Numerical planet formation on exascale architectures

Timothée David–Cléris

N° d'ordre NNT :

THÈSE

en vue de l'obtention du grade de Docteur, délivré par

l'ÉCOLE NORMALE SUPERIEURE DE LYON

École Doctorale 52

École doctorale de Physique et astrophysique

Spécialité de doctorat : Physique – Astrophysique

Soutenue publiquement le 24 septembre 2024, par :

Timothée David--Cléris

Numerical planet formation on exascale architectures

—

Formation planétaire numérique sur architecture exascale

Devant le jury composé de :

LESUR Geoffroy, Directeur de recherche, IPAG/Université Grenoble Alpes	Rapporteur
GRANDGIRARD Virginie, Directrice de Recherche, CEA, IRFM	Rapporteuse
TREMBLIN Pascal, Chercheur CEA HDR, CEA Saclay	Examinateur
BENOIT Anne, Maître de conférences-HDR, LIP, ENS de Lyon	Examinatrice
TERRASSE Isabelle, Personnalité scientifique, Airbus	Examinatrice
LAIBE Guillaume, Professeur des universités, CRAL ENS de Lyon	Directeur de thèse

Résumé

Les phénomènes astrophysiques se caractérisent par une interaction complexe entre des processus multi-physiques, multi-échelles, hors d'équilibre et non linéaires. Récemment, la puissance de calcul des supercalculateurs a augmenté jusqu'à atteindre la performance Exascale, à savoir un quintillion d'opérations par seconde. En principe, cette puissance de calcul permet de résoudre des questions cruciales sur la formation des planètes, grâce à des simulations d'une précision sans précédent. Pour y parvenir, il est nécessaire de développer un code basé sur des algorithmes capables de tirer parti de cette nouvelle puissance de calcul.

L'objectif de cette thèse est de développer Shamrock, le premier code Exascale astrophysique qui utilise des méthodes multiples (particules ou grilles adaptatives). Le cœur de ce travail est l'adaptation et l'optimisation d'une nouvelle procédure de construction et de traversée d'arbre pour trouver des voisins distribués aléatoirement, qui est entièrement parallélisée sur des architectures utilisant des cartes graphiques, ainsi qu'une nouvelle approche de la décomposition de domaine pour abstraire l'équilibrage de la charge et la communication. La version actuelle de l'integrateur SPH de Shamrock atteint une efficacité parallèle supérieure à 90% sur les supercalculateurs et fournit des facteurs d'accélération de plusieurs milliers par rapport aux simulations standard de pointe, ouvrant la voie à la résolution de nouveaux problèmes astrophysiques.

Abstract

Astrophysics phenomenon feature a complex interplay between processes that are multi-physics, multiscale, out of equilibrium, and non-linear. Recently, the computing power of supercomputers increased up to Exascale performance, namely a quintillion operations per seconds. In principle, this computing power makes it possible to resolve crucial questions about planet formation, thanks to simulations of unprecedented accuracy. To achieve this, it is necessary to develop code based on algorithms capable of taking advantage of this new computing power.

The aim of this thesis is to develop Shamrock, the first astrophysical Exascale code that employs multi-methods (particles or adaptive grids). The core of this work is the adaptation and optimisation of a novel tree building and traversal procedure for finding randomly distributed neighbours, which is fully parallelised on architectures using graphics cards, as well as a novel approach to domain decomposition to abstract load balancing and communication. The current version of the Shamrock SPH solver achieves parallel efficiency that exceeds 90% on supercomputers and delivers acceleration factors of several thousand compared to standard state-of-the-art simulations, paving the way towards tackling novel astrophysical problems.

Remerciements

Contents

Notations

References

When available the bibliography style features three different links associated with three different colors: links to the journal/editor website or to a numerical version of the paper are in red, links to the ADS website are in blue and links to the arXiv website are in green.

Results

The important passages of the thesis are highlighted in grey boxes.

Name

If the result is named (e.g. equations of $\cdots$) it will be displayed like so instead.

Warning

! Important caveats or warning of the thesis will be highlighted like this.

Definitions

Definition ▶ style of definition

Definition are always named like so

Theorems

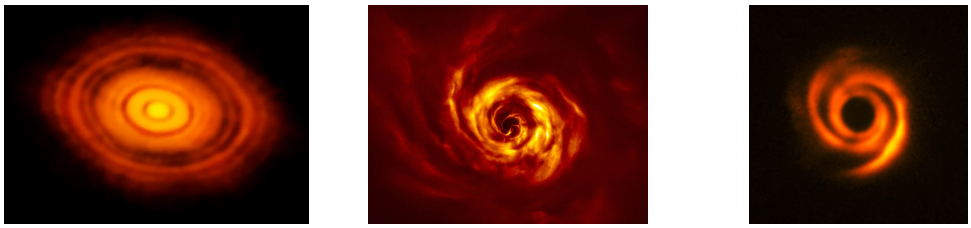
Theorem ▶ style of theorems

Theorems are always named like so

Acronyms

SPH	Smoothed Particle Hydrodynamics
AMR	Adaptive Mesh Refinement
GPU	Graphics Processing Unit
MHD	Magneto-Hydro-Dynamics
CFL	Courant-Friedrichs-Lewy condition
SI	Streaming-instability
PSI	Polydisperse Streaming-instability

Preamble



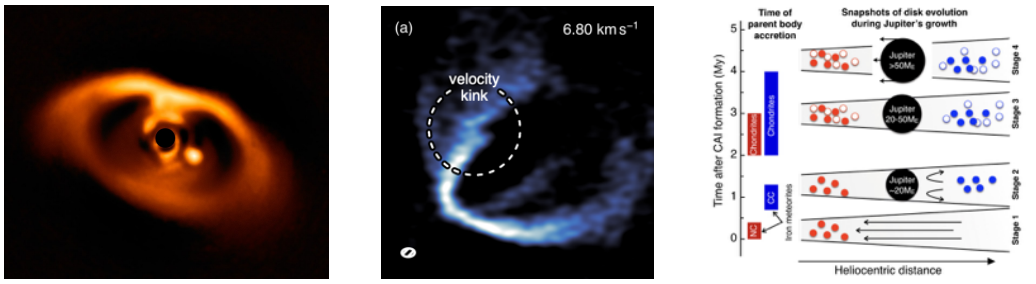
(a) (b) (c)

Figure 1: Images of discs presenting substructures. (a) Image of HL Tau observed using the ALMA radio interferometer (Credit: ALMA,NRAO/ESO/NAOJ) (age $\sim 2\,\mathrm{Myr}$) (b) Disc arbound AB Aurigae observed in infrared polarized light (Credit: VLT/SPHERE) (c) HD135344B observed in infrared polarized light (Credit: VLT/-SPHERE)

Since the initial discovery of exoplanets by Mayor & Queloz in 1995, thousands of them have been discovered. Since planets are the cradle of life, the understanding of the origines of life then requires the understanding of why, where and how planets forms. In particular, planets are thought to be formed in young discs of gas and dust orbiting around young stars. This assertion is supported by the presence of substructures (example shown in Fig. 1) which in most case are consequences of the presence of planets as confirmed by kinematic signature, direct observations and meteoritic constraints (see example in Fig. 2). In particular, when observed, most discs do present a variety of sub-structures. These sub-structures are ubiquitous as they are observed in most discs, even in young ones. This, in turns, hint that most discs may likely host planets. In particular, the presence of planets in young discs suggest that their formation must be though in connexion with the formation of the star, its discs and larger scales in general.

However, processes leading to planet formation are multi-scales, multi-physics and out-of-equilibrium, making the analytical treatment of such processes unfeasible in its entirety. We therefore need numerical simulations, although they are limited in the scales and effects they can handle by the computing power available.

Understanding how to increase computational power using existing and already available hardware is therefore key to unlock the capacity of permorming simulations of planet formation with unprecedented resolution and predictive capacities. In a computer the CPU (central processing unit) can be thought as the brain of the computer. It can perform all kind of standard computations. Because of its versatility, we normally relies on a CPU to perform computations. On the other hand, most modern computer also have a GPU (graphics processing unit), which is a dedicated



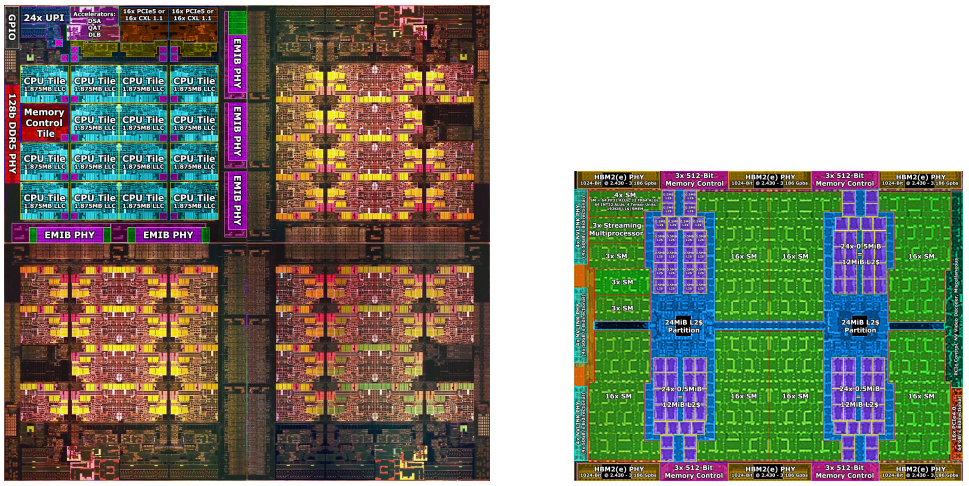
(a) (b) (c)

Figure 2: Examples of evidences of planet's presence in discs (see Chapt. 1 for more details). (a) Direct observation of a planet: observation of Pds 70b observed in infrared polarized light (Credit: VLT/SPHERE) (b) Velocity kinks: Observation of Doppler shift in ^{12}CO line measuring deviation to Kepler's orbits indicating the presence of a perturbating gravitational potential (Credit : Pinte et al., 2022) (c) Meteoritic constraints: In our solar system, a strong dichotomy is observed in the isotopic compositions of meteorites, suggesting that the disc of our system was physically separated after 2 or 3 million years.

piece of hardware to perform graphical operations or massively parallel operations in general. A typical compute dedicated GPU will outperform a similar class of CPU by a factor ten in both bandwidth and computing power while consuming the same amount of energy. Due to this efficiency, computers with GPUs were naturally linked together using a network whose bandwidth is large enough to not be a bottleneck. Such network was scaled to thousands of computers (called computing node in this instance), such structure is called a heterogeneous supercomputer, as it makes use of both CPUs and GPUs. It is however important to note that while GPUs jobs are to perform computations CPUs are here used mostly to schedule operations on GPUs and communications between them. As an example, the largest supercompter today (Frontier) is built sing an heterogeneous architecture and has a staggering performance of 1.2 exaflops. This means that it is able to perform 1.2 quintillion floating point operations per seconds. This comes at a cost of $22.10^6 W$ which is about a 50th of a nuclear power plant throughput. However, while this is a large power consumption, it is about 4.5 times more efficient than the largest CPU only supercomputer.

Nowadays, more and more heterogeneous supercomputer are being built due to the pressure toward higher power efficiency, higher computing density and the growth of AI applications. This result in old CPU only supercomputer being slowly replaced by heterogeneous ones. This is typically the case of the most powerful supercomputer in France, Adastra, which replaced its CPU only predecessor, the supercomputer Oxygen. As supercomputer migrate towards GPU based architectures, new astrophysics codes need to be developed to utilize those modern supercomputers. Their development, is therefore fulfilling the need for more computing power

to resolve a larger amount of scales and effects, as well as the need to transition to follow supercomputers evolutions.



(a) Intel Saphire rapids CPU die shot, 15 CPU Cores are contained in the cyan area (credit : TechPowerUp)

(b) Nvidia A100 GPU die shot, 6912 GPU 'Cores' are contained in the green area (credit : Locuza)

Figure 3: Exemples of CPU and GPU die shot, relative scale is correct.

However, migrating astrophysical codes from CPU to GPU codes is hard. The reason is due to the differences between CPUs and GPUs. A modern CPU posses a few large CPU cores which can execute independent tasks simultaneously. Those CPU cores are fairly large (see Fig. 3) because they carry additional transistors to digest the program complexity. This makes their use easier as they are more versatile, however those additional transistors do increase the power consumption in return. GPUs instead integrate many (thousands) of GPU cores which are way smaller. They do not carry those additional transistors to digest the same level of complexity, making them highly specialized, small and power efficient. They are also slower than their CPU counterpart, but their sheer number makes the GPU about 10 time faster than the CPU. Also, GPUs do not use the same programming paradigm as CPUs, they rely on a SPMD (Single Program Multiple Data) programming paradigm. Additionally, they require to be given even work to be efficient, meaning that one must submit enough work in parallel to leverage their computing power, and they are more sensible to data access pattern than CPUs. This render most CPUs not portable to GPU architectures, requiring the development of new codes.

In recent years many codes were ported to GPU architectures, to name a few Idefix, Parthenon and codes derived from the AMRex framework. However, all of them are grid based codes. However, many astrophysical systems posses complex geometries for which grid based methods can be overly diffusive. An alternative to mesh based methods is to use a particle based meshless method named Smoothed Particle

Hydrodynamics (SPH) to simulate compressible hydrodynamics. Additionally, numerical results should be method agnostic, meaning that they must be reproducible with multiple methods. The SPH method was used successfully for many decades in the astrophysics community, in particular, in the context of discs through the SPH code Phantom. However, Phantom is limited by its lack of GPU support and scalability on multiple computing node. The core of the Phantom code consists in 3 principal algorithms. Firstly, the neighbour search algorithm in Phantom is base on a KD-tree which can not be ported to GPUs as there is no fast parallel building algorithm for such tree, and updating it would be too costly to update it in the event of load balancing or domain decomposition operations. Secondly, Phantom uses a single loop approach to evolve the particle size and the fields. However, the particle size needs to be updated then communicated before evolving the fields, rendering the single loop approach impossible for a code that is scalable on multiple nodes. Lastly, Phantom uses a dynamic cache for neighbours, which is not applicable on GPUs. Overall, Phantom can not be ported on multi-GPU architectures. This led to the decision of developing a new multi methods code during this PhD, with a particular focus on its SPH scheme as particle are numerically the most challenging aspect of a multi-GPU code.

It is this context that we present the following manuscript, which will be divided in the following manner. We will first introduce the astrophysical context through a review of planet formation in astrophysics through the scope of numerical methods, simulation, and codes. In particular, we will draw specific attention to challenges related to the simulation of phenomenon of importance in the context of planet formation. As we aim at developing a multi-method code, we need to detail the principal scheme used in astrophysics to be able to identify a possible abstraction that allows the implementation of multiple schemes (particle or grid based) in a single code. This will be done in this second chapter, which will then cover the three main numerical methods used in astrophysics in order, namely, the finite element scheme at the basis of the Zeus code, a typical finite-difference code use in thre community of astrophysical. The finite volume scheme at the basis of the Ramses code, one of the most used code in astrophysics which is based on the Godunov's methods. Lastly, this chapter will cover the details of the SPH scheme. Before starting to implement a code, it is crucial to gain a good understanding of the underlying hardware that we want to target. This will be the goal of the third chapter, dedicated modern computing hardware and especially GPUs. In particular, we will cover in greater details the differences between GPUs and CPUs to understand how to adapt algorithm to newer architectures, as well as possible challenges associated to GPUs. Having introduced the astrophysical context, the numerical schemes as well as modern computing hardware, we will then present in the fourth chapter our novel approach in the Shamrock framework. This framework consist of a novel patch based domain decomposition, where each of the patches are updated on the GPUs. The patch update is performed by a novel on the fly built tree, built using fully parallelized tree building procedure, as alternatives of updating the tree would be

too slow in the event of decomposition or load balancing operations. In addition to this tree algorithm, we introduce novel approaches to handle communications and abstract the load balancing from the code. Additionally, the framework abstracts the notion of particle or grid cell, allowing for the implementation of the methods previously described. A last chapter before the conclusion of the manuscript will finally detail the SPH solver implementation in Shamrock, as well as numerical tests to validate and benchmark it. In addition to those chapter, we will present in the appendices preliminary work that are direct extensions of the work presented in detail in the manuscript, and which will be further developed in the following years. A first one will detail the extension of the streaming instability to a continuum of dust specie in the presence of magnetic fields, showing the possibility of streaming instability under the presence of a magnetic field in the polydisperse case. The next two appendices will be dedicated to studying the precision, and extending the dust models in SPH. In order to study local phenomenon such as streaming instability, an appendix will be dedicated to the implementation of shearing box in SPH in Shamrock. In the last appendix, we will present the preliminary work performed to implement the Fast Multipole Method (FMM) in Shamrock to solve self-gravitating fluid equations.

Chapter 1

A shorthand on numerical planet formation

Contents

1. Context

1.1. Understanding planet formation

The question of the origins of life begins with understanding the formation of planets, including our own. In particular, it is essential to understand how and where planets form with which properties. A planet, roughly, is an object with a true mass below the limiting mass for thermonuclear fusion of deuterium that orbits stars other than our sun (the semantics are still under debate by the International Astronomical Union). On top of the known planets in our system, more than 5000 exoplanets (source : `https://exoplanet.eu`) have been discovered since the initial discovery of exoplanets by Mayor & Queloz (1995), who discovered the first exoplanet orbiting a star, 51 Pegasi b. Statistics obtained by the Kepler space telescope, the James Webb Space Telescope (JWST), suggest that the majority of stars possess multiple exoplanets within their systems (e.g. Currie et al. 2023; Lissauer et al. 2023). Those will be completed in a few years by the Extremely Large Telescope (ELT).

One striking observation is that exoplanets come in a very high diversity, in mass and size, with small planets (similar to Mercury for example) and hot Jupiters (several times the mass of Jupiter for example). There is diversity in localization, from hot-Jupiters orbiting so close to the host star that they perform a full revolution in few hours to others that perform a full revolution in thousands of years to planets within the so-called habitable zone of the host star (zone where water can be liquid on the surface under enough atmospheric pressure). The composition of the planets is also diverse, with giant planets made of mostly hydrogen and helium

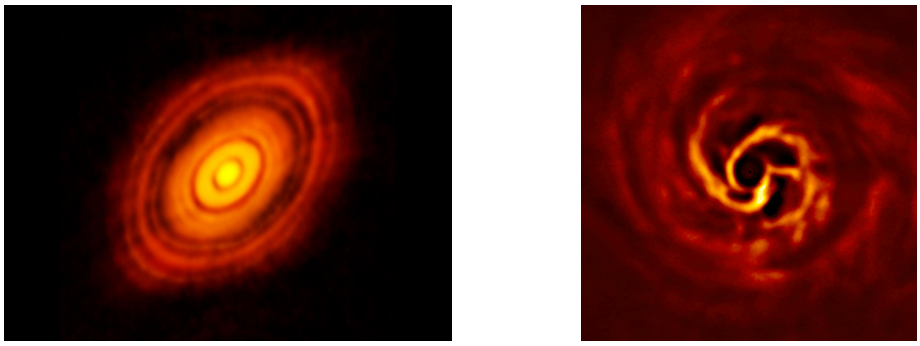
and rocky planets made of mostly iron, oxygen, silicon, and magnesium. The atmosphere of such planets can also be made of different molecules (oxygen, nitrogen, methane, and carbon dioxide to name a few). In order to understand the likelihood of the emergence of life, it is imperative to comprehend the fundamental principles underlying such diversity.

The first effort toward understanding planet formation started with studies made by Galilée, Descartes, and Swedenborg who studied the formation of our own system, trying to explain how such structures can be formed. Influenced by observations of spiral "nebulas", which later turned out to be galaxies, Moulton and Chamberlin in 1904 formulated the planetesimal hypothesis, thinking that spirals were hosting planet formation. This hypothesis was later interrogated by Von Weisäcker's 1944 model, which introduced the notion of turbulence into these nebulas through colliding counterrotating vortices, in order to explain solar system formation. Lastly, an historical contribution is Whipple's 1948 model, which postulated that the Sun formed from a collapsing cloud of "smoke" and that the planets resulted from secondary clouds captured by the main one.

The breakthrough that led to more quantitative studies of planet formation was the observational work of Mendoza V. (1966), who analyzed the spectrum of T Tauri stars. T Tauri stars are young variable class stars that exhibit variation in luminosity. Their names derive from the typical star T Tauri located in the Taurus constellation. T Tauri stars exhibit excess infrared emission in comparison to our sun. They interpret the existence of this excess by stating that 'The short-wavelength photometry refers to a small core and long-wavelength photometry to a large envelope' (Mendoza V., 1966). This was formalized in Mendoza V. (1968) by stating that 'If the infrared luminosity is many times the visual luminosity, then a thick circumstellar dust cloud absorbs the visual radiation and reradiates in the infrared the energy produced by the parent stars.' In other words, T Tauri stars are likely to have a dusty disc surrounding the central object, which is commonly referred to as a circumstellar disc.

Since the 1970s telescopes in visible and infrared light, such as the Spitzer, Hubble Space Telescope, VLT/SPHERE and the ALMA radio interferometer have provided many observations of circumstellar discs in multiple wavelengths that are more spatially resolved. This revealed unexpected complex substructures such as rings, gaps and horseshoes (Andrews et al., 2018; Huang et al., 2018a,b; Kurtovic et al., 2018; Birnstiel et al., 2018; Dullemond et al., 2018; Zhang et al., 2018b; Guzmán et al., 2018; Isella et al., 2018; Pérez et al., 2018). Some examples of such discs are shown in Fig. 1.1.

Since a few years, modern studies suggest that planets can coexist with circumstellar discs. Firstly, we have direct observations of planets in discs (e.g. Currie et al. 2023) starting with the first such observation, PDS 70 b shown in Fig. 1.2. Secondly, discs exhibiting substructure are thought to be likely to host planets (e.g. Bae et al. 2023). Thirdly, newer disc kinematic observations are now able to observe the kinematic signature of the presence of planets in discs (e.g. Pinte et al. 2024, 2023).



(a) (b)

Figure 1.1: (a) Image of HL Tau (Credit: ALMA (NRAO/ESO/NAOJ)) (age $\sim$ 2 Myr). (b) AB Aurigae, observed by the VLT with the instrument SPHERE en in infrared polarized light (age : $\sim$ 1 Myr) (Boccaletti et al., 2020). Substructures such as rings and spirals are observed.

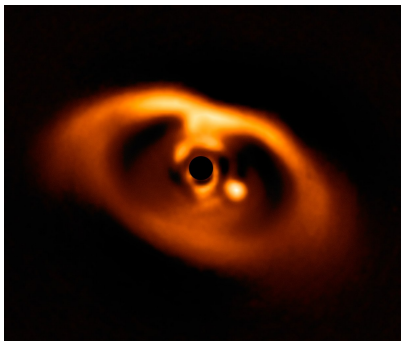


Figure 1.2: Image of Pds 70b, observed by the VLT with the SPHERE instrument in infrared, age : 5.4 ± 1.0Myr (Müller et al., 2018)

Lastly, meteoritic analysis of the solar system shows that solids were physically separated around 3 Myr after the solar system formed, which suggests that there was an early substructure in the young solar system disc, thought to be the result of Jupiter (e.g. Kruijer et al. 2017). Estimating the age of a young stellar system with models of stellar evolution provides $\sim 1 - 10$Myr for the age of planet-hosting discs. This is early compared to the > 100Myr predicted by the former cold isolated nebula scenario. That would be suggested by the viscous diffusion in a stellar nebula (e.g. Hartmann et al. 1998). Those clues imply that planet formation is likely to start very early in disc life, if not before, and be due to short timescale phenomena.

! As a planet may form and evolve in a circumstellar disc, we shall refer to such discs as protoplanetary discs as well.

1.2. On the origins of discs

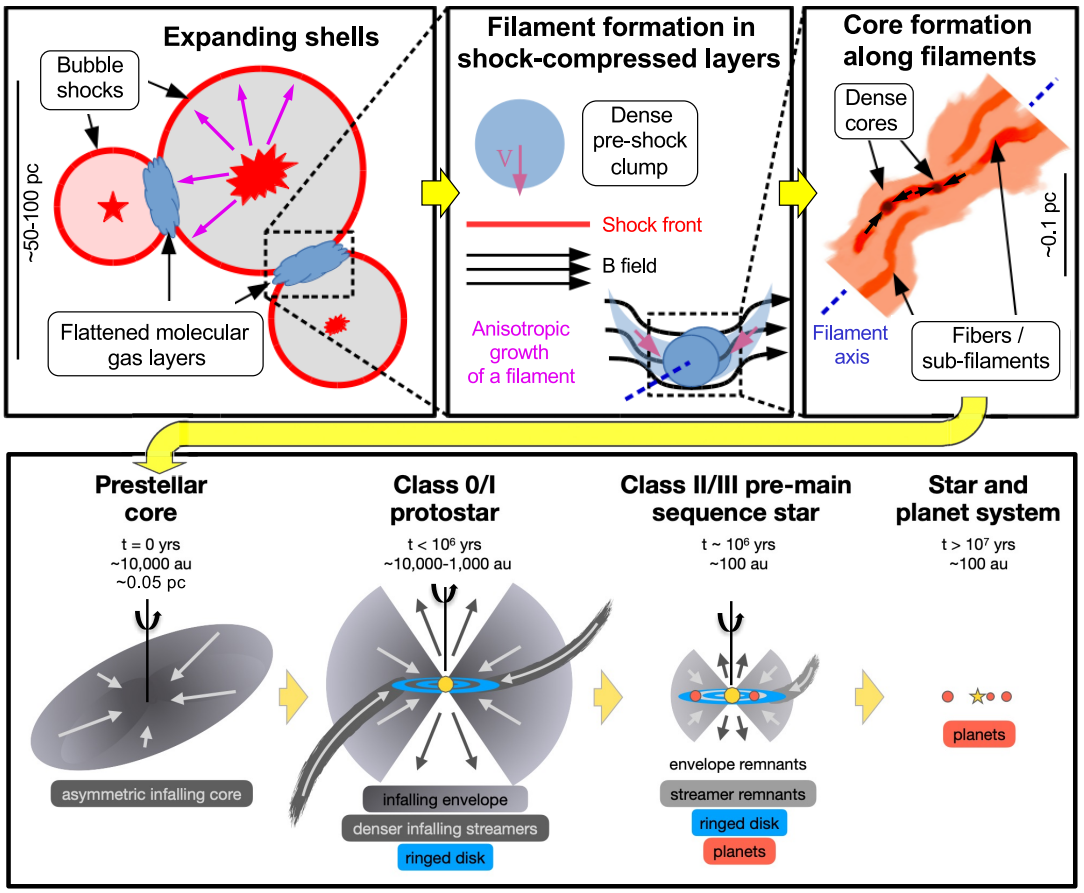


Figure 1.3: Succession of processes leading to the formation of a star and its disc from a protostellar nebula to the formation of planets (Source: Pineda et al. (2023))

Early planet formation implies a connection with the formation of the disc, which in turn also implies a connection with the formation of the star. Therefore, it is not considered that disc and planet formation are in isolation. Observations, simulations, and analytical models support the idea that the formation of stars and discs is part of the same process (e.g. Pineda et al. 2023). Under processes that are still debated (e.g. shocks, compressible turbulence), some regions, of molecular clouds are sufficiently dense and cold such that they exhibit a thermal pressure support weaker than the influence of their own gravity. This causes the cloud to collapse under its own gravitational force. During the protostellar collapse, angular momentum and pressure (thermodynamic as well as magnetic) are opposing this gravitational pull. Angular momentum is conserved during the collapse, implying that rotational speed increases until it opposes further collapse of the gas. As a result, the protostellar collapse results in the formation of a single or multiple protostars surrounded by a circumstellar disc that is supported by its own rotation, which was inherited from

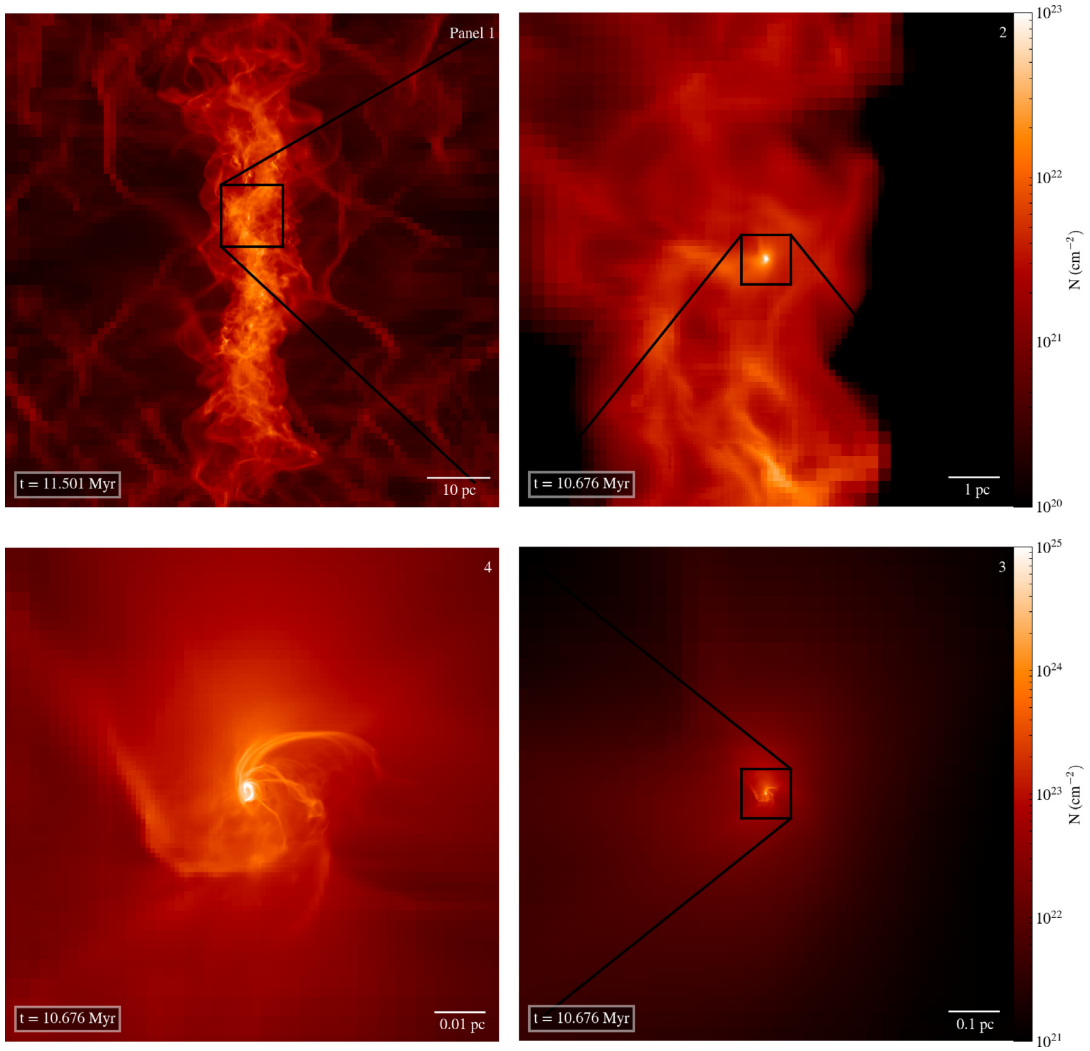


Figure 1.4: Numerical simulation of a multiscale collapse of a molecular cloud leading to star formation (Source: Zhang et al. (2018a))

the protostellar collapse. An example of a simulation of a protostellar cloud leading to star formation is shown in Fig. 1.4 and the complete process is schematically represented in Fig. 1.3.

