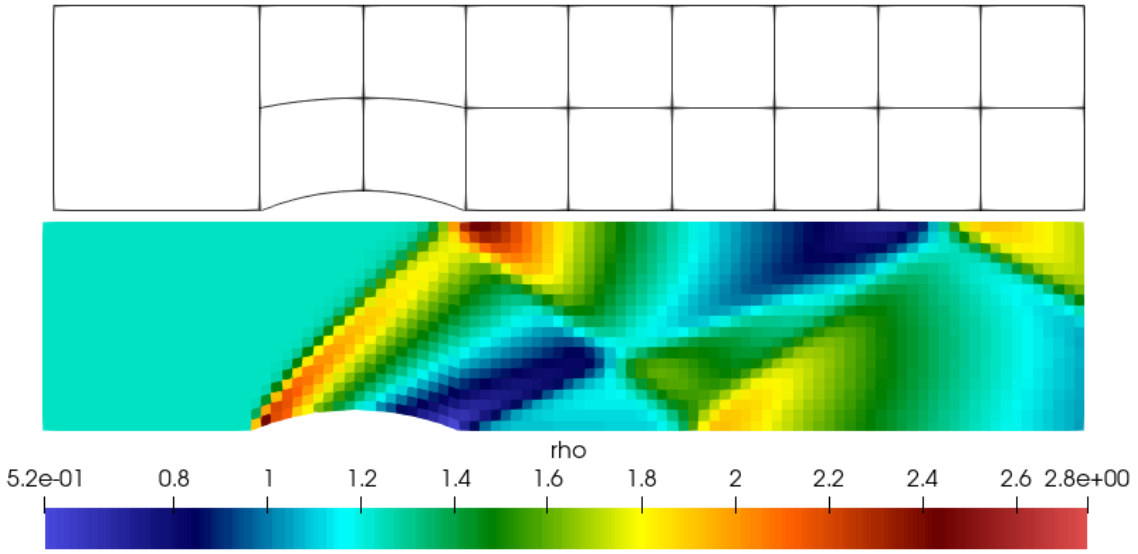


Figure 7.12: Two-Dimensional Flow with a Mach number of 2 over a bump with one level of refinement using a discontinuous-Galerkin Hancock scheme.

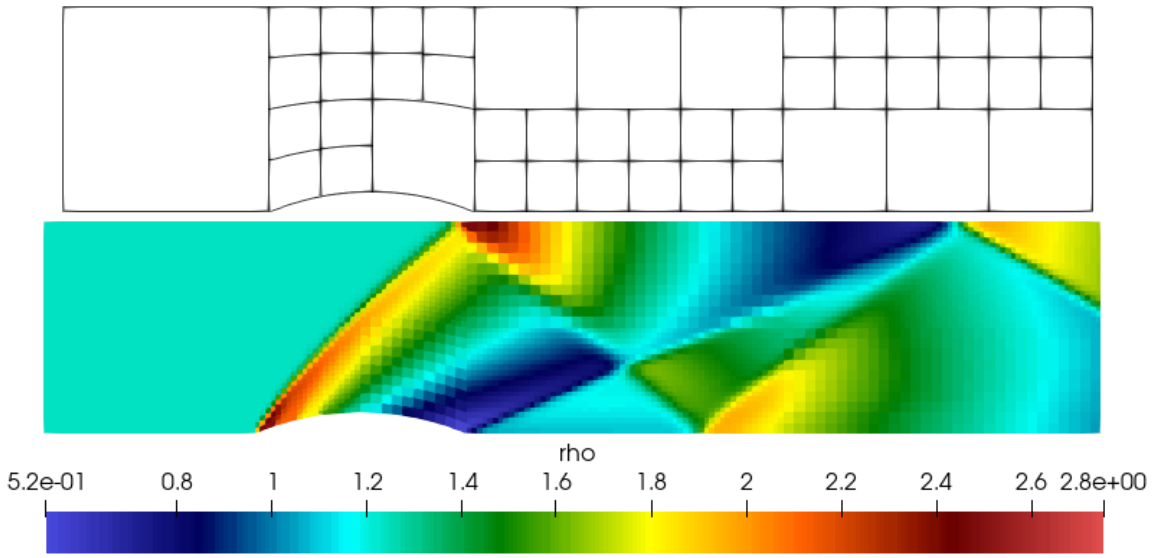


Figure 7.13: Two-Dimensional Flow with a Mach number of 2 over a bump with two levels of refinement using a discontinuous-Galerkin Hancock scheme.

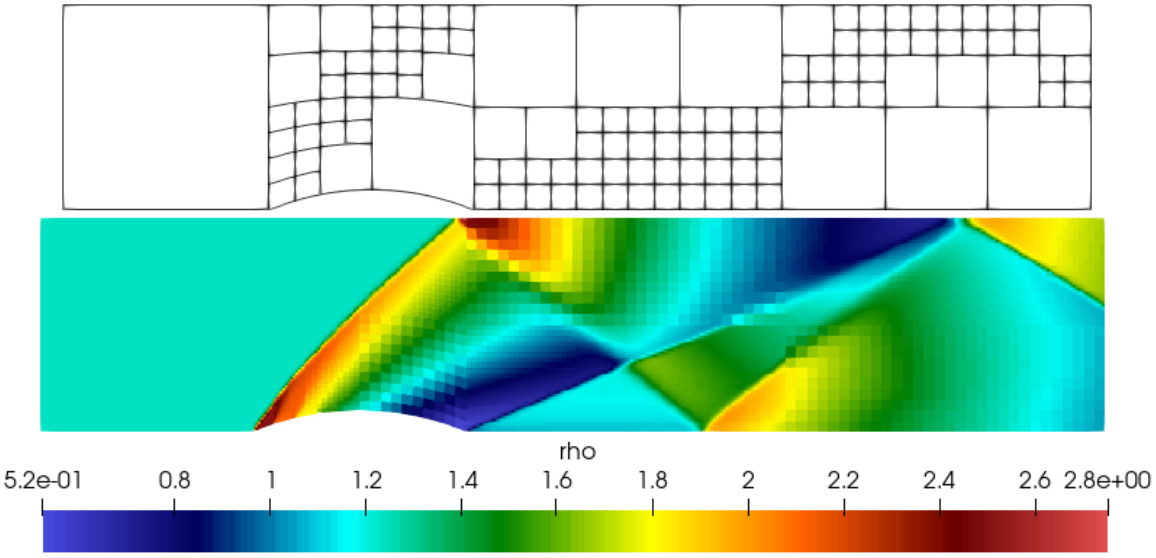


Figure 7.14: Two-Dimensional Flow with a Mach number of 2 over a bump with three levels of refinement using a discontinuous-Galerkin Hancock scheme.

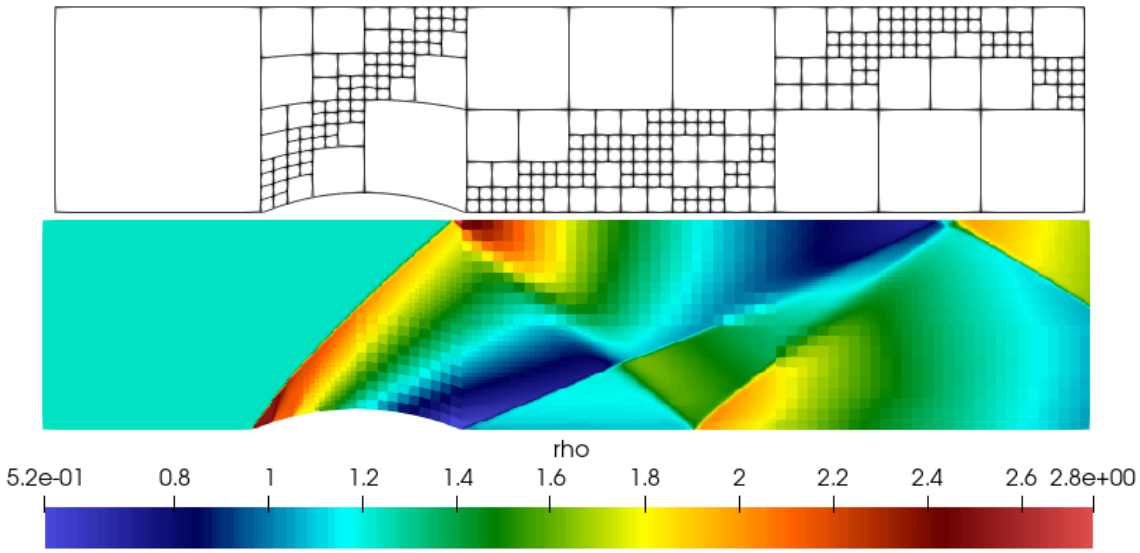


Figure 7.15: Two-Dimensional Flow with a Mach number of 2 over a bump with four levels of refinement using a discontinuous-Galerkin Hancock scheme.

is advantageous as future students will only have to write their numerical schemes or partial differential equation algorithms on one block and it will be transferable to a multiblock framework with almost no additional effort.

7.3 Polydisperse Gaussian Model

Although the polydisperse Gaussian model was not developed as a part of this thesis, obtaining the first results in three dimensions was a contribution of this work, as only one-dimensional results were previously obtained. The first step is a space homogeneous problems, for which an exact solution was available. Then a three-dimensional shell of particles is given an outward velocity, to replicate the detonation of a radiological dispersal device, and subjected to gravity, as well as a background wind. The following results were obtain using the equations of Section 3.3.3.

7.3.1 Space Homogeneous Problem

The third case considered is a space homogeneous case. The goal of this problem is to verify the implementation of the source term, and ensure it agrees with the available exact solution of the full kinetic equation for a quiescent background flow. The transport equation, defined as

$$\frac{\partial \mathcal{F}}{\partial t} + v_i \frac{\partial \mathcal{F}}{\partial x_i} = -\frac{\partial}{\partial v_i}\left(\frac{V_i - v_i}{\tau}\right) - \frac{\partial}{\partial v_i}(\phi_i \mathcal{F}), \tag{7.12}$$

can be difficult to solve analytically, but when the background flow velocity is zero and the initial particle distribution is given by

$$\mathcal{F}_0(x_i, v_i, d), \tag{7.13}$$

it is possible to obtain a closed-form solution. The exact solution to the kinetic equation is given by

$$\mathcal{F}(x_i, v_i, d, t) = A\mathcal{F}_0(B_i, C_i, d), \tag{7.14}$$

where $A = \exp\left(\frac{3t}{\tau}\right)$, $B_i = x_i + \tau\left((v_i + \phi_i\tau)(1 - \exp\left(\frac{t}{\tau}\right)) - \phi_i t\right)$ and $C_i = (v_i + \phi_i\tau)\exp\left(\frac{t}{\tau}\right) + \phi_i t$ [8]. This exact solution is included in Figures 7.16-7.18 as a reference only, the model is expected to have some modelling errors. The curves representing the exact solution are denoted as $u(t)$ exact, $\Theta_{xx}(t)$ exact and $\Psi_{xd}(t)$ exact.