1.3. Why is planet formation a complex problem ?

Planet and star formations are the byproducts of the balance between gravity and processes that oppose it. These can be either

- Inertial effects, either in the form of angular momentum, where the rotation counteracts the gravitational pull, or in the form of microscopic inertia of the gas atoms, which results in gas pressure at a macroscopic scale.

- Magnetic effects, such as magnetic pressure for ionized gas flows, or radiative effects as a consequence of the radiation of the central star, which heats its environment, thereby elevating its temperature and, consequently, its pressure.
- Dust effects, whereby solid aggregates that have been formed through local contacts favored by surface tension, can be destroyed before being significantly agglomerated together and assembled in a single larger body by gravity. This is the so-called fragmentation barrier, wherein meter-in-size grains are unlikely to grow to larger sizes through collisions and are more likely to fragment in impacts. This is also the case with the meter drift barrier, where meter-sized dust solids are likely to drift and ultimately be destroyed by falling into the star. We will discuss this effect in the next section.

Aside from opposing gravity, these effects also alter the medium's properties or its composition (amount of charges, opacities, chemical composition). Furthermore, the majority of the effects involved in planet formation are non-linear and out-of-equilibrium, as we will discuss in this chapter.

The chain of events leading to planet formation begins with the protostellar collapse, which corresponds at the start to the typical scale of the parsec ($1\,\mathrm{pc} \simeq 3 \cdot 10^{16}\,\mathrm{m}$), which is roughly the typical distance between two stars in our galactic neighborhood. This can range down to the formation of potentially small planets having a typical scale of a few thousand kilometers (mercury mean radius $\simeq 2{\cdot}10^{6}\,\mathrm{m}$). Gravity concentrates matter. This implies that resolving all scales relevant to planet formation corresponds to resolving 10 orders of magnitude in size.

In conclusion, understanding the conditions and outcomes of planet formation requires comprehending such balance and side effects, necessitating an understanding of all those effects, both individually and collectively, across all involved scales. The conditions and results of planet formation are, therefore, inherently multiscale, non-linear, and multi-effect out-equilibrium physics. In addition to resolving all effects, the formation of planets is fundamentally an out-of-equilibrium phenomenon, necessitating the study of all processes across all relevant time scales.

2. Basic physics of a protoplanetary disc.

In order to better understand the physical processes involved in planet formation, we first present a minimal model of a disc, in the spirit of the early model of planet formation pioneered e.g. by Safronov (1969). This model will serve as a basis for describing the state of the art of disc physics.

2.1. Scale-lengths and fluid approximation

Fig. 1.5 shows the different length scales involved in a protoplanetary disc. A typical protoplanetary disc is composed of gas and dust, which are radially distributed

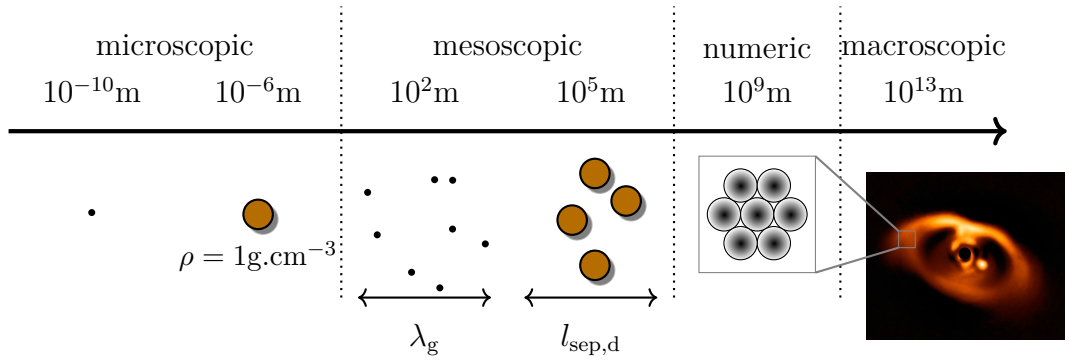


Figure 1.5: An illustration of the different scale lengths involved in protoplanetary discs. Gas molecules are represented by black dots and dust by brown circles. We additionally represent both the numerical and global scale of the problem. In a protoplanetary disc, from microscopical to astronomical, a large span of scale length is involved. Both the micro and mesoscopic scales are sufficiently separated from the numerical scale to allow the use of a fluid approximation for both gas and dust.

around one or several stars. The typical inner radius is of the order of $0.1\,\text{au} \simeq 1.5 \cdot 10^{10}\,\text{m}$ and the outer radius is of the order of $100\,\text{au} \simeq 1.5 \cdot 10^{13}\,\text{m}$. The gas consists primarily of hydrogen and helium in molecular form. Dust grains are porous and fractal aggregates essentially made of silicates originating from the interstellar medium. They range in size from nanometer-sized grains up to meter sized grains. The typical pressure and temperature conditions within the disc result in a typical mean free path for the gas to be of the order of $10 - 100\,\text{m}$ at $1\,\text{au}$. In the context of planet formation, processes of interest take place above $10^6 - 10^7\,\text{m} \simeq 6 \cdot 10^{-6} - 6 \cdot 10^{-5}\,\text{au}$. It is therefore convenient to model the gas as a fluid. Dust grains are typically separated by a few hundred kilometers. As the dynamics of dust grains are largely restricted by the gas, without including dust-dust collisions, and their mean separation being below the scale of interest, they are also approximated to be a fluid. It should be noted that this matter is currently subject to active debate (e.g. Lynch et al. 2024).

2.2. Minimal disc model

We begin by considering a disc made entirely of gas (without magnetic fields), in an external inertial frame, centered on a single star. We use cylindrical coordinates (r, ϕ, z) where the midplane of the disc is aligned in the coordinates r, ϕ, see Fig. 1.6. In this simple model, in a first approach, the following physical conditions are assumed

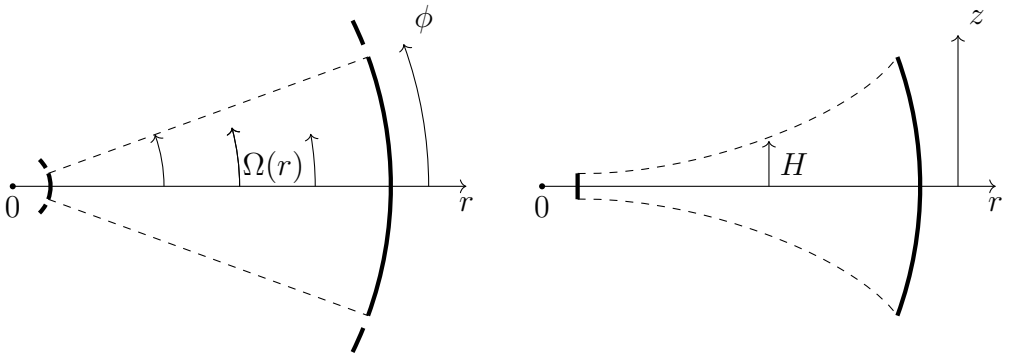


Figure 1.6: An illustration of a simple protoplanetary disc, depicted face-on on the left and edge-on on the right, complemented with typical disc parameters. Ω denotes the angular frequency of close to Keplerian orbits, at a radius r, and H is the pressure scale height.

Minimal disc model assumptions

- the disc is isolated and orbits a single star of mass M_*.
- the mass of the disc is negligible compared to the mass of the star $M_{\text{disc}} \simeq 0.01 M_*$ corresponding to typical discs in a nebula.
- the disc is circular and axisymmetric (all fields are independent of ϕ).
- the disc is thin, meaning that its vertical extension H, which will be properly defined, is negligible compared to the distance to the star $H(r) \ll r$. This does correspond to a cold disc.
- the vertical and radial structures are decoupled.

2.2.1. Rotation profile

The gas can be described by an Euler equation, in which only the external gravitational force of the central object is considered

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{1.1}$$

$$\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{1}{\rho} \nabla P + \mathbf{f}_G, \tag{1.2}$$

where $\mathbf{f}_G = -GM_* \mathbf{r}/||r||^3$ is the gravitational force of the central object, ρ the gas density, P the gas pressure, and $\mathbf{v}$ its velocity.

Neglecting the vertical direction, accounting for the axisymmetry and seeking

steady-state solutions one obtains

$$(\mathbf{v}\cdot\nabla\mathbf{v})_r = -\frac{1}{\rho}(\nabla P)_r - GM_*\frac{1}{r^2}, \tag{1.3}$$

$$(\mathbf{v}\cdot\nabla\mathbf{v})_\phi = 0, \tag{1.4}$$

which results in

$$v_\phi = \sqrt{\frac{GM_*}{r} + \frac{r}{\rho}\frac{\partial P}{\partial r}}, \tag{1.5}$$

where the angular frequency is defined according to $\Omega \equiv v_\phi/r$. The second term of Eq. 1.5 is corrective (see justification below). The disc is therefore here supported radially in majority by its rotation. The proximity of the inner, denser, region of the disc to the star results in a higher temperature, which leads to a higher sound speed and subsequent elevated pressure. Therefore, the radial component of the pressure gradient is generally negative. This implies, from Eq. 1.5 that the gas is orbiting at an orbital velocity lower than the Keplerian velocity $v_K = \sqrt{GM_*/r}$, which is referred to as a sub-Keplerian rotation.

It must be noted that having a rotational profile on circular orbits with an angular rotation of $\Omega_{\rm K}(r) \simeq \sqrt{GM_*/r^3}$ results in an angular momentum density of $l[\Omega_{\rm K}](r) = r^2\Omega_{\rm K} = \sqrt{GM_*r}$. Angular momentum increases with radius, implying that some angular momentum must be lost in order for an orbit to shrink in radius. The same holds for sub-keplerian rotation, as pressure only is a small correction of order $c_{\rm s}^2/v_K^2$ since the disc is thin, as we will show.

2.2.2. Vertical density profile

The disc is assumed to be thin ($H(r) \ll r$), which implies that we can approximate the vertical component of the gravitational force from the central star at first order to be

$$f_{G,z} = -\frac{GM_*}{r^3}z + \mathcal{O}(z^2) = -\Omega_{\rm K}^2 z + \mathcal{O}(z^2), \tag{1.6}$$

where M_* is the central star mass, $\Omega_{\rm K} = \sqrt{GM_*/r^3}$ the corresponding Keplerian angular velocity at a distance r. This yields the following equation for the vertical hydrostatic equilibrium

$$0 = -\frac{1}{\rho}\partial_z P - \Omega_{\rm K}^2 z. \tag{1.7}$$

In this minimal model, since the gas is optically thin and the resulting cooling timescale is short compared to the orbital timescale, it is possible to assume the equation of state is locally and vertically isothermal ($P = c_s(r)^2\rho$), where the sound

speed depends on the distance to the host star (e.g. D'Alessio et al. 1998; Miotello et al. 2023). Under this assumption Eq. 1.7 becomes

$$\partial_z \rho = -\frac{\Omega_{\rm K}^2}{c_s(r)^2}\rho z, \tag{1.8}$$

which admits for solution the following Gaussian vertical density profile

$$\rho(r,z) = \rho_0(r)\exp\left(-\frac{\Omega_{\rm K}^2}{c_s(r)^2}\frac{z^2}{2}\right). \tag{1.9}$$

Using the scale height $H = c_s/\Omega_{\rm K}$ and the surface density profile $\Sigma(r,\theta) = \int \mathrm{d}z\, \rho(r,\theta,z)$ we can rewrite Eq. 1.9 as

$$\rho(r,z) = \frac{\Sigma}{H\sqrt{2\pi}}\exp\left(-(z/H)^2/2\right). \tag{1.10}$$

In conclusion, the disc is supported both vertically by pressure and radially by rotation. A summary of the corresponding parameters can be found in Fig. 1.6.

2.2.3. Supersonic orbital motion

The minimal disc model is vertically stratified, and its velocity is only azimuthal. This implies that the orbital flow is divergence-free. However, assuming a thin disc approximation is equivalent to assuming that the gas flow within the disc is highly supersonic as

$$\frac{H}{r} = \frac{c_s}{v_\phi} = \frac{1}{\mathcal{M}} \ll 1,$$

where in a standard protoplanetary disc ($M_* = 1\mathrm{M}_\odot, T = 300\mathrm{K}$) at $r = 1\mathrm{au}$.

$$c_s = 1.03\mathrm{km\,s^{-1}} \ll v_K = 29.8\mathrm{km\,s^{-1}} \Rightarrow \frac{H}{r} \simeq 0.03 \ll 1.$$

Therefore, the orbital motion is a supersonic divergence-free flow. This implies that the mean flow is incompressible, but that strong perturbations (e.g. planets or other effects) will lead to the presence of shocks.

2.2.4. Power laws

The disc's surface density and its sound speed are usually parametrized by power laws. For the density we use

$$\Sigma(r) = \Sigma_0\,(r/r_0)^{-p}, \tag{1.11}$$

where Σ_0 is the surface density at r_0. For the sound speed profile, we use

$$c_s(r) = c_0(r/r_0)^{-q}, \tag{1.12}$$

where c_0 is the sound speed at radius r_0. This choice of sound speed profile yields a scale height

$$H(r) = \left(\frac{c_0}{\sqrt{GM/r_0^3}}\right)(r/r_0)^{\frac{3}{2}-q}, \tag{1.13}$$

where for $q < 1/2$ the disc is flared ($H(r)/r$ is increasing a function of r), as represented on Fig. 1.6. In summary for a simple, gas-only, thin, axisymmetric disc we have in first approximation as density and rotation profile:

$$\rho(r,z) = \frac{\Sigma_0}{H\sqrt{2\pi}} \exp\left(-(z/H)^2/2\right)(r/r_0)^{-p}, \tag{1.14}$$

$$H(r) = \left(c_0/\sqrt{GM/r_0^3}\right)(r/r_0)^{\frac{3}{2}-q}, \tag{1.15}$$

$$\Omega = \frac{1}{r}\sqrt{\frac{r}{\rho}\frac{\partial P}{\partial r} + \frac{GM_*}{r}}. \tag{1.16}$$

A more complete version can be found (e.g. in Nelson et al. 2013) without the vertical/radial decoupling (azimuthal speed depends on height) and a better treatment of the thermodynamic. Rotation and thermodynamic support are derived consistently to ensure consistency of the vorticity equation.

2.3. Dust evolution

2.3.1. Two fluid model

Let us now understand the basics of dust evolution on this simple disc. Dust grains in protoplanetary discs are porous fractal aggregates made mostly of silicates. Grains are modeled by spheres of mean radius s for small grains ($s \lesssim 10$cm) and with a density of $\rho \sim 1\mathrm{g\cdot cm^{-3}}$ when compacted (Love et al., 1994). In a typical protoplanetary disc, the relative density of dust relative to the gas is taken to be a conservative value of the order of the percent (e.g. Testi et al. 2014). Furthermore, the mean free path of the gas is of the order of 100m, and the dust particles are smaller than a centimeter, which implies that the interactions between the dust and gas particles are collisional. This regime is known as Epstein-regime (Epstein, 1924), in which the drag is solely collisional. The dust is approximated to be a pressureless fluid following an Euler equation which is coupled to the gas through a drag force, which takes the form of an average force $\mathbf{f}_{\mathrm{drag}}$. Assuming that the dust is made of spherical compact grains, it can be written (as in Bae et al. 2023) as

$$\mathbf{f}_{\mathrm{drag}} = -\frac{4}{3}\pi a^2 \rho_g v_{\mathrm{th}} \Delta\mathbf{v}, \tag{1.17}$$

where the drag force $\mathbf{f}_{\mathrm{drag}}$ depends on the cross-section of the dust grain πa^2, the density of the gas ρ_g, $\Delta\mathbf{v} = \mathbf{v}_{\mathrm{d}} - \mathbf{v}_{\mathrm{g}}$ the differential velocity between the gas fluid

and the dust particle, and $v_{\rm th}$ is the thermal velocity of the gas molecules. Drag implies a transfer of momentum between gas and dust. Momentum conservation therefore requires a so-called back reaction of the dust onto the gas, which takes the form of a force $-\mathbf{f}_{\rm drag}$. In summary, the equations for a mixture of dust and gas here can be written as follows:

$$\frac{\partial \rho_{\rm g}}{\partial t} + (\mathbf{v}_{\rm g} \cdot \nabla)\, \rho_{\rm g} = -\rho_{\rm g}\, (\nabla \cdot \mathbf{v}_{\rm g})\,, \tag{1.18}$$

$$\frac{\partial \rho_{\rm d}}{\partial t} + (\mathbf{v}_{\rm d} \cdot \nabla)\, \rho_{\rm d} = -\rho_{\rm d}\, (\nabla \cdot \mathbf{v}_{\rm d})\,, \tag{1.19}$$

$$\frac{\partial \mathbf{v}_{\rm g}}{\partial t} + (\mathbf{v}_{\rm g} \cdot \nabla)\, \mathbf{v}_{\rm g} = -\frac{\nabla P}{\rho_{\rm g}} - \mathbf{f}_{\rm drag} + \mathbf{f}_G, \tag{1.20}$$

$$\frac{\partial \mathbf{v}_{\rm d}}{\partial t} + (\mathbf{v}_{\rm d} \cdot \nabla)\, \mathbf{v}_{\rm d} = \mathbf{f}_{\rm drag} + \mathbf{f}_G, \tag{1.21}$$

! In such a model, approximating the dust grains by a fluid neglects the multi-valuedness of the velocity at a given position. When the dust is taken to be pressureless, it then equates to neglecting the width of the velocity dispersion of the dust particles modeled by the fluid. Special care should ideally be taken, for example in a protoplanetary disc where crossing orbits of dust particles does correspond under such a framework to a dust pressure without requiring the presence of dust-dust collisions (Lynch et al., 2024).

2.3.2. Stopping time and Stokes number

The typical drag time, which measures the intensity of the drag force, is also called the stopping time $t_{\rm s}$. It is the characteristic time on which the velocity of a single grain is damped to the velocity of the gas. Formally, the stopping time is defined as

$$t_{\rm s} = \frac{m|\Delta v|}{|\mathbf{f}_{\rm drag}|} \underset{Epstein}{=} \frac{\rho_s}{\rho_g} \frac{a}{v_{\rm th}}. \tag{1.22}$$

This stopping time can be compared to the global timescale of the studied problem, which is the orbital timescale. This dimensionless number is in an astrophysical context called the Stokes number and is defined as

$$\mathrm{St} = \Omega_{\rm K} t_{\rm s}. \tag{1.23}$$

By definition, a Stokes number lower than one represents dust grains being well coupled to gas, and a Stokes number larger than one does correspond to grains mostly decoupled from gas. In typical discs, sub-millimeter grains are expected to have $\mathrm{St} \lesssim 1$, whereas pebbles and larger bodies are expected to have $\mathrm{St} \gtrsim 1$.

2.3.3. Terminal velocity approximation

Grains with St $\ll 1$, corresponding to small grains, are heavily coupled to the gas. Their stopping time being very small compared to the global timescale implies that their velocity can be taken to always be the terminal velocity. This is called the terminal velocity approximation, where the differential velocity can be rewritten as

$$\Delta \mathbf{v} = \mathbf{v}_\mathrm{d} - \mathbf{v}_\mathrm{g} = t_s \frac{\boldsymbol{\nabla} P_g}{\rho_\mathrm{g}}, \tag{1.24}$$

where $\Delta \mathbf{v} = \mathbf{v}_\mathrm{d} - \mathbf{v}_\mathrm{g}$ is the differential velocity, and subscript g denotes quantities related to the gas, and subscripts d to the dust. It is important to note that Eq. 1.24 implies that dust is diffusing towards local maxima of pressure in the terminal velocity approximation.

2.3.4. Dust drift

The gas, in absence of dust, is sub-Keplerian, however the dust does not have any similar pressure support. On the other hand, pressureless dust would orbit at Keplerian speed in absence of gas. Drag between gas and dust therefore results in a transfer of angular momentum from the dust to the gas, making the dust drift towards the center of the disc and pushing the gas outwards (Whipple, 1972; Adachi et al., 1976; Weidenschilling, 1977; Nakagawa et al., 1986). Formally, such drift speed can be computed with a perturbative approach in $c_\mathrm{s}^2/v_K^2$and is (as written in Bae et al. 2023)

$$v_{\mathrm{d},r} = \frac{-\eta v_K}{\mathrm{St} + \mathrm{St}^{-1}}, \tag{1.25}$$

where η quantifies the sub-keplerian correction to the velocity profile

$$\eta = -\left(\frac{c_s}{v_K}\right)^2 \frac{\mathrm{d}\ln P}{\mathrm{d}\ln r}. \tag{1.26}$$

Additional terms should be considered if the gas has a radial motion (e.g. viscous diffusion, see below). The drift speed has a maxima for St $\simeq 1$, which corresponds, historically metre-sized grains (nebulae structure were purely speculative) that drift onto the stars in about 1000 years. Under those conditions, meter-sized dust grains will always be lost because of drift, hence the designation of the meter barrier as given by Weidenschilling (1977). Nowadays, nebulae structures are constrained by observations and St ~ 1 correspond to mm-sized grains.

2.3.5. Dust settling

Apart from the radial drift previously mentioned, drag in protoplanetary discs also creates vertical motion in the dust. As shown in Sec. 2.2, the vertical density profile is Gaussian, $P = c_\mathrm{s}^2 \rho \propto e^{-(z/\sqrt{2}H)^2}$, as is the pressure since the equation of state

is locally isothermal. In the terminal velocity approximation, this results in dust settling and concentrating toward $z = 0$. This phenomenon is known as dust settling, as the dust is moving towards the disc midplane. This is also true outside the terminal velocity approximation, where instead dust grains will experience weakly damped oscillations toward the midplane. Without an additional source of dust diffusivity, such as turbulence, grains would concentrate to an infinitely thin layer and be likely be prone to a gravitational instability (e.g. Goldreich & Ward 1973).

3. State of the art (Processes)

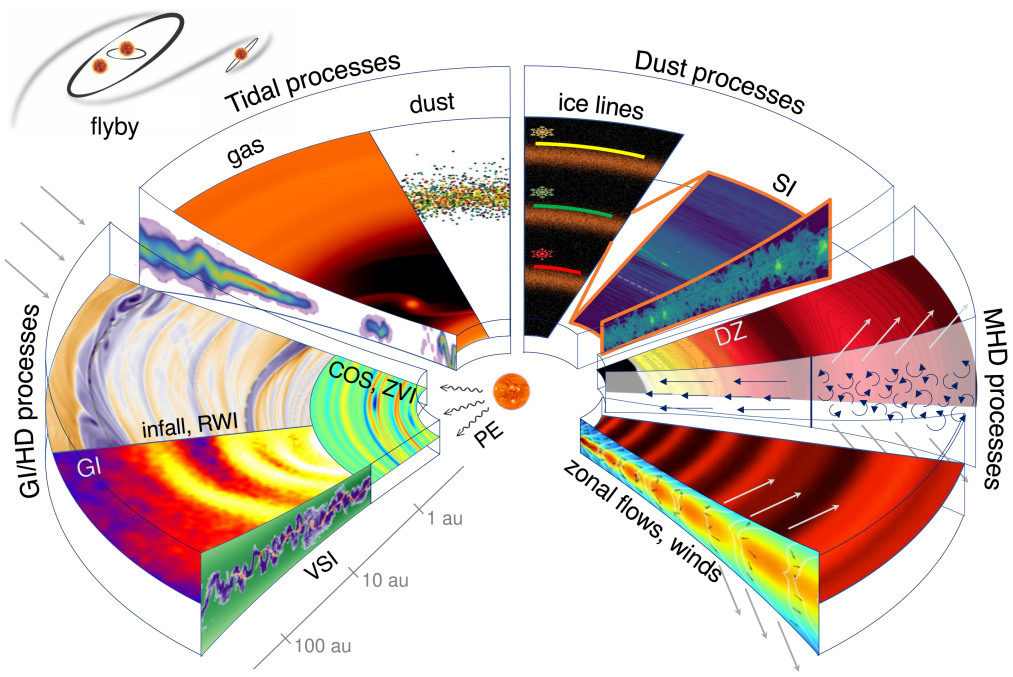


Figure 1.7: Illustration of the principal effects leading to substructure formation, adapted from Bae et al. (2023). This figure shows the different physical processes involved in a disc as well as a visual representation and localization of the corresponding phenomenon. This shows the different scales involved as well as the corresponding substructures possible.

Real discs are more complex than our simple model and include many additional processes. In the next two sections, we present a short overview of the mechanisms so far thought to be important for planet formation, most of which have been recently reviewed in Bae et al. (2023); Lesur et al. (2023a); Paardekooper et al. (2023). We aim at identifying the key processes that have to be integrated in a numerical simulation of planet formation, in relation to the numerical locks and challenges associated. We first present an overview of phenomena present in protoplanetary

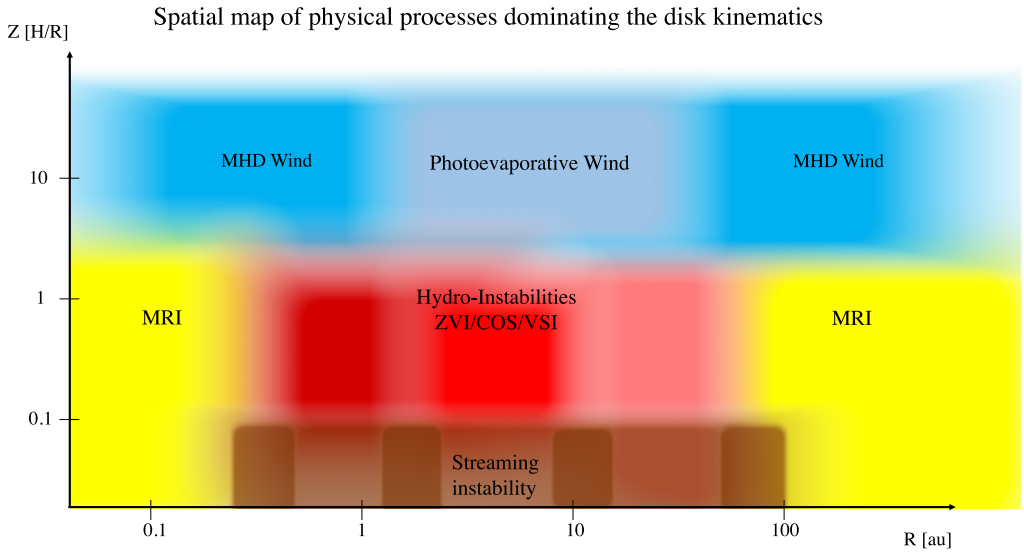


Figure 1.8: Representation of the radial and vertical localization of principal processes involved in protoplanetary disc evolution (Source: Lesur et al. (2023a)). This figure shows the different physical processes and instabilities occurring in the disc's regions, involving different physical processes.

discs, along with their corresponding localization and associated physical processes, as shown in Fig. 1.7.

3.1. Accretion and angular momentum transport

3.1.1. Observations

In protoplanetary observations, accretion rates on the central stars are measured to be of the order of $\sim 10^{-10} - 10^{-7} \mathrm{M}_{\odot} \cdot \mathrm{yr}^{-1}$ (e.g. Venuti et al. 2014; Hartmann et al. 2016; Manara et al. 2016). Beside disc mass being accreted on the central stars, mass can also be expelled from the system altogether in the form of winds. When ejected or accreted, material carry angular momentum with it. This implies that in order to understand the evolution of disc radius, mass, and density profiles, we need to understand how angular momentum and mass are transported in discs.

3.1.2. Turbulent transport

In order to explain accretion rates and angular momentum transport, Shakura & Sunyaev (1973) have in the context of black holes proposed to add an effective Navier Stokes shear viscosity $\nu = \alpha_S c_s H$ to Eq. 1.1- 1.2. Turbulence is considered to be the source of this pseudo-viscosity when averaged on larger scales. This simple model of turbulence with one parameter is still widely used in the disc community, despite more refined models such as the k-epsilon one (Launder & Spalding, 1974) being

often considered in other communities. Indeed, the alpha-model remains generally sufficient to interpret both observations and numerical simulations, and is easy to use in analytical models. Viscosity creates accretion on the central object, therefore, by conservation of angular momentum, transports outward, implying a disc expanding in radius. Current observational constraints suggest that, if they are effectively viscous, discs have a low viscosity of $\alpha_S < 10^3 - 10^4$ (Flaherty et al., 2015; Teague et al., 2016; Flaherty et al., 2017, 2018). However, accurate modeling of turbulence requires a more complex model or resolving turbulence directly.

3.1.3. Winds

Outflows (material becoming gravitationally unbound) are an alternative mechanism, initially proposed by Blandford & Payne (1982) to explain the accretion rates of young stars since they carry, in the form of winds or jets, material and angular momentum out of the disc. Outflows can have different origins (e.g. Lesur et al. 2023a). The first type of outflows are thermal outflows, where radiative sources (radiation from the star, or local currents, for example) may result in heating of the disc surface above the liberation temperature of the disc, resulting in thermally unbound material. In addition to thermal outflows, ejection of material can also originate from magnetohydrodynamical processes (see Sec. 3.3), where ionized material will follow the magnetic field lines that are twisted as they are advected by the orbital motion. If the diffusion of the field lines is weak enough, the resulting magnetic pressure leads to material on the surface of the disc becoming unbound gravitationally.

3.1.4. Summary

Mechanisms leading to an effective pseudo viscosity, such as instabilities, redistribute angular momentum in the disc. Redistribution of angular momentum triggered by a dissipative effect leads to mass flow and accretion onto the central star, while the disc expands to conserve the total angular momentum. On the contrary, outflows carry angular momentum outside the system, which results in a net decrease in the total momentum, leading to the disc shrinking in radius. Overall, angular momentum can be either redistributed in the disc with mechanisms leading to an effective pseudo-viscosity, such as instabilities, or extracted from the disc by outflows that will carry it outside the system. This balance sets the evolution of the surface density profile of the disc as well as the accretion rate onto the star.

3.1.5. Numerical challenges

A disc combined the challenges of celestial mechanics related to the advection around the central object, and local hydrodynamical processes.

Orbital parameters The orbital aspects of a disc flow are related to the fundamental principles of mechanics, which are best formalized in a Lagrangian formalism. Additionally, a strong emphasis is drawn toward the conservation of angular momentum and energy when studying celestial mechanics (or more generally, orbital elements). In Lagrangian numerical methods such as Smoothed Particle Hydrodynamics (SPH), it is possible to conserve angular momentum to machine precision, making them suitable for such simulations. Additionally, the use of a symplectic integrator in particle-based methods allows for better conservation of energy (exact up to $\mathcal{O}(\Delta t^2)$ for the leapfrog, when Δt is constant). This ensures the stability of the system, which is not necessarily the case with a non-conservative scheme of higher orders. Lastly, the scheme, being Lagrangian in nature, results in accuracy of advection and conservation properties that are exact, provided with an exact time integrator, unlike grid-schemes. On the other hand, the behavior of grid-based method depends on the scheme used. For example, finite volume Godunov-type schemes can be very diffusive for flows that do not align with the grid axis. This can be mitigated using a cylindrical geometry such that the flow is almost aligned with the grid. However, if the disc presents a more complex dynamical structure, such as a warp (when the position of the disc midplane is not constant in z), this would again cause a loss of precision. Alternatively, it is possible to reformulate the numerical scheme to improve the conservation of angular momentum while accelerating the computation of orbital advection. An example of such a case is the FARGO algorithm (also called fast orbital advection), initially introduced in Masset (2000), where the advection on the mean velocity profile is subtracted from the numerical scheme and calculated independently, resulting in better accuracy and a reduction in the number of timesteps required to integrate a given system in general.

Hydrodynamics For the local hydrodynamical processes, discs are stratified, both radially and vertically, with large density contrasts (e.g. Gaussian vertical density profile, shocks, planet embryos). Discs being supersonic also implies that strong perturbations will create local structure due to strong compression along the orbits. Numerical methods for discs must capture relevant physical effects in both regions of large densities (formation of planet embryos) and low densities (atmospheres) and resolve compressible hydrodynamics. Additionally, hydrodynamical shocks resulting from the dynamics must be also resolved either using Riemann-Solvers based methods (see Chapter 3) or methods artificial viscosity (see Chapter 2). With regard to the α_S viscosity in discs, two strategies can be adopted to simulate transport originating from small scales in a protoplanetary disc. The first one is to implement an effective viscosity that mimics the prescription of Shakura & Sunyaev (1973). Even though the resolution must be sufficient for the physical viscosity to dominate over the effective numerical viscosity that results from truncation errors in the numerical scheme (e.g. Meheut et al. 2015). A more physical alternative to the α_S prescription consists in simulating the transport of physical quantities at small scales directly. However, such task requires an even larger resolution. Additionally,

simultaneously resolving regions of large and low density is challenging, such as a vertical slab of the disc with outflows that have as a basis a thin, irradiated layer of the disc atmosphere. They often correspond to low-density regions, which can be resolved by meshless methods (see Sec. 3.4). Additionally, in the traditional formulations of SPH, the resolution follows the density, resulting in lower resolution in the outflows compared to grid-based methods. Alternative formulations, such as the use of unequal mass particles, also imply difficulties related to the numerical stability of the underlying lattice of particles. Finally, methods such as SPH tend, because of inherent stochasticity of particle-based methods, to result in high numerical viscosities compared to their grid-based counterparts. To date, hydrodynamical solvers rely on explicit schemes. Consequently, the computational cost scales with the fourth power of the resolution due to the three spatial dimensions and the Courant-Friedrichs-Lewy (CFL) condition. The memory cost of the algebra of high-order implicit schemes, associated with their additional algorithmic complexity, has prevented them from being used in the community so far.

High resolution constraints In order to achieve suitable results for angular momentum transport, high resolution is required. However, disc simulations can be expensive computationally, as, for example, resolving a disc orbit can necessitate thousands of timesteps, and this for hundreds of orbits for some processes. Adding additional physics, such as magnetohydrodynamics (MHD), can further increase constraints on the timesteping Courant-Friedrichs-Lewy (CFL) condition, further increasing the computational cost. This highlights the need for methods that adapt to the density gradient or the density itself to resolve regions of interest without incurring the cost of regions that would not be as relevant for the study. Examples include Adaptive Mesh Refinement, which dynamically refines the grid used to simulate the problem in regions of strong gradients, or Smoothed Particle Hydrodynamics, where the resolution follows the mass, thus having a resolution that adapts itself to the disc density. Another option is also to rescale the grid, such as using a spherical grid where the radial coordinate is logarithmic (as in Lesur et al. 2023b, for example).

In order to further improve the speed or comparatively reduce the numerical cost. Strategies involving individual or hierarchical timestepping have been used. They are schemes where particles or cells in the simulation are updated with respect to their individual timestepping criteria (or per bin, in hierarchical timestepping). This allows the updating of cells or particles only when necessary. This results in a net decrease in the amount of computation to perform per orbit, as in disc simulation, the timestepping criterion can vary by multiple orders of magnitude between the innermost and outermost orbits of the disc. Such methods have been implemented in both particle and grid-based methods, with the example of hierarchical timestepping in SPH, detailed in Price et al. (2018), or the Adaptive Mesh Refinement Godunov method with adaptive timestepping described in Teyssier (2002). This method, in SPH comes to the cost of breaking the rigorous conservation of

momentum. Another possibility is to leverage schemes that lift some constraints on time-stepping Courant-Friedrichs-Lewy (CFL) conditions, such as the FARGO algorithm Masset (2000). The key challenge of these approaches is the conservation of orbital quantities.

Boundary conditions Lastly, boundary conditions in disc problems are not given by simple mathematical conditions a priori, but are rather related to complex physical processes developing there. For example, the inner part of the disc is connected to the magnetosphere of the star and potentially the star itself. On the outer part, the boundary condition is the circumstellar envelope, the interstellar medium itself, or an infall of material on the disc. In the vertical direction, the boundary conditions are the disc atmosphere, which can include winds or an external medium if the disc is embedded in an envelope of infalling material. The surface is also sometimes treated to be free. It should be noted that in the external layers, the molecular density is sometimes too low for the fluid approximation to be valid anymore. Those boundary conditions have to be correctly captured by the chosen numerical method, and limitations associated to the choice of boundaries should be properly identified and discussed. In that regard, grid-based methods allow for most kinds of boundaries, except moving boundaries, which require a change in the geometry of the problem or free boundary conditions, as grid methods are only able to represent finite volumes. In particle-based methods such as SPH, moving boundaries and free boundaries are possible. However, fixed complex boundaries are more complex to implement compared to their grid counterparts (see, for example, the use of ghost particles to approximate a Dirichlet boundary condition in Price et al. (2018)).

Summary In summary, it is now generally admitted that for problems that involve smooth flows that can be aligned with grid geometry, grid-based methods can be suitable as they provide more flexibility with regard to the boundary condition, numerical viscosity, and grid refinement. However, for problems involving complex geometries or mechanical evolutions, SPH may be more appropriate.

3.2. Dust evolution

3.2.1. Coagulation & fragmentation

Dust grains observed in protoplanetary discs correspond to complex aggregates of nanometer-sized grains that can be fluffy or compacted, as shown in Fig. 1.9. Most grains are made of multiple spherical monomers. Such aggregates grow by sticking together due to electrostatic forces (Blum, 2018). During dust-dust collisions many outcomes are possible. Dust grains can stick ($s < 10\mu m$), or bounce ($10\mu m < s < 100\mu m$). When more kinetic energy is involved, the outcome of collisions can be more complex, with mass transfer between the grains or, in extreme cases, fragmentation of the grains upon collisions (Blum & Wurm, 2008). Exam-

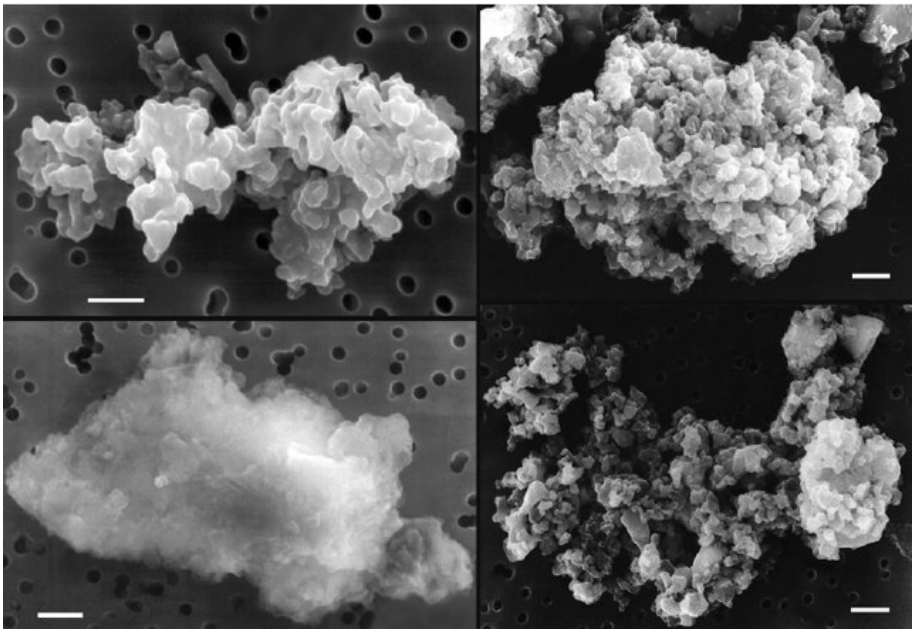


Figure 1.9: Example pictures of interplanetary dust particles. The white line in each panel corresponds to 1 μm size. Source: (Zubko, 2012, Fig. 2.1)

ples of possible outcomes for collisions of two different dust grains are shown in Fig. 1.10. Formally, the description of coagulation and fragmentation follows the

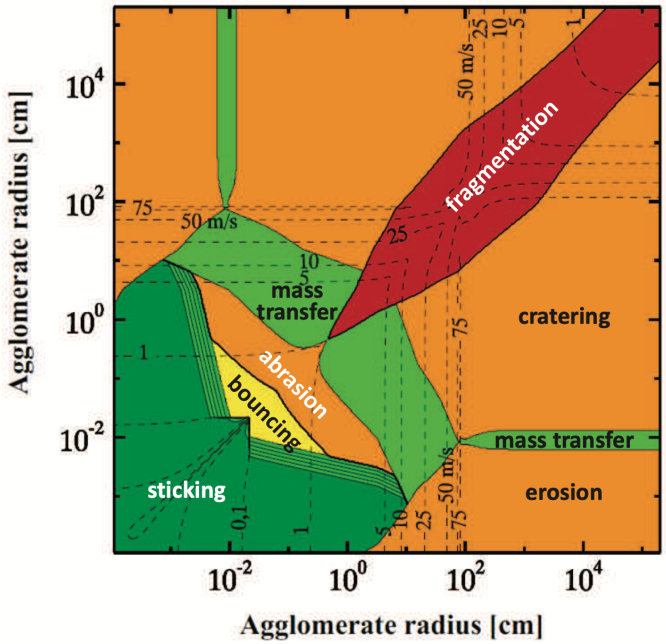


Figure 1.10: Example of possible outcomes of dust-dust collisions for silicate monomer particles having a radius of 0.75µm. Taken from (Blum, 2018, Fig. 1)

Smoluchowski equation (Smoluchowski, 1918). It is an integro-differential equation that generalizes mass conservation for a continuous distribution in the size of aggregates. The averaged microphysics of the collision is encoded in a collision kernel that depends in particular on the averaged relative velocities between the colliding objects, describing the dust-to-dust collisions shown in Fig. 1.10.

3.2.2. Interplay with dynamics

Dust growth directly affects the dust size distribution, which in turn impacts the dynamics of the dust and subsequent instabilities. It also determines the formation of grains of intermediate sizes (St $\sim$ 1), for which the decoupling with the gas is the most effective. This implies more effective concentration and a more intense back-reaction. Dynamics determines the outcome of growth in two ways: directly, by setting the relative velocities between the particles, or indirectly, by enhancing collision rates by concentrating particles at certain locations of the disc. The interplay between coagulation and fragmentation leads to the formation of regions with strong back-reaction gradients. As a response, gas is pushed outwards differentially, and the associated compression can possibly lead to the formation of a self-induced pressure maximum that, in return, will further trap and concentrate solids (Gonzalez et al., 2017).

3.2.3. Numerical challenges

Dust dynamics Simulating a dust-gas mixture can be done using multiple approaches. A first one is to model dust as Lagrangian particles of finite masses. In this *dust as particles* approach, the dust grains are modeled using particles that are coupled with the gas through drag forces. In SPH, typically, this approach is done using two sets of SPH particles, where one represents the gas (see Sec. 4) and the other the dust. A key difference for the dust is that no artificial viscosity is used, allowing particle interpenetration. This approach can be seen as a sub-sampling of a collisionless Boltzmann equation with particular initial conditions rather than a fluid. In particular, multivalued velocity distribution can be modeled by Lagrangian particles (see details in Price et al. 2018). A similar kind of approach can also be used on grid-based methods using tracer particles, which can deposit and take momentum from the grid cells according to the drag force. An issue with the Lagrangian dust-as-particle approach is that since the singlevaluedness of the velocity is not guaranteed, it is a priori required to sample the velocity space in addition to the position space. This corresponds to so-called Particles-In-Cell (PIC) methods more than hydrodynamics. However, such a modeling is more costly as it requires simulations of six-dimensional spaces rather than three-dimensional ones. However, for small grains (St $<$ 1), the strong coupling of the dust to the gas results in short stopping times that must be resolved by the simulation. This results in the integration of small dust sizes dominating the CFL, slowing, in turn, the simulation. This led to the development of so-called *one-fluid* methods, as introduced in Laibe

& Price (2014). In a one-fluid or monofluid approach, the dust, as well as the gas, is modeled directly as a fluid. Instead of simulating the two fluids having two velocities $(\mathbf{v}_\mathrm{g}, \mathbf{v}_\mathrm{d})$ and two densities $(\rho_\mathrm{g}, \rho_\mathrm{d})$, one density ρ, one velocity $\mathbf{v}$, and two internal quantities $(\epsilon, \Delta\mathbf{v})$ are used, keeping the same number of degrees of freedom. The dusty mixture is modeled directly as a single fluid. The transformation is as follows: For density, we use the total density of the two fluids, $\rho = \rho_\mathrm{g} + \rho_\mathrm{d}$, and the velocity is taken to be the barycentric velocity, $\mathbf{v} = (\rho_\mathrm{g}\mathbf{v}_\mathrm{g} + \rho_\mathrm{d}\mathbf{v}_\mathrm{d})/\rho$. The two additional internal variables are the fraction of dust $\epsilon = \rho_\mathrm{d}/\rho$ and the differential velocity $\Delta\mathbf{v} = \mathbf{v}_\mathrm{d} - \mathbf{v}_\mathrm{g}$. In this formalism, the terminal velocity approximation can naturally be taken as the $\Delta\mathbf{v}$, which is constrained by the terminal velocity approximation. The monofluid formalism can then be used in the terminal velocity approximation. This allows the removal of the CFL constraints associated with the small stopping time of dust grains, as this is accounted for by the terminal velocity approximation, but in turn leads to stringent conditions for large grains. The monofluid approach with terminal velocity approximation has been used in SPH (e.g. Price et al. 2018). Recently, extensions to grid methods have also been made in the AMR code Ramses (Lebreuilly et al., 2020). However, the use of monofluid methods in the terminal velocity approximation does not allow the use of large grains ($\mathrm{St} \simeq 1$). This motivated the work of extending current monofluid formalism to include large dust grains that do not follow the terminal velocity approximation that will be presented Appendix C (our novel formulation conserves the dust total momentum in absence of drag). This is of special interest in SPH methods, since SPH dust particles do not reorganize as dust is pressureless. However, deriving a full-monofluid approach is challenging in SPH as current attempts do not conserve the momentum of dust grains (without drag) and can yield wrong results. In AMR, implementing dust-gas mixtures requires the development of dusty Riemann solvers, which can be equally challenging. Additionally, to monofluid methods, two-fluid methods were used to simulate dust grain dynamics in grid-codes such as FARGO3D (Benítez-Llambay et al., 2019). In general, accuracy of methods for dusty mixture are still widely debated (e.g. Commerçon et al. (2023)). Lastly, efforts have been made toward having a consistent framework for the dust pressure in dust-as-fluid approaches, as dust must have a non-null pressure in reality (orbit crossing).

Dust evolution In addition to the dust dynamics, it is necessary to be able to simulate dust growth and fragmentation accurately. The population of large grains is related to the formation of planet embryos, while small grains set the thermodynamical, chemical, and electrical properties of the disc. It is therefore important to simulate them simultaneously. However, simulating dust growth in hydrodynamical simulation is challenging for multiple reasons. Firstly, a proper model of coagulation as well a fragmentation must be used to properly model dust evolution. Crude approaches have been used (Stepinski & Valageas, 1997; Laibe et al., 2008). However, they are not able to capture simultaneously small and large grains, and are not guaranteed to be the correct limit of the Smoluchowski equation. Secondly, when

using a model resolving the Smoluchowski equation by sampling pairwise collisions, such as Brauer et al. (2008), one is required to have many dust sizes ($N_{\text{species}} > 100$) simulated in a given simulation of a protoplanetary to resolve coagulation processes to avoid numerical diffusion. However, as mentioned, each dust size is a fluid that we have to simulate. Resolving coagulation processes would then requires running the equivalent, computationally, of hundreds of simulations at the same time. Solving for the coagulation equation also involves linear algebra algorithms whose complexity does not scale linearly with the number of dust species. To alleviate such limitations, a possible solution is to exploit higher order schemes to achieve similar resolution for the growth processes while reducing the number of species required in the simulation. This was done in Lombart & Laibe (2021) using a Galerkin scheme capable of resolving dust growth while using ten times fewer dust species. However, such a scheme has yet to be coupled with existing codes to be able to perform simulations of protoplanetary discs, including dust growth. One of the associated challenges is that, in practice, computational time spent to solve coagulation and fragmentation should remain of the same order of magnitude as the computational time for a hydrodynamical timestep. This implies the need for strategies of implicit integration or sub-cycling when the CFL condition associated with dust coagulation and fragmentation is much shorter than the hydrodynamical one.

Summary In summary, dust can be modeled in simulations either as Lagrangian particles, in which case they correspond more to a collisionless Boltzmann equation, or as a fluid using two-fluid or one-fluid methods. For simulations using Lagrangian particles, the proper resolution of the model would require a very large number of dust particles to sample the velocity distribution. Using one or two fluid methods for dust gas mixtures allows resolution of the dust to a proper fluid, eliminating the need for velocity space sampling; however, dust pressure is not yet included in such models. Additionally, using a one-fluid method allows for the use of the terminal velocity approximation to simulate small dust grains without the CFL limitation that would be associated with them in other methods. In that same objective, efforts are made toward having a complete one-fluid formalism without the terminal velocity approximation. Lastly, realistic dust size distribution is continuous. Theoretical results have demonstrated that accounting for the polydisperse nature of dust grains results in changes to the dynamics of dust regarding instabilities. No current methods to date are able to simulate all effects related to polydisperse dust-gas mixtures.

3.3. Magnetohydrodynamics in discs

3.3.1. Are disc magnetic ?

The gas in the disc can be ionized due to the central star radiation, or on the outer part of the protoplanetary discs by non-thermal sources such as X-rays in the inter-

stellar medium (Glassgold et al., 2017). Even if no direct evidence of the existence of a magnetic field in discs was found, its presence is suggested by the outflows. Indeed, in observed discs, outflows and jets can be observed with speeds of a few $\mathrm{km\,s^{-1}}$, which are unlikely to be the result of hydrodynamics alone. Magnetohydrodynamical processes, however, can produce such outflows without heating the gas (Lesur et al., 2023a), which does suggest the magnetization of discs.

3.3.2. Magnetohydrodynamics

In ionized fluids, the movement of charges within the fluid results in an induced magnetic field in Maxwell's equations. Additionally, the resulting field exerts feedback on the flow, through the Lorentz force applied to the plasma. This two ways coupling of the magnetic field and the flow is called Magneto-Hydro-Dynamics (MHD). As summarized by Lesur et al. (2023a), differential dynamics between the charges species leads to the Ohm law, where the electric field in the local rest frame of the gas ($\mathbf{E}'$) depends on the current density ($\mathbf{J}$) through a tensorial electrical conductivity. This is due to the magnetic field deflecting mobile charges in the plasma resulting in a current that is not parallel to the electric field (Wardle & Ng, 1999). In protoplanetary discs, the ionized material mass is negligible compared to the gas density, and the electric field in the local rest frame is the sum of contributions of three terms (Balbus & Terquem, 2001; Bai, 2014)

$$\mathbf{E}' = \eta_{\mathrm{Ohm}}\mathbf{J} + \eta_{\mathrm{Hall}}\,\mathbf{J}\times\hat{\mathbf{B}} + \eta_{\mathrm{AD}}\hat{\mathbf{B}}\times(\mathbf{J}\times\hat{\mathbf{B}}), \tag{1.27}$$

where η_{Ohm} is the Ohmic diffusivity, η_{Hall} the Hall diffusivity, and η_{AD} the ambipolar diffusivity. Therefore, the induction equation is (Mouschovias et al., 2009; Tsukamoto et al., 2023)

$$\begin{aligned}\frac{\partial\mathbf{B}}{\partial t} = {}& \boldsymbol{\nabla}\times(\mathbf{v}\times\mathbf{B}) - \boldsymbol{\nabla}\times\Big\{\eta_{\mathrm{Ohm}}\boldsymbol{\nabla}\times\mathbf{B} \\ & + \eta_{\mathrm{Hall}}\,(\boldsymbol{\nabla}\times\mathbf{B})\times\frac{\mathbf{B}}{B} + \eta_{\mathrm{AD}}\frac{\mathbf{B}}{B}\times\left[(\boldsymbol{\nabla}\times\mathbf{B})\times\frac{\mathbf{B}}{B}\right]\Big\},\end{aligned} \tag{1.28}$$

where the magnetic field follows the solenoidal condition

$$\boldsymbol{\nabla}\cdot\mathbf{B} = 0. \tag{1.29}$$

Lastly, the equation for the flow of the gas is, in MHD,

$$\partial_t\mathbf{v} + (\mathbf{v}\cdot\nabla)\mathbf{v} = -\frac{\boldsymbol{\nabla}P}{\rho} + \mathbf{f}_{\mathrm{ext}} + \frac{1}{4\pi\rho}(\boldsymbol{\nabla}\times\mathbf{B})\times\mathbf{B}. \tag{1.30}$$

Those equations are the non-ideal MHD equations. An approximate version can be used by neglecting the additional conductivities ($\eta_{\mathrm{Ohm,Hall,AD}}$) in Eq. 1.28. This approximation is the ideal-MHD. With the identity

$$\frac{1}{2}\nabla(\mathbf{B}\cdot\mathbf{B}) = (\mathbf{B}\cdot\nabla)\mathbf{B} + \mathbf{B}\times(\nabla\times\mathbf{B}), \tag{1.31}$$

it is possible to rewrite the Lorentz force in Eq. 1.30 as

$$\mathbf{f}_M = \frac{1}{4\pi\rho}(\mathbf{B}\cdot\nabla)\mathbf{B} - \frac{1}{8\pi\rho}\nabla(\mathbf{B}^2), \tag{1.32}$$

where the term $-\frac{1}{8\pi\rho}\nabla(\mathbf{B}^2)$ is the magnetic pressure. The relative influence of thermodynamics and magnetic pressure is characterized by a non-dimensional number called β-plasma parameter, defined as

$$\beta = 8\pi P/B^2, \tag{1.33}$$

where a typical value in a protoplanetary disc in a typical T-tauri disc is of the order of $\beta \sim 10^4$ (Lesur, 2021a). All the non-ideal effects are important at different steps of the formation and evolution of the disc (e.g. Tsukamoto et al. 2023; Lesur et al. 2023a; Cui & Bai 2022), requiring, ideally, the treatment of the complete non-ideal MHD. In particular the non-ideal MHD effects on a protoplanetary disc is a current active research topic as they affect magnetically driven turbulence (MRI turbulence, see Sec. 4) and can create axisymmetric long-lived features in non-ideal regions (Lesur et al., 2023a).

3.3.3. Numerical challenges

The first universal challenge associated with MHD is that the scheme must enforce the solenoidal condition (Eq. 1.29), which, when not satisfied, results in magnetic monopoles yielding unphysical results. A naive approach consists of using potentials, such as the vector potential or Clebsch quantities, but the problem translates into maintaining a similar accuracy on the gauge constain (see, e.g. Price 2010). Simulating MHD is then commonly done using two approaches. The first one is the so-called divergence cleaning method, where an additional field coupled to the magnetic field is introduced with the role of diluting the residual of numerical errors by propagating them before dissipating them. This is the 8-wave scheme described in Powell et al. (1999). However, since magnetic monopoles are dissipated by the scheme, the magnetic field is not strictly divergence-free. Another approach is to use so-called contrained transport schemes, where the balance of magnetic flux is enforced to machine precision. This method is particularly suited when volumes are explicit in the simulation (e.g. finite volume methods). In the procedure, detailed in Evans & Hawley (1988), the magnetic field is represented as face-averaged quantities that are evolved using constrained transport. It relies on the Stokes theorem to evolve the magnetic field using the countour integral of the electric field on the edges of the cell. This ensures that the divergence of the resulting magnetic field is null to machine precision. However, errors due to floating point precision can still accumulate resulting in monopoles. This can be avoided by instead using constraint transport on the vector potential on the edges of the cell (Lesur et al., 2023b). Additional difficulties are associated with non-ideal MHD effects. A specific difficulty consists of simulating the Hall effect in non-ideal MHD. In this case, multiple

wave modes are possible, most of which do not cause issues with numerical codes. Only one of those waves, called the whistler wave, is dispersive but non-diffusive, propagating with speed.

$$c_{\text{whistler}} = \frac{\omega}{k} = \frac{\eta_H k}{2} + \sqrt{\left(\frac{\eta_H k}{2}\right)^2 + v_A^2}, \tag{1.34}$$

where $v_A = B/\sqrt{\rho}$ is the Alfvén speed (Zier et al., 2024; Marchand et al., 2018). Here, the speed of the soundwave increases with the value of k, thus, it is not bounded, and the smaller the perturbation, the larger the speed, and without dissipation. This behavior is non-physical as it can be shown that it is a result of neglecting electron inertia and assuming charge neutrality. Without those approximations the speed of the whistler wave is actually bound by the electron cyclotron frequency, as detailed in Zier et al. (2024). In order to treat them, it is possible to use an approximate dissipative Riemann solver, as detailed in Lesur et al. (2014). It is also possible to add explicit diffusion, making whistler waves dissipative as well, which results in damping on the whistler waves to stabilize the numerical scheme. In Smoothed Particle Hydrodynamics methods, implementations of non-ideal MHD are used (e.g. Price et al. 2018, and Price 2012 in ideal-MHD), however the stability of the scheme with regard to whistler waves is not discussed (Wurster et al., 2016). To our knowledge no discussion of the precision of whistler wave modes, aside of test at lower frequencies, is present in the SPH literature.

3.4. Planets

Planets are expected to form at some point. Their gravitational interaction with the disc induces a transfer of angular momentum between the disc and the planet, which modifies the orbit of the planet, and modifies the morphology of the disc while affecting its dynamics (e.g. Lin & Papaloizou 1979). In general, planets are modeled as point masses whose regularised gravitational force is applied to the disc. Momentum is exchanged between the disc and the planet, which in return can modify its orbital parameters while keeping total momentum conserved. This results in a potential migration of the planet, implying the difficulty for planet formation to happen *in situ*. Migration of planets can be classified into multiple categories depending on the mechanism that locally dominates the exchange of angular momentum. For this description, we follow the classification given in Paardekooper et al. (2023); McNally et al. (2019) shown in Fig. 1.11.

3.4.1. Type I migration

For small planets, the planet-disc interaction can result in so called viscous type-I migration. It was the first studied regime of migration in Goldreich & Tremaine (1979, 1980), where it is assumed that the planet creates linear perturbations in the disc density that is asymmetric, which in turn creates a torque on the planet that

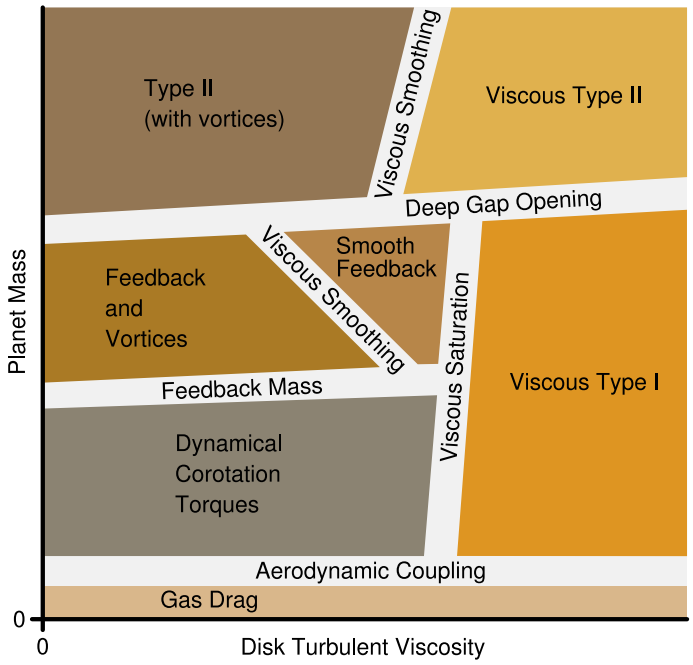


Figure 1.11: Map of the different regimes of planet migration present in discs. Source: (McNally et al., 2019, Fig.1). We omit the drag regimes, as they only apply to small objects that can be considered to be large dust grains.

can be estimated. Additionally, in the frame of the planet, corotating horshoe orbits exists (see Fig. 1.12) and are asymmetric due to the disc density and temperature profile being not uniform. Those pertubations also creat a torque on the planet Ward (1991). This regime corresponds to a planet creating only linear pertubations in the disc density, the migration is classified as *type-1*.