It is desired to compare the results obtained with the implementation of the PGM source term with the derived analytical solution for the system of Eqs. (3.41)-(3.46), to observe numerical errors. The primitive variables n, μ and Ψ_{dd} remain constant in time,

$$n(t) = n(0) \tag{7.15}$$

$$\mu(t) = \mu(0) \tag{7.16}$$

$$\Psi_{dd}(t) = \Psi_{dd}(0). \tag{7.17}$$

Based on this, it is possible to derive equations for the evolution of $u(t)$, Θ_{xx} and Ψ_{xd}. The full derivation of the following equations was done by Forgues et al. [8].

$$u(t) = \frac{\mathcal{V}_0 + \mathcal{V}_1}{4\alpha} \tag{7.18}$$

$$\Psi_{xd}(t) = \frac{(\lambda_0 + 1)\mathcal{V}_0 + (\lambda_1 + 1)\mathcal{V}_1}{8\alpha} \tag{7.19}$$

$$\Theta_{xx}(t) = \frac{1}{\Psi_{dd}}\left(\left(\Theta_{xx}(0)\Psi_{dd} - \Psi_{xd}(0)^2\right)\exp\left(-2\frac{t}{\tau_G}\right) + \Psi_{xd}(t)^2\right) \tag{7.20}$$

where

$$\lambda_{0,1} = -(1 + 2\Psi_{dd} \pm 2\alpha), \tag{7.21}$$

$$\alpha = \sqrt{\Psi_{dd}(\Psi_{dd} + 1)}, \tag{7.22}$$

$$C_0 = -(\lambda_1 + 1)\left(\frac{V_x}{\tau_G} + \phi_x\right) - 4\frac{V_x\Psi_{dd}}{\tau_G}, \tag{7.23}$$

$$C_1 = -(\lambda_0 + 1)\left(\frac{V_x}{\tau_G} + \phi_x\right) + 4\frac{V_x\Psi_{dd}}{\tau_G}, \tag{7.24}$$

$$\mathcal{V}_0 = \left(2\Psi_{xd}(0) - (\lambda_1 + 1)u(0) + \frac{C_0\tau_G}{\lambda_0}\right)\exp\left(\frac{\lambda_0 t}{\tau_G}\right) - \frac{C_0\tau_G}{\lambda_0}, \tag{7.25}$$

$$\mathcal{V}_1 = \left(-2\Psi_{xd}(0) + (\lambda_0 + 1)u(0) + \frac{C_1\tau_G}{\lambda_1}\right)\exp\left(\frac{\lambda_1 t}{\tau_G}\right) - \frac{C_1\tau_G}{\lambda_1}. \tag{7.26}$$

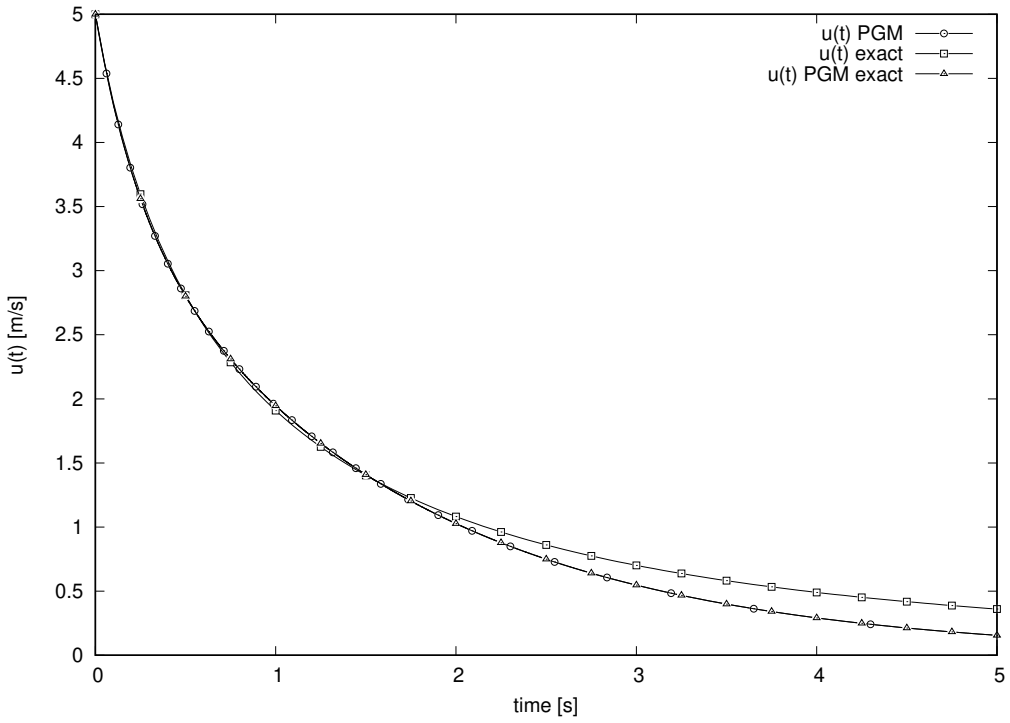


Figure 7.16: Exact and PGM evolution of the average particle velocity in a space homogeneous problem.

All of the particle in the domain of the simulation have an initial average velocity of $5.0\,\mathrm{m/s}$, a variance $\Theta_{xx} = 1.0\,\mathrm{m^2/s^2}$, and no covariance, $\Psi_{xd} = 0$. The average logarithm of the diameters of the particles is $\mu = \ln(28 \times 10^{-6})$ and $\Psi_{dd} = 0.25$ [8]. The background fluid is air, with a density of $1.225\,\mathrm{kg/m^3}$ and dynamic viscosity of $1.81 \times 10^{-5}\,\mathrm{kg/ms}$. The particle density in Eq. (2.3) is chosen such that $\tau = 1.0\,\mathrm{s}$. The source term is the only thing affecting the evolution of the distribution function of particles, because the state is the same across the whole domain. The source term will relax the particle velocity due to particle drag.

As shown in Figure 7.16, the particles relax to a terminal velocity due to the buoyancy, drag and gravity forces. The covariance, shown in Figure 7.18, remains positive, meaning that the smaller particles decelerate faster than the bigger particles. This is also reflected in the variance, shown in Figure 7.17, as it initially increases. As shown in Figures 7.16-7.18, the curves obtained with the PGM, denoted $u(t)$ PGM, $\Theta_{xx}(t)$ PGM and $\Psi_{xd}(t)$ PGM closely follow the curves representing the exact

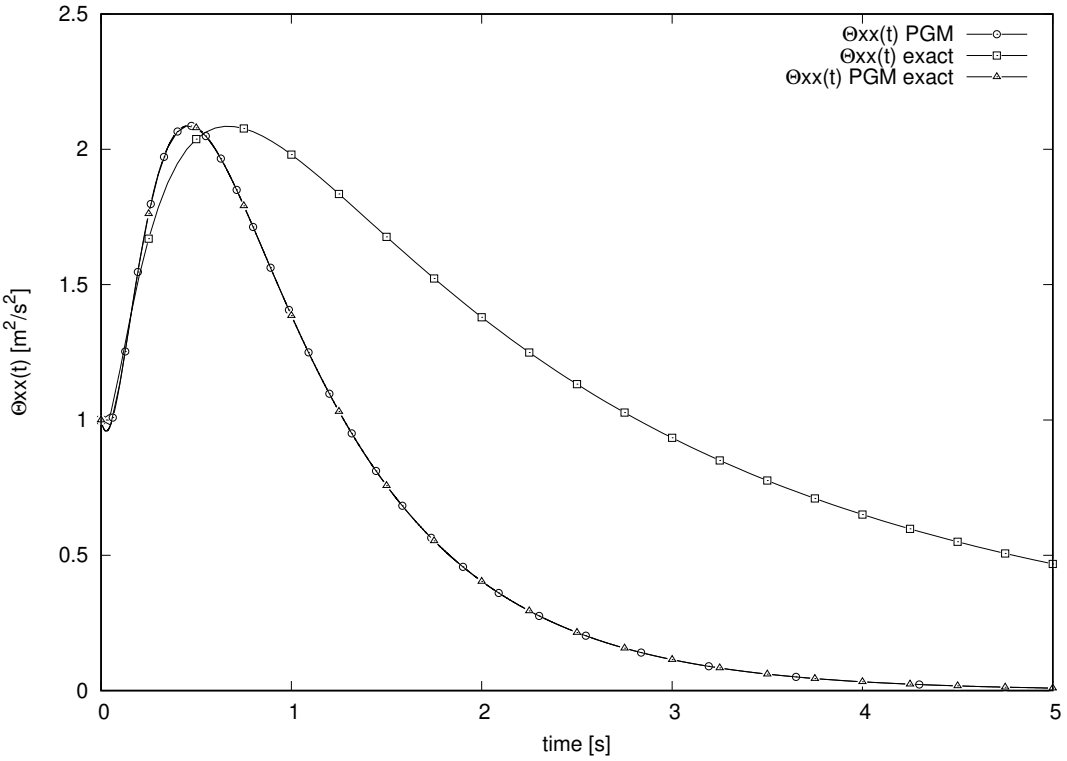


Figure 7.17: Exact and PGM evolution of the variance in a space homogeneous problem.

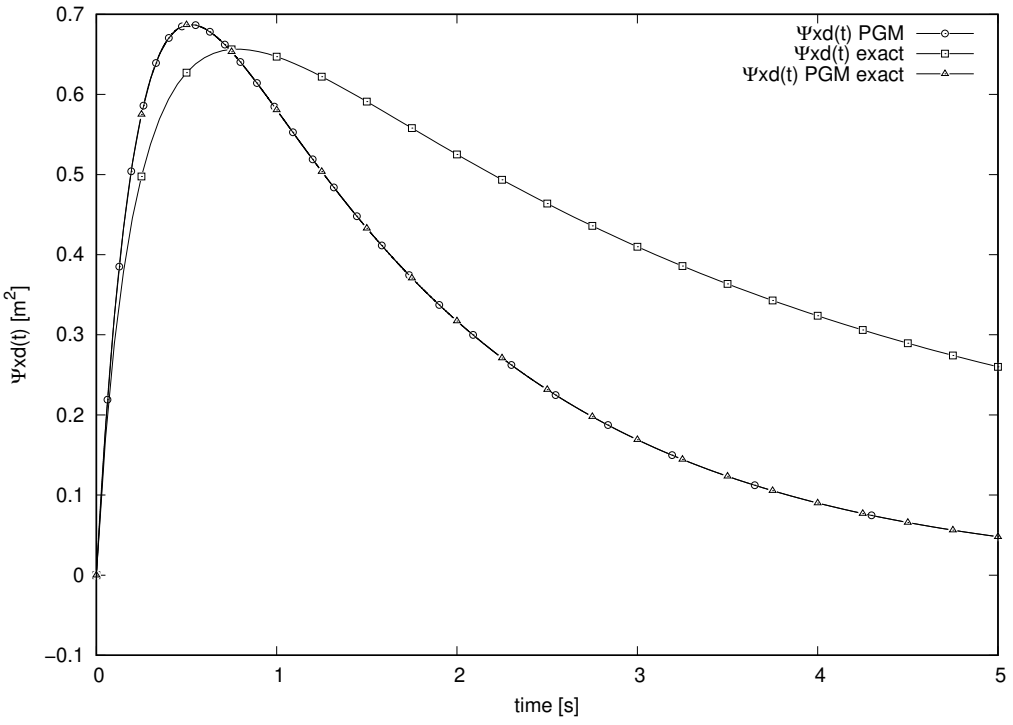


Figure 7.18: Exact and PGM evolution of the covariance in a space homogeneous problem.

solution to the PGM, proving that there is little to no numerical error and that the implementation of the source term is accurate. The discrepancies between the curves obtained with the PGM and the exact solution to the kinetic equations are due to modelling error, not numerical error.

7.3.2 Three-Dimensional Shell of Particles

To model RDDs, the fourth case considered is a shell of particles that are given an outward velocity and are subjected to drag, gravity and buoyancy. The domain considered is a rectangular prism, where $-12.0\,\mathrm{m} < x < 12.0\,\mathrm{m}$, $-2.0\,\mathrm{m} < y < 2.0\,\mathrm{m}$ and $-14.0\,\mathrm{m} < z < 10.0\,\mathrm{m}$. A spherical shell is introduced at the origin, with a minimum radius of 0.3 m and a maximum radius of 0.63 m, as shown in Figure 7.19. Inside the thickness of the shell, the number density of particle is 300000.0 particles per cubic meter and the norm of the average velocity of the particles is 24.0 m/s and is oriented radially outwards. Outside the shell the particle density is 0.0001 particles per cubic meter and the particles do not have a velocity. In the whole domain, the variance in particles velocity is set to $1.0\,\mathrm{m}^2/\mathrm{s}^2$ and there is no covariance. The average logarithm of the diameters of the particles is $\mu = \ln(28 \times 10^{-6})$ and $\Psi_{dd} = 0.25$ [8]. The background fluid is air, with a density of $1.225\,\mathrm{kg/m}^3$ and dynamic viscosity of $1.81 \times 10^{-5}\,\mathrm{kg/ms}$. The particle density in Eq. (2.3) is chosen such that $\tau = 1.0\,\mathrm{s}$. The background fluid has a velocity in the x-direction of 8.0 m/s. Each block has 20 cells in all directions. The CFL number used is 0.4 and the final time of the calculation is $t = 0.5\,\mathrm{s}$.