Early results on this type of migration include work done by Korycansky & Pollack (1993) who used a model reduced to a static single-dimensional model per azymuthal Fourrier component to carry a numerical estimate of the resulting torque or Tanaka et al. (2002) which provided three dimensional estimates by using hermites polynomial vertically instead to account for the vertical structure. When the viscosity is lowered the reservoir of angular momentum close to the planet is not replenished sufficiently fast by the viscosity leading to saturation of the torque applied to the planet. In general, aside from large viscosities, the corotation torque is given by the non-linear horseshoe drag, as shown by Paardekooper & Papaloizou (2009); Casoli & Masset (2009) using analytical models.

3.4.2. Type II migration

When the planet-to-star mass ratio increases, the torque applied to the disc becomes non-linear and forms a planetary gap (i.e. an annulus of much lower gas density) centered on the orbit of the planet. This regime corresponds to type II migration

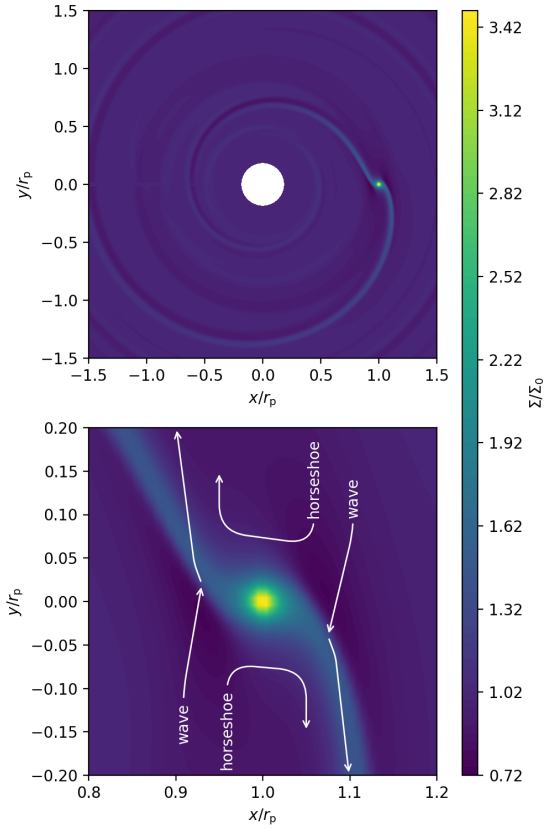


Figure 1.12: Figure from Paardekooper et al. (2023), showing the differential flow of gas close to the planet.

(Paardekooper et al., 2023). In type II migration the low density gap on the planet's orbit implies that the mechanisms of type I migration are ineffective as they only involve material close to the planet (Ward, 1997; Lin & Papaloizou, 1986). Instead, material crosses the gap on horseshoe orbits, as shown by hydrodynamical simulations such as the study from Duffell et al. (2014); Dürmann & Kley (2015, 2017). This can result in efficient angular momentum transfer between the planet and the disc resulting in migration even though the mechanism is significantly different from type I migration (Kanagawa et al., 2018)

3.4.3. Planet in dusty discs

Observations of discs provided by ALMA show substructures in the millimeter-sized dust grain distribution. Dust evolution in a disc hosting a planet essentially results from a competition between the tidal torque exerted by the planet and the drag torque, the latter being indirectly affected by the presence of the planet as it modifies the local gas density (Dipierro et al., 2018). Dipierro et al. (2016), whose results are partially shown in Fig. 1.13, demonstrated that, small planets may carve a gap in the dust but not in the gas as long as $\mathrm{St} \gtrsim 1$. A larger planet may open a gap in the gas.

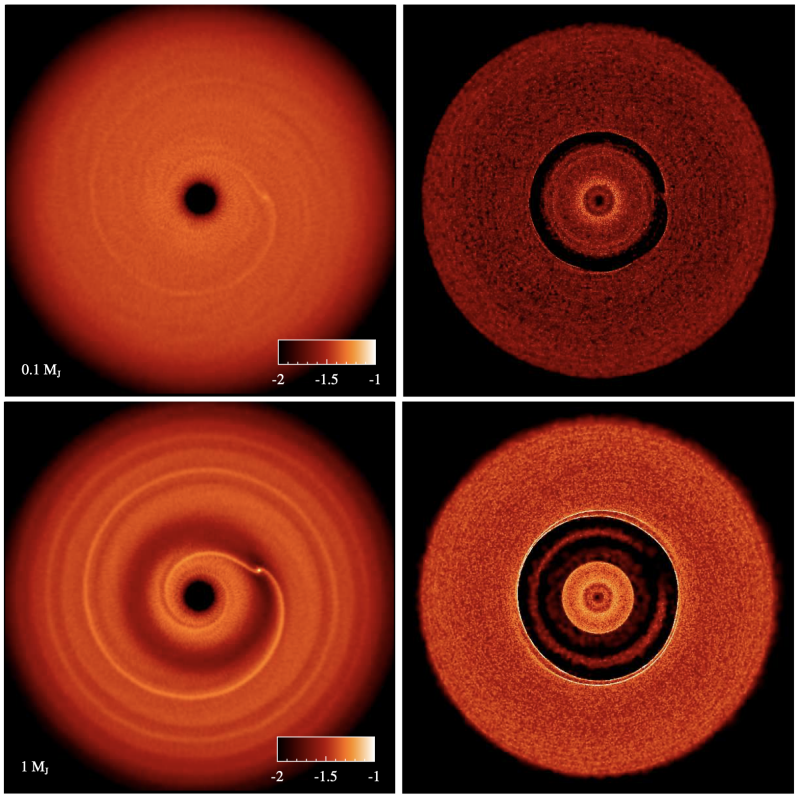


Figure 1.13: Figure adapted from Dipierro et al. (2016) showing the influence of a planet on dust and gas for a small (top panels) and a large (bottom panels), respectively of $0.1M_J$ and $1M_J$. The right panels shows the gas column density for a planet of $0.1M_J$ and $1M_J$ respectively on top and bottom. The left panels shows the column dust column density for the same simulations.

The related structures are sharper in the dust due to the combined effect of drift at pressure maxima, resonances, and stability at corotation. In particular, dust species can be separated into different regions by the presence of a small planet, which can create two distinct populations of dust grains, in agreement with the scenario proposed by Kruijer et al. 2017.

3.4.4. Circumplanetary disc

The last interesting object in planet-disc interaction is the circumplanetary disc. Planets, when put in a protoplanetary disc, gather material within their Hill radius (i.e. the radius into which the gravity of the planet dominates over the gravity of the star) into a smaller disc orbiting the planet instead (example shown in Fig. 1.14). It is from this disc that the planete accrete material (Machida et al., 2008). Additionally, drift and accretion of solids, especially of pebble size, are expected to be of critical importance to form solid cores in an efficient manner (e.g. Johansen & Lambrechts 2017). Therefore, properly resolving the circumplanetary discs is neces-

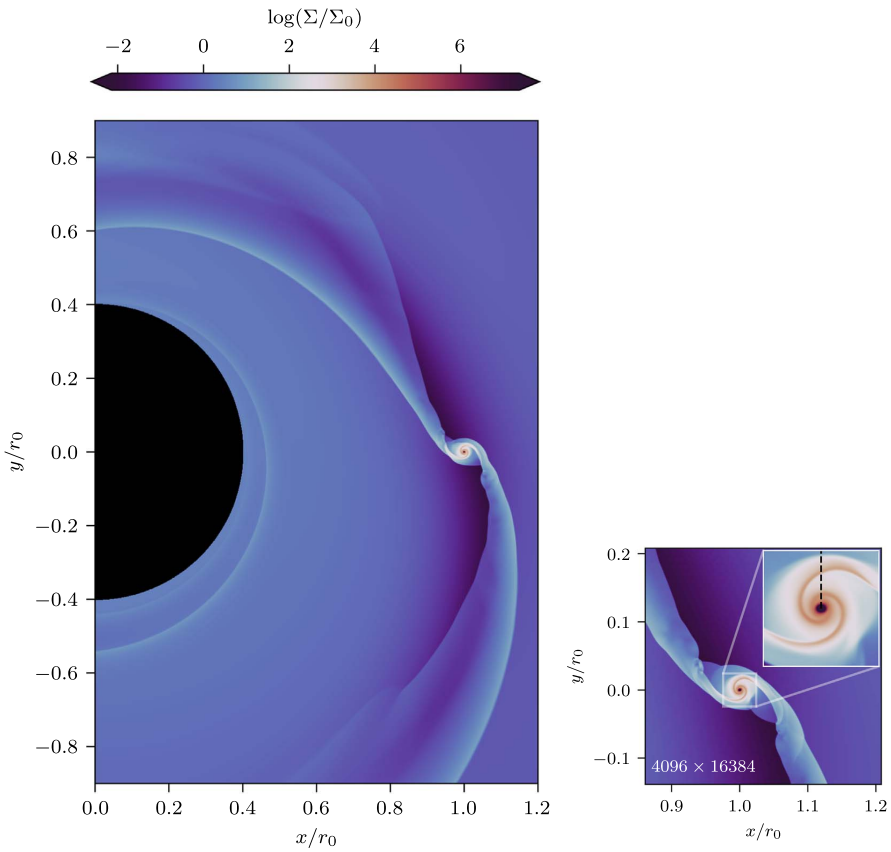


Figure 1.14: Figure adapted from Benítez-Llambay et al. (2023) to showcase the shape of circumplanetary discs.

sary to measure the realistic accretion rate on the planet. Similarly to the hosting protoplanetary disc, the disc is supported by local rotation, and the gas has to remove angular momentum in order to be accreted on the planet. However, due to the scale, density, and proximity of the planet, the conditions are different compared to protoplanetary discs. Firstly, radiative transfer must be accounted for as cooling is not efficient enough at this time scale, requiring the need to account for the opacity of the gas-dust mixture instead of using locally isothermal models (e.g. Szulágyi 2017; Schulik et al. 2020). Secondly, as the circumplanetary disc is embedded in the protoplanetary discs, it is subject to infalls of material from the protoplanetary disc onto the circumplanetary (e.g. Machida et al. 2008; Tanigawa et al. 2012). Lastly, the proximity of the actively accreting planet results in a potential coupling of the planet's atmosphere with the inner regions of the circumplanetary discs, with the planet itself being the inner boundary of the disc. So far, the dynamics and evolution of the circumplanetary disc remain poorly constrained. In particular, the transport of angular momentum that results in the accretion of dust and gas is key to understanding the formation and evolution of the planet through core accretion. However, the proximity to the planet and the corresponding complex boundary con-

ditions associated with the planet's atmosphere, as well as resolving the physics of such discs, including inflows, and the presence of eventual satelites, are challenging.

3.4.5. Numerical challenges of planet-disc interactions

Simulating planet-disc interactions poses a few numerical challenges.

Planet modeling Two options are possible to simulate the presence of a planet in a simulation. A first approach consists in simulating a disc hosting a planet, involves using a rotating potential to model the star and the planet (e.g. de Val-Borro et al. 2006). However, this does not allow the gas to exert a torque on the planet. Therefore, angular momentum is not globally conserved, making it not a suitable method to simulate planet migration mechanisms. A possible alternative is to use a so-called sink particle, which is a point-mass gravitating body modeled by a particle that is evolved according to the gravitational force of the star and the disc, and then applies its gravitational force to the disc as well. Additionally, it is also possible to model the star using another sink particle, allowing conservation of momentum in all gravitational interactions (e.g. Bate & Bonnell 1997). In addition to the gravitational interaction, sink particles can, as their names suggest, account for the accretion of nearby material. Various criteria can be used to parametrize the accretion of a sink. In SPH, for example, a typical criterion is to check that the gas particle to be accreted is bound within a fraction of the hill radius of the sink particle (Price et al., 2018). This condition would typically correspond to particles being contained in the circumplanetary disc. Sink particles can also carry other quantities, such as angular momentum, magnetic flux, etc. However, when modeling a planet in the context of a circumplanetary disc, no work to date has accounted for radiative emissions from the sink onto the gas and dust. The accuracy and the dependence of a simulation to the use of sinks is still widely debated, both in the fields of planet and star formation (e.g. Hennebelle et al. 2020).

Large timescales, small lenghtscales Another challenge of planet-disc interaction is the timescale to be simulated. As shown in Paardekooper et al. (2023), a simulation with moderate resolution requires at least hundreds of orbits to resolve the timescale of planet migration, each of which takes about thousands of seconds with current computational capacities, making up for long numerical simulations. To alleviate this problem, multiple options are available. They include adaptive or hierarchical timestepping, relaxing the CFL constraints, or porting on modern, faster computing architectures. Additionally, resolving the small scales around a planet can be limiting for numerical codes that do not allow refinement when treating circumplanetary disc simulations, for example.

Miscellaneous As explained in Paardekooper et al. (2023), in order to correctly resolve planet-disc interactions, conservation of vortensity and entropy is important.

This would make methods such as SPH suitable, as they can naturally conserve those quantities on any closed contour defined by a set of particles. However, most SPH simulations result in high numerical viscosity, requiring high resolutions to achieve realistic α_S. Additionally, the planet can creat low density regions and hydrodynamical shocks close to its orbits, requiring proper treatment of shocks to ensure the quality of the solution, which tends to favor schemes based on Riemann solvers (e.g. de Val-Borro et al. 2006). In practice, multiple methods can be used, as each comes with several advantages, highlighting the need to perform simulations with multiple different methods to avoid method bias. Lastly, observations hint toward low viscosities in discs, as mentioned in Sec. 3.1. This implies a higher dependency on initial conditions, hence the need for special care in its treatment. And the presence of winds in realistic discs needs to be resolved as well (see Wafflard-Fernandez & Lesur 2023), requiring MHD simulations.

Circumplanetary discs Resolving the accretion rate of a planet requires resolving the circumplanetary disc as well. However, due to its tiny radius compared to the hosting circumstellar disc, this is challenging for multiple reasons. Firstly, material is continuously falling from the circumstellar disc onto the circumplanetary disc. This requires resolving the complete disc in order to have realistic infalls. Secondly, because of its scale, it is necessary to perform local simulations or global simulations with extreme resolution. The first studies were performed, for example, using a locally refined grid close to the planet (D'Angelo et al., 2003; Machida et al., 2008), or through local approximations (Ormel et al., 2015), or non-uniform geometry (Schulik et al., 2020). Lastly the proximity of the planet makes the radiative transfer effects critical, requiring studies to resolve them properly to measure a realistic accretion rate (see Szulágyi 2017).

4. State of the art (instabilities)

Most of the effects discussed above lead, in combination with the orbital motion of the disc, to the development of instabilities at small scales that are key processes for planet formation. In this section, we review some of the most relevant ones and discuss associated numerical challenges. Effects on protoplanetary discs can be classified by their relative position to the disc midplane ($z = 0$) and their distance to the host star. We provide the expected location of key instabilities in protoplanetary discs in Fig. 1.8.

4.1. Hydrodynamical instabilities

As mentioned in Sec. 2.2, in a realistic disc, the rotational speed profile depends on the distance to the central star but also varies vertically. This vertical gradient of rotational velocity exists in a protoplanetary disc if the disc is not globally isothermal, results in the amplification of vertical motion disturbances called Vertical Shear

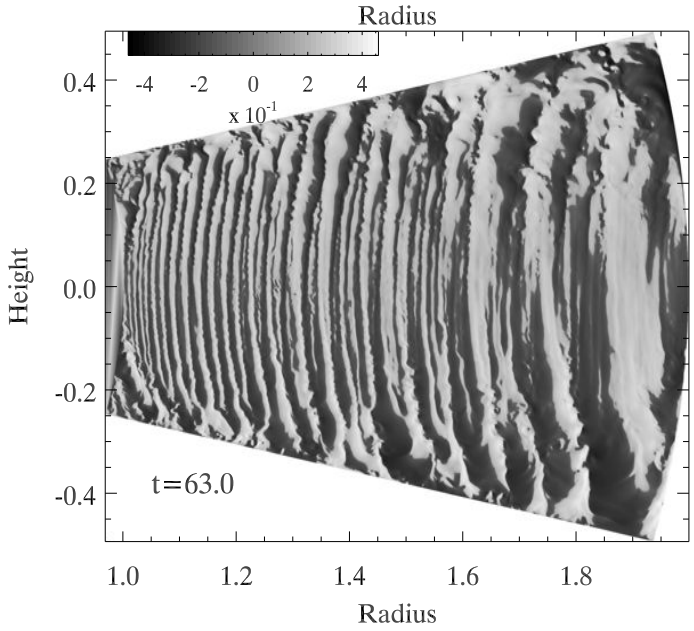


Figure 1.15: An example of a simulation of the Vertical Shear Instability, adapted from Nelson et al. (2013). In this example, we observe the fully grown, non-linear state of the instability in corrugation modes.

Instability (VSI) (Arlt & Urpin, 2004; Nelson et al., 2013). This instability may occur within the disc layer ($z/r \lesssim 4$) and for $10\,\mathrm{au} \lesssim r \lesssim 100\,\mathrm{au}$ since it is suppressed in the inner disc due to inefficient cooling, which stabilizes the instability. When grown to a fully non-linear state, vertical shear instability should lead to the presence of nearly axisymmetric corrugation modes (an example is shown in Fig. 1.15) that should be observable using the ALMA telescope (Barraza-Alfaro et al., 2021). As we will see later, dust-gas instabilities are heavily dependent on the dust concentration which can be strongly affected by the presence of VSI in the disc which can lift grains into the disc atmosphere ($z/r > 4$) and concentrate them radially into rings like structures (Flock et al., 2020). Thus, VSI can significantly affect the condition of planet formation justifying interest toward such studies. On the other end, VSI can be suppressed by dust-settling (Lin, 2019), the presence of dust growth (Fukuhara et al., 2021), and strong magnetic fields (Latter & Papaloizou, 2018; Cui & Lin, 2021). This shows the need for global simulation with coupling of multiple physical effects to properly model VSI in protoplanetary discs. Additionally, VSI can trigger secondary Kelvin-Helmotz like instabilities between the corrugated flows leading typically to Rossby-Wave-Instability for example (Richard et al., 2016), which require sufficient resolution as well as long timescale integration of the simulation (hundreds of orbits) to be observed.

Aside from VSI hydrodynamical instabilities in discs, other instabilities are studied, but their impact on planet formation is yet to be fully assessed. They include the Convective Over-Stability (COS) ($z/r \lesssim 4, 0.5\,\mathrm{au} \lesssim r \lesssim 100\,\mathrm{au}$) (Klahr & Hub-

bard, 2014; Lyra, 2014) and Zombie-Vortex Instability (ZVI) ($z/r \lesssim 4$, $0.5\,\mathrm{au} \lesssim r \lesssim 100\,\mathrm{au}$) (Barranco & Marcus, 2005) which can participate toward creating turbulence in the disc thus have influence on the angular momentum transport. Additionally, the COS can create vortices, which can be an efficient dust trapping mechanism (Raettig et al., 2015; Lyra et al., 2018). However, aside from dust trapping vortices, little was studied about the interaction of dust and COS in the context of planet formation, as pointed out by Lesur et al. (2023a). As for ZVI little is known about its consequences on global protoplanetary discs and interactions with other processes as it requires high resolution simulation to be resolved properly (Lesur & Latter, 2016).

4.2. Dust-Gas instabilities

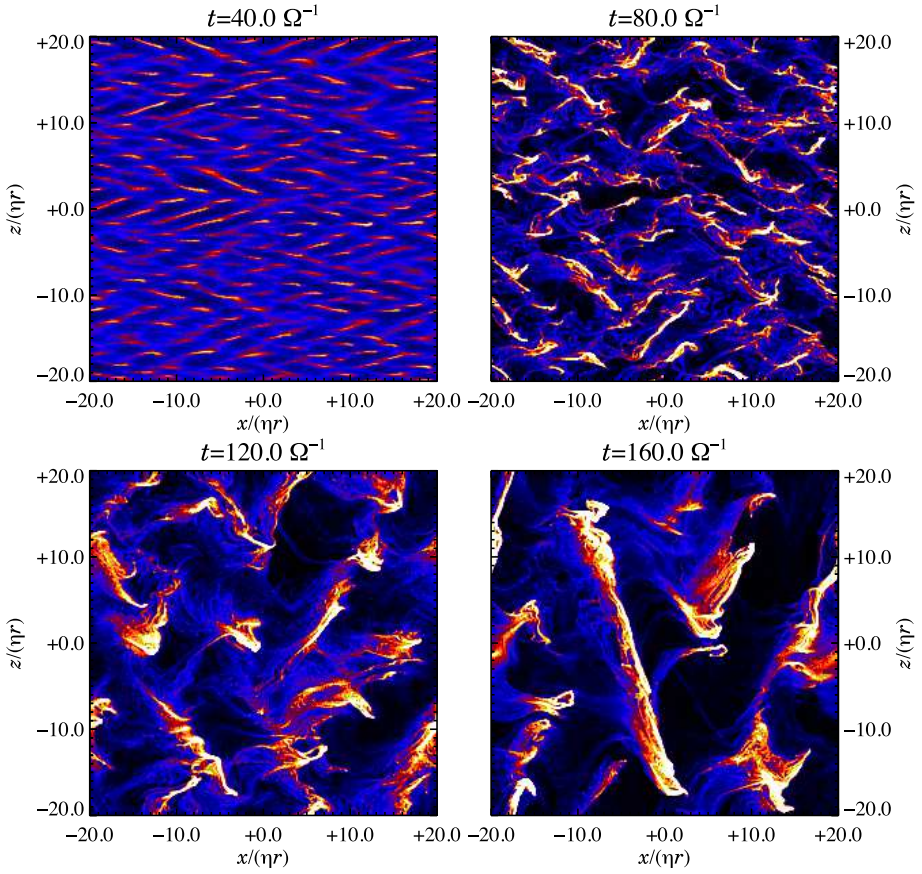


Figure 1.16: An example simulation of the Streaming Instability adapted from Johansen & Youdin (2007). In this example, we observe the development of the SI in a local unstratified shearing box at multiple timesteps.

Planet formation as per Safronov (1969) requires objects above $0.1 - 100$km so that they are gravitationally bound and become planetesimal. However, dust growth is limited by the meter drift barrier as mentioned in Sec. 2.3 and the fragmentation

barrier as in Sec. 3.2. In order to form planets, dust must be concentrated without growth or gravity. One of the compelling mechanisms to do so is the so-called Streaming Instability (SI) (Youdin & Goodman, 2005), which tends to concentrate dust effectively into clumps (example of simulation of the SI Fig. 1.16), which could in turn lead to pebbles leading to the formation of planetesimals. SI has recently been interpreted as a particular type of resonant drag instability (Squire & Hopkins, 2018b,a), where the concentration of dust in local density maxima by radial epicyclic motion is in phase with gas recirculation due to epicycles in the vertical direction(e.g. Magnan et al. 2024). However, such instability is expected to develop at scales that are fractions of order $\sim (H/r)^2$ of the disc scale height (near hundredths of a disc scale height). This makes the SI challenging to capture in a global simulation. Growth of SI during its linear phase was first studied Youdin & Goodman (2005). However, exploring its non-linear phase required the need for numerical methods such as dust as particles methods. The first pioneering simulations were carried out by Youdin & Johansen (2007), where the method was validated on the linear phase of the SI (Johansen & Youdin, 2007). The method was rendered possible by the use of the local shearing box method, which is a local corotating Cartesian box where the local shear is linearized, hence its name. Shearing boxes are key to carrying out local, small-scale simulations in shearing flows such as a disc. Most SI simulations were done using the code Pencil (Youdin & Johansen, 2007; Johansen & Youdin, 2007) or Athena (Li & Youdin, 2021; Carrera et al., 2021) using the shearing box approximation with dust as Lagrangian particles. Alternatively, SI has also been studied using multifluid methods in FARGO3D (Krapp et al., 2019) using the FARGO algorithm to integrate precisely the contribution of the shear. To date, no simulation of streaming instability with SPH has been published. The key challenge consists of correctly capturing the linear growth of the instability without being dominated by the truncature errors associated with the method. Recent work on streaming instability includes its extension to pressure maxima, where the existence of the linear phase of the instability has been shown by Auffinger & Laibe (2018); Xu & Bai (2022). The instability has also been extended in the presence of a magnetic field in Lin & Hsu (2022), where hybrid unstable modes can be observed. Recent work is also aimed at extending it to multiple species of dust (multiple sizes of grains) (Krapp et al., 2019; Benítez-Llambay et al., 2019), where the instability tends to be less efficient than its single-specie counterpart. The SI has also been shown to be strongly quenched by gas viscosity and dust diffusion (Umurhan et al., 2020; Chen & Lin, 2020). Additionally, we know that pebbles formed from streaming instability can lead to planet formation (Johansen et al., 2015; Simon et al., 2016; Lambrechts et al., 2019). However, extensions of the Streaming Instability toward poly-disperse dust distribution (continuum in dust sizes) suggest that SI might be ineffective for small dust concentrations ($\epsilon = \rho_{\rm d}/\rho_{\rm g} < 1$) (Paardekooper et al., 2020, 2021; McNally et al., 2021). To solve this tension, simulations of a combined model of dust growth with dusty hydrodynamics (for example, using the Lombart & Laibe (2021) scheme in a hydrodynamics code) are required to verify if

the combination of both effects can restore the SI or if the coupling between MHD and dusty hydrodynamics leads to combined instabilities, as in Lin & Hsu (2022), but in the poly-disperse case. The SI is also heavily dependent on the dust to gas ratio ($\epsilon = \rho_d/\rho_g$), so the effects of instabilities such as VSI on the global scale can heavily influence the efficiency and outcomes of SI requiring the need for global SI simulations to study the outcomes of VSI and SI combined. However, such simulations require integration on long timescales as well as very high resolutions and thus require a lot of computing power.

4.3. Magnetohydrodynamical instabilities

As detailed in Sec. 3.1 angular momentum transfer is related to the presence of turbulence, which results in pseudo-viscosity in the Shakura & Sunyaev (1973) model. One of the main sources of turbulence in protoplanetary discs is considered to be the Magneto-Rotational Instability (MRI) (Balbus & Hawley, 1991; Hawley & Balbus, 1991), which in its non-linear regime results in MRI-induced turbulence if the magnetic field is weak enough. It is present in the inner and outer parts of the disc (Lesur, 2021b) because of weak mag field $\beta \geq 1$ and high enough ionization. Early simulations of MRI were made possible by the use of the local shearing box, as for the Streaming Instability, and constrained transport for the magnetohydrodynamics in the code Zeus, whose hydrodynamics scheme will be presented in more detail in Chapter 2-Sec. 2. In numerical simulation, MRI-induced turbulence results in induced $\alpha_S = 0.01 - 1$ depending on the strength of the instability in the quasi-ideal MHD regime. However, as mentioned in Sec. 3.1, observations suggest a lower level of turbulence, implying that it must be mitigated by other phenomena. Additionally, when studied in the non-ideal regime, non-ideal effects may become significant, and reduce the efficiency of the turbulence generated. (Bai, 2015; Li et al., 2018). In protoplanetary discs, this corresponds to larger radii. However, non-ideal effects are more sensible to density profiles and disc parameters, which need to be studied further.

When studied coupled with radiative transfer, Flock et al. (2019) found that the strength of the induced turbulence is highly dependent on the temperature and pressure profiles. In an optically thick regime, MRI increases temperature, which in turn increases scale height, which enhances turbulence and results in convection (Hirose et al., 2014; Scepi et al., 2018). Finally, MHD magnetic reconnection events result in heating (Ross & Latter, 2018), which in the context of MRI requires high resolution studied to be measured numerically.

4.4. Rossby Wave Instability

Another important instability of interest in protoplanetary discs is the Rossby Wave Instability (RWI) (e.g. Lovelace et al. (1999); Li et al. (2000)). It can be created by local minima in the vortensity radial profile, which are creating velocity shear, creat-

ing in turn Kelvin-Helmholtz type waves (Bae et al., 2023) resulting in the presence of vortices. It is a special instability as it is often the result of coupling with other effects such as, at the edge of MRI dead zones (coupling Ideal-MHD, hydrodynamics) (Varnière & Tagger, 2006), planet gaps (de Val-Borro et al., 2007), the edges of gas inflow on the disc (Bae et al., 2015), or between flows created by VSI (Richard et al., 2016). In its non-linear phase, RWI-induced vortices merge together into a larger one, which can be stable for thousands of orbits. Such vortices are studied as they offer a potential way to concentrate dust particles inside. However, simulating RWI is complex as it requires high resolution to achieve a sufficiently low numerical viscosity for long integration times of the RWI. Additionally, as RWI often results in non-axisymmetric features, simulations need to resolve the disc in the azimuthal direction to not artificially truncate the first Fourier azymuthal mode. This may result in stable axisymmetric features in a partial simulation being unstable in a global simulation (Cui & Bai, 2022). In general, the outcome of RWI is strongly related to the dimensionality of the system. In two dimensions, large structures are stabilized, whereas vortex stretching in 3D may disrupt them. Large dust concentrations may also destroy vortices. These points are still strongly debated.

4.5. Self-gravity

4.5.1. Self-gravitating discs

When the mass of the disc is not negligible anymore compared to the mass of the star, the gravity of the disc becomes important and should be accounted for. Such discs are referred as self-gravitating discs. However, considering the mass of the disc alone or not, is not sufficient to determine if the gravity of the disc is expected to have a strong effect. In the case of a cloud of gas subjected to its own gravity, it is possible to know whether a perturbation in its medium will collapse onto itself by comparing the balance between the pressure of the gas and the gravitational pull created by the perturbation. This is called the Jeans instability (Jeans, 1902). However, in a rotating medium this is not the correct criterion, as additional support can be provided by rotation that stabilizes large perturbations. Instead, the pendant of the Jeans criterion in an isothermal disc is the Toomre parameter $Q \equiv c_s\Omega/(\pi G\Sigma)$ (Toomre, 1964), where density perturbation will grow unstable due to gravity when $Q \lesssim 1$. In those discs, this results in spirals that are analogous to the ones observed in spiral galaxies since they share a common origin.

According to reviews on the subject of gravitational instability (GI) in protoplanetary discs (e.g. Bae et al. 2023; Kratter & Lodato 2016), the effect of self-gravity in discs can be summarized as follows: Firstly, as reflected by the expression of the Toomre parameter, the disc needs to be massive and cold to be unstable. Secondly, when a disc is gravitationally unstable and is forming spirals that tend to be non-axisymmetric. This results in efficient angular momentum transport, which dissipates some energy as well. This results in an heating of the disc, reducing the efficiency of GI. Therefore, depending on the cooling rate the disc can also remain

gravitationally unstable, leading to the formation of giant planets Gammie (2001); Rice et al. (2003); Cossins et al. (2009) or bound clumps of matter Rafikov (2005); Cossins et al. (2010); Zhu et al. (2012). However, if the cooling rate is sufficiently low GI-induced spirals will only be transient as the system will not cool quickly enough to sustain the instability Cossins et al. (2009); Béthune et al. (2021). In general a strong interplay between gravity, dynamics and thermal effects is shown. This means that realistic cooling is required to model GI accurately, hence the need for coupling realistic radiative transfer with self-gravitating hydrodynamical simulations.

4.5.2. Numerical challenges

Simulating self-gravitating discs poses a few challenges. Firstly, the gravitational potential satisfies a Poisson equation, which must be solved to determine the local gravitational force. However, the Poisson equation is an elliptic equation, which implies that one has to model long-range interactions in a numerical simulation (e.g. every element of the simulation technically interacts gravitationally with every element). In practice, multiple methods are used to resolve self-gravity. On grid-based methods, the first one is the conjugated gradient method, in which the Poisson equation is converted into a system of linear equations to be solved and iterated toward a solution for that linear system, which in this case corresponds to the gravitational potential. The conjugate gradient method is used e.g. in RAMSES (Teyssier, 2002) or Idefix (Lesur et al., 2023b). Since it iterates toward a solution for the Poisson equation, having a good preconditioner can significantly reduce the number of steps necessary for the solution to converge. Since the solver for gravity is called at each timestep, we refer to the iteration of the Poisson solver scheme nested within the hydro timestepping scheme as subcyles. One drawback of iterative methods is that the number of subcyles is not guaranteed. An extension of such a method is the use of multigrid methods for self-gravity solvers. It was recently implemented in the community code Athena++ (Tomida & Stone, 2023). It relies on solving the Poisson equation on a degraded grid to increase convergence speed, reusing that solution as the starting point for the finer grid, and doing so using many levels of nested grids. It can be viewed abusively as a nested solver with a smarter preconditioner informed by the coarser level. This, in principle, leads to faster convergence on complex problems and is close to the implementation of AMR codes being implemented with nested grid structures such as RAMSES and Athena++. Another approach that does not rely on the use of a grid consists is the Fast Multipole Method (FMM), typically used in meshless hydrodynamical and N-body codes such as Gadget-4 (Springel et al., 2021) and Phantom (Price et al., 2018). FMM offers the advantage of providing the upper bound of the error for a given parametrization of the algorithm independently of the object being simulated, and it does not use subcyles. Therefore, the computational cost is known in advance and is nearly constant. It relies on the analytical solution of the Poisson equation as a sum of Green functions when the sources are point masses and performing a

linear expansion to group particles together in cells. This last method is detailed together with its implementation in Shamrock, in Appendix E.

4.6. Dust-Gravitational instability

Ultimately, as mentioned, dust must become gravitationally unstable to ultimately form planets. An important result of early-70s planet formation studies was the so-called Golreich-Ward mechanism (Goldreich & Ward, 1973) where dust can collapse into planetesimals. They have been revisited recently in e.g. Longarini et al. (2023a,b) discussing the possibility of dust collapse induced by gravitational instabilities. Solar system observations tend to support that planetesimals and planetary cores form by gravitational collapse (Morbidelli et al., 2009; Nesvorný et al., 2010, 2019; McKinnon et al., 2020). This suggests that Gravitational Instabilities (GI) are likely to be present during the planet formation process. Even if gas is expected to be mostly stable with respect to gravitation instabilities in protoplanetary discs, it may be important in the early phases of GI as it has a heavy influence on SI, which, if efficient, can facilitate the gravitation instability (Li & Youdin, 2021). Overall, state-of-the-art studies on dust gravitational instabilities consist mainly of two-dimensional simulations. More long-term instabilities, called secular gravitational instabilities, are also discussed in the literature. However, they have not been probed in global simulations yet.

5. Conclusion

To conclude, resolving modern questions of planet formation requires to understand the interplay between different physical processes such as magnetohydrodynamics, dust, and self-gravity that interplay together both at local and global scalse. It is mandatory to be able to resolve complex geometries such as protostellar collapse down to the disc or circumplanetary discs embedded in protoplanetary discs, the geometry, the properties, the boundaries and the environment of the system evolving as the young stellar system forms. In order to resolve statistics, many studies varying the initial conditions, to confront models with observational statistics are required. Moreover, although some numerical methods are more favorable to study specific processes, it is important to demonstrate that the results obtained do not depend on the solver used. Overall, developing a comprehensive framework for planet formation requires an unprecedented numerical computational power and efficiency.

It is in this context that we first aim to build a powerful, versatile and robust hydrodynamic framework that can easily be extended to include MHD and self-gravity, as well as the dust physics. The first step of this ambitious project is the work that is presented in this manuscript.

References

Adachi I., Hayashi C., Nakazawa K., 1976, *The Gas Drag Effect on the Elliptic Motion of a Solid Body in the Primordial Solar Nebula*, Progress of Theoretical Physics, 1756-1771

Andrews S. M., et al., 2018, *The Disk Substructures at High Angular Resolution Project (DSHARP). I. Motivation, Sample, Calibration, and Overview*, ApJ, 869, L41

Arlt R., Urpin V., 2004, *Simulations of vertical shear instability in accretion discs*, A&A, 426, 755-765

Auffinger J., Laibe G., 2018, *Linear growth of streaming instability in pressure bumps*, MNRAS, 473, 796-805

Bae J., Hartmann L., Zhu Z., 2015, *Are Protoplanetary Disks Born with Vortices? Rossby Wave Instability Driven by Protostellar Infall*, ApJ, 805, 15

Bae J., Isella A., Zhu Z., Martin R., Okuzumi S., Suriano S., in *Protostars and Planets VII*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M., p. 423 (arXiv:2210.13314), doi:10.48550/arXiv.2210.13314

Bai X.-N., 2014, *Hall-effect-Controlled Gas Dynamics in Protoplanetary Disks. I. Wind Solutions at the Inner Disk*, ApJ, 791, 137

Bai X.-N., 2015, *Hall Effect Controlled Gas Dynamics in Protoplanetary Disks. II. Full 3D Simulations toward the Outer Disk*, ApJ, 798, 84

Balbus S. A., Hawley J. F., 1991, *A Powerful Local Shear Instability in Weakly Magnetized Disks. I. Linear Analysis*, ApJ, 376, 214

Balbus S. A., Terquem C., 2001, *Linear Analysis of the Hall Effect in Protostellar Disks*, ApJ, 552, 235-247

Barranco J. A., Marcus P. S., 2005, *Three-dimensional Vortices in Stratified Protoplanetary Disks*, ApJ, 623, 1157-1170

Barraza-Alfaro M., Flock M., Marino S., Pérez S., 2021, *Observability of the vertical shear instability in protoplanetary disk CO kinematics*, A&A, 653, A113

Bate M. R., Bonnell I. A., 1997, *Accretion during binary star formation - II. Gaseous accretion and disc formation*, MNRAS, 285, 33-48

Benítez-Llambay P., Krapp L., Pessah M. E., 2019, *Asymptotically Stable Numerical Method for Multispecies Momentum Transfer: Gas and Multifluid Dust Test Suite and Implementation in FARGO3D*, ApJS, 241, 25

Benítez-Llambay P., Krapp L., Ramos X. S., Kratter K. M., 2023, *RAM: Rapid Advection Algorithm on Arbitrary Meshes*, ApJ, 952, 106

Béthune W., Latter H., Kley W., 2021, *Spiral structures in gravito-turbulent gaseous disks*, A&A, 650, A49

Birnstiel T., et al., 2018, *The Disk Substructures at High Angular Resolution Project (DSHARP). V. Interpreting ALMA Maps of Protoplanetary Disks in Terms of a Dust Model*, ApJ, 869, L45

Blandford R. D., Payne D. G., 1982, *Hydromagnetic flows from accretion disks and the production of radio jets.*, MNRAS, 199, 883-903

Blum J., 2018, *Dust evolution in protoplanetary discs and the formation of planetesimals: What have we learned from laboratory experiments?*, Space Science Reviews, 52

Blum J., Wurm G., 2008, *The growth mechanisms of macroscopic bodies in protoplanetary disks.*, ARA&A, 46, 21-56

Boccaletti A., et al., 2020, *Possible evidence of ongoing planet formation in AB Aurigae. A showcase of the SPHERE/ALMA synergy*, A&A, 637, L5

Brauer F., Dullemond C. P., Henning T., 2008, *Coagulation, fragmentation and radial motion of solid particles in protoplanetary disks*, A&A, 480, 859-877

Carrera D., Simon J. B., Li R., Kretke K. A., Klahr H., 2021, *Protoplanetary Disk Rings as Sites for Planetesimal Formation*, AJ, 161, 96

Casoli J., Masset F. S., 2009, *On the Horseshoe Drag of a Low-Mass Planet. I. Migration in Isothermal Disks*, ApJ, 703, 845-856

Chen K., Lin M.-K., 2020, *How Efficient Is the Streaming Instability in Viscous Protoplanetary Disks?*, ApJ, 891, 132

Commerçon B., Lebreuilly U., Price D. J., Lovascio F., Laibe G., Hennebelle P., 2023, *Dynamics of dust grains in turbulent molecular clouds. Conditions for decoupling and limits of different numerical implementations*, A&A, 671, A128

Cossins P., Lodato G., Clarke C. J., 2009, *Characterizing the gravitational instability in cooling accretion discs*, MNRAS, 393, 1157-1173

Cossins P., Lodato G., Clarke C., 2010, *The effects of opacity on gravitational stability in protoplanetary discs*, MNRAS, 401, 2587-2598

Cui C., Bai X.-N., 2022, *Turbulence in outer protoplanetary discs: MRI or VSI?*, MNRAS, 516, 4660-4668

Cui C., Lin M.-K., 2021, *On the vertical shear instability in magnetized protoplanetary discs*, MNRAS, 505, 2983-2998

Currie T., Biller B., Lagrange A., Marois C., Guyon O., Nielsen E. L., Bonnefoy M., De Rosa R. J., in *Protostars and Planets VII*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M., p. 799 (arXiv:2205.05696), doi:10.48550/arXiv.2205.05696

D'Alessio P., Cantö J., Calvet N., Lizano S., 1998, *Accretion Disks around Young Objects. I. The Detailed Vertical Structure*, ApJ, 500, 411-427

D'Angelo G., Kley W., Henning T., 2003, *Orbital Migration and Mass Accretion of Protoplanets in Three-dimensional Global Computations with Nested Grids*, ApJ, 586, 540-561

Dipierro G., Laibe G., Price D. J., Lodato G., 2016, *Two mechanisms for dust gap opening in protoplanetary discs*, MNRAS, 459, L1-L5

Dipierro G., Laibe G., Alexander R., Hutchison M., 2018, *Gas and multispecies dust dynamics in viscous protoplanetary discs: the importance of the dust back-reaction*, MNRAS, 479, 4187-4206

Duffell P. C., Haiman Z., MacFadyen A. I., D'Orazio D. J., Farris B. D., 2014, *The Migration of Gap-opening Planets is Not Locked to Viscous Disk Evolution*, ApJ, 792,

L10

Dullemond C. P., et al., 2018, *The Disk Substructures at High Angular Resolution Project (DSHARP). VI. Dust Trapping in Thin-ringed Protoplanetary Disks*, ApJ, 869, L46

Dürmann C., Kley W., 2015, *Migration of massive planets in accreting disks*, A&A, 574, A52

Dürmann C., Kley W., 2017, *The accretion of migrating giant planets*, A&A, 598, A80

Epstein P. S., 1924, *On the Resistance Experienced by Spheres in their Motion through Gases*, Physical Review, 23, 710-733

Evans C. R., Hawley J. F., 1988, *Simulation of Magnetohydrodynamic Flows: A Constrained Transport Model*, ApJ, 332, 659

Flaherty K. M., Hughes A. M., Rosenfeld K. A., Andrews S. M., Chiang E., Simon J. B., Kerzner S., Wilner D. J., 2015, *Weak Turbulence in the HD 163296 Protoplanetary Disk Revealed by ALMA CO Observations*, ApJ, 813, 99

Flaherty K. M., et al., 2017, *A Three-dimensional View of Turbulence: Constraints on Turbulent Motions in the HD 163296 Protoplanetary Disk Using DCO^+*, ApJ, 843, 150

Flaherty K. M., Hughes A. M., Teague R., Simon J. B., Andrews S. M., Wilner D. J., 2018, *Turbulence in the TW Hya Disk*, ApJ, 856, 117

Flock M., Turner N. J., Mulders G. D., Hasegawa Y., Nelson R. P., Bitsch B., 2019, *Planet formation and migration near the silicate sublimation front in protoplanetary disks*, A&A, 630, A147

Flock M., Turner N. J., Nelson R. P., Lyra W., Manger N., Klahr H., 2020, *Gas and Dust Dynamics in Starlight-heated Protoplanetary Disks*, ApJ, 897, 155

Fukuhara Y., Okuzumi S., Ono T., 2021, *Effects of Dust Evolution on the Vertical Shear Instability in the Outer Regions of Protoplanetary Disks*, ApJ, 914, 132

Gammie C. F., 2001, *Nonlinear Outcome of Gravitational Instability in Cooling, Gaseous Disks*, ApJ, 553, 174-183

Glassgold A. E., Lizano S., Galli D., 2017, *Deep-down ionization of protoplanetary discs*, MNRAS, 472, 2447-2453

Goldreich P., Tremaine S., 1979, *The excitation of density waves at the Lindblad and corotation resonances by an external potential.*, ApJ, 233, 857-871

Goldreich P., Tremaine S., 1980, *Disk-satellite interactions.*, ApJ, 241, 425-441

Goldreich P., Ward W. R., 1973, *The Formation of Planetesimals*, ApJ, 183, 1051-1062

Gonzalez J. F., Laibe G., Maddison S. T., 2017, *Self-induced dust traps: overcoming planet formation barriers*, MNRAS, 467, 1984-1996

Guzmán V. V., et al., 2018, *The Disk Substructures at High Angular Resolution Program (DSHARP). VIII. The Rich Ringed Substructures in the AS 209 Disk*, ApJ, 869, L48

Hartmann L., Calvet N., Gullbring E., D'Alessio P., 1998, *Accretion and the Evolution of T Tauri Disks*, ApJ, 495, 385-400

Hartmann L., Herczeg G., Calvet N., 2016, *Accretion onto Pre-Main-Sequence Stars*, ARA&A, 54, 135-180

Hawley J. F., Balbus S. A., 1991, *A Powerful Local Shear Instability in Weakly Magnetized Disks. II. Nonlinear Evolution*, ApJ, 376, 223

Hennebelle P., Commerçon B., Lee Y.-N., Charnoz S., 2020, *What determines the formation and characteristics of protoplanetary discs?*, A&A, 635, A67

Hirose S., Blaes O., Krolik J. H., Coleman M. S. B., Sano T., 2014, *Convection Causes Enhanced Magnetic Turbulence in Accretion Disks in Outburst*, ApJ, 787, 1

Huang J., et al., 2018a, *The Disk Substructures at High Angular Resolution Project (DSHARP). II. Characteristics of Annular Substructures*, ApJ, 869, L42

Huang J., et al., 2018b, *The Disk Substructures at High Angular Resolution Project (DSHARP). III. Spiral Structures in the Millimeter Continuum of the Elias 27, IM Lup, and WaOph 6 Disks*, ApJ, 869, L43

Isella A., et al., 2018, *The Disk Substructures at High Angular Resolution Project (DSHARP). IX. A High-definition Study of the HD 163296 Planet-forming Disk*, ApJ, 869, L49

Jeans J. H., 1902, *The Stability of a Spherical Nebula*, Philosophical Transactions of the Royal Society of London Series A, 199, 1-53

Johansen A., Lambrechts M., 2017, *Forming Planets via Pebble Accretion*, Annual Review of Earth and Planetary Sciences, 45, 359-387

Johansen A., Youdin A., 2007, *Protoplanetary Disk Turbulence Driven by the Streaming Instability: Nonlinear Saturation and Particle Concentration*, ApJ, 662, 627-641

Johansen A., Mac Low M.-M., Lacerda P., Bizzarro M., in *IAU General Assembly*, booktitle, p. 2229132

Kanagawa K. D., Tanaka H., Szuszkiewicz E., 2018, *Radial Migration of Gap-opening Planets in Protoplanetary Disks. I. The Case of a Single Planet*, ApJ, 861, 140

Klahr H., Hubbard A., 2014, *Convective Overstability in Radially Stratified Accretion Disks under Thermal Relaxation*, ApJ, 788, 21

Korycansky D. G., Pollack J. B., 1993, *Numerical Calculations of the Linear Response of a Gaseous Disk to a Protoplanet*, Icarus, 102, 150-165

Krapp L., Benítez-Llambay P., Gressel O., Pessah M. E., 2019, *Streaming Instability for Particle-size Distributions*, ApJ, 878, L30

Kratter K., Lodato G., 2016, *Gravitational Instabilities in Circumstellar Disks*, ARA&A, 54, 271-311

Kruijer T. S., Burkhardt C., Budde G., Kleine T., 2017, *Age of Jupiter inferred from the distinct genetics and formation times of meteorites*, Proceedings of the National Academy of Science, 114, 6712-6716

Kurtovic N. T., et al., 2018, *The Disk Substructures at High Angular Resolution Project (DSHARP). IV. Characterizing Substructures and Interactions in Disks around Multiple Star Systems*, ApJ, 869, L44

Laibe G., Price D. J., 2014, *Dusty gas with one fluid*, MNRAS, 440, 2136-2146

Laibe G., Gonzalez J. F., Fouchet L., Maddison S. T., 2008, *SPH simulations of grain growth in protoplanetary disks*, A&A, 487, 265-270

Lambrechts M., Morbidelli A., Jacobson S. A., Johansen A., Bitsch B., Izidoro A., Raymond S. N., 2019, *Formation of planetary systems by pebble accretion and migration. How the radial pebble flux determines a terrestrial-planet or super-Earth growth mode*, A&A, 627, A83

Latter H. N., Papaloizou J., 2018, *Vortices and the saturation of the vertical shear instability in protoplanetary discs*, MNRAS, 474, 3110-3124

Launder B., Spalding D., 1974, *The numerical computation of turbulent flows*, Computer Methods in Applied Mechanics and Engineering, 269-289

Lebreuilly U., Commerçon B., Laibe G., 2020, *Protostellar collapse: the conditions to form dust-rich protoplanetary disks*, A&A, 641, A112

Lesur G., 2021a, *Magnetohydrodynamics of protoplanetary discs*, Journal of Plasma Physics, 87, 205870101

Lesur G. R. J., 2021b, *Systematic description of wind-driven protoplanetary discs*, A&A, 650, A35

Lesur G. R. J., Latter H., 2016, *On the survival of zombie vortices in protoplanetary discs*, MNRAS, 462, 4549-4554

Lesur G., Kunz M. W., Fromang S., 2014, *Thanatology in protoplanetary discs. The combined influence of Ohmic, Hall, and ambipolar diffusion on dead zones*, A&A, 566, A56

Lesur G., et al.,in *Protostars and Planets VII*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M., p. 465

Lesur G. R. J., Baghdadi S., Wafflard-Fernandez G., Mauxion J., Robert C. M. T., Van den Bossche M., 2023b, *IDEFIX: A versatile performance-portable Godunov code for astrophysical flows*, A&A, 677, A9

Li R., Youdin A. N., 2021, *Thresholds for Particle Clumping by the Streaming Instability*, ApJ, 919, 107

Li H., Finn J. M., Lovelace R. V. E., Colgate S. A., 2000, *Rossby Wave Instability of Thin Accretion Disks. II. Detailed Linear Theory*, ApJ, 533, 1023-1034

Li R., Youdin A. N., Simon J. B., 2018, *On the Numerical Robustness of the Streaming Instability: Particle Concentration and Gas Dynamics in Protoplanetary Disks*, ApJ, 862, 14

Lin M.-K., 2019, *Dust settling against hydrodynamic turbulence in protoplanetary discs*, MNRAS, 485, 5221-5234

Lin M.-K., Hsu C.-Y., 2022, *Streaming Instabilities in Accreting and Magnetized Laminar Protoplanetary Disks*, ApJ, 926, 14

Lin D. N. C., Papaloizou J., 1979, *Tidal torques on accretion discs in binary systems with extreme mass ratios.*, MNRAS, 186, 799-812

Lin D. N. C., Papaloizou J., 1986, *On the Tidal Interaction between Protoplanets and the Protoplanetary Disk. III. Orbital Migration of Protoplanets*, ApJ, 309, 846

Lissauer J. J., Batalha N. M., Borucki W. J., in *Protostars and Planets VII*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M.,

p. 839 (arXiv:2311.04981), doi:10.48550/arXiv.2311.04981

Lombart M., Laibe G., 2021, *Grain growth for astrophysics with discontinuous Galerkin schemes*, MNRAS, 501, 4298-4316

Longarini C., Lodato G., Bertin G., Armitage P. J., 2023a, *The role of the drag force in the gravitational stability of dusty planet forming disc - I. Analytical theory*, MNRAS, 519, 2017-2029

Longarini C., Armitage P. J., Lodato G., Price D. J., Ceppi S., 2023b, *The role of the drag force in the gravitational stability of dusty planet-forming disc - II. Numerical simulations*, MNRAS, 522, 6217-6235

Love S. G., Joswiak D. J., Brownlee D. E., 1994, *Densities of Stratospheric Micrometeorites*, Icarus, 111, 227-236

Lovelace R. V. E., Li H., Colgate S. A., Nelson A. F., 1999, *Rossby Wave Instability of Keplerian Accretion Disks*, ApJ, 513, 805-810

Lynch E. M., Lovell J. B., Sefilian A. A., 2024, *Why dust pressure matters in debris discs*, MNRAS, 529, L147-L151

Lyra W., 2014, *Convective Overstability in Accretion Disks: Three-dimensional Linear Analysis and Nonlinear Saturation*, ApJ, 789, 77

Lyra W., Raettig N., Klahr H., 2018, *Pebble-trapping Backreaction Does Not Destroy Vortices*, Research Notes of the American Astronomical Society, 2, 195

Machida M. N., Kokubo E., Inutsuka S.-i., Matsumoto T., 2008, *Angular Momentum Accretion onto a Gas Giant Planet*, ApJ, 685, 1220-1236

Magnan N., Heinemann T., Latter H. N., 2024, *A physical picture for the acoustic resonant drag instability*, MNRAS, 529, 688-701

Manara C. F., et al., 2016, *Evidence for a correlation between mass accretion rates onto young stars and the mass of their protoplanetary disks*, A&A, 591, L3

Marchand P., Commerçon B., Chabrier G., 2018, *Impact of the Hall effect in star formation and the issue of angular momentum conservation*, A&A, 619, A37

Masset F., 2000, *FARGO: A fast eulerian transport algorithm for differentially rotating disks*, A&AS, 141, 165-173

Mayor M., Queloz D., 1995, *A Jupiter-mass companion to a solar-type star*, Nature, 378, 355-359

McKinnon W. B., et al., 2020, *The solar nebula origin of (486958) Arrokoth, a primordial contact binary in the Kuiper Belt*, Science, 367, aay6620

McNally C. P., Nelson R. P., Paardekooper S.-J., Benítez-Llambay P., 2019, *Migrating super-Earths in low-viscosity discs: unveiling the roles of feedback, vortices, and laminar accretion flows*, MNRAS, 484, 728-748

McNally C. P., Lovascio F., Paardekooper S.-J., 2021, *Polydisperse streaming instability - III. Dust evolution encourages fast instability*, MNRAS, 502, 1469-1486

Meheut H., Fromang S., Lesur G., Joos M., Longaretti P.-Y., 2015, *Angular momentum transport and large eddy simulations in magnetorotational turbulence: the small Pm limit*, A&A, 579, A117

Mendoza V. E. E., 1966, *Infrared Photometry of T Tauri Stars and Related Objects*, ApJ, 143, 1010

Mendoza V. E. E., 1968, *Infrared Excesses in T Tauri Stars and Related Objects*, ApJ, 151, 977

Miotello A., Kamp I., Birnstiel T., Cleeves L. C., Kataoka A., in *Protostars and Planets VII*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M., p. 501 (arXiv:2203.09818), doi:10.48550/arXiv.2203.09818

Morbidelli A., Bottke W. F., Nesvorný D., Levison H. F., 2009, *Asteroids were born big*, Icarus, 204, 558-573

Mouschovias T. C., Kunz M. W., Christie D. A., 2009, *Formation of interstellar clouds: Parker instability with phase transitions*, MNRAS, 397, 14-23

Müller A., et al., 2018, *Orbital and atmospheric characterization of the planet within the gap of the PDS 70 transition disk*, A&A, 617, L2

Nakagawa Y., Sekiya M., Hayashi C., 1986, *Settling and growth of dust particles in a laminar phase of a low-mass solar nebula*, Icarus, 67, 375-390

Nelson R. P., Gressel O., Umurhan O. M., 2013, *Linear and non-linear evolution of the vertical shear instability in accretion discs*, MNRAS, 435, 2610-2632

Nesvorný D., Youdin A. N., Richardson D. C., 2010, *Formation of Kuiper Belt Binaries by Gravitational Collapse*, AJ, 140, 785-793

Nesvorný D., Li R., Youdin A. N., Simon J. B., Grundy W. M., 2019, *Trans-Neptunian binaries as evidence for planetesimal formation by the streaming instability*, Nature Astronomy, 3, 808-812

Ormel C. W., Shi J.-M., Kuiper R., 2015, *Hydrodynamics of embedded planets' first atmospheres - II. A rapid recycling of atmospheric gas*, MNRAS, 447, 3512-3525

Paardekooper S. J., Papaloizou J. C. B., 2009, *On the width and shape of the corotation region for low-mass planets*, MNRAS, 394, 2297-2309

Paardekooper S.-J., McNally C. P., Lovascio F., 2020, *Polydisperse streaming instability - I. Tightly coupled particles and the terminal velocity approximation*, MNRAS, 499, 4223-4238

Paardekooper S.-J., McNally C. P., Lovascio F., 2021, *Polydisperse streaming instability - II. Methods for solving the linear stability problem*, MNRAS, 502, 1579-1595

Paardekooper S., Dong R., Duffell P., Fung J., Masset F. S., Ogilvie G., Tanaka H., in *Protostars and Planets VII*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M., p. 685 (arXiv:2203.09595), doi:10.48550/arXiv.2203.09595

Pérez L. M., et al., 2018, *The Disk Substructures at High Angular Resolution Project (DSHARP). X. Multiple Rings, a Misaligned Inner Disk, and a Bright Arc in the Disk around the T Tauri star HD 143006*, ApJ, 869, L50

Pineda J. E., et al.,in *Protostars and Planets VII*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M., p. 233 (arXiv:2205.03935), doi:10.48550/arXiv.2205.03935

Pinte C., Teague R., Flaherty K., Hall C., Facchini S., Casassus S., in *Protostars and Planets VII*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M., p. 645 (arXiv:2203.09528), doi:10.48550/arXiv.2203.09528

Pinte C., Benisty M., Facchini S., Fukagawa M., Teague R., in *AAS/Division for Extreme Solar Systems Abstracts*, booktitle, p. 400.01

Powell K. G., Roe P. L., Linde T. J., Gombosi T. I., De Zeeuw D. L., 1999, *A Solution-Adaptive Upwind Scheme for Ideal Magnetohydrodynamics*, Journal of Computational Physics, 154, 284-309

Price D. J., 2010, *Smoothed Particle Magnetohydrodynamics - IV. Using the vector potential*, MNRAS, 401, 1475-1499

Price D. J., 2012, *Smoothed particle hydrodynamics and magnetohydrodynamics*, Journal of Computational Physics, 231, 759-794

Price D. J., et al., 2018, *Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics*, PASA, 35, e031

Raettig N., Klahr H., Lyra W., 2015, *Particle Trapping and Streaming Instability in Vortices in Protoplanetary Disks*, ApJ, 804, 35

Rafikov R. R., 2005, *Can Giant Planets Form by Direct Gravitational Instability?*, ApJ, 621, L69-L72

Rice W. K. M., Armitage P. J., Bate M. R., Bonnell I. A., 2003, *The effect of cooling on the global stability of self-gravitating protoplanetary discs*, MNRAS, 339, 1025-1030

Richard S., Nelson R. P., Umurhan O. M., 2016, *Vortex formation in protoplanetary discs induced by the vertical shear instability*, MNRAS, 456, 3571-3584

Ross J., Latter H. N., 2018, *Dissipative structures in magnetorotational turbulence*, MNRAS, 477, 3329-3342

Safronov V. S., , booktitle, https://api.semanticscholar.org/CorpusID:140184681

Scepi N., Lesur G., Dubus G., Flock M., 2018, *Turbulent and wind-driven accretion in dwarf novae threaded by a large-scale magnetic field*, A&A, 620, A49

Schulik M., Johansen A., Bitsch B., Lega E., Lambrechts M., 2020, *On the structure and mass delivery towards circumplanetary discs*, A&A, 642, A187

Shakura N. I., Sunyaev R. A., 1973, *Black holes in binary systems. Observational appearance.*, A&A, 24, 337-355

Simon J. B., Armitage P. J., Li R., Youdin A. N., 2016, *The Mass and Size Distribution of Planetesimals Formed by the Streaming Instability. I. The Role of Self-gravity*, ApJ, 822, 55

Smoluchowski M. v., 1918, *Versuch einer mathematischen Theorie der Koagulationskinetik kolloider Lösungen*, Zeitschrift für physikalische Chemie, 129–168

Springel V., Pakmor R., Zier O., Reinecke M., 2021, *Simulating cosmic structure formation with the GADGET-4 code*, MNRAS, 506, 2871-2949

Squire J., Hopkins P. F., 2018a, *Resonant drag instabilities in protoplanetary discs: the streaming instability and new, faster growing instabilities*, MNRAS, 477, 5011-5040