As shown in Figure 7.19, the shell is located in the centre of the domain. More blocks have been created in this region to properly capture the initial conditions. Then, the case is run using the adaptive mesh refinement algorithm of Section 6. The refinement criterion chosen is based on the particle number density and the criterion is evaluated every 150 iterations. If some cells in a block have a particle number density that is higher than 550.0 particles per cubic meter, the block is flagged for refinement. As shown in Figure 7.20, blocks are added where the particle number

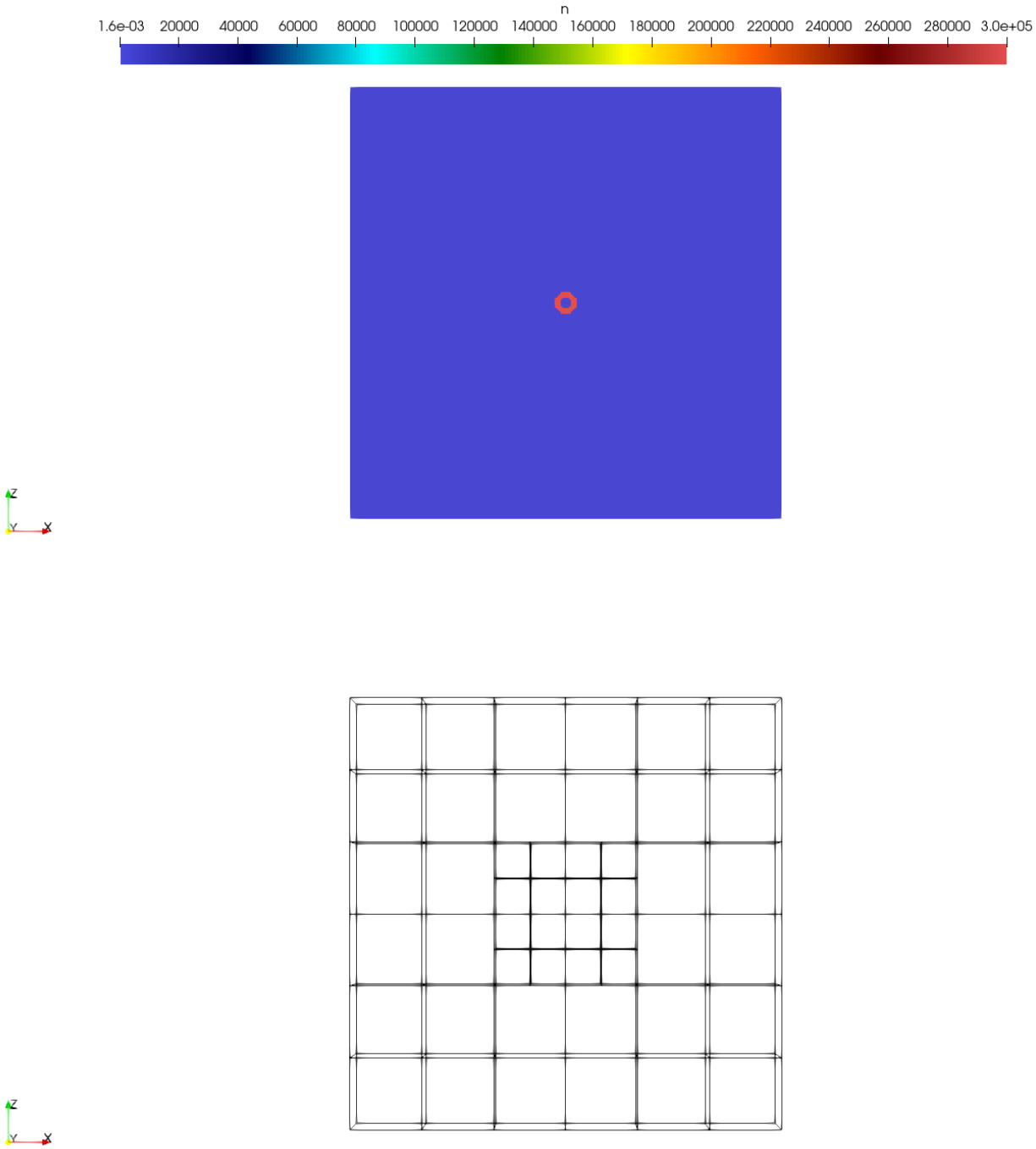


Figure 7.19: Initial condition mesh of the particle number density in a shell of particles problem.

density is high. As expected, more blocks are added towards the centre of the domain, as the particles initially start there. The final number of blocks is 344, for a total number or cells of 2752000. Figures 7.19 and 7.20 represent a cut in the x-z plane of the three-dimensional domain.

Initially, the spherical shell is in the centre of the domain, and then expands radially, as shown in Figure 7.21 due to the initially outwards velocity of the particles. The lack of circular symmetry observed is due to the gravity, drag and buoyancy forces and the background wind velocity. There are more particles to the right of the due to the left-to-right background wind. The left side of the ring is thicker than the right side also because of the background velocity spreading particles of different sizes.

As shown in Figure 7.21, there are essentially no particles in the corners and centre of the domain. Because of this, when the particle density is below 5.0 particles per cubic meters, the value of the particles' velocity is set to 0.0, in order to be able to observe the data where the particle density is significant. As shown in Figure 7.22, the particles with the fastest x-velocity component are concentrated towards the extreme left and right sides of the domain. There is a circular region of particles in the centre of the domain that have reached the background wind velocity. This is due to the fact that in the centre of the shell, the particles had no initial velocity and have accelerated to the background velocity. As shown in Figure 7.23, the majority of the particles have a negative z-velocity component, meaning that most particles are moving towards the bottom of the domain. The maximum z-velocity component is less than the initial outwards velocity due to the gravity that is pulling the particles downwards.

As shown in Figure 7.21, there are essentially no particles in the corners and center of the domain, which leads to high values of variance and covariance in the velocity components of particles in the corners and centre of the domain. Because of this, when the particle density is below 5.0 particles per cubic meters, the value of variance and covariance is set to $0.0\,\mathrm{m}^2/\mathrm{s}^2$, in order to be able to observe the data where the particle density is significant. Figures 7.24 and 7.26 show that the local variance in the x-

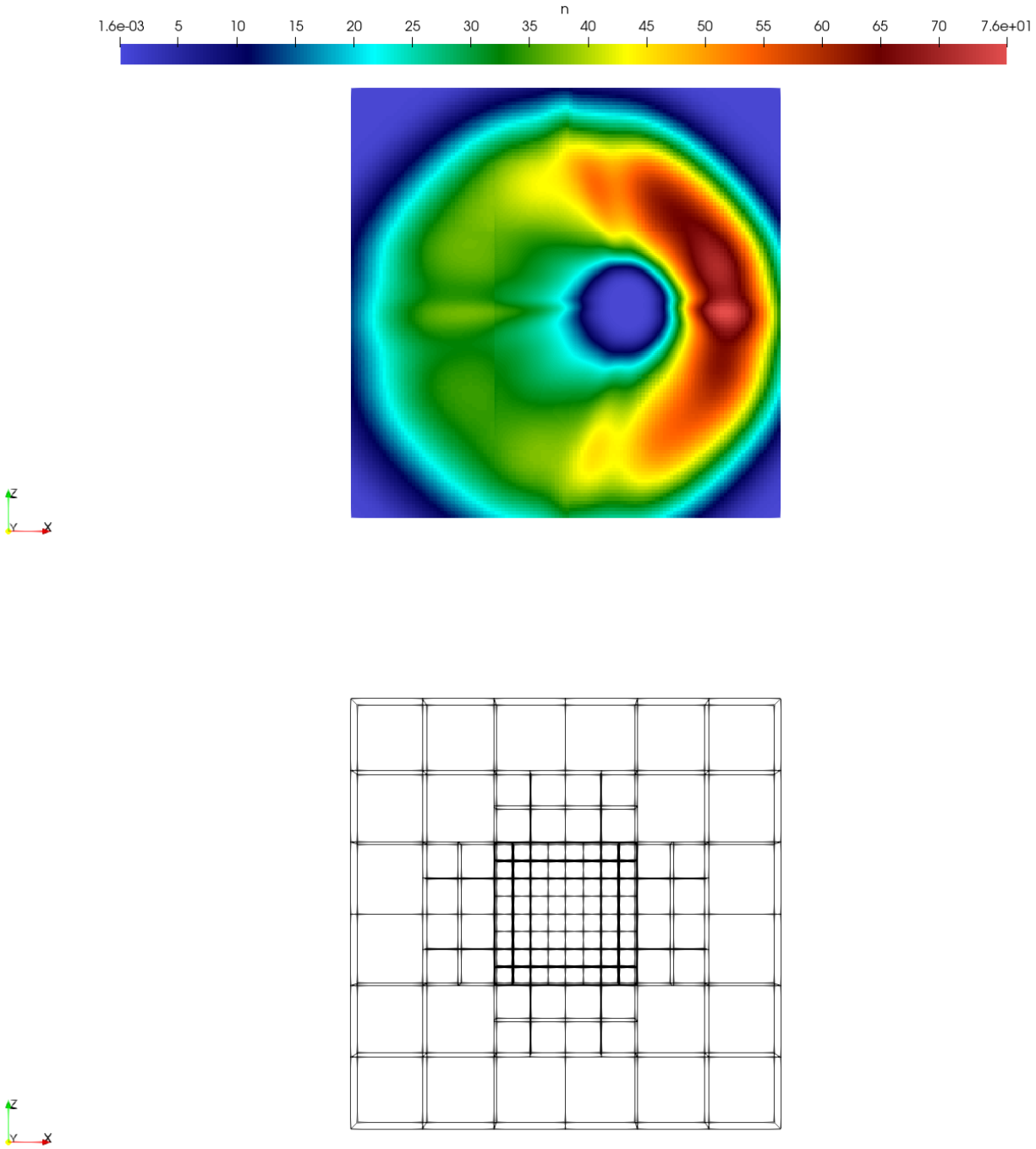


Figure 7.20: Particle number density mesh in a shell of particles problem after 0.5 second.

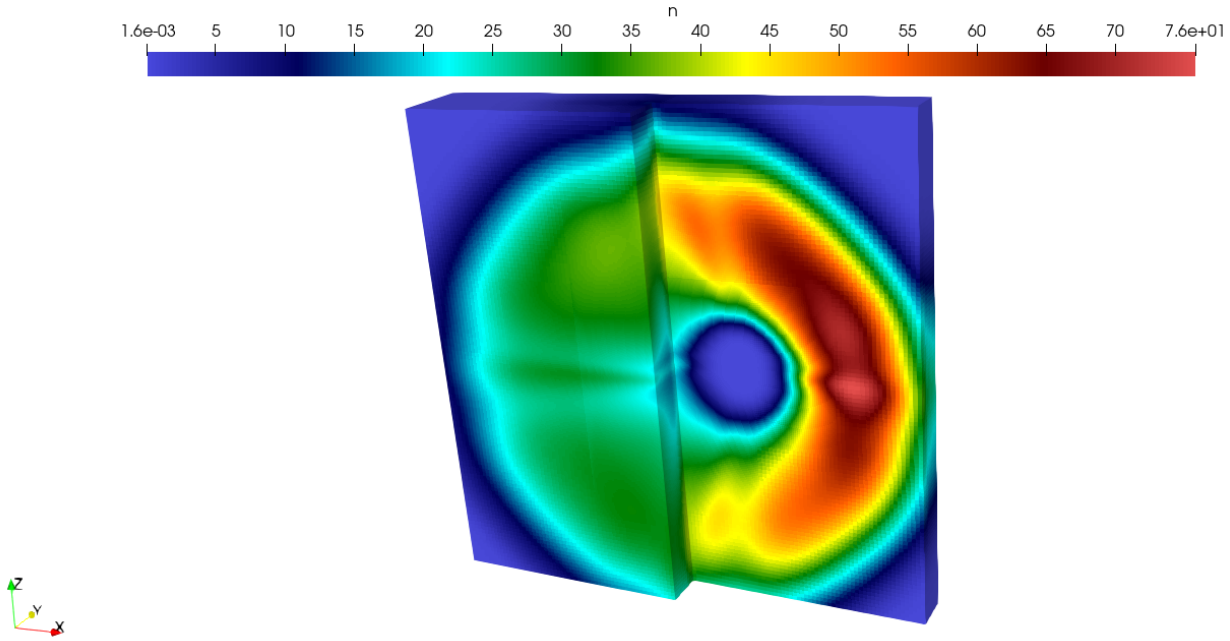


Figure 7.21: Particle number density in a shell of particles problem after 0.5 second.

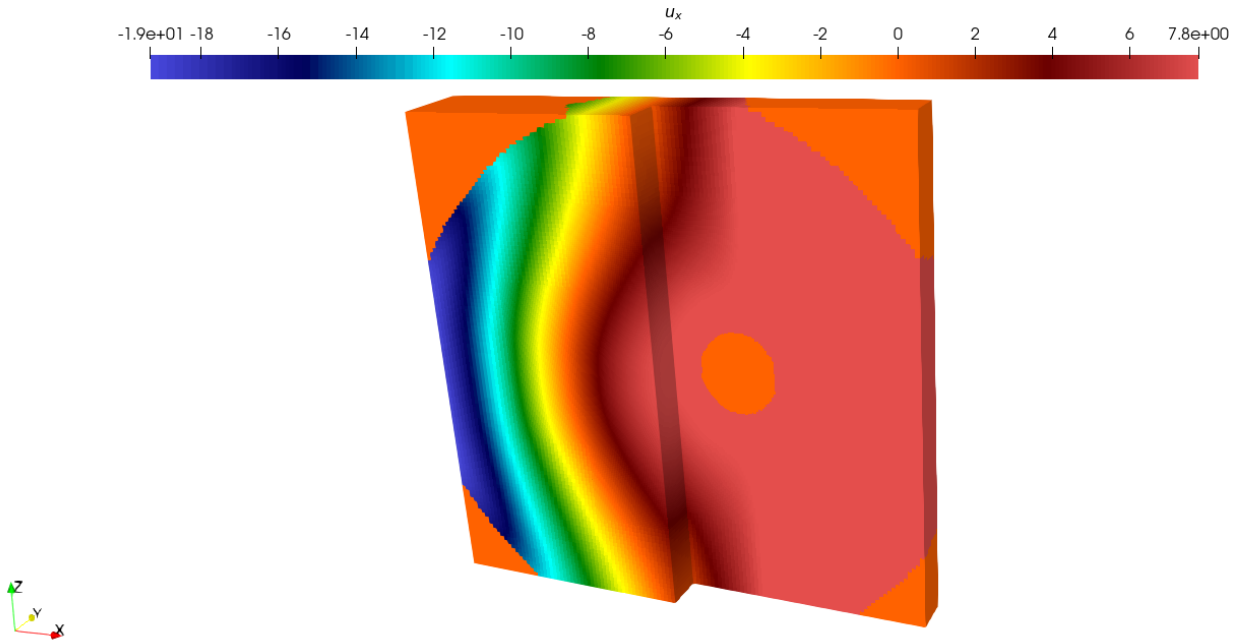


Figure 7.22: Velocity component in the x-direction in a shell of particles problem after 0.5 second.

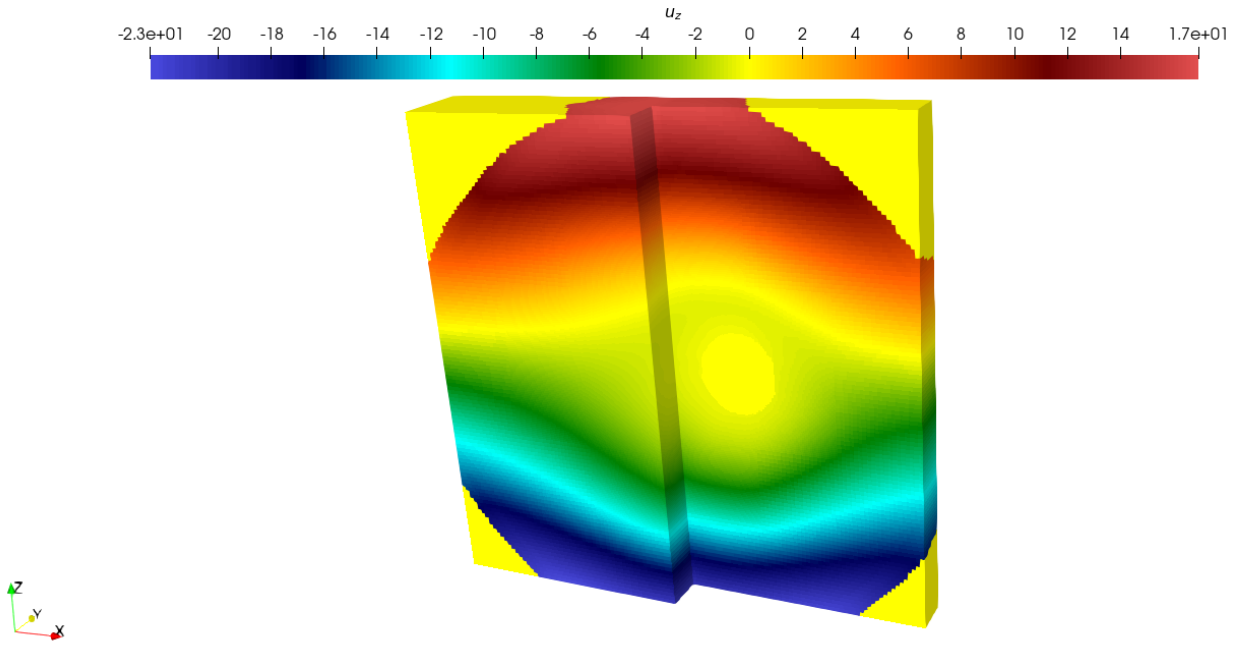


Figure 7.23: Velocity component in the z-direction in a shell of particles problem after 0.5 second.

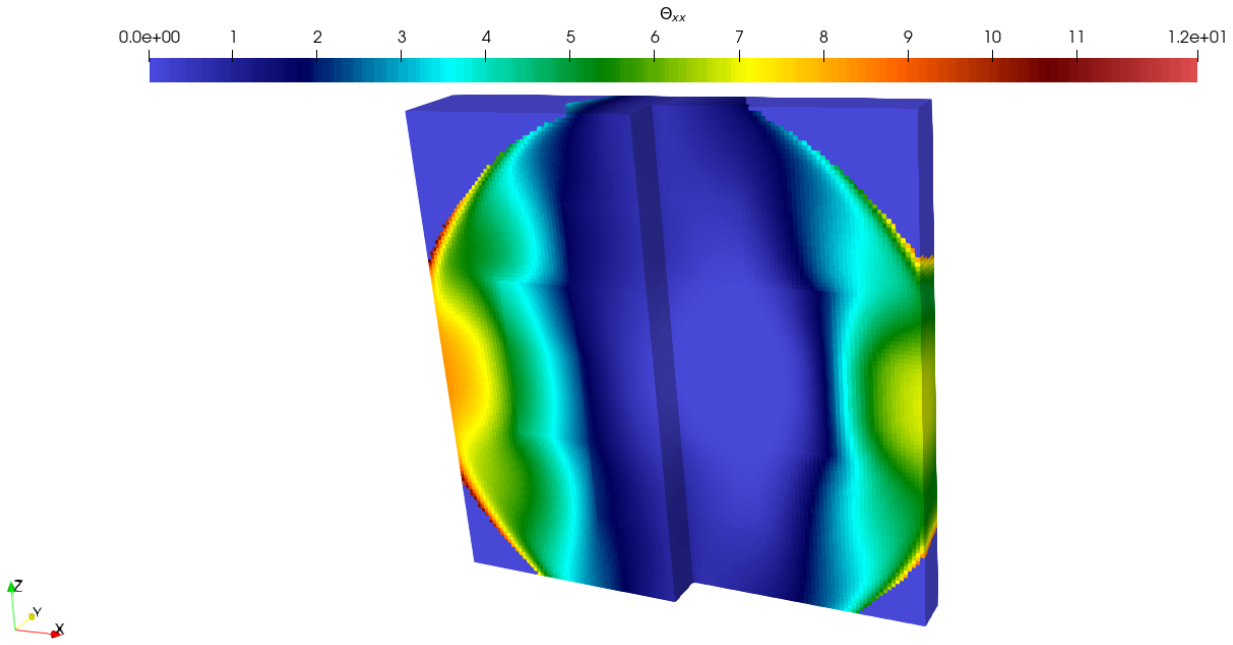


Figure 7.24: Local variance in the x-direction velocity component in a shell of particles problem after 0.5 second.

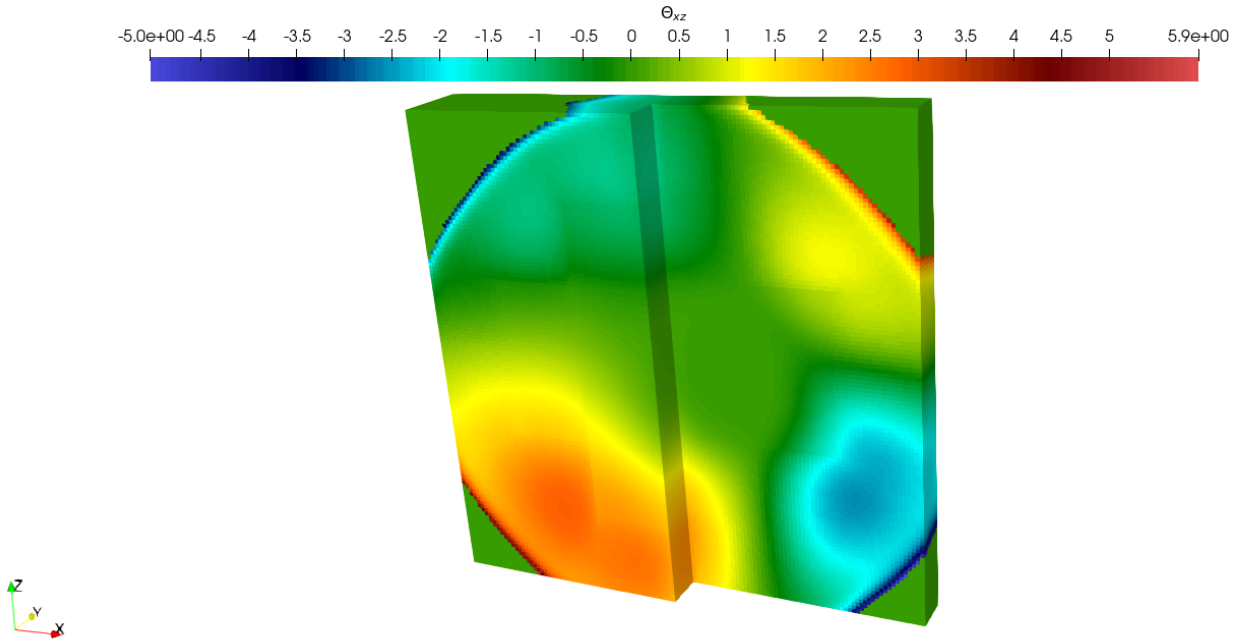


Figure 7.25: Local covariance between the x-direction z-direction velocity components in a shell of particles problem after 0.5 second.