Squire J., Hopkins P. F., 2018b, *Resonant Drag Instability of Grains Streaming in Fluids*, ApJ, 856, L15

Stepinski T. F., Valageas P., 1997, *Global evolution of solid matter in turbulent protoplanetary disks. II. Development of icy planetesimals.*, A&A, 319, 1007-1019

Szulágyi J., 2017, *Effects of the Planetary Temperature on the Circumplanetary Disk and on the Gap*, ApJ, 842, 103

Tanaka H., Takeuchi T., Ward W. R., 2002, *Three-Dimensional Interaction between a Planet and an Isothermal Gaseous Disk. I. Corotation and Lindblad Torques and Planet Migration*, ApJ, 565, 1257-1274

Tanigawa T., Ohtsuki K., Machida M. N., 2012, *Distribution of Accreting Gas and Angular Momentum onto Circumplanetary Disks*, ApJ, 747, 47

Teague R., et al., 2016, *Measuring turbulence in TW Hydrae with ALMA: methods and limitations*, A&A, 592, A49

Testi L., et al.,in *Protostars and Planets VI*, booktitle, edited by Beuther, Henrik and Klessen, Ralf S. and Dullemond, Cornelis P. and Henning, Thomas, pp 339–361 (arXiv:1402.1354), doi:10.2458/azu˙uapress˙9780816531240-ch015

Teyssier R., 2002, *Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES*, A&A, 385, 337-364

Tomida K., Stone J. M., 2023, *The Athena++ Adaptive Mesh Refinement Framework: Multigrid Solvers for Self-gravity*, ApJS, 266, 7

Toomre A., 1964, *On the gravitational stability of a disk of stars.*, ApJ, 139, 1217-1238

Tsukamoto Y., et al.,in *Astronomical Society of the Pacific Conference Series*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M., p. 317

Umurhan O. M., Estrada P. R., Cuzzi J. N., 2020, *Streaming Instability in Turbulent Protoplanetary Disks*, ApJ, 895, 4

Varnière P., Tagger M., 2006, *Reviving Dead Zones in accretion disks by Rossby vortices at their boundaries*, A&A, 446, L13-L16

Venuti L., et al., 2014, *Mapping accretion and its variability in the young open cluster NGC 2264: a study based on u-band photometry*, A&A, 570, A82

Wafflard-Fernandez G., Lesur G., 2023, *Planet-disk-wind interaction: The magnetized fate of protoplanets*, A&A, 677, A70

Ward W. R., in *Lunar and Planetary Science Conference*, booktitle, p. 1463

Ward W. R., 1997, *Protoplanet Migration by Nebula Tides*, Icarus, 126, 261-281

Wardle M., Ng C., 1999, *The conductivity of dense molecular gas*, MNRAS, 303, 239-246

Weidenschilling S. J., 1977, *Aerodynamics of solid bodies in the solar nebula.*, MNRAS, 180, 57-70

Whipple F. L., in *From Plasma to Planet*, booktitle, edited by Elvius, Aina, p. 211

Wurster J., Price D. J., Bate M. R., 2016, *Can non-ideal magnetohydrodynamics solve the magnetic braking catastrophe?*, MNRAS, 457, 1037-1061

Xu Z., Bai X.-N., 2022, *Turbulent Dust-trapping Rings as Efficient Sites for Planetesimal Formation*, ApJ, 937, L4

Youdin A. N., Goodman J., 2005, *Streaming Instabilities in Protoplanetary Disks*, ApJ, 620, 459-469

Youdin A., Johansen A., 2007, *Protoplanetary Disk Turbulence Driven by the Streaming Instability: Linear Evolution and Numerical Methods*, ApJ, 662, 613-626

Zhang S., Hartmann L., Zamora-Avilés M., Kuznetsova A., 2018a, *On estimating angular momenta of infalling protostellar cores from observations*, MNRAS, 480, 5495-5503

Zhang S., et al., 2018b, *The Disk Substructures at High Angular Resolution Project (DSHARP). VII. The Planet-Disk Interactions Interpretation*, ApJ, 869, L47

Zhu Z., Hartmann L., Nelson R. P., Gammie C. F., 2012, *Challenges in Forming Planets by Gravitational Instability: Disk Irradiation and Clump Migration, Accretion, and Tidal Destruction*, ApJ, 746, 110

Zier O., Mayer A. C., Springel V., 2024, *Non-ideal magnetohydrodynamics on a moving mesh II: Hall effect*, MNRAS, 527, 8355-8368

Zubko E., 2012, *Light scattering by irregularly shaped particles with sizes comparable to the wavelength*. pp 39–74, doi:10.1007/978-3-642-15531-4˙2

de Val-Borro M., et al., 2006, *A comparative study of disc-planet interaction*, MNRAS, 370, 529-558

de Val-Borro M., Artymowicz P., D'Angelo G., Peplinski A., 2007, *Vortex generation in protoplanetary disks with an embedded giant planet*, A&A, 471, 1043-1055

Chapter 2

Numerical Computation of astrophysical flows

Contents

1. Introduction

In this chapter, we introduce the main hydrodynamic schemes relevant for astrophysics that we have implemented in Shamrock. For simplicity, we restrain ourselves to pure hydrodynamics and neglect dust, magnetic, and radiative effects here.

1.1. Euler's equation

Overall, the basis of most hydrodynamics problems we aim to study derives from a form of the Euler equation with potential addition of other terms or fields.

The Euler equation can be written in the following form

$$(\partial_t + \mathbf{v}\cdot\boldsymbol{\nabla})\rho = -\boldsymbol{\nabla}\cdot\mathbf{v}, \tag{2.1}$$

$$(\partial_t + \mathbf{v}\cdot\boldsymbol{\nabla})\mathbf{v} = -\frac{1}{\rho}\boldsymbol{\nabla}P + \mathbf{f}, \tag{2.2}$$

$$(\partial_t + \mathbf{v}\cdot\boldsymbol{\nabla})u = -\frac{P}{\rho}\boldsymbol{\nabla}\cdot\mathbf{v}, \tag{2.3}$$

where ρ is the density of the fluid, $\mathbf{v}$ its velocity, u its specific internal energy, and $\mathbf{f}$ denotes other external forces. The fluid pressure P is defined through an astrophysical equation of state, which closes the system of equations. For example, when using an adiabatic equation of state, $P = (\gamma - 1)\rho u$, or $P = c_s^2\rho$ when the equation of state is isothermal. This form of the Euler equation is referred to as the

convective form. The latter can also be transformed into a so-called conservative form.

$$\partial_t \rho + \boldsymbol{\nabla} \cdot (\rho \mathbf{v}) = 0 \tag{2.4}$$

$$\partial_t (\rho \mathbf{v}) + \boldsymbol{\nabla} \cdot (\rho \mathbf{v}\mathbf{v} + P\mathbf{I}) = \rho \mathbf{f} \tag{2.5}$$

$$\partial_t E + \boldsymbol{\nabla} \cdot [(E + P)\mathbf{v}] = \rho \mathbf{v} \cdot \mathbf{f} \tag{2.6}$$

where $E = \rho u + \frac{1}{2}\rho \mathbf{v}^2$ is the total energy density of the fluid.

1.2. Rankine-Hugoniot conditions

Euler's equations consist of a set of partial differential equations that describe the evolution of inviscid smooth flows in the form of solutions that are differentiable in both space and time. However, the dynamics resulting from the Euler equation result in non-linearities whose derivatives can sharpen until they become discontinuities. Such flows are referred to as shocks. For example, wave steepening can occur for nonlinear acoustic waves. In order to solve problems with such discontinuities, we can use the so-called Rankine-Hugoniot equations. They are obtained by using the conservative form of the Euler equation. Consider, for example, mass conservation in one dimension, this equation can be written under the form

$$\partial_t \rho + \partial_x (\rho v_x) = 0.$$

Let us integrate spatially a stationary shock located at position χ over a width ϵ

$$\int_{\chi-\epsilon}^{\chi+\epsilon} (\partial_t \rho) \mathrm{d}x + \int_{\chi-\epsilon}^{\chi+\epsilon} \partial_x (\rho v_x) \, \mathrm{d}x = 0.$$

The use of the Leibniz theorem on the left integral gives

$$\partial_t \int_{\chi}^{\chi+\epsilon} \rho \mathrm{d}x + \partial_t \int_{\chi-\epsilon}^{\chi} \rho \mathrm{d}x + [\rho v_x]_{\chi-\epsilon}^{\chi+\epsilon} = 0.$$

Taking the limit $\epsilon \to 0$ provides the Rankine-Hugoniot condition for a stationary shock on the mass equation

$$(\rho v_x)_{\text{left}} = (\rho v_x)_{\text{right}}, \tag{2.7}$$

which expresses the continuity of the mass flux.

As detailed in Toro (2013), this is the case in general, where the Rankine-Hugoniot conditions for stationary shocks consist of the equality of the flux on both sides (the flux must be continuous across the shock). From the Euler equations, in one dimension, one obtains the three conditions.

$$\Delta (\rho v_x) = 0, \tag{2.8}$$

$$\Delta (\rho v_x v_x + P) = 0, \tag{2.9}$$

$$\Delta [(E + P) v_x] = 0, \tag{2.10}$$

where Δ denotes the difference between the left and right values of the shock. Those conditions are used to solve the case of discontinuities in the Euler equation (see Sec. 3.2).

2. Finite elements (Zeus & Fargo)

The first scheme that we discuss is the Stone & Norman (1992a) scheme. Since it corresponds to the actual scheme implemented in the eponym community code, we will commonly refer to this scheme as the ZEUS scheme in the manuscript. We have chosen this scheme for multiple reasons. Firstly, it is a simple scheme, allowing to test the behavior of the grid in SHAMROCK and track potential bottlenecks because of the inherent speed of the scheme associated with the underlying numerical framework. Fixing those issues can improve the performance of the other schemes in SHAMROCK. Secondly, the ZEUS solver is the solver on the basis of the FARGO, which is very useful to perform both discs and shearing box simulations because of its fast advection algorithm. The ZEUS solver has recently been ported to the GPU (Benítez-Llambay & Masset, 2016), allowing potential comparisons with its performance. Lastly, the ZEUS scheme conserves internal energy rather than total energy, which can facilitate the numerical integration of instabilities relying on the thermodynamics of the fluid (e.g. Rayleigh-Taylor, Kelvin Helmholtz).

2.1. Functional form of the equations

The ZEUS scheme solves the basic Euler equation in the following form:

$$\partial_t \rho + \boldsymbol{\nabla} \cdot (\rho \mathbf{v}) = 0, \tag{2.11}$$

$$(\partial_t + \mathbf{v} \cdot \boldsymbol{\nabla})\, \mathbf{v} = -\frac{1}{\rho} \boldsymbol{\nabla} P + \mathbf{f}, \tag{2.12}$$

$$(\partial_t + \mathbf{v} \cdot \boldsymbol{\nabla}) \left(\frac{U}{\rho} \right) = -\frac{P}{\rho} \boldsymbol{\nabla} \cdot \mathbf{v}, \tag{2.13}$$

where instead of manipulating the specific internal energy, the internal energy U is used. This choice has the advantage in this scheme that it avoids the need for a floating point division, which is the most expensive basic floating point operation. We will come back to this point later.

2.2. Operator splitting

It is an operator splitting scheme where source terms are integrated using the Forward Time-Centered Space method (FTCS), but transport is done using conservative upwinding. The different steps can be highlighted as follows:

$$\partial_t \rho + \boldsymbol{\nabla} \cdot (\rho \mathbf{v}) = 0, \tag{2.14}$$

$$(\partial_t + \mathbf{v} \cdot \boldsymbol{\nabla})\, \mathbf{v} = -\frac{1}{\rho} \boldsymbol{\nabla} p + \mathbf{f}, \tag{2.15}$$

$$(\partial_t + \mathbf{v} \cdot \boldsymbol{\nabla}) \left(\frac{U}{\rho} \right) = -\frac{p}{\rho} \boldsymbol{\nabla} \cdot \mathbf{v}. \tag{2.16}$$

As we will explain, the scheme can be summarized as a succession of four operators to perform a full timestep Δt:

- Substep 1 : Add pressure and external forces to the velocity with timestep Δt.
- Substep 2 : Add artificial viscosity terms to the velocity and internal energy with timestep Δt.
- Substep 3 : Implicit update with timestep Δt of the internal energy with respect to the term $-\frac{p}{\rho}\boldsymbol{\nabla}\cdot\mathbf{v}$ if possible, or a predictor corrector method otherwise.
- Transport step : Using the variables in conservative form, perform advection with timestep Δt.

2.3. Staggered mesh

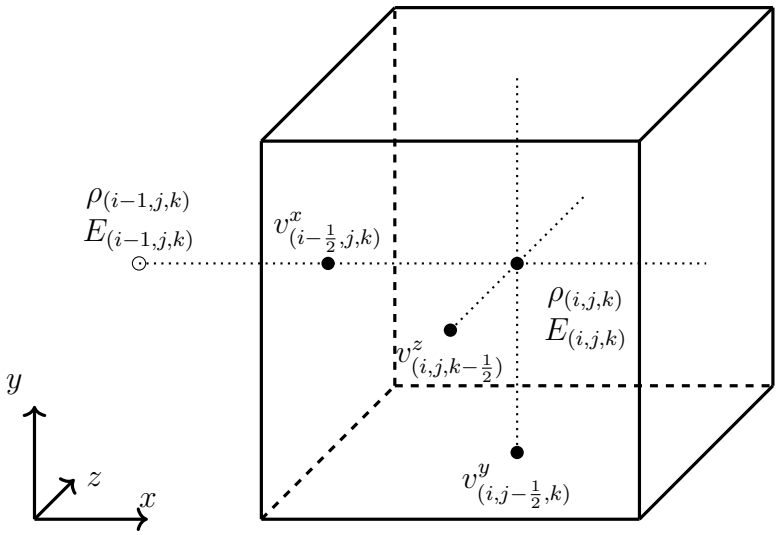


Figure 2.1: Staggered mesh setup: the black dot represents values that belong to the cell (i, j, k).

The Zeus scheme uses a staggered mesh to represent the discretization points. In it, the scalar quantities are represented on cell centers, whereas vector quantities such as velocity are represented on cell faces, the corresponding quantities are represented in Fig. 2.1. The state vector of a cell here is

$$\mathbb{U}_{(i,j,k)} = (\rho_{(i,j,k)}, v^x_{(i-\frac{1}{2},j,k)}, v^y_{(i,j-\frac{1}{2},k)}, v^z_{(i,j,k-\frac{1}{2})}, E_{(i,j,k)})$$

as we store quantities per cell even if they are technically on the faces. This means that, provided that we interpolate cell-centered quantities to the cell faces, the

fluxes can simply be computed using their analytic expressions, without any need for reconstruction.

Unlike Godunov-type methods (see Sec. 3), using a staggered mesh removes the need for a Riemann solver, reducing the numerical cost of the scheme significantly. However, the lack of Riemann solvers means that an alternative method has to be used in order for the numerical scheme to handle hydrodynamic shocks.

Boundary conditions can be handled in such a scheme by setting the values of cells in a so-called ghost zone accordingly to create a boundary condition. For a periodic domain, the cells on one edge of the simulation will be mirrored on the other side of the periodic boundary.

2.4. Artificial viscosity

In computation fluid dynamics, two main methods are used to resolve shock properties in a numerical scheme. The first one is the Riemann solver (see Sec. 3), which, as stated, is not in use in this scheme. The other option is the so-called artificial viscosity originally introduced by the unpublished reports Richtmyer (1948a,b) and whose results were included in the landmark paper Von Neumann & Richtmyer (1950). The solution is to introduce a non-physical additional viscosity to automatically take care of the shocks in the scheme by the additional dissipation, which will 'smear' the shock (Richtmyer, 1948a). The other insight is that it is possible to provide such additional dissipation while maintaining the Rankine-Hugoniot conditions and having the correct entropy increase due to the shock.

In the Zeus scheme, the artificial viscosity is taken as being either by default the quadratic form from Von Neumann & Richtmyer (1950) where the additional viscosity is proportional to the velocity gradient squared per axis. Additionally, as detailed by Stone & Norman (1992a), in cases of strong shocks, it can be suitable to introduce another artificial viscosity in the linear form. Overall, accounting for the artificial viscosity terms the solved equations can be written as

$$\partial_t \rho + \boldsymbol{\nabla} \cdot (\rho \mathbf{v}) = 0 \tag{2.17}$$

$$(\partial_t + \mathbf{v} \cdot \boldsymbol{\nabla})\, \mathbf{v} = -\frac{1}{\rho} \boldsymbol{\nabla} \cdot \tilde{P} + \mathbf{f} \tag{2.18}$$

$$(\partial_t + \mathbf{v} \cdot \boldsymbol{\nabla}) \left(\frac{U}{\rho} \right) = -Tr \left(\frac{\tilde{P}^T}{\rho} \cdot \boldsymbol{\nabla} \mathbf{v} \right) \tag{2.19}$$

$$\tilde{P}_{\mu\nu} = P(\rho, U)\delta_{\mu\nu} + \chi_1 \rho c_s \delta_{\mu\nu} (\partial_\mu v_\mu)^+ + \chi_2 \rho \delta_{\mu\nu} ((\partial_\mu v_\mu)^+)^2 \tag{2.20}$$

where $(\cdot)^+$ is the positive part, $\rho c_s \delta_{\mu\nu}(\partial_\mu v_\mu)$ is the linear artificial viscosity, and $\rho \delta_{\mu\nu}(\partial_\mu v_\mu)^2$ is the quadratic one. It can be noted that such a form is not a proper tensor because of the velocity derivative squared, not even accounting for the numerical scheme; the underlying equations are not invariant under the coordinate transform. This issue, however, can be addressed by replacing the artificial viscosity with a tensor (Stone & Norman, 1992a, Appendix b) inspired by the physical form

in Mihalas Mihalas (Mihalas & Mihalas, 1984, § 25). Such an extension does not impact the underlying scheme, so it can easily be added at a later stage. In our case the Zeus scheme is being used to explore the feasibility grid scheme in Shamrock, but it is not yet a priority to implement it as of now.

In the following section, we detail the different steps of the Zeus scheme, as it is hard to find them in a closed form in the literature. We omit the linear artificial viscosity term as it is just a modification of the applied formula for the artificial viscosity and thus can be simply added to the formulas presented here.

2.5. Substep 1 (Pressure gradient)

In substep 1, the goal is to add the contribution of the pressure gradient and of the external forces. As per Stone & Norman (1992a),

$$\frac{v^{x,s1}_{(i-\frac{1}{2},j,k)} - v^{x,n}_{(i-\frac{1}{2},j,k)}}{\Delta t} = -\frac{P_{(i,j,k)} - P_{(i-1,j,k)}}{\Delta x(\rho^n_{(i,j,k)} + \rho^n_{(i-1,j,k)})/2} + f^x_{(i-\frac{1}{2},j,k)}, \tag{2.21}$$

$$\frac{v^{y,s1}_{(i,j-\frac{1}{2},k)} - v^{y,n}_{(i,j-\frac{1}{2},k)}}{\Delta t} = -\frac{P_{(i,j,k)} - P_{(i,j-1,k)}}{\Delta y(\rho^n_{(i,j,k)} + \rho^n_{(i,j-1,k)})/2} + f^y_{(i,j-\frac{1}{2},k)}, \tag{2.22}$$

$$\frac{v^{z,s1}_{(i,j,k-\frac{1}{2})} - v^{z,n}_{(i,j,k-\frac{1}{2})}}{\Delta t} = -\frac{P_{(i,j,k)} - P_{(i,j,k-1)}}{\Delta z(\rho^n_{(i,j,k)} + \rho^n_{(i,j,k-1)})/2} + f^z_{(i,j,k-\frac{1}{2})}, \tag{2.23}$$

where superscript $s1$ refers to the result of the substep, and superscript n refers to the initial state at the beginning of the timestep. The pressure P is computed just before computing its gradient according to the equation of state, where for the adiabatic case we use $P_{(i,j,k)} = (\gamma - 1)E^n_{(i,j,k)}$.

2.6. Substep 2 (Artificial viscosity)

Terms related to artificial viscosity are then computed using

$$q^x_{(i,j,k)} = C_2\rho^n_{(i,j,k)}((v^{x,n}_{(i+\frac{1}{2},j,k)} - v^{x,n}_{(i-\frac{1}{2},j,k)})^+)^2, \tag{2.24}$$

$$q^y_{(i,j,k)} = C_2\rho^n_{(i,j,k)}((v^{y,n}_{(i,j+\frac{1}{2},k)} - v^{y,n}_{(i,j-\frac{1}{2},k)})^+)^2, \tag{2.25}$$

$$q^z_{(i,j,k)} = C_2\rho^n_{(i,j,k)}((v^{z,n}_{(i,j,k+\frac{1}{2})} - v^{z,n}_{(i,j,k-\frac{1}{2})})^+)^2, \tag{2.26}$$

where C_2 is a constant corresponding qualitatively to the smearing length that will be applied to the shock. In practice, we use $C_2 \simeq 3$, as in Stone & Norman (1992a). The contribution due to the stress associated to the artificial viscosity is then dif-

ferentiated and added to the velocity and energy fields.

$$\frac{v^{x,s2}_{(i-\frac{1}{2},j,k)} - v^{x,s1}_{(i-\frac{1}{2},j,k)}}{\Delta t} = -\frac{q^x_{(i,j,k)} - q^x_{(i-1,j,k)}}{\Delta x(\rho^n_{(i,j,k)} + \rho^n_{(i-1,j,k)})/2}, \tag{2.27}$$

$$\frac{v^{y,s2}_{(i,j-\frac{1}{2},k)} - v^{y,s1}_{(i,j-\frac{1}{2},k)}}{\Delta t} = -\frac{q^y_{(i,j,k)} - q^y_{(i,j-1,k)}}{\Delta y(\rho^n_{(i,j,k)} + \rho^n_{(i,j-1,k)})/2}, \tag{2.28}$$

$$\frac{v^{z,s2}_{(i,j,k-\frac{1}{2})} - v^{z,s1}_{(i,j,k-\frac{1}{2})}}{\Delta t} = -\frac{q^z_{(i,j,k)} - q^z_{(i,j,k-1)}}{\Delta z(\rho^n_{(i,j,k)} + \rho^n_{(i,j,k-1)})/2}, \tag{2.29}$$

$$\begin{aligned}\frac{E^{s2}_{(i,j,k)} - E^n_{(i,j,k)}}{\Delta t} = &-q^x_{(i,j,k)}\left(\frac{v^{x,n}_{(i+\frac{1}{2},j,k)} - v^{x,n}_{(i-\frac{1}{2},j,k)}}{\Delta x}\right) - q^y_{(i,j,k)}\left(\frac{v^{y,n}_{(i,j+\frac{1}{2},k)} - v^{y,n}_{(i,j-\frac{1}{2},k)}}{\Delta x}\right) \\ &- q^z_{(i,j,k)}\left(\frac{v^{z,n}_{(i,j,k+\frac{1}{2})} - v^{z,n}_{(i,j,k-\frac{1}{2})}}{\Delta x}\right).\end{aligned} \tag{2.30}$$

As mentioned above, the linear artificial viscosity can be added by modifying the expressions Eq. 2.24-2.26 to include it.

2.7. Substep 3 (Compressional heating)

In order to complete the source step with the compressional heating term in Stone & Norman (1992a), an implicit update is performed using the time-centered pressure.

$$\frac{E^{s3}_{(i,j,k)} - E^{s2}_{(i,j,k)}}{\Delta t} = -\frac{P^{n+1} + P^n}{2}\boldsymbol{\nabla}\cdot\mathbf{v}. \tag{2.31}$$

When using an adiabatic equation of state, it can be rewritten as

$$E^{s3}_{(i,j,k)} = \left[\frac{1 - (\Delta t/2)(\gamma-1)(\boldsymbol{\nabla}\cdot\mathbf{v}^n)_{(i,j,k)}}{1 + (\Delta t/2)(\gamma-1)(\boldsymbol{\nabla}\cdot\mathbf{v}^n)_{(i,j,k)}}\right] E^{s2}_{(i,j,k)} \tag{2.32}$$

In ZEUS, this step was chosen to be implicit in order to improve energy conservation. Alternatively, when the equation of state is not adiabatic, a predictor corrector scheme is used (e.g. Stone & Norman 1992a).

2.8. Transport step

We choose to implement the transport step as it is performed in the code FARGO3D since only cell-centered quantities are transported, making the transport consistent for all fields. Secondly, in order to implement the FARGO fast advection algorithm, the control volume has to be the same for all variables (Benítez-Llambay & Masset, 2016, Section 3.4.1). The transport step being the same for all fields additionally means that it is possible to write only one routine for it and reuse it for all fields in the scheme, which is convenient.

The initial issue lies in the fact that velocities are face-centered quantities and densities are cell-centered, but we need conservative variables for the transport step. In order to define momentum, one would need to either interpolate densities to the faces or interpolate the velocities to the cell center. In the strategy adopted in the FARGO code, neither option is used. Instead, we define two cell-centered moments per direction, e.g.

$$\Pi^{+x,n}_{(i,j,k)} = \rho^{n}_{(i,j,k)} v^{x,s2}_{(i+1/2,j,k)}, \tag{2.33}$$

$$\Pi^{-x,n}_{(i,j,k)} = \rho^{n}_{(i,j,k)} v^{x,s2}_{(i-1/2,j,k)}, \tag{2.34}$$

doing so yields six momentas that are cell-centered and that can be advected. At the end of the step, velocities are reconstructed from the advected cell-centered moments as follows:

$$v^{x,n+1}_{(i-\frac{1}{2},j,k)} = \frac{\Pi^{+x,n+1}_{(i-1,j,k)} + \Pi^{-x,n+1}_{(i,j,k)}}{\rho^{n+1}_{(i-1,j,k)} + \rho^{n+1}_{(i,j,k)}}, \tag{2.35}$$

and similarly for each axis.

The list of variables to advect during the transport step can be conveniently written as an 8-dimensional vector:

$$\mathbf{Q}_{(i,j,k)} \equiv \{\rho, \Pi^{+x}, \Pi^{-x}, \Pi^{+y}, \Pi^{-y}, \Pi^{+z}, \Pi^{-z}, E\}_{(i,j,k)}, \tag{2.36}$$

The transport step can be summarized under the following equations:

$$\partial_t \iiint_v \mathbf{Q} \mathrm{d}V + \iint_{\partial V} \mathbf{Q}\mathbf{v} \cdot \mathrm{d}\mathbf{S} = 0, \tag{2.37}$$

which, translated to our staggered grid, becomes:

$$\frac{\mathbf{Q}^{n+1}_{(i,j,k)} - \mathbf{Q}^{n}_{(i,j,k)}}{\Delta t} = \Big[\mathcal{F}^{x-}_{(i-1/2,j,k)} + \mathcal{F}^{y-}_{(i,j-1/2,k)} + \mathcal{F}^{z-}_{(i,j,k-1/2)} \tag{2.38}$$

$$- \mathcal{F}^{x+}_{(i+1/2,j,k)} - \mathcal{F}^{y+}_{(i,j+1/2,k)} - \mathcal{F}^{z+}_{(i,j,k+1/2)}\Big]/V, \tag{2.39}$$

where we have chosen, as in Benítez-Llambay & Masset (2016), the convention that the flux is always oriented in the same direction as the axis. However, unlike in the FARGO3D code, where directions are done as three successive one-dimensional problems, we have chosen to perform the three directions simultaneously to treat them uniformly, as it is better suited to how the code is designed, and to make the treatment of the three directions consistent.

Unlike in Godunov-type methods using staggered meshes, it is possible to write the flux as

$$\mathcal{F}^{x+}_{(i+1/2,j,k)} = v^{x}_{(i+1/2,j,k)} \mathbf{Q}^{*,x}_{(i+1/2,j,k)} S_{(i+1/2,j,k)}, \tag{2.40}$$

where $\mathbf{Q}^*$ is the upwinded quantites from cell centers and by half a step in time, corresponding, for example, at the face $i + 1/2$ (in 1D) to the value being at the position $x^* = x_{i+1/2} - v^x_{i+1/2}\Delta t/2$. In Stone & Norman (1992a); Benítez-Llambay & Masset (2016) multiple options are available: donner-cell (1st order), slope limiter (1st or 2nd order), and Piecewise Parabolic Advection (3rd order). In our case we only studied donnor cell and slope limiter, as PPA is more complex to implement and comes at a significant performance cost.

For a slope limiter case, they can be written as being

$$\mathbf{Q}^{*x}_{(i+1/2,j,k)} \equiv \begin{cases} \mathbf{Q}_{(i,j,k)} + \mathbf{a}^x_{(i,j,k)} \left(\Delta x - v^x_{(i+1/2,j,k)}\Delta t\right)/2 & \text{if } v^x_{(i+1/2,j,k)} \geqslant 0 \\ \mathbf{Q}_{(i+1,j,k)} - \mathbf{a}^x_{(i+1,j,k)} \left(\Delta x + v^x_{(i+1/2,j,k)}\Delta t\right)/2 & \text{if } v^x_{(i+1/2,j,k)} < 0 \end{cases}, \tag{2.41}$$

and similarly on the other axis, where $\mathbf{a}$ is the limited slope related to $\mathbf{Q}$. In this case, the donnor cell method corresponds to $\mathbf{a} = 0$.