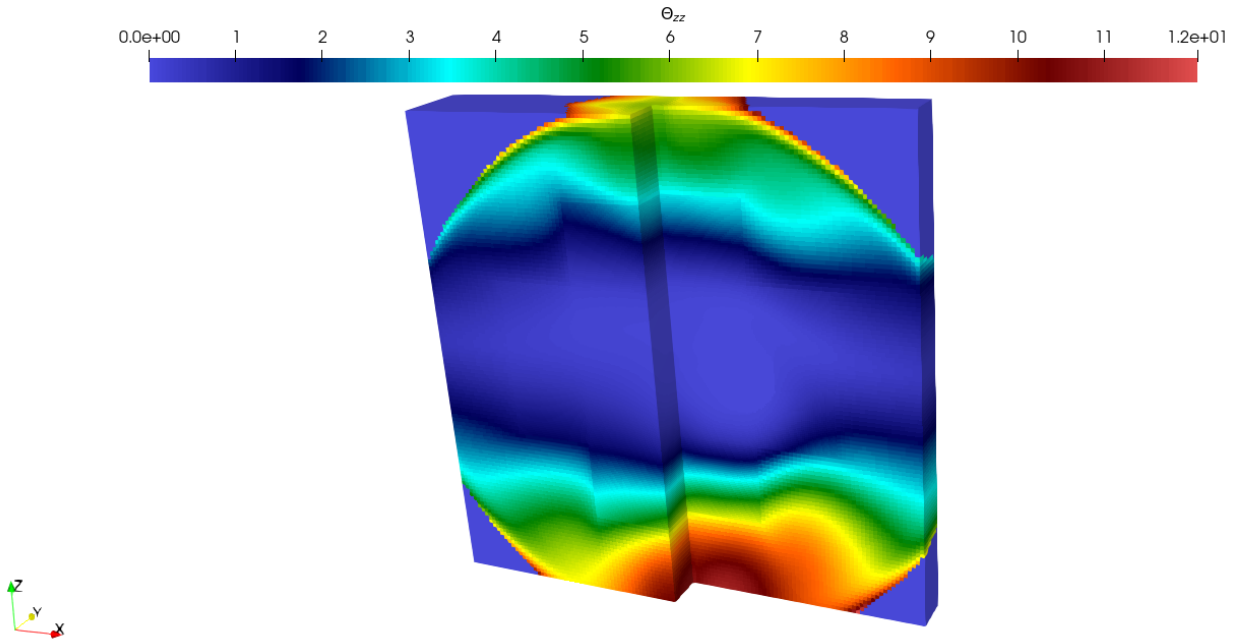


Figure 7.26: Local variance in the z-direction velocity component in a shell of particles problem after 0.5 second.

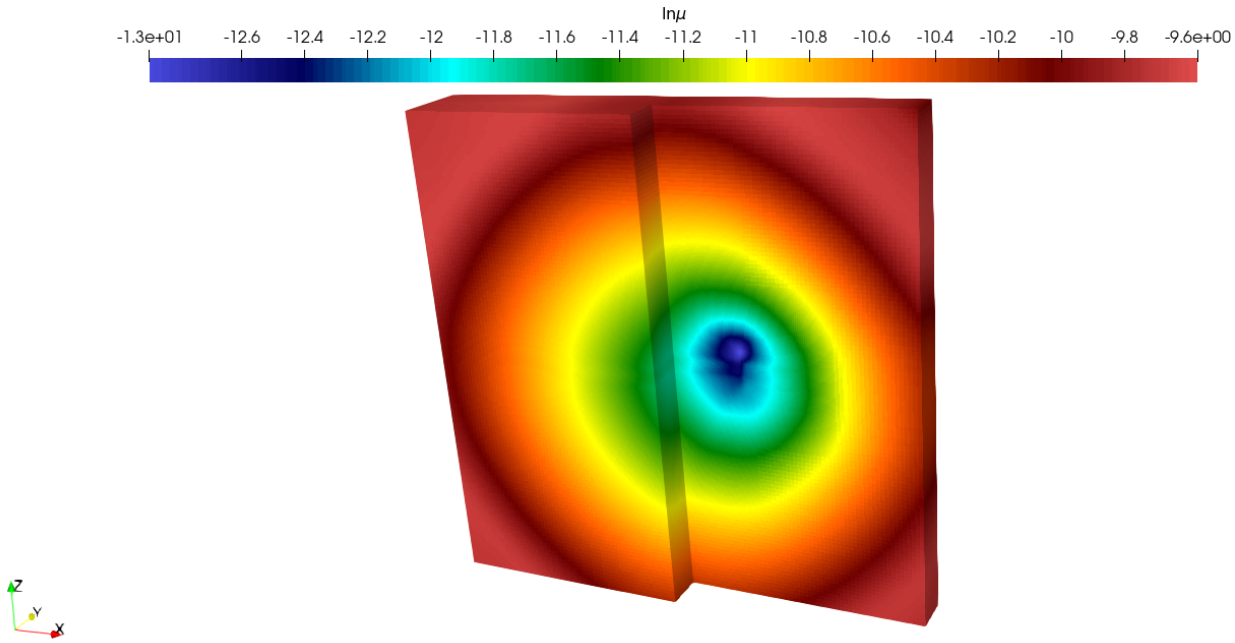


Figure 7.27: Local average logarithm of the particle size in a shell of particles problem after 0.5 second.

direction and z-direction velocity components are positive across the whole domain. As shown in Figure 7.24, the local variance is highest at the left and right edges of the domain. As shown in Figure 7.26, the local variance is highest at the upper and lower edges of the domain. This means that in these locations, the velocity of the particles are further from the local average. This is expected because particles with a higher velocity will travel further in the domain and is consistent with Figures 7.22 and 7.23. Where the local variance is the smallest, the particles mostly have x-direction and z-direction velocity components that are close to the local average. Figure 7.25 shows that the maximum positive and negative local covariance between the x-direction and z-direction velocity components are reached towards the corners of the domain. This makes sense because the particles that have large x-direction velocity component are more likely to have large z-direction velocity components. This is also supported by Figures 7.22 and 7.23, where the maximum particle velocities are reached at the top, bottom, left and right of the domain.

Figure 7.27 shows the average logarithm of the particle diameters. To obtain the

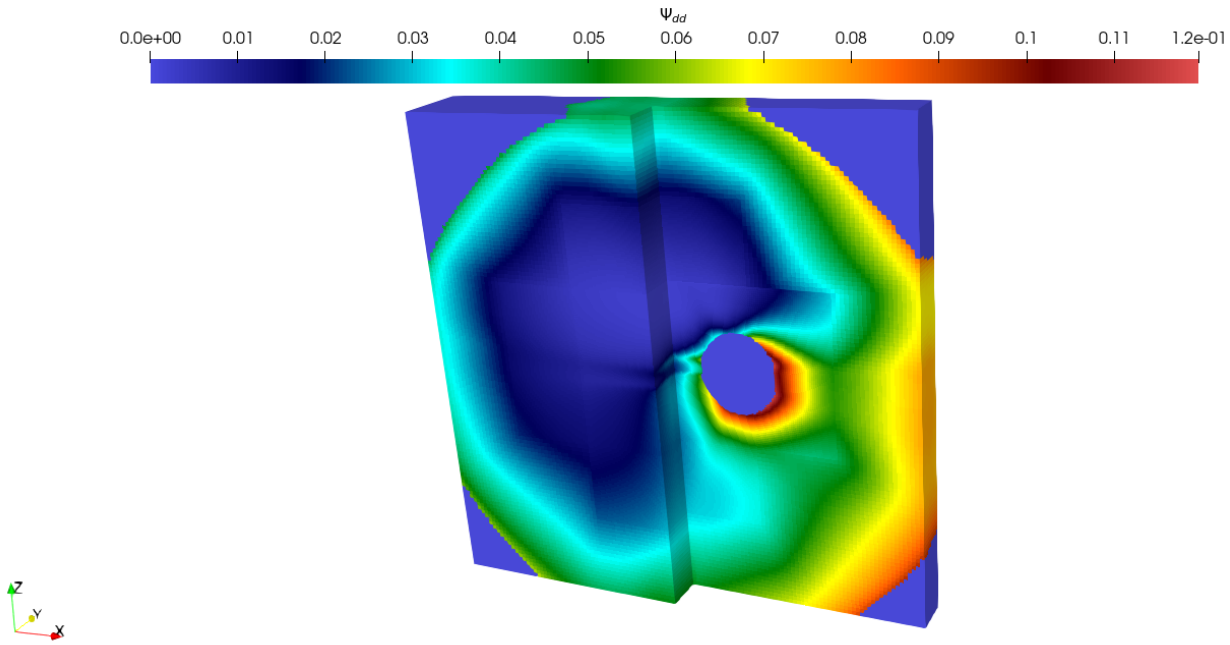


Figure 7.28: Local variance in the particle size in a shell of particles problem after 0.5 second.

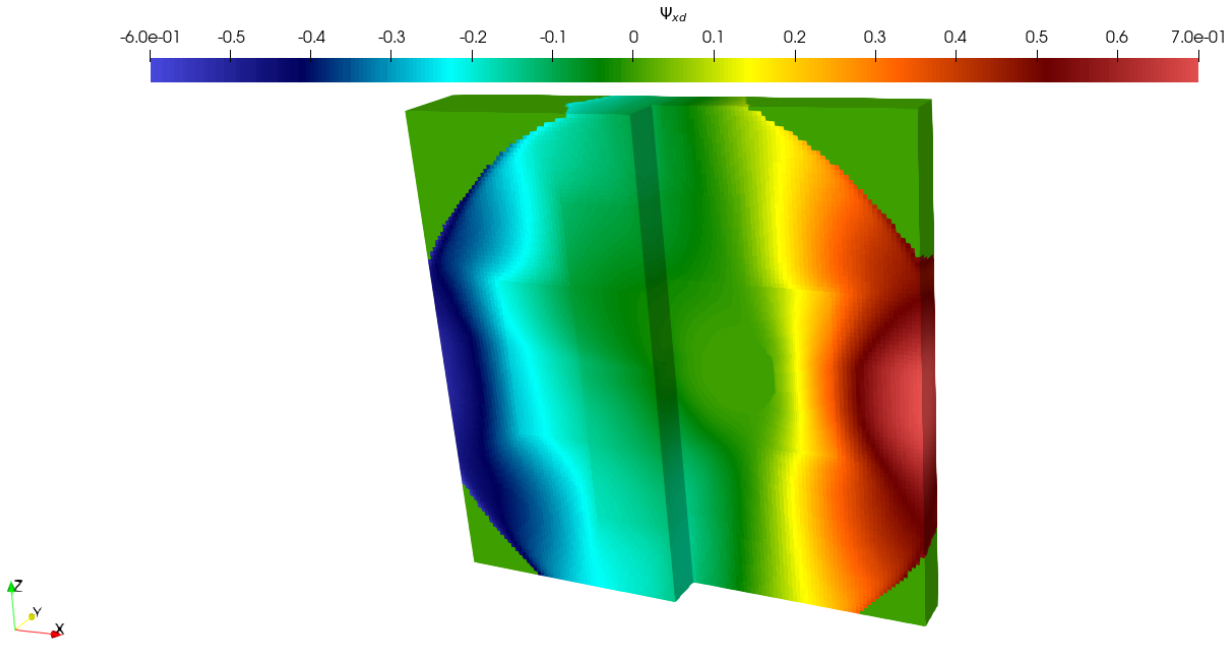


Figure 7.29: Local covariance between the particle size and the x-direction velocity component in a shell of particles problem after 0.5 second.