To estimate the limited slope $\mathbf{a}$ on each face, first we compute the slope on each face

$$\Delta\mathbf{Q}_{(i+1/2,j,k)} = \left(\mathbf{Q}_{(i+1,j,k)} - \mathbf{Q}_{(i,j,k)}\right)\Delta x, \tag{2.42}$$

$$\Delta\mathbf{Q}_{(i,j+1/2,k)} = \left(\mathbf{Q}_{(i,j+1,k)} - \mathbf{Q}_{(i,j,k)}\right)\Delta y, \tag{2.43}$$

$$\Delta\mathbf{Q}_{(i,j,k+1/2)} = \left(\mathbf{Q}_{(i,j,k+1)} - \mathbf{Q}_{(i,j,k)}\right)\Delta z. \tag{2.44}$$

Using those slopes, it is possible to write the limited slope as being

$$\mathbf{a}^x_{(i,j,k)} \equiv \Phi\left(\Delta\mathbf{Q}_{(i+1/2,j,k)}, \Delta\mathbf{Q}_{(i-1/2,j,k)}\right), \tag{2.45}$$

$$\mathbf{a}^y_{(i,j,k)} \equiv \Phi\left(\Delta\mathbf{Q}_{(i,j+1/2,k)}, \Delta\mathbf{Q}_{(i,j-1/2,k)}\right), \tag{2.46}$$

$$\mathbf{a}^z_{(i,j,k)} \equiv \Phi\left(\Delta\mathbf{Q}_{(i,j,k+1/2)}, \Delta\mathbf{Q}_{(i,j,k-1/2)}\right), \tag{2.47}$$

where Φ is the slope limiter function times one of the two slopes, for example, when using the Van Leer slope limiter

$$\Phi(\Delta\mathbf{Q}_-, \Delta\mathbf{Q}_+) = \begin{cases} 0, & \text{if } \Delta\mathbf{Q}_-\,\Delta\mathbf{Q}_+ < 0 \\ 2\dfrac{\Delta\mathbf{Q}_-\,\Delta\mathbf{Q}_+}{\Delta\mathbf{Q}_- + \Delta\mathbf{Q}_+}, & \text{otherwise} \end{cases}. \tag{2.48}$$

Stone & Norman (1992a) mentions the possibility of using so-called consistent transport to reduce the diffusivity of the transport step, in which case the transported quantities are instead the ones divided by the density. Lastly, as mentioned in Benítez-Llambay & Masset (2016) the transport scheme as presented conserves momentum.

2.9. Courant-Friedrichs-Lewy condition

The stability criterion, also called Courant-Friedrichs-Lewy (CFL), for this scheme to not develop grid-scale oscillations and be stable over the course of the simulation is, as detailed by Stone & Norman (1992a),

$$\Delta t \leq \min(\Delta x)/(|\mathbf{v}| + c_s), \tag{2.49}$$

where Δt is the timestep duration, Δx the grid cell size, $\mathbf{v}$ the fluid velocity vector, and c_s the sound speed. This condition expresses that information must not travel more than a grid cell per timestep, see Richtmyer & Dill (1959) for the rigorous derivation.

2.10. Performance

On most architectures, floating-point division is one of the most costly instructions. The main advantage of the Zeus scheme is revealed when we count them. Assuming that division by constants can be precomputed such that we multiply by the inverse instead, this leaves 3 divisions per cell in substep 1, which can be reused in substep 2. In substep 3, only one division per cell is applied. In the transport step, there is one division per expected quantity in the Van Leer slope plus one per cell to reconstruct the velocities at the end of the step. This amounts to a total of 11 divisions per cell, which is very low. For comparaison in a HLL Riemann solver, there are roughly, per direction, 2 square roots for the soundspeeds, 3 for the conservative to primitive conversion, and 2 in the HLL flux itself. This results in 15 divisions plus 6 square roots only for the Riemann solver. This shows why the Zeus is appealing computationally. We note, however, that this may not necessarily be true on modern hardware, which would be memory-bound instead of compute-bound since the Zeus scheme involves more memory-bound than a Godunov-type solver.

3. Finite volume (Godunov)

3.1. Formulation of hydro equations

An alternative approach to solve the hydrodynamics equation in astrophysics is to use Godunov-type schemes. They have many advantages compared to traditional finite element schemes since they are, by construction, conservative. In order to present Godunov's type schemes (schemes built using the Godunov method), it is necessary to initially introduce several concepts. We follow hereafter the approach of Toro (2013).

The starting point of the Godunov method is to rewrite the basic conservative form of the Euler equations (Eq. 2.4- Eq. 2.6) as a hyperbolic conservation law

$$\partial_t \mathbb{U} + \boldsymbol{\nabla} \cdot \mathbf{F}(\mathbb{U}) = 0, \tag{2.50}$$

where $\mathbb{U} = (\rho, \rho\mathbf{v}, E)$ denotes the state vector made of the conservative variables, and $\mathbf{F} = (\rho\mathbf{v}, \rho\mathbf{v}\mathbf{v} + P\mathbf{I}, (E + P)\mathbf{v})$ the flux function. This form contrasts with the quasi-linear form of the Euler equation that involves the physical or primitive variables, for which $\mathbb{W} = (\rho, \mathbf{v}, P)$ is the primitive variable state vector. For instance, in a single dimension the primitive form of the Euler equation is:

$$\partial_t \mathbb{W} + \mathbf{A}(\mathbb{W})\partial_x \mathbb{W} = 0, \quad \mathbf{A}(\mathbb{W}) = \begin{pmatrix} v_x & \rho & 0 \\ 0 & v_x & 1/\rho \\ 0 & \rho c_s^2 & v_x \end{pmatrix}, \tag{2.51}$$

where c_s is the sound speed as determined by the equation of state. The matrix $\mathbf{A}$ can be diagonalized using 3 eigenvalues and eigenvectors corresponding to the eigenvalues $(v_x - c_s, v_x, v_x + c_s)$. Since the eigenvalues are real numbers, the equation is said to be hyperbolic. Additionally, its eigenvalues have the dimension of a velocity, meaning that an eigenvector will move at the speed corresponding to its eigenvalue. Those velocities are commonly referred to as characteristic speeds. In this particular instance, the three eigenvalues correspond to the so-called three-wave solutions. For more complex hyperbolic conservation laws involving a larger number of variables, they will result in a greater number of degrees of freedom and, consequently, more distinctive characteristic speeds. In MHD problems, typically, the number of waves is 7.

3.2. Riemann problem

In the case of the Zeus scheme presented in Sec. 2, artificial viscosities smear shocks, which results in the absence of discontinuities. This implies that the problem is smooth, allowing the use of a large class of finite elements methods. However, in the case of the Godunov method, the opposite approach is taken. The method relies exclusively on the treatment of those discontinuities. Such discontinuities correspond to an initial value problem, which is known as the Riemann problem.

The Riemann problem consists of solving the hyperbolic conservation law (Eq. 2.50) for a discontinuity expressed as the following initial condition:

$$\mathbb{U}(x,0) = \begin{cases} \mathbb{U}^L, & x < 0 \\ \mathbb{U}^R, & x > 0 \end{cases}. \tag{2.52}$$

In practice, the Riemann problem is frequently solved in an approximate manner by employing a Riemann solver, whose objective is to provide or approximate the solution state $\mathbb{U}(x,t)$ at time t from that initial condition.

Furthermore, a possible generalization of the Riemann problem is one where the values of the derivatives up to an order of $\mathbb{U}$ on both sides of the discontinuity and possibly a source term are included. Such a problem is commonly referred to as the generalized Riemann problem (see Toro 2013, Chapter 19).

3.3. Cell averaging

In a finite element method, one seeks to discretize a problem into a set of values at given positions, where the values would ideally match the analytical solution at the given locations of the discretization points. In a finite volume method, space is also discretized as a finite set of cells. However, the so-called cell values represent instead an average of the values of a given field within the cell, instead of their value at a given location. Formally, for a cell corresponding to the subspace Ω_i whose volume is V_i the cell value $\mathbb{U}_i^t$ at time t is defined as

$$\mathbb{U}_i^t = \frac{1}{V_i} \iiint_{\Omega_i} \mathbb{U}(\mathbf{r},t)\mathrm{d}V, \tag{2.53}$$

where $\mathbb{U}(\mathbf{r},t)$ is the field that is discretized.

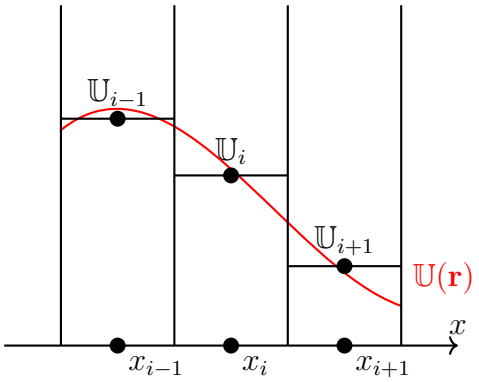


Figure 2.2: Illustration of the cell averaging procedure: the red curve is the real field, and the black horizontal line represents the averaged values on each cell.

Godunov's method's key insight is that such a discretized field is actually made of cells whose value within the cell is constant, resulting in discontinuities at the interfaces with other cells. Formally, the state of the discretized field in one dimension

at a given timestep is

$$\tilde{\mathbb{U}}(x,t) = \sum_i \Pi_{[x_{i-1/2},x_{i+1/2}]}(x)\,\mathbb{U}_i^t. \tag{2.54}$$

where Π is the Step function. Similar logic can be applied to three dimensions. In such a situation, there is a Riemann problem to be solved, at each interface between two cells. By assuming that over a duration Δt information from one interface cannot reach any other interfaces, every Riemann problem is independent. Therefore, for a cell i, solving the two Riemann problems $RP(\mathbb{U}_{i-1}^t, \mathbb{U}_i^t)$ and $RP(\mathbb{U}_i^t, \mathbb{U}_{i+1}^t)$ allows computing the state of the field $\tilde{\mathbb{U}}(x, t+\Delta t)$ whose initial condition is the discretized field $\tilde{\mathbb{U}}(x,t)$ Eq. 2.54. Lastly, by applying the averaging procedure to the obtained field in each cell, one can compute the new cell values

$$\mathbb{U}_i^{t+\Delta t} = \frac{1}{V}\iiint_{V_i} \tilde{\mathbb{U}}(x, t+\Delta t)\mathrm{d}V. \tag{2.55}$$

In practice, Godunov-type schemes can be described as cell updating based on the sum of incoming fluxes for all given fields. Formally,

$$\mathbb{U}_i^{t+\Delta t} = \mathbb{U}_i^t - \frac{\Delta t}{V_i}\left[\sum_{\text{cells } j} \mathcal{F}_{i\to j}\right], \tag{2.56}$$

where $\mathcal{F}_{i\to j}$ is the flux between a cell i and a cell j (rigorously also over a duration Δt), which is not null only if they share a common face, and $F_{i\to j}$ must be antisymmetric in order for the total quantity to be conserved. Aside from the time integration, the challenge of such method lies in computing this flux. In order for the fluxes in Eq. 2.55 to be equal to the result of the averaging procedure Eq. 2.56, one should select fluxes corresponding to each face such that they reproduce the averaging of the solutions of the Riemann problem. We therefore seek a flux function that corresponds to the averaged outcome of the Riemann problem. Specifically, we want a flux function that depends on the cell values on both sides of the Riemann problem

$$\mathcal{F}_{i\to j} = RS(\mathbb{U}_i^t, \mathbb{U}_j^t), \tag{2.57}$$

where RS is called a Riemann solver.

Mathematically, the Godunov method can be thought of as follows: we begin by integrating in time and space over a cell for a length of time Δt the hyperbolic conservation law. The initial conditions are the discontinuous approximation of the field shown in Eq. 2.54:

$$\text{Integrated law :} \quad \iiint_{V_i} \mathrm{d}V \int_t^{t+\Delta t} \mathrm{d}t\,[\partial_t \mathbb{U} + \boldsymbol{\nabla}\cdot\mathbf{F}(\mathbb{U})] = 0, \tag{2.58}$$

$$\text{Initial condition :} \quad \mathbb{U}(\mathbf{r}, t) = \sum_i \Pi(\mathbf{r}\in V_i)\mathbb{U}_i^t. \tag{2.59}$$

We can integrate the time derivative part with respect to time and space and use the Gauss theorem for the flux term. Hence,

$$\frac{\mathbb{U}_i^{t+\Delta t} - \mathbb{U}_i^t}{\Delta t} + \frac{1}{V_i} \oint_{\partial V_i} \mathrm{d}S \frac{1}{\Delta t} \int_t^{t+\Delta t} \mathrm{d}t \mathbf{n} \cdot \mathbf{F}(\mathbb{U}) = 0, \tag{2.60}$$

where the surface integral can be divided into a sum over the faces

$$\frac{\mathbb{U}_i^{t+\Delta t} - \mathbb{U}_i^t}{\Delta t} + \frac{1}{V_i} \left[\sum_{j,\text{common face}} \int_{V_i \cap V_j} \mathrm{d}S \frac{1}{\Delta t} \int_t^{t+\Delta t} \mathrm{d}t \mathbf{n} \cdot \mathbf{F}(\mathbb{U}) \right] = 0. \tag{2.61}$$

It is possible to express the Godunov scheme stated above by defining the fluxes in Eq. 2.56 as being

$$\mathcal{F}_{i \to j} = \int_{V_i \cap V_j} \mathrm{d}S \frac{1}{\Delta t} \int_t^{t+\Delta t} \mathrm{d}t \mathbf{n} \cdot \mathbf{F}(\mathbb{U}), \tag{2.62}$$

where we observe that the seeked flux function comes from a procedure of averaging, both in time and space. It is the actual quantity estimated by a Riemann solver. In practice, this means that the choice of scheme is solely based on the time integration and the choice of a Riemann solver. Such a numerical method is commonly referred to as a first-order Godunov method, as it was published in Godunov (1959). It is a first order scheme in both space and time. However, even if the resulting scheme is stable, it is very diffusive (Toro, 2013).

3.4. High order space reconstruction

This issue of over-diffusivity remained unsolved for a long time. It was van Leer (1979) that showed that it could be fixed by instead using a so-called high-order Godunov scheme, where the higher order refers to both space and time integration. The key insight was the realization that inside a cell, a piecewise linear reconstruction could be used to reconstruct a slope within the cell instead of a constant value to improve accuracy when interpolating the value to the faces. Instead of a plateau, each cell now has a slope that is reconstructed from neighboring cells. Therefore, the initial condition at the beginning of the time step is

$$\mathbb{U}(\mathbf{r}, t) = \sum_i \Pi(\mathbf{r} \in V_i) \left(\mathbb{U}_i^t + (\mathbf{r} - \mathbf{r}_i) \cdot (\boldsymbol{\nabla} \mathbb{U})_i^t \right). \tag{2.63}$$

Eq. 2.63 implies that on both edges of the cells, we have generalized Riemann problems instead of standard ones, rendering the direct utilization of the Riemann solver previously introduced incorrect as they only solve for the standard Riemann problem. To address this limitation, an alternative approach is to employ the MUSCL-Hancock method, wherein a predictor corrector scheme is used to estimate the integral Eq. 2.62 by calling twice the Riemann solver to correct the standard Riemann problem solution into its generalized case. It is important to note that, a Riemann

solver capable of exactly solving the Riemann problem Eq. 2.62 can be done in one dimension, down to floating point precision. However, when using a predictor corrector scheme, even with an exact Riemann solver, the estimation of Eq. 2.62 will always be approximate unless one directly solves the generalized Riemann problem.

However, the MUSCL-Hancock approach has the drawback of requiring two calls to the Riemann solver, which is already expensive. This resulted in the development of alternative methods to assess the outcomes of the slopes. For example, we will present the MUSCL-midpoint method (Toro, 2013, Sect. 14.4.2) that is used in the Godunov scheme of the code RAMSES (Teyssier, 2002). Firstly, the reconstruction is performed on the primitive variables instead of the conservative ones. As noted by Van Leer (2006), it is advisable to employ primitive variables during interpolation as it ensures that $E > \rho \mathbf{v}^2/2$ thereby preventing any negative pressures. Formally, in one dimension, in a cell i at time t with the primitive variables $\mathbb{W}_i^t$ and their numerically estimated gradient $\partial_x \mathbb{W}_i^t$, at the interface $i + 1/2$, the left and right values are:

$$\mathbb{W}_{i\to i+1/2}^{t+\Delta t/2} = \mathbb{W}_i + \left(\frac{\partial \mathbb{W}}{\partial t}\right)_i^t \frac{\Delta t}{2} + \left(\frac{\partial \mathbb{W}}{\partial x}\right)_i^t \frac{\Delta x}{2}, \tag{2.64}$$

$$\mathbb{W}_{i+1\to i+1/2}^{t+\Delta t/2} = \mathbb{W}_{i+1} + \left(\frac{\partial \mathbb{W}}{\partial t}\right)_{i+1}^t \frac{\Delta t}{2} - \left(\frac{\partial \mathbb{W}}{\partial x}\right)_{i+1}^t \frac{\Delta x}{2}, \tag{2.65}$$

in which, using the quasi-linear form of the Euler equation on the primitive variables

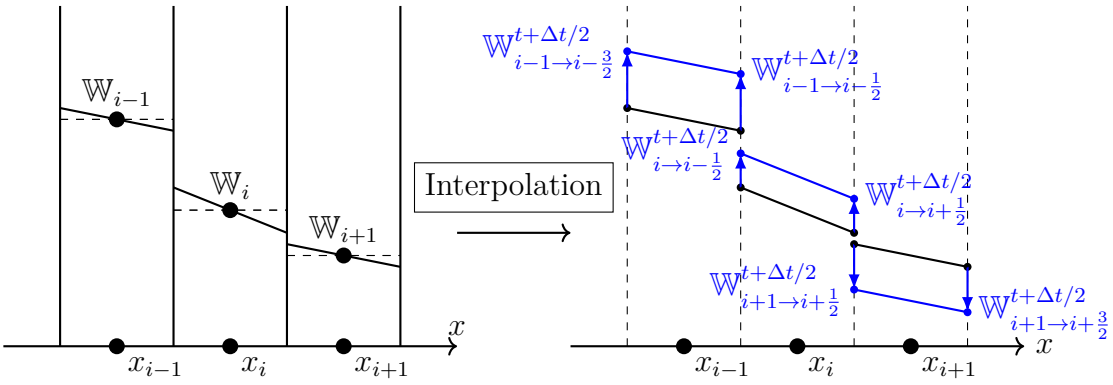


Figure 2.3: Illustration of the interpolation of values onto faces by utilizing the reconstructed slopes of cells. On the left, the cells and their respective values are depicted, along with the slope that corresponds to the estimated gradient. On the right, we demonstrate the process of first moving the slope by half a timestep and then interpolating the values using the slope to the faces. The complete procedure corresponds to Eq. 2.66 and Eq. 2.67. Here, the quantities corresponding to time t are in black, and the ones at time $t + \Delta t/2$ are in blue.

Eq. 2.51, one can rewrite those under the convenient form:

$$\mathbb{W}^{t+\Delta t/2}_{i\to i+1/2} = \mathbb{W}_i + \left(\mathbf{I}\frac{\Delta x}{2} - \mathbf{A}\frac{\Delta t}{2}\right)\left(\frac{\partial \mathbb{W}}{\partial x}\right)^t_i \tag{2.66}$$

$$\mathbb{W}^{t+\Delta t/2}_{i+1\to i+1/2} = \mathbb{W}_{i+1} + \left(-\mathbf{I}\frac{\Delta x}{2} - \mathbf{A}\frac{\Delta t}{2}\right)\left(\frac{\partial \mathbb{W}}{\partial x}\right)^t_{i+1} \tag{2.67}$$

Finally, the flux is estimated using the first-order Godunov method, but with the corrected values on the faces.

$$\mathcal{F}_{i\to i+1} = RS(\mathbb{W}^{t+\Delta t/2}_{i\to i+1/2}, \mathbb{W}^{t+\Delta t/2}_{i+1\to i+1/2}), \tag{2.68}$$

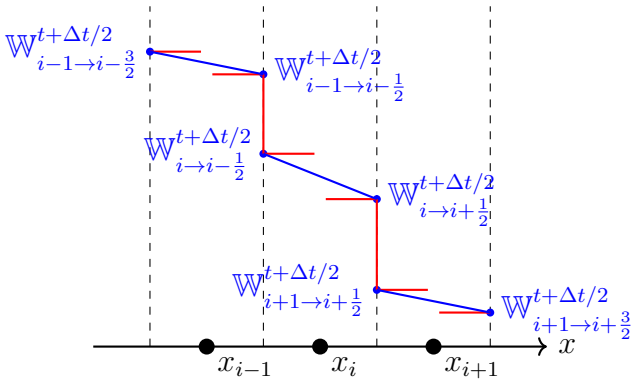


Figure 2.4: Representation of the corresponding Riemann problem that is solved in the MUSCL-midpoint method. Here, the figure corresponds to the situation at the end of the interpolation in Fig. 2.3, where the Riemann problems solved in practice are in red. From this figure, we see that the problem is a generalized Riemann problem at each interface, which is here approximated by a standard one. It should be noted that the discontinuity located in the middle of the cell is not accounted for and does not correspond to a Riemann problem that the model accounts for.

In practice, this procedure leads to a second-order scheme in both space and time. We did not find a proof that such an approximation of the generalized Riemann problem here by a standard one still yields the correct result. We believe that the explanation should rely on the following argument: the Cauchy-Kowalewski theorem allows for the expansion of the generalized Riemann into a standard Riemann problem and a perturbation of a higher order. When utilized in conjunction with the interpolated values in space and time, this correction is of a higher order than the solver's order, thereby allowing us to disregard the distinction between the generalized and the standard Riemann problem in this instance.

3.5. TVD slopes

In practice, the scheme will be of second-order when studying smooth flows, but it may also lead to additional spurious oscillations in the presence of discontinuities. The reason for this is the implementation of the slope reconstruction procedure. When we reconstruct the slopes, it is possible to create new maxima, as shown in Fig. 2.5. The property of not creating new extrema that would have led to spurious

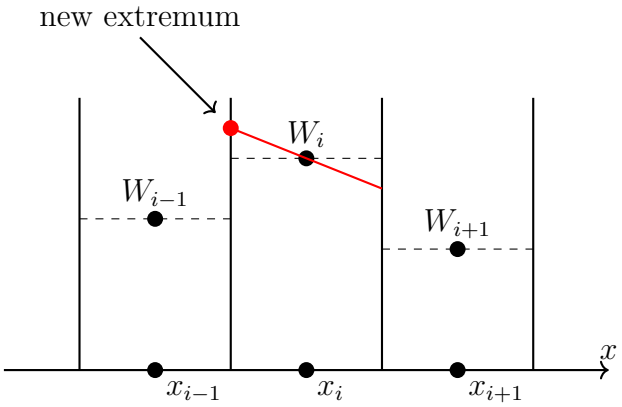


Figure 2.5: Example of a slope reconstruction leading to the creation of a new extremum.

oscillations is when the scheme is said to be TVD (Total Variation Diminishing). TVD schemes were initially developed in van Leer (1974) and the concept of the Total Variation Diminishing Scheme was introduced by Harten with its theorem stating that 'a Total Variation Diminishing (TVD) scheme is monotonicity preserving' (Harten, 1983). It is also possible to design schemes that are TVD graphically from the graph of Sweby (1984). To formalize this concept, we base our work on the work of Zou (2021).

Firstly, for a cell i and its neighbors $i-1$ and $i+1$, we can define the three differences:

$$\Delta_- = \mathbb{W}_i - \mathbb{W}_{i-1}, \tag{2.69}$$
$$\Delta_+ = \mathbb{W}_{i+1} - \mathbb{W}_i, \tag{2.70}$$
$$\Delta_t = \Delta_- + \Delta_+ \tag{2.71}$$
$$\tag{2.72}$$

and the related ratios are defined as

$$s_r = \frac{\Delta_t}{2\Delta x}, \tag{2.73}$$
$$f = \frac{\Delta_-}{\Delta_t}, \tag{2.74}$$
$$r = \frac{\Delta_-}{\Delta_+}, \tag{2.75}$$

where we assume that cells have a constant size Δx. In order for the reconstructed slopes to be TVD, we compute the new slope s of the cell such that

$$s = \phi(f) s_r = \phi\left(f = \frac{\Delta_-}{\Delta_t}\right) \frac{\Delta_t}{2dx},$$

where $\phi(f)$ is called a slope-limiting function. As presented in Zou (2021), being TVD corresponds to having the following properties on the slope limiting function $\phi(f)$

TVD rules

- If $\mathbb{W}_i$ is a local extremum, $-\infty < f < 0$ and $1 < f < +\infty$ the slope must be null to avoid creating new extrema, i.e.

$$\phi(f) = 0, f \in (-\infty, 0] \cup [1, +\infty)$$

- When $\mathbb{W}_i$ is in between $\mathbb{W}_{i+1}$ and $\mathbb{W}_{i-1}$, $0 \leq f \leq 1$, the slope must not create new extremas(as on Fig. 2.5). This condition on the new limited slope corresponds to $\phi(f) \leq 4f$ and $\phi(f) \leq 4(1-f)$ on the interval $[0, 1]$.

However, this condition while restoring the TVD property is not sufficient to ensure that the provided reconstruction results in a high-order Godunov scheme. For instance, a valid TVD slope limiting function is $\phi(f) = 0$. However, this does correspond to the absence of reconstruction, which is the first-order Godunov scheme. The TVD condition on slope limiters can be extended to a so-called high-order TVD limiter, which ensures that the resulting scheme is also high-order. This condition, graphically explained in Zou (2021), is that the reconstructed slope must lie in between the smallest of the following four slopes: current cell center to the edge of the neighbor cell, current cell center to the centers of the neighbor cell, on both neighbors. Formally, this corresponds, using the slope ratios to

High-order TVD rules

If $\min_2(a, b, c, d)$ gives the interval bound by the two smallest values of the four values a, b, c, d.

- If $\mathbb{W}_i$ is a local extremum, $-\infty < f < 0$ and $1 < f < +\infty$ the slope must be null to avoid creating new extrema, i.e.

$$\phi(f) = 0, f \in (-\infty, 0] \cup [1, +\infty)$$

- $\phi(f) \in \min_2(\{2f, 4f, 2(1-f), 4(1-f)\}), f \in (0, 1)$,

where $2f, 4f, 2(1-f), 4(1-f)$ are the aforementioned four slopes.

Graphically, the TVD rules can be summarized as shown in Fig. 2.6.

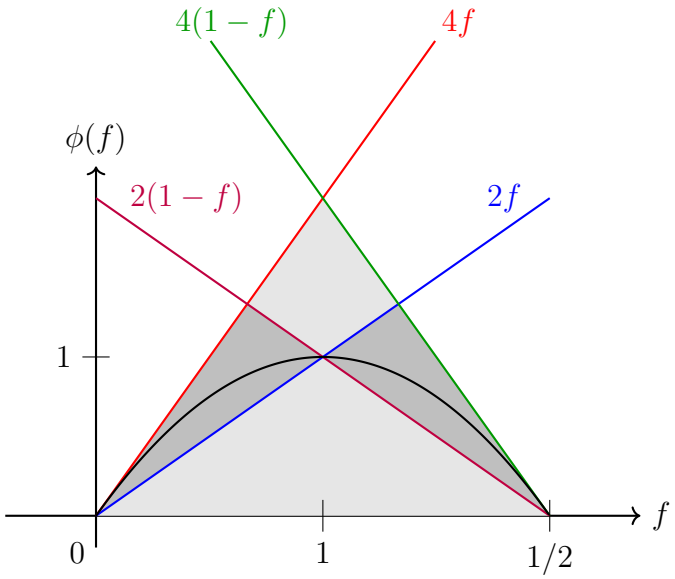


Figure 2.6: Illustration of the TVD regions: The light gray region is the TVD region, and the gray region is the high-order TVD region. The black curve depicted in this graph represents the Van Leer slope limiter.

In summary, using a slope limiter, the primitive value interpolation is now made as follows:

$$\left(\frac{\partial \mathbb{W}}{\partial x}\right)_i^t = \phi\left(f = \frac{\Delta_-}{\Delta_t}\right)\frac{\Delta_t}{2dx} \tag{2.76}$$

which, when developed, result in the same expression as used in ZEUS for the Van Leer slope limiter (Eq. 2.48).

3.6. Courant-Friedrichs-Lewy condition

The CFL condition for the Godunov scheme is analogous to that of the ZEUS scheme,

$$\Delta t \leq \min(\Delta x)/(|\mathbf{v}| + c_s), \tag{2.77}$$

where Δt is the timestep duration, Δx the grid cell size, $\mathbf{v}$ the fluid velocity vector, and c_s the sound speed. From the characteristic speed presented Sec. 3.1 we note that this corresponds to the condition that the largest characteristic speed must not propagate by more than a cell every timesteps.

3.7. Summary of the scheme

In summary, a second-order in space and time TVD Godunov scheme can be achieved using the steps described in this section. The steps are summarized in Alg. 1. As

Algorithm 1: Simplified pseudocode of a Godunov using MUSCL midpoint interpolation

Data: $\mathbb{U}_i^t$ the cell averaged values at time t,
Result: $\mathbb{U}_i^{t+\Delta t}$ the cell averaged values at time $t + \Delta t$.

foreach *cells* i **do**
  // Conservative to primitive
  $\mathbb{W}_i^t \leftarrow \texttt{constoprim}(\mathbb{U}_i^t)$;
  Compute slope limited gradients $\left(\frac{\partial \mathbb{W}}{\partial x}\right)_i^t$;
foreach *faces* $i \to j$ **do**
  interpolate quantity to the face at half timestep on both sides $\mathbb{W}_{i\to j}^{t+\Delta t/2}, \mathbb{W}_{j\to i}^{t+\Delta t/2}$;
  $\mathcal{F}_{i\to j} \leftarrow RS(\mathbb{W}_{i\to j}^{t+\Delta t/2}, \mathbb{W}_{j\to i}^{t+\Delta t/2})$;
foreach *cells* i **do**
  $\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \frac{\Delta t}{V_i}\left[\sum_{\text{faces } i\to j} \mathcal{F}_{i\to j}\right]$;

mentioned in the case of Zeus, Godunov schemes involve more operations, especially floating-point divisions, making them computationally expensive compared to simpler ones. However, such limitations as discussed are not as significant on modern architectures, since they are memory-bound rather than compute-bound. Godunov-type schemes also offer significant modularity, as the interpolation and Riemann solver are two distinct and relatively independent components of the scheme.

3.8. Extension to mesh refinement

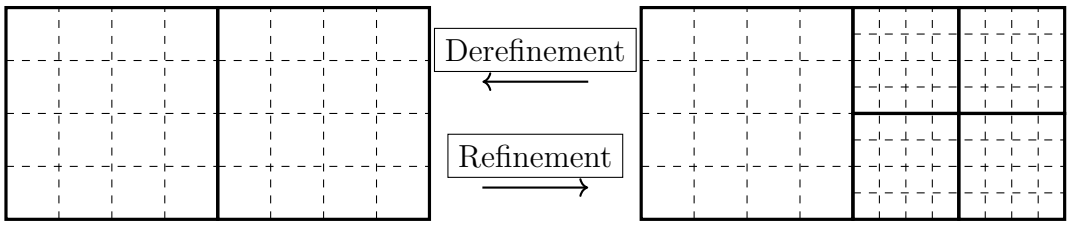


Figure 2.7: Example of a dynamic refinement procedure. In this case, an initially uniform grid is refined. As a result of the refinement, the number of neighbors per cell is subject to variation.