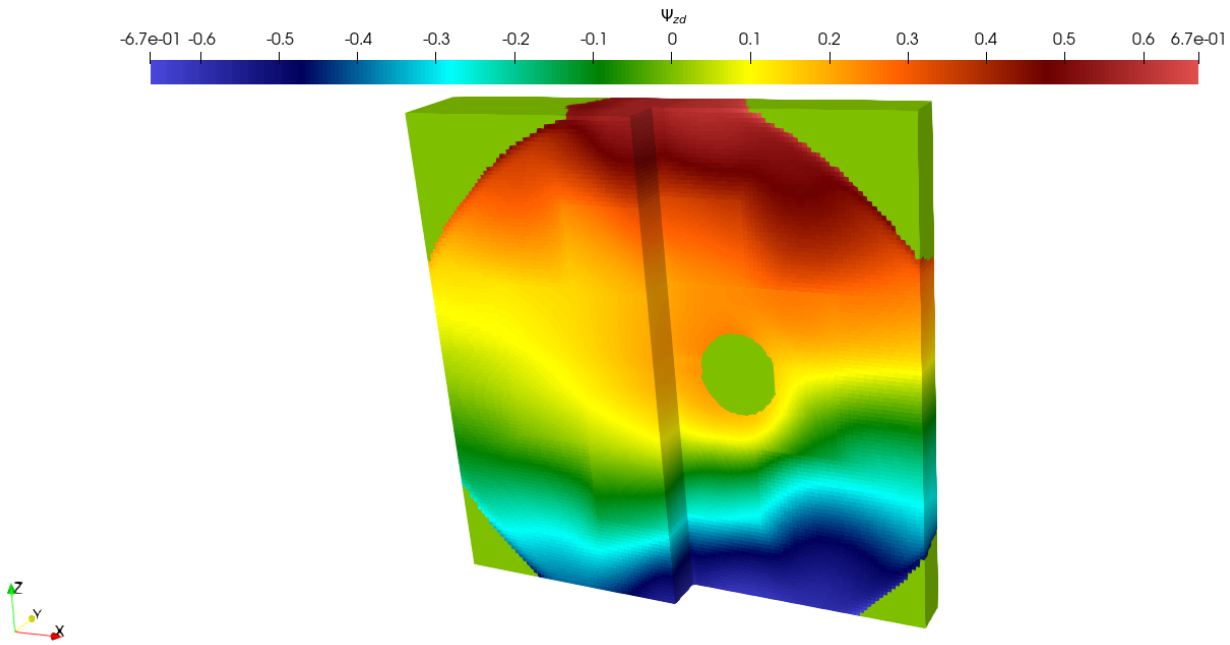


Figure 7.30: Local covariance between the particle size and the z-direction velocity component in a shell of particles problem after 0.5 second.

actual average particle diameter, $\exp(\mu)$ is used. In Figure 7.27, the local particle diameters range from 2.26 m in dark blue to 67.73 m in red. As expected, the bigger particles have moved further than the smaller particles. As shown in Figure 7.21, there are essentially no particles in the corners and centre of the domain, which led to high values of variance in particle size in the corners and centre of the domain. Because of this, when the particle density is below 5.0 particles per cubic meters, the value of the variance is set to 0.0, in order to be able to observe the data where the particle density is significant. Figure 7.28 shows the local variance in particle sizes. Across the whole domain, the variance is mostly close to zero, which means that the particle sizes are close to the local average.

As shown in Figure 7.21, there are essentially no particles in the corners and centre of the domain, which led to high values of covariance between the velocity components and sizes of the particles in the corners and centre of the domain. Because of this, when the particle density is below 5.0 particles per cubic meters, the value of the covariance is set to 0.0, in order to be able to observe the data where the particle

density is significant. Figure 7.29 illustrates the local covariance between the particle size and its x-direction velocity component. The maximum positive and negative Ψ_{xd} are reached at the left and right edges of the domain, which means that the bigger particles are more likely to have a larger positive and negative x-direction velocity component, as shown in Figures 7.22 and 7.27. Figure 7.30 illustrates the local covariance between the particle size and its z-direction velocity component. The maximum positive and negative Ψ_{zd} are reached at the top and bottom extremities of the domain, which means that the bigger particles are more likely to have a larger positive and negative z-direction velocity component, as shown in Figures 7.23 and 7.27.

The simulation shown in Figures 7.19-7.30 was run using the Niagara cluster, from the University of Toronto and operated by SciNet and Compute Canada, as described in Section 7.5. One node was used with a total of 40 cores.

7.4 Convection-Relaxation Study

In order to demonstrate the order of accuracy of the scheme and the advantages of the AMR implementation, a simple convection-relaxation equation is solved using the discontinuous-Galerkin-Hancock scheme described in Section 4.2 with and without AMR. The PDE that is solved is

$$\frac{\partial \phi}{\partial t} + v_i \frac{\partial \phi}{\partial x_i} = -\frac{1}{\tau}\phi, \tag{7.27}$$

where ϕ is a quantity that is convected, v_i is the convection velocity and is chosen to be $-1.0\,\mathrm{m/s}$ in the x, y and z-directions, and τ, the relaxation time, is $1.0\,\mathrm{s}$. The domain is a cube with $-10.0\,\mathrm{m} < x < 10.0\,\mathrm{m}$, $-10.0\,\mathrm{m} < y < 10.0\,\mathrm{m}$ and $-10.0\,\mathrm{m} < z < 10.0\,\mathrm{m}$. The initial conditions are given by

$$\phi_0 = \exp\left(-\frac{1}{2}\left(x^2 + y^2 + z^2\right)\right). \tag{7.28}$$

In order to determine the order of accuracy of the scheme, an exact solution is

Table 7.1: Solution errors for a convection-relaxation study without AMR.

# of Cells	$\sqrt[3]{\#\text{ of Cells}}$	l^2 error	Order
1000	10	0.04532	—
8000	20	0.02009	1.17360
64000	40	0.00538	1.89978
512000	80	0.0009	2.57606
4096000	160	0.00012	2.87476

Table 7.2: Solution errors for a convection-relaxation study with AMR.

# of Cells	$\sqrt[3]{\#\text{ of Cells}}$	l^2 error	Effective Order
1000	10	0.04532	—
8000	20	0.02009	1.17360
64000	40	0.00538	1.89978
232000	60.44	0.0009	4.32581
904000	96.69	0.00012	4.24058

necessary. The exact solution is

$$\phi = \exp\left(-\frac{t}{\tau}\right)\exp\left(-\frac{1}{2}\left((x-v_x t)^2+(y-v_y t)^2+(z-v_z t)^2\right)\right). \tag{7.29}$$

The final time for the simulations is 3 seconds and the CFL number chosen is 0.3. For the case where AMR is used, the refinement criteria is based on the density in the cell. If the density is larger than 0.00001, the cell is flagged for refinement. The criteria is checked before the simulation is launched, as well as every 10 time steps. At the end of the simulation, the l^2 norm of the error is computed using

$$\text{Error} = \sqrt{\sum_{n=0}^{N}(\phi_{exact\,n}-\phi_n)V_n}, \tag{7.30}$$

where V_n is the volume of the cell, $\phi_{exact\,n}$ is the exact solution, ϕ_n is the cell average solution from the scheme and N is the total number of cells.

Table 7.1 shows the convergence properties of the DGH scheme without using AMR. The scheme approaches third-order accuracy as the grid is refined. Table 7.2 shows the convergence properties of the DGH scheme with AMR. Though the order

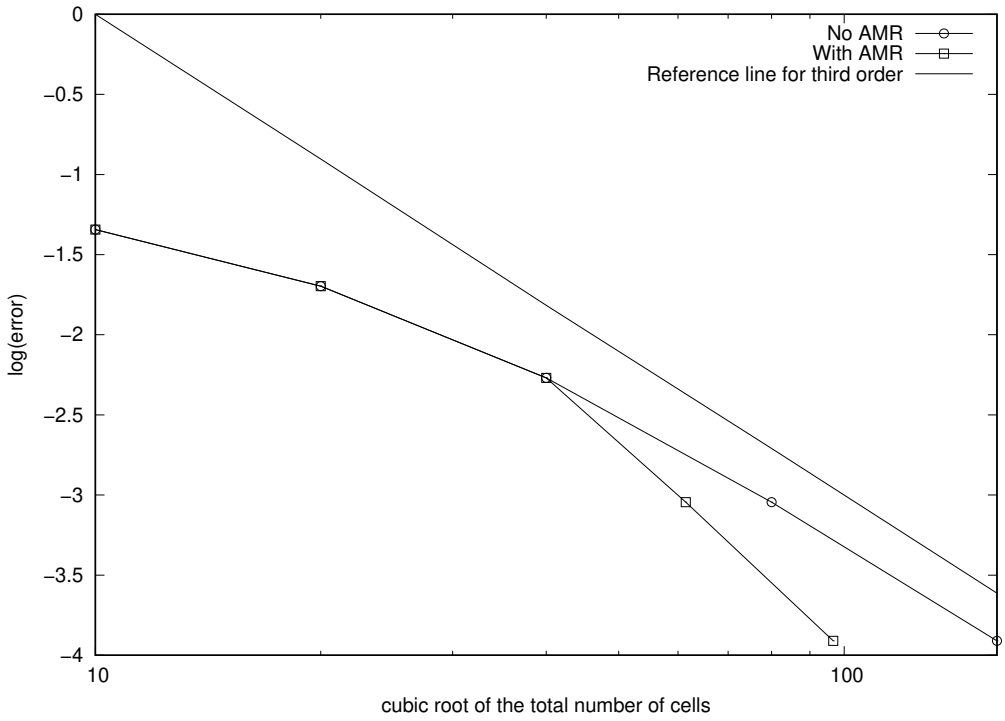


Figure 7.31: l^2 norms of the errors with a reference line showing third-order accuracy.

of accuracy is really only defined when the mesh is refined uniformly, an effective order of accuracy can be defined as

$$\text{effective order} = \frac{\log(l^2 \text{ error}_2) - \log(l^2 \text{ error}_1)}{\log(\sqrt[3]{\#\text{ of cells}_2}) - \log(\sqrt[3]{\#\text{ of cells}_1})}. \tag{7.31}$$

Using AMR, it is possible to obtain the same error as Table 7.1, but with less cells, which is advantageous. Figure 7.31 depicts the order of accuracy of the scheme without AMR, the effective order of the scheme with AMR, as well as a reference line of slope -3. The DGH scheme essentially achieves third-order accuracy without the use of AMR and is even more accurate when using AMR, as the same error is obatained, but with less cells.

Figure 7.31 depicts the initial conditions of the convection-relaxation problem. Figures 7.33 and 7.34 shows the initial grid with AMR. The starting number of cells is 568000. Figure 7.35 shows the final solution of the convection-relaxation problem. Figures 7.36 and 7.37 shows the final grid with AMR. The final number of cells is 904000.

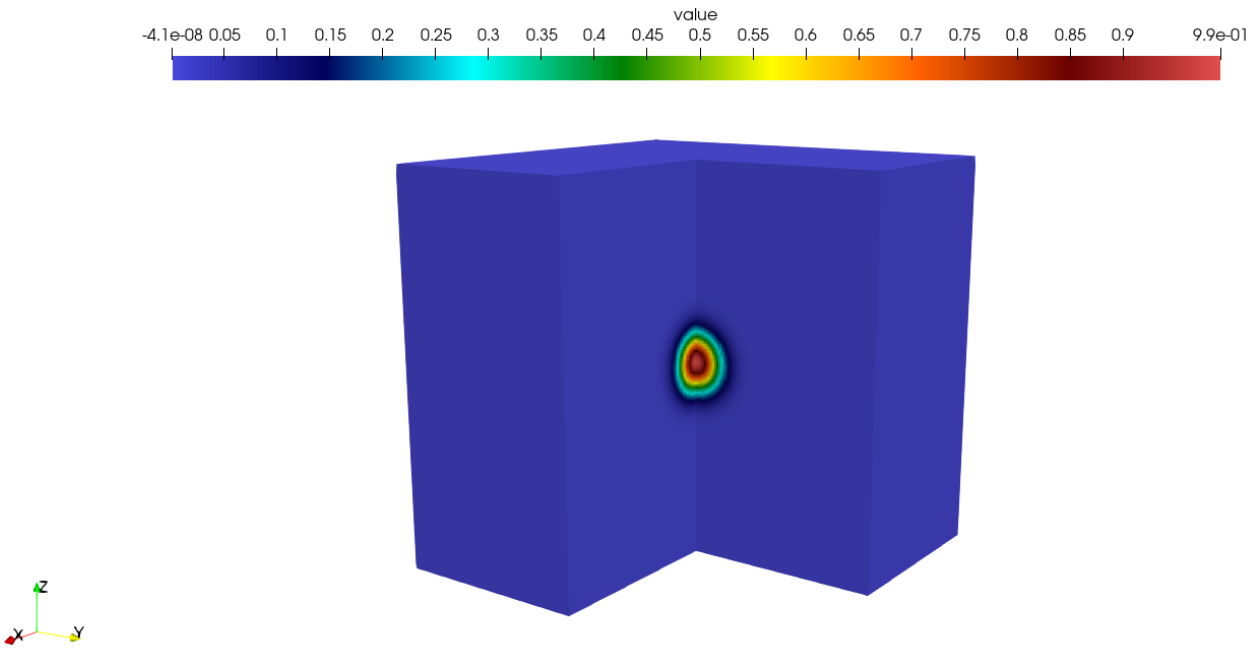


Figure 7.32: Initial conditions of the convection-relaxation study.

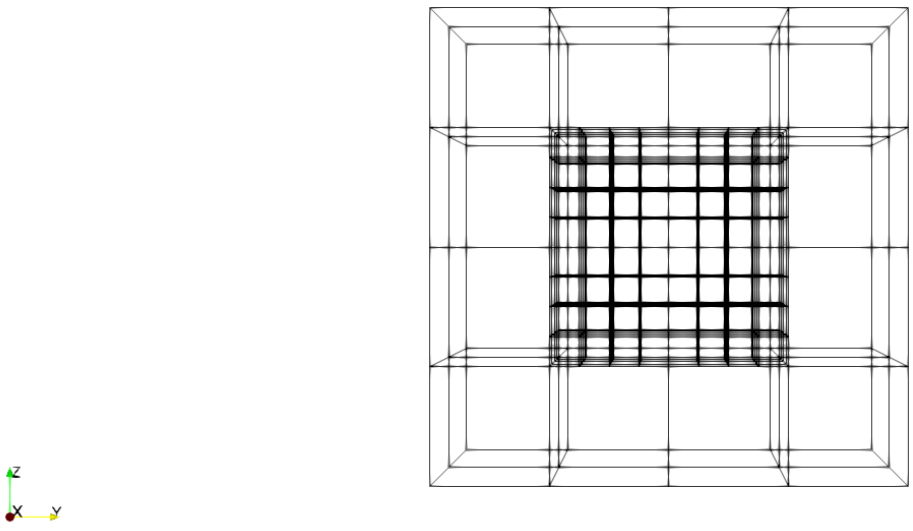


Figure 7.33: Section of the initial grid of the convection-relaxation study taken along the $Y - Z$ plane.

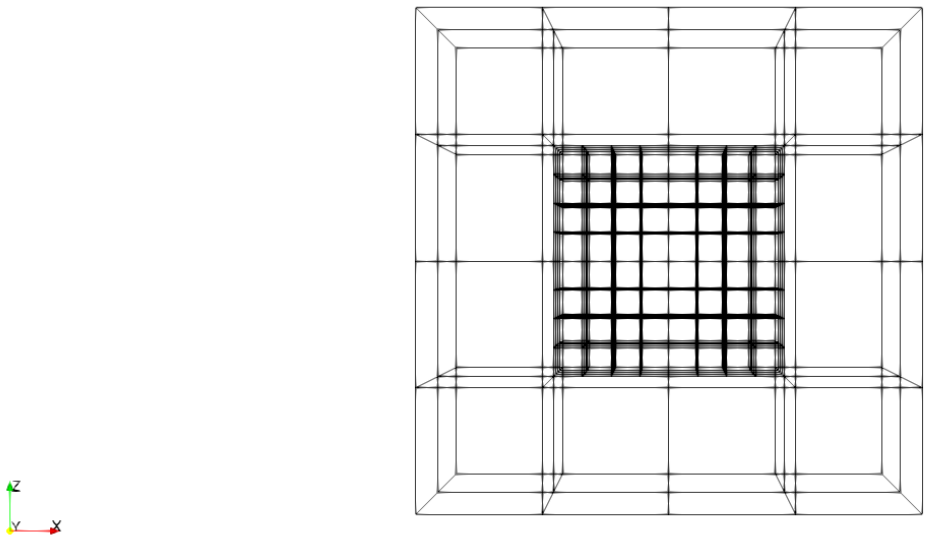


Figure 7.34: Section of the initial grid of the convection-relaxation study taken along the $X - Z$ plane.

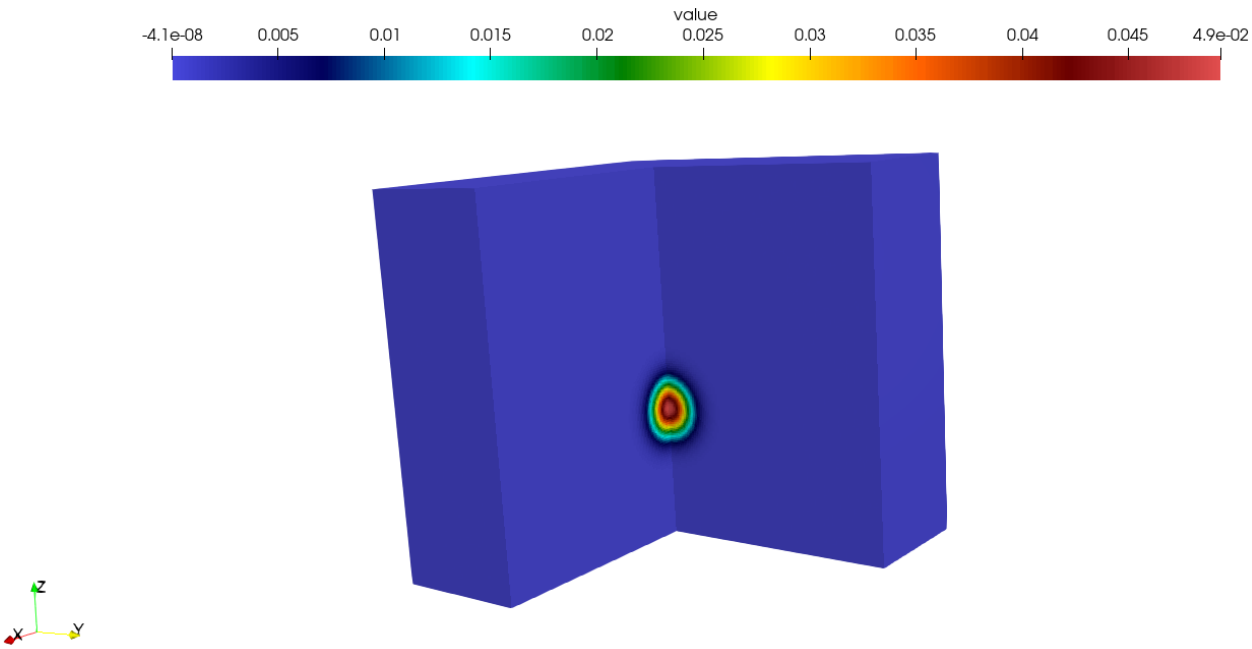


Figure 7.35: Solution of the convection-relaxation study after 3 seconds.

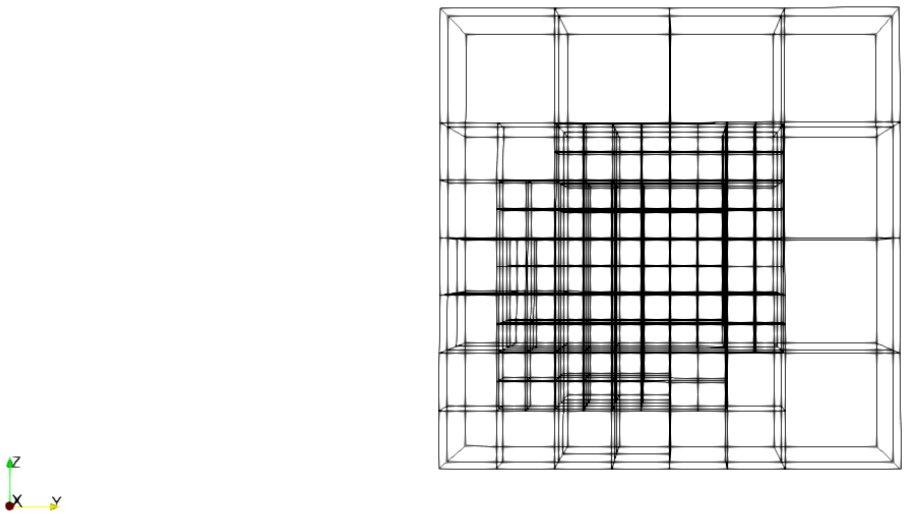


Figure 7.36: Section of the final grid of the convection-relaxation study taken along the $Y - Z$ plane.

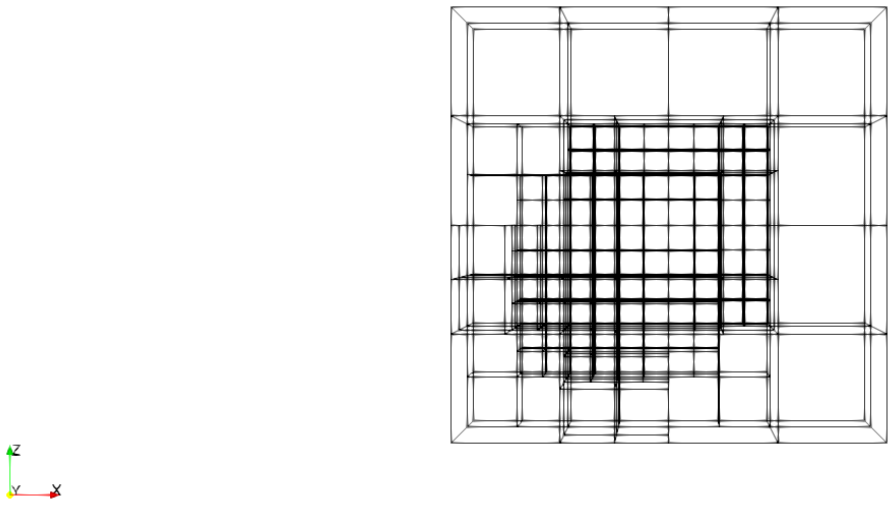


Figure 7.37: Section of the final grid of the convection-relaxation study taken along the $X - Z$ plane.

7.5 Parallel Efficiency

The block-based adaptive mesh refinement algorithm has been implemented within a parallel academic code, written using the C++ programming language, OpenMP and the MPI (message passing interface) library. In order to test the parallel efficiency of the algorithm, computations were done in parallel with a first-order finite volume scheme using the Niagara cluster from the University of Toronto and operated by SciNet and Compute Canada. The cluster has 1548 nodes with 40 Intel Skylake Xeon Gold 6148 processors with a clock speed of 2.4GHz and 468 nodes with 40 Intel Cascade Lake Xeon Gold 6248 processors with a clock speed of 2.5GHz, for a total of 2016 nodes and 80640 cores. Each node has 202 GB of RAM. The scaling study presented here was done using a strong scaling analysis, meaning that that total problem size is constant and the number of cores is varied.

The parallel speed-up, S_n, is computed using

$$S_n = \frac{t_1}{t_n}, \tag{7.32}$$

where t_1 is the time a computation takes on one node and t_n is the time a computation takes on n nodes. The parallel efficiency, E_n, is computed using

$$E_n = \frac{S_n}{n}, \tag{7.33}$$

where n is the number of nodes used.

The first problem considered is the shock cube of Section 7.1 with 4096 blocks, each block has 20 cells in each of the three directions, for a total of 8000 computational cells per block and 32 768 000 computational cells in the mesh. The second problem considered is also the shock cube of Section 7.1 with 4096 blocks, but this time each block has 50 cells in each of the three directions, for a total of 125 000 computational cells per block and 512 000 000 computational cells in the mesh. Calculations of the parallel performance and scalability are shown in Figures 7.38 and 7.39.

If the scalability were perfect, the speed-up shown in Figures 7.38 and 7.39 would

Table 7.3: Wall time for the shock cube of Section 7.1 on different number of nodes with 4096 blocks, each block having 8000 cells, for a total of 32 768 000 computational cells.

# of Nodes	# of Cores	Wall Time (s)	Speed-Up	Efficiency
1	32	1006	1.0	1.0
2	64	508	1.98	0.99
4	128	247	4.07	1.02
8	256	128	7.86	0.98
16	512	63	15.97	0.99

Table 7.4: Wall time for the shock cube of Section 7.1 on different number of nodes with 4096 blocks, each block having 125 000 cells, for a total of 512 000 000 computational cells.

# of Nodes	# of Cores	Wall Time (s)	Speed-Up	Efficiency
1	32	36 725	1.0	1.0
2	64	18420	1.99	0.99
4	128	9260	3.96	0.99
8	256	4601	7.98	0.99
16	512	2402	15.28	0.96

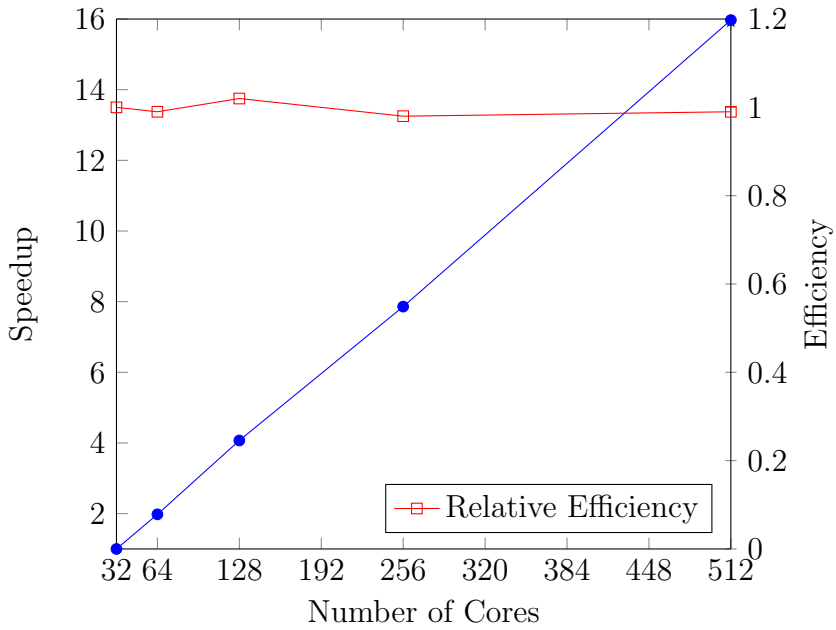


Figure 7.38: Parallel efficiency for the shock cube problem of Section 7.1 with 4096 block of 8000 cells each.

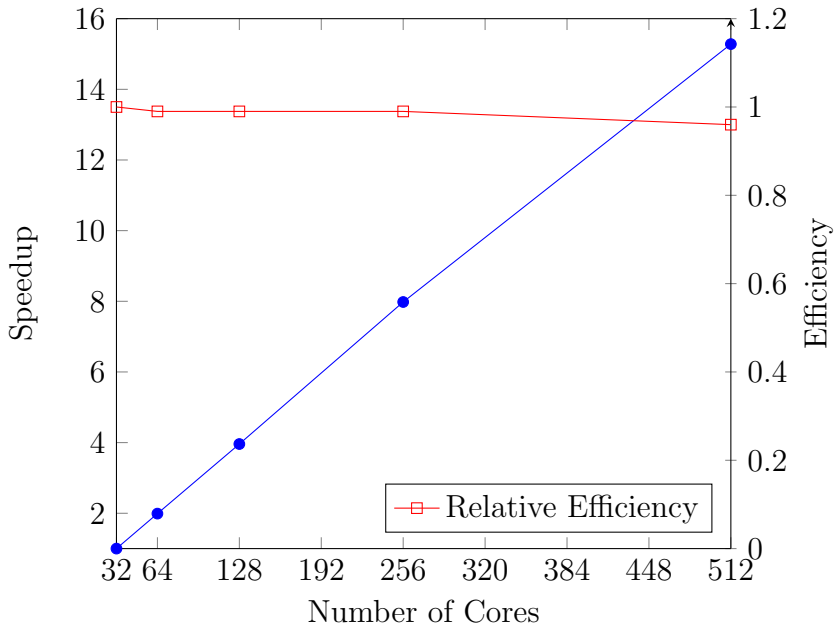


Figure 7.39: Parallel efficiency for the shock cube problem of Section 7.1 with 4096 block of 125 000 cells each.

have a slope of 1 and the efficiency would be 100%. However, because of inter-CPU communication, this is not the case. Furthermore, the fact that a first-order finite volume scheme was used also played a role in the less than perfect scalability, as the scheme is simple and has little local computational work. In Figure 7.38, one will note that the efficiency goes above 1. It is possible this is due to the problem getting smaller and fitting in the cache and means that the efficiency is close to perfect. Based on the scaling observed for a first-order scheme, the scaling is expected to be even better for more complex, higher-order schemes that have more local work.

Chapter 8

Conclusions and Future Work

8.1 Discussion and Conclusion

As shown in this thesis, this work contains two main contributions: implementing the PGM in a three-dimensional solver and developing a block-based AMR algorithm that does not rely on a data tree structure to keep track of the block. Firstly, as shown in Section 7.3, the PGM was tested in a space homogeneous case and the results were compared to the exact solution. The data obtained with the PGM agreed with the exact solution of this model and the deviations from the true kinetic equation are due to modelling errors. Then, the PGM was used to model the detonation of an RDD and the results obtained agreed with the expected behaviour of such a device. Secondly, as shown in Sections 7.1 and 7.2, the AMR algorithm was tested with different problems. The AMR implementation using boundary signatures, block signatures, interblock faces and composite interblock faces instead of a hierarchical tree structure was shown to work well and be efficient in parallel.

In this thesis, moment methods applied to the kinetic theory of gases are described. Different moment closure methods are presented, as well as the one chosen for the present thesis. The advantages of the resulting first-order hyperbolic partial differential equations are put forward.

The first-order finite-volume scheme used is presented. It was chosen because it is robust and computationally inexpensive. It can be a useful tool for model develop-

ment. The domain and time discretization specific to this scheme are presented. The third-order discontinuous-Galerkin scheme used is also presented.

The different techniques of adaptive mesh refinement available are described, as well as their advantages and drawbacks. Block-based adaptive mesh refinement was chosen for this thesis because of its ease of use with parallel networks. In traditional block-based adaptive mesh refinement techniques, the block connectivity is stored using a hierarchical tree structure. However, this method can lead to problems when the mesh is flagged for coarsening. A way to store the connections between the blocks that does not use a tree structure is presented. This technique relies on object data types called boundary signatures, block signatures, interblock faces and composite interblock faces. The block signatures represent the refinement history that a block has been through. The first entry is a root number, and then the refinement events are added to the signature as needed. This results in a unique collection of data that allows one to identify different blocks. The boundary signatures hold information about the two blocks that share a boundary. Each boundary has a unique signature that allows communication coordination through the comm hub. A composite interblock face represents a boundary face. Inside each composite face is a single interblock face, or a collection of interblock faces. These interblock faces represent the boundary faces of the block on the other side of the boundary. The interblock faces have the boundary signature that allows communication of fluxes between blocks. The AMR scheme developed as part of this thesis will form the basis of many different solvers written by future students. The way it is written allows for one to write their solver having only one block in mind. If refinement is necessary, the scheme takes care of the block communication independently from the solver or PDE used.

The mesh refinement techniques was tested first with uniform mesh refinement across the domain and then with adaptive mesh refinement to capture the shock waves. As predicted, more cells are added around the shock waves in both the two-dimensional and three-dimensional meshes.

Finally, the new polydisperse Gaussian model was tested in three dimensions for

the first time, with local mesh refinement where needed. As expected the particles moved outwardly and to the right, while still being subject to gravitational acceleration.

8.2 Future Work

The current work on mesh refinement could be extended in a number of directions. Immediate advancements could cover

- Mesh coarsening should be implemented, in order to remove cells where they are not needed anymore.
- The PGM should be coupled to more complex background flow, namely time-varying background flows that are typical after an RDD explosion.

These extensions are both currently being developed.

References

[1] Marsha Berger and Randall Leveque. An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries. In *9th Computational Fluid Dynamics Conference*, page 1930, 1989.

[2] Marsha J Berger. Adaptive mesh refinement for hyperbolic partial differential equations. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1982.

[3] Marsha J Berger, Phillip Colella, et al. Local adaptive mesh refinement for shock hydrodynamics. *Journal of computational Physics*, 82(1):64–84, 1989.

[4] Marsha J Berger and Jeff S Saltzman. AMR on the CM-2. *Applied Numerical Mathematics*, 14(1-3):239–253, 1994.

[5] Graeme A Bird and JM Brady. *Molecular gas dynamics and the direct simulation of gas flows*, volume 42. Clarendon press Oxford, 1994.

[6] Sydney Chapman, Thomas George Cowling, and David Burnett. *The mathematical theory of non-uniform gases: an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases*. Cambridge university press, 1990.

[7] Bernardo Cockburn and Chi-Wang Shu. The Runge-Kutta local projection-discontinuous-Galerkin finite element method for scalar conservation laws. *ESAIM: Mathematical Modelling and Numerical Analysis*, 25(3):337–361, 1991.

[8] François Forgues, Lucian Ivan, Alexandre Trottier, and James G McDonald. A Gaussian moment method for polydisperse multiphase flow modelling. *Journal of Computational Physics*, 398, 2019.

[9] Lucie Freret and Clinton P Groth. Anisotropic non-uniform block-based adaptive mesh refinement for three-dimensional inviscid and viscous flows. In *22nd AIAA Computational Fluid Dynamics Conference*, page 2613, 2015.

[10] Tamas I Gombosi and Atmo Gombosi. *Gaskinetic theory*. Cambridge University Press, 1994.

[11] Harold Grad. On the kinetic theory of rarefied gases. *Communications on pure and applied mathematics*, 2(4):331–407, 1949.

[12] Lucian Ivan, David Hummel, and Luke Lebel. A MCREXS modelling approach for the simulation of a radiological dispersal device. *Journal of environmental radioactivity*, 192:551–564, 2018.

[13] Luke Lebel, Pierre Bourgouin, Sohan Chouhan, Nils Ek, Volodymyr Korolevych, Alain Malo, Dov Bensimon, and Lorne Erhardt. Radiation modeling and finite cloud effects for atmospheric dispersion calculations in near-field applications: Modeling of the full scale RDD experiments with operational models in canada, part ii. *Health physics*, 110(5):518–525, 2016.

[14] Pierre Lesaint and Pierre-Arnaud Raviart. On a finite element method for solving the neutron transport equation. *Publications mathématiques et informatique de Rennes*, pages 1–40, 1974.

[15] C David Levermore. Moment closure hierarchies for kinetic theories. *Journal of statistical Physics*, 83(5-6):1021–1065, 1996.

[16] C David Levermore and William J Morokoff. The Gaussian moment closure for gas dynamics. *SIAM Journal on Applied Mathematics*, 59(1):72–96, 1998.

[17] James Clerk Maxwell. Illustrations of the dynamical theory of gases. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 20(130):21–37, 1860.

[18] James McDonald and Manuel Torrilhon. Affordable robust moment closures for CFD based on the maximum-entropy hierarchy. *Journal of Computational Physics*, 251:500–523, 2013.

[19] James J Quirk and Ulf R Hanebutte. A parallel adaptive mesh refinement algorithm. Technical report, Institute for computer applications in science and engineering Hampton VA, 1993.

[20] Wm H Reed and TR Hill. Triangular mesh methods for the neutron transport equation. *Los Alamos Report LA-UR-73-479*, 1973.

[21] Yoshifumi Suzuki. *Discontinuous Galerkin Methods for Extended Hydrodynamics.* PhD thesis, University of Michigan, 2008.

[22] Yoshifumi Suzuki and Bram van Leer. A discontinuous Galerkin method with Hancock-type time integration for hyperbolic systems with stiff relaxation source terms. In *Computational Fluid Dynamics 2006*, pages 59–64. Springer, 2009.

[23] Aymeric Vié, Hadi Pouransari, Rémi Zamansky, and Ali Mani. Particle-laden flows forced by the disperse phase: comparison between Lagrangian and Eulerian simulations. *International Journal of Multiphase Flow*, 79:144–158, 2016.

[24] Michael Williamschen and Clinton P Groth. Parallel anisotropic block-based adaptive mesh refinement algorithm for three-dimensional flows. In *21st AIAA Computational Fluid Dynamics Conference*, page 2442, 2013.