As mentioned in the introduction, we want to refine simulated problems around regions of interest. To do so, we use primitive blocks called AMR blocks, which correspond to a regular grid made of only a few cells forming a cube. Such blocks can be dynamically refined into smaller blocks under a criterion that can be specified by the user. This results in the following possible configurations shown in Fig. 2.7.

Here we still have a list of cells and faces. The difference compared to the regular grid case is that now the number of faces per cell can vary. The Godunov scheme can here be easily adapted in its basic form to such a change, as it is already described per cell and face. The possible changes then include the criteria to control grid refinement, interpolation, and slope limiting. A typical example of such a method of using an AMR grid in conjunction with a Godunov scheme is the numerical code RAMSES (Teyssier, 2002).

3.9. Discussion

Godunov-type schemes offer some significant advantages to simulate compressible astrophysical flows. Firstly, they naturally provide good accuracy when resolving shocks by design. Secondly, being written as a sum of fluxes over neighbors, they can naturally be rewritten as operations on a graph of cells, where a link between two cells corresponds to a common face. In such representation, fluxes are represented on the edges of the graph. This approach can naturally be transposed to massively parallel algorithms, thanks to the abstraction provided by this representation. Additionally, such representation makes the Godunov scheme similar algorithmically to the SPH scheme, allowing for the factorization of many common algorithms for both schemes. Thirdly, Godunov-type schemes have been found to be robust when extended to Adaptive Mesh Refinement methods. Lastly, higher-order methods such as discontinuous Galerkin schemes and ADER rely on different Riemann solvers, slope limiters, and basis functions. However, the structure of those schemes is very similar. This means that implementing a Godunov-type scheme is a good first step when aiming for the implementation of higher-order methods, as they share many algorithms in their implementations.

Godunov-type schemes are therefore suitable for many applications. However, they also have some significant drawbacks. Firstly, as mentioned in Sec. 2.10, the Godunov scheme is in general more expensive computationally than the Zeus scheme, while both schemes result in second-order accuracy in both space and time. Secondly, the CFL as for the Zeus scheme depends on the local velocity of the cell, meaning that for advection-dominated systems such as protoplanetary discs, the scheme will be limited by the velocity within the cells. This is not the case for Lagrangian methods, where the CFL depends instead on the accelerations or the local sound speed, reducing the CFL constraints in protoplanetary discs. This issue can, however, be mitigated by the use of the FARGO advection algorithm to relax some CFL constraints. Thirdly, it may not be appropriated for flows with complex geometry that evolve dynamically.

In general, we view the Godunov scheme as a mandatory scheme to implement in Shamrock for its versatility, modularity, expendability to higher-order methods, and AMR capabilities, as well as aligning with the state-of-the-art in the community.

4. Meshless (Smoothed particle hydrodynamics)

4.1. Simulated equations

Another approach widely employed in astrophysics is Smoothed Particle Hydrodynamics (SPH), initially introduced by Lucy (1977); Gingold & Monaghan (1977). In SPH, instead of relying on grid cells to simulate the flow, we instead use interpolation point that move with velocities that approximate the one of the fluid. This results in particles moving with the flow, and is therefore a Lagrangian method. Lagrangian methods are used for their capacity to handle complex geometries, adapt resolution to follow the mass, address free boundary conditions, and offer an alternative approach to grid-based methods for validating nonlinear solutions. In a Lagrangian form, the Euler equation Eq. 2.1 - Eq. 2.3 can be rewritten in a pseudo-Lagrangian form.

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\boldsymbol{\nabla}\cdot\mathbf{v}, \tag{2.78}$$

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\frac{1}{\rho}\boldsymbol{\nabla}P + \mathbf{f}, \tag{2.79}$$

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -\frac{P}{\rho}\boldsymbol{\nabla}\cdot\mathbf{v}, \tag{2.80}$$

where $\frac{\mathrm{d}}{\mathrm{d}t} = \partial_t + \mathbf{v}\cdot\boldsymbol{\nabla}$ is called a convective derivative, ρ, $\mathbf{v}$, P, and u denote the density, the velocity, the pressure, and the specific internal energy of the fluid, respectively.

4.2. SPH density interpolation

The SPH method is based on the assumption that the density estimates for each particle (a) are obtained using density interpolation.

Definition ▶ SPH density estimate

The density estimate ρ_a of an SPH particle a is

$$\rho_a = \sum_b m_b W_{ab}(h_a), \tag{2.81}$$

$$W_{ab}(h_a) = W(\mathbf{r}_a - \mathbf{r}_b, h_a), \tag{2.82}$$

where m_b is the mass of the particle b, h_a its smoothing length, $W_{ab}(h_a)$ the SPH kernel.

This definition expresses that the density of a SPH particle is a weighted sum of the masses of its neighbors. The interpolation kernel W is a bell-shaped function that weakly converges towards a delta Dirac distribution when the smoothing length h goes to zero. It must satisfy multiple properties.

1. W must have a norm of unity :
$$\iiint W(\mathbf{r}, h)\mathrm{d}V = 1, \quad \forall h > 0, \tag{2.83}$$
2. W is spherically symmetric:
$$W(\mathbf{r}, h) \equiv W(|\mathbf{r}|, h), \tag{2.84}$$
3. W is positive and monotonically decreasing from its center,
4. $(\boldsymbol{\nabla}_{\mathbf{r}} W)(0, h) = 0$ to avoid instabilities when two particles are too close.

For convenience, the SPH kernel W is defined through a SPH kernel generator function f such that, in three dimensions:
$$W_{ab}(h_a) = W(\mathbf{r}_a - \mathbf{r}_b, h_a) = W(|\mathbf{r}_a - \mathbf{r}_b|, h_a) = \frac{C_{\text{norm}}}{h_a^3} f\left(\frac{|\mathbf{r}_a - \mathbf{r}_b|}{h_a}\right), \tag{2.85}$$
In practice, the kernel generator f and consequently the kernel W are taken to have compact support. Although Gaussian kernels are excellent for SPH, they would be too costly for large simulations, motivating the choice of a compactly supported function to ensure computational efficiency (see Morris (1996); Price (2012); Dehnen & Aly (2012) for details). When taken to be with compact support, we define the kernel generator radius $R_{\text{kern},f}$, which is the radius of the compact support of f. A function f having a kernel generator radius of $R_{\text{kern},f}$ implies that the compact support radius of $W(\cdot, h)$ is $R_{\text{kern},f}h$.

Modern SPH codes tend to mostly use the Schoenberg (1946) B-Spline (Price, 2012) functions defined as being
$$f_{M_n}(q, h) = \frac{1}{2\pi}\int_{-\infty}^{\infty} [\text{sinc}(kh/2)]^n \cos(kq)\mathrm{d}k, \tag{2.86}$$
where $\text{sinc}(x) = \sin(x)/x$ is the sinus cardinal function. They correspond to a gate function convoluted with itself n times, hence ensuring compact support in real space.

The two main SPH kernel generators stand out as being the most used in practice. The first one is the cubic spline.
$$f_{M_4}(q) = \begin{cases} \frac{1}{4}(2-q)^3 - (1-q)^3, & 0 \le q \le 1 \\ \frac{1}{4}(2-q)^3, & 1 \le q \le 2 \\ 0, & \text{otherwise} \end{cases} \tag{2.87}$$
which has a kernel generator radius $R_{kern,M_4} = 2$ and a 3d norm $C_{\text{norm},M_4,3d} = 1/\pi$ and the quintic spline
$$f_{M_6}(q) = \begin{cases} (3-q)^5 - 6(2-q)^5 + 15(1-q)^5, & 0 \le q \le 1 \\ (3-q)^5 - 6(2-q)^5, & 1 \le q \le 2 \\ (3-q)^5, & 2 \le q \le 3 \\ 0, & \text{otherwise} \end{cases} \tag{2.88}$$

which has a kernel generator radius $R_{kern,M_6} = 3$ and a 3d norm $C_{\text{norm},M_6,3d} = 1/120\pi$. The M_4 is convenient since it can provide a correct result while having a small, compact support and computational cost, while the M_6 is more precise but also more costly both in size and computation. Some examples of typical kernels are shown in Fig. 2.8.

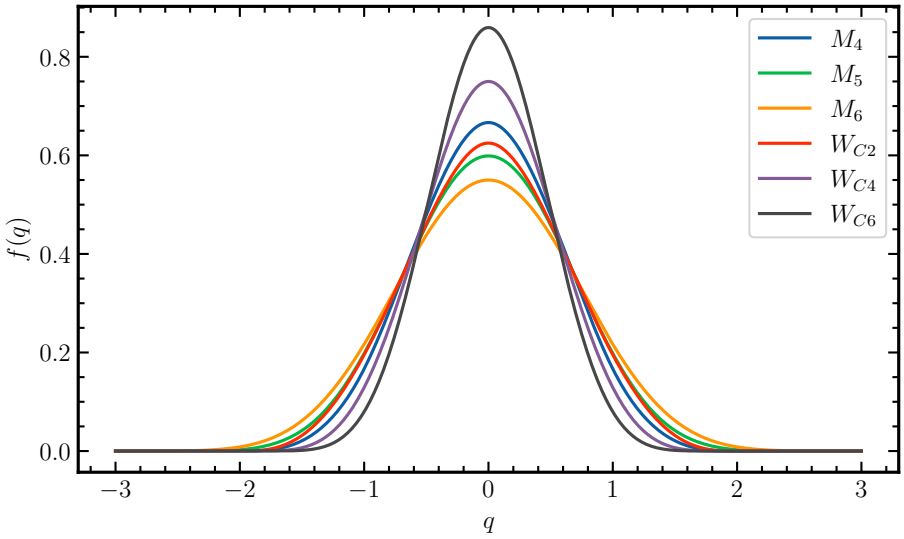


Figure 2.8: Example of some of the most used SPH kernels generators.

4.3. Field interpolation in SPH

In SPH, as stated, we only have discrete particles, which are defined only by their positions, smoothing lengths, and masses. The weight when performing the SPH interpolation is the particle mass. This means that in order to interpolate a field, one must use the masses as weight and divide by the density estimate to keep the equation homogeneous. In practice

$$\langle A\rangle_a = \sum_b m_b \frac{A_b}{\rho_b} W_{ab}(h_a) \tag{2.89}$$

$$\simeq_{ctn} \int \rho(\mathbf{r}')\mathrm{d}V \ \frac{A(\mathbf{r}')}{\rho(\mathbf{r}')} W(\mathbf{r}-\mathbf{r}', h(\mathbf{r})), \tag{2.90}$$

which, in the continuous limit, would be equivalent to a weighted integral of the masses. We will discuss this limit later.

As detailed in Price (2012), the derivatives can be estimated as such by deriving the SPH kernels:

$$\langle \boldsymbol{\nabla} A\rangle_a = \sum_b m_b \frac{A_b}{\rho_b} \boldsymbol{\nabla}_a W_{ab}(h_a). \tag{2.91}$$

However, many possible forms of the derivatives are possible, which yield the same continuous limit but have very different properties. For example, the symmetric form is:

$$\langle \boldsymbol{\nabla} A \rangle_a = \sum_b m_b \frac{A_b - A_a}{\rho_b} \boldsymbol{\nabla}_a W_{ab}(h_a) \tag{2.92}$$

and the antisymmetric form:

$$\langle \boldsymbol{\nabla} A \rangle_a = \sum_b m_b \rho_a \left(\frac{A_a}{\rho_a^2} + \frac{A_b}{\rho_b^2} \right) \boldsymbol{\nabla}_a W_{ab}(h_a). \tag{2.93}$$

Both yield the same continuous limit, but one is symmetric under permutation of particles and the other is antisymmetric. The form of the derivative can change the properties of the equations of motion.

4.4. Equation of motion

Equations of motion for the SPH particles can be derived from a Lagrangian (e.g. Monaghan & Price 2001; Price 2012)

$$L = \sum_b m_b \left[\frac{1}{2} \mathbf{v}_b^2 - u_b(\rho_b, s_b) \right], \tag{2.94}$$

giving a parallel to its continuous counterpart (Seliger & Whitham, 1968)

$$\mathcal{L} = \int \rho_0 \left[\frac{1}{2} \left(\frac{\partial \mathbf{x}_i}{\partial t} \right)^2 - u \right] \mathrm{d}\boldsymbol{\alpha}. \tag{2.95}$$

From a variational principle, one obtains

$$\frac{\mathrm{d}\mathbf{x}_a}{\mathrm{d}t} = \mathbf{v}_a, \tag{2.96}$$

$$\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} = -\sum_b m_b \left(\frac{P_a}{\rho_a^2 \Omega_a} \nabla_a W_{ab}(h_a) + \frac{P_b}{\rho_b^2 \Omega_b} \nabla_a W_{ab}(h_b) \right). \tag{2.97}$$

$$\Omega_a = 1 - \frac{\partial h_a}{\partial \rho_a} \sum_b m_b \frac{\partial W_{ab}(h_a)}{\partial h_a}, \tag{2.98}$$

The variational method guarantees that Eq. 2.96 – 2.97 preserve total linear momentum, angular momentum, and energy conservation up to machine precision. The term Ω_a arises from the fact that h_a is chosen in practice to be a function of the density, and as such, depends on the positions of the particles (Monaghan, 2002; Springel & Hernquist, 2002).

For the internal energy equation, Eq. 2.81 provides an estimate of the local expansion rate of an elementary volume of the fluid $\mathrm{d}V$

$$\frac{\mathrm{d}}{\mathrm{d}t} \mathrm{d}V = (\nabla \cdot \mathbf{v})\, \mathrm{d}V, \tag{2.99}$$

without defining volumes explicitly. Indeed, taking the derivative of Eq. 2.81 with respect to time yields

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{m_a}{\rho_a}\right) = -\frac{1}{\rho_a}\frac{\mathrm{d}\rho_a}{\mathrm{d}t}\left(\frac{m_a}{\rho_a}\right). \tag{2.100}$$

The evolution of internal energy Eq. 2.80 is subsequently calculated according to

$$\frac{\mathrm{d}u_a}{\mathrm{d}t} = \frac{P_a}{\rho_a^2}\frac{\mathrm{d}\rho_a}{\mathrm{d}t} = \frac{P_a}{\rho_a^2\Omega_a}\sum_b m_b \mathbf{v}_{ab}\cdot\nabla_a W_{ab}(h_a), \tag{2.101}$$

where $\mathbf{v}_{ab} \equiv \mathbf{v}_a - \mathbf{v}_b$, and the pressure P_a is related to the density ρ_a and other variables through the equation of state.

4.5. Conserved quantities

4.5.1. Momentum

The total momentum in SPH can be written as being

$$\mathbf{P} = \sum_a m_a \mathbf{v}_a. \tag{2.102}$$

Deriving Eq. 2.102 with respect to time and using the balance of forces Eq. 2.97, one obtains

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{P} &= \sum_a m_a \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_a \\ &= -\sum_a\sum_b m_a m_b \underbrace{\left(\frac{P_a}{\rho_a^2\Omega_a}\nabla_a W_{ab}(h_a) + \frac{P_b}{\rho_b^2\Omega_b}\nabla_a W_{ab}(h_b)\right)}_{\text{antisymetric}} \\ &= 0.\end{aligned}$$

Since the time derivative of the velocity is the sum of an antisymmetric SPH operator, this results in the conservation of the total momentum. This would not be the case if a symmetric form of the operator was chosen instead, as mentioned with Eq. 2.93. Additionally, when derived from a Lagrangian, which is translation-invariant, the form must be antisymmetric as the momentum is conserved due to Noether's theorem.

4.5.2. Total energy

The total energy in SPH can be written as being

$$E = \sum_a m_a\left(\frac{1}{2}\mathbf{v}_a\cdot\mathbf{v}_a + u_a\right). \tag{2.103}$$

Its time derivative is

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}E &= \sum_a m_a \left(\mathbf{v}_a \cdot \frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} + \frac{\mathrm{d}u_a}{\mathrm{d}t} \right) \\
&= -\sum_a \sum_b m_a m_b \left(\frac{P_a \mathbf{v}_a}{\rho_a^2 \Omega_a} \cdot \nabla_a W_{ab}(h_a) + \frac{P_b \mathbf{v}_a}{\rho_b^2 \Omega_b} \cdot \nabla_a W_{ab}(h_b) \right) \\
&\quad + \sum_a m_a \frac{P_a}{\rho_a^2 \Omega_a} \sum_b m_b (\mathbf{v}_a - \mathbf{v}_b) \cdot \nabla_a W_{ab}(h_a) \\
&= -\sum_a \sum_b m_a m_b \underbrace{\left(\frac{P_a \mathbf{v}_b}{\rho_a^2 \Omega_a} \cdot \nabla_a W_{ab}(h_a) + \frac{P_b \mathbf{v}_a}{\rho_b^2 \Omega_b} \cdot \nabla_a W_{ab}(h_b) \right)}_{\text{antisymetric}} \\
&= 0.
\end{aligned}$$

This means that the time derivative of the total energy is always null. However, due to the leapfrog integration, variations of the total energy can be measured in simulations by an order of $\mathcal{O}(\Delta^2)$ since the time integrator is symplectic. Formally, the scheme conserves exactly the energy corresponding to an effective Hamiltonian whose difference with the simulated one is of order $\mathcal{O}(\Delta t^2)$.

4.5.3. Angular momentum

Another important conserved quantity in the context of protoplanetary discs is the conservation of angular momentum. In SPH it can be written as being

$$\mathbf{L} = \sum_a m_a \mathbf{r}_a \times \mathbf{v}_a. \tag{2.104}$$

Its time derivative is

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{L} &= \sum_a m_a \left(\mathbf{r}_a \times \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_a \right) \\
&= -\sum_a \sum_b m_a m_b \left(\frac{P_a}{\rho_a^2 \Omega_a} \mathbf{r}_a \times \nabla_a W_{ab}(h_a) + \frac{P_b}{\rho_b^2 \Omega_b} \mathbf{r}_a \times \nabla_a W_{ab}(h_b) \right) \\
&= -\sum_a \sum_b m_a m_b \left(\frac{P_a}{\rho_a^2 \Omega_a} \mathbf{r}_a \times \nabla_a W_{ab}(h_a) - \frac{P_a}{\rho_a^2 \Omega_a} \mathbf{r}_b \times \nabla_a W_{ab}(h_a) \right) \\
&= -\sum_a \sum_b m_a m_b \left(\frac{P_a}{\rho_a^2 \Omega_a} \underbrace{(\mathbf{r}_a - \mathbf{r}_b) \times \nabla_a W_{ab}(h_a)}_{=0} \right) \\
&= 0.
\end{aligned}$$

The angular momentum is always conserved as the force is always antisymmetric and along the axis between the two particles, resulting in the sum of the angular momentum derivative being null.

In summary, as SPH derives from a Lagrangian, it also conserves the corresponding Noether's invariants. When extending this basic version of SPH, special care is required to ensure that no invariants are broken. In general, in SHAMROCK or PHANTOM, the conservation of those quantities is checked at all timesteps to be within the floating point errors (around $10^{-15} - 10^{-17}$) for the momentum, angular momentum, and $\frac{\mathrm{d}}{\mathrm{d}t}E$. However, since the time integrator is a symplectic one, the value of the total energy E is not conserved exactly. In a Leapfrog scheme, it is constant at order $\mathcal{O}(\Delta t^2)$. This deviation from the initial value can be mitigated by lowering the CFL criterion for SPH.

4.6. Artificial viscosity

Similarly to the approach used in the ZEUS scheme, in SPH, since we represented smoothed fields through kernel interpolation, this means that we do not have a means to represent the Riemann problem in traditional SPH. This was the starting point for the development of Godunov SPH methods (Inutsuka, 2002) to reintroduce the Riemann problem and associated Riemann solvers into SPH. However, to date, the results are either on par or worse than traditional SPH (Price et al., 2018). In traditional SPH, however, to address this issue, a Von Neumann shock viscosity with linear and quadratic terms (Von Neumann & Richtmyer, 1950; Landshoff, 1955; Margolin & Lloyd-Ronning, 2022) is employed for velocities to circumvent this limitation. In the SPH solver, we use the shock-capturing terms from Price & Federrath (2010); Lodato & Price (2010) modified from the original formation of Monaghan (1997a). The extended equation of motion becomes

$$\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} = \sum_b m_b \left(\frac{P_a + q^a_{ab}}{\rho_a^2 \Omega_a} \nabla_a W_{ab}(h_a) + \frac{P_b + q^b_{ab}}{\rho_b^2 \Omega_b} \nabla_a W_{ab}(h_b) \right), \tag{2.105}$$

where

$$q^a_{ab} = \begin{cases} -\frac{1}{2}\rho_a v_{\mathrm{sig},a} \boldsymbol{v}_{ab} \cdot \hat{\boldsymbol{r}}_{ab}, & \boldsymbol{v}_{ab} \cdot \hat{\boldsymbol{r}}_{ab} < 0 \\ 0 & \text{otherwise,} \end{cases}, \tag{2.106}$$

$$v_{\mathrm{sig},a} = \alpha^{\mathrm{AV}}_a c_{s,a} + \beta |\mathbf{v}_{ab} \cdot \hat{\boldsymbol{r}}_{ab}| \ , \ \alpha^{\mathrm{AV}}_a \in [0,1]. \tag{2.107}$$

To properly capture energy discontinuities, a shock conductivity (also known as artificial conductivity) is employed for the internal energy (e.g. Noh 1987; Margolin & Lloyd-Ronning 2022). Eq. 2.101 extends to (e.g. Chow & Monaghan 1997; Price 2012, 2008)

$$\frac{\mathrm{d}u_a}{\mathrm{d}t} = \frac{P_a + q^a_{ab}}{\rho_a^2 \Omega_a} \sum_b m_b \mathbf{v}_{ab} \cdot \nabla_a W_{ab}(h_a) + \Lambda_{\mathrm{cond}}, \tag{2.108}$$

where

$$\Lambda_{\rm cond} = \sum_b m_b \beta_u v^u_{\rm sig}(u_a - u_b)\frac{1}{2}\left[\frac{F_{ab}(h_a)}{\Omega_a\rho_a} + \frac{F_{ab}(h_b)}{\Omega_b\rho_b}\right], \tag{2.109}$$

$$v^u_{\rm sig} = \sqrt{\frac{|P_a - P_b|}{(\rho_a + \rho_b)/2}}, \tag{2.110}$$

using $F_{ab}(h_a) = \hat{\mathbf{r}}_{ab}\cdot\nabla_a W_{ab}(h_a)$. We use the symbol β_u to represent the shock conductivity parameter instead of the conventional α_u. This change clarifies that α_u is related to the quadratic part of the artificial viscosity $\beta|\mathbf{v}_{ab}\cdot\hat{\boldsymbol{r}}_{ab}|$, rather than the linear part $\alpha_a^{\rm AV}c_{{\rm s},a}$ (Von Neumann & Richtmyer, 1950; Noh, 1987; Margolin & Lloyd-Ronning, 2022). Writing Eq. 2.105&2.108 with a shock viscosity expressed as a modified pressure ensures consistent application of the corresponding terms in both the velocity and energy equations.

We note that the chosen form for the linear plus quadratic artificial viscosity is similar to the one used in Zeus, which is not a tensor; this can lead to issues potentially in shear flow (private communication with E. Lynch). This motivates a potential study on the feasibility of reusing the tensorial artificial viscosity of Zeus (Stone & Norman, 1992b) in SPH.

4.7. Shock detection

To provide shock detection in order to enable shock viscosity only in regions of interest, we use the method from Cullen & Dehnen (2010) that was implemented in Phantom (Price et al., 2018), which is an improved version of the Morris & Monaghan (1997) switch (see Price 2008, 2012). The value of the shock viscosity parameter α_a is evolved using

$$\frac{{\rm d}\alpha_a}{{\rm d}t} = -\frac{(\alpha_a - \alpha_{{\rm loc},a})}{\tau_a}. \tag{2.111}$$

The targeted value of the shock viscosity parameter $\alpha_{{\rm loc},a}$ is defined using

$$\alpha_{{\rm loc},a} \equiv \min\left(10A_a\frac{h_a^2}{c^2_{{\rm s},a}}, \alpha_{\rm max}\right), \tag{2.112}$$

where

$$A_a \equiv \xi_a \max\left[-\frac{\rm d}{{\rm d}t}\left(\nabla\cdot\boldsymbol{v}_a\right), 0\right], \tag{2.113}$$

is the shock indicator, and ξ_a is the corrective factor (Balsara, 1995)

$$\xi \equiv \frac{|\nabla\cdot\boldsymbol{v}|^2}{|\nabla\cdot\boldsymbol{v}|^2 + |\nabla\times\boldsymbol{v}|^2}. \tag{2.114}$$

The rising time $\tau_a \equiv h_a/(c_{s,a}\sigma_d)$ is parameterized by the decay parameter $\sigma_d = 0.1$, a typical value for practical cases. In practice, $\alpha_a(t)$ is set directly to $\alpha_{\mathrm{loc},a}$ if $\alpha_{\mathrm{loc},a} > \alpha_a(t)$. Similarly to the approach used in PHANTOM (Cullen & Dehnen, 2010), we use SPH derivatives that are exact to the linear order to compute

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\nabla \cdot \boldsymbol{v}_a\right) = \sum_i \frac{\partial a_a^i}{\partial x_a^i} - \sum_{i,j} \frac{\partial v_a^i}{\partial x_a^j}\frac{\partial v_a^j}{\partial x_a^i}, \tag{2.115}$$

where for a given field ϕ, this accurate SPH derivative is

$$R_a^{ij}\frac{\partial \phi_a^k}{\partial x_a^j} = \sum_b m_b \left(\phi_b^k - \phi_a^k\right)\frac{\partial W_{ab}\left(h_a\right)}{\partial x^i}, \tag{2.116}$$

where,

$$R_a^{ij} = \sum_b m_b \left(x_b^i - x_a^i\right)\frac{\partial W_{ab}\left(h_a\right)}{\partial x^j} \approx \delta^{ij}. \tag{2.117}$$

Inverting R_a^{ij} and applying it to Eq. 2.116 provides the desired derivative.

4.8. Adaptive smoothing length

In astrophysics, a typical choice consists of choosing h_a in a way that the resolution follows the density

$$\rho(h) = m\left(\frac{h_{\mathrm{fact}}}{h}\right)^3, \tag{2.118}$$

where h_{fact} is a tabulated dimensionless constant that depends on the kernel (e.g. $h_{\mathrm{fact}} = 1.2$ for the M_4 cubic kernel). This specific form also implies that the averaged number of neighbors within the compact support of a given SPH particle is roughly constant throughout the simulation. Eq. 2.118 must itself be consistent with the definition of density Eq.5.2, since h depends on ρ and vice versa. Achieving this requires density and smoothing length to be calculated simultaneously by minimizing the function.

$$\delta\rho = \rho_a - \rho(h_a). \tag{2.119}$$

This approach allows an accurate use of $\rho(h_a)$ in the algorithms rather than calculating the SPH sum. In practice, the iterative procedure is conducted with a Newton-Raphson algorithm. This will be covered in more detail in the SPH implementation in SHAMROCK.

4.9. Time stepping

4.9.1. Leapfrog integration

By construction, standard SPH is conservative and achieves second-order accuracy in space in smooth flows. To ensure consistency, time integration is performed using

symplectic second-order leapfrog integrator (in the same form as used in Price et al. (2018)), or 'Kick-drift-kick' (e.g. Verlet 1967; Hairer et al. 2003):

$$\boldsymbol{v}^{n+\frac{1}{2}} = \boldsymbol{v}^{n} + \frac{1}{2}\Delta t \boldsymbol{a}^{n}, \tag{2.120}$$

$$\boldsymbol{r}^{n+1} = \boldsymbol{r}^{n} + \Delta t \boldsymbol{v}^{n+\frac{1}{2}}, \tag{2.121}$$

$$\boldsymbol{v}^{*} = \boldsymbol{v}^{n+\frac{1}{2}} + \frac{1}{2}\Delta t \boldsymbol{a}^{n}, \tag{2.122}$$

$$\boldsymbol{a}^{n+1} = \boldsymbol{a}\left(\boldsymbol{r}^{n+1}, \boldsymbol{v}^{*}\right), \tag{2.123}$$

$$\boldsymbol{v}^{n+1} = \boldsymbol{v}^{*} + \frac{1}{2}\Delta t \left[\boldsymbol{a}^{n+1} - \boldsymbol{a}^{n}\right], \tag{2.124}$$

where $\mathbf{r}^n$, $\mathbf{v}^n$ and $\mathbf{a}^n$ denote positions, velocities and acceleration at the $n^{-\text{th}}$ time step Δt. In the scheme presented in Price et al. 2018, a combined iteration is used to calculate the acceleration $\boldsymbol{a}^{n+1}$ and update the smoothing length at the same time. After the derivative update a corrector step is applied to the velocity, and its results is used as a reference to check that the resulting solution is reversible over time. The correction applied at the end of the leapfrog scheme is as follows

$$\Delta \mathbf{v}_i = \frac{1}{2}\Delta t \left[\boldsymbol{a}_i^{n+1} - \boldsymbol{a}_i^{n}\right]. \tag{2.125}$$

We use the result of Eq. 2.125 to verify that the maximum correction does not exceed a fraction ϵ_{v} of the mean square correction

$$\max_i \left(|\Delta \mathbf{v}_i| / \sqrt{\frac{1}{N}\sum_j |\Delta \mathbf{v}_j|^2|} \right) < \varepsilon_{\text{v}}. \tag{2.126}$$

If any particles fail to meet this criterion, we recalculate the acceleration and apply the correction step again with $\boldsymbol{v}^{*} \leftarrow \boldsymbol{v}^{n+1}$ instead.

Alternative options for the time integrations include Runge-Kutta type schemes, which are not symplectic, thus not ensuring conservation of energy, and individual time-stepping schemes, which will not be discussed here.

4.9.2. Choice of the timestep

One of the drawbacks of the previous methods is that they feature a CFL dependent on the local fluid velocity. This can be avoided in a lagrangian fluid since the method is invariant under the Galilean transform. The CFL does not depend on the local velocity but rather on the local signal velocity and local acceleration. The value of the explicit time step is governed by the Courant-Friedrich-Levy stability condition (Courant et al., 1928). Following Price et al. (2018) from Lattanzio et al. (1986); Monaghan (1997b),

$$\Delta t \equiv \min(C_{\text{cour}} \frac{h_a}{v_{\text{sig},a}^{\text{dt}}}, C_{\text{force}} \sqrt{\frac{h_a}{|\boldsymbol{a}_a|}}). \tag{2.127}$$

The first term allows for the correct capture of the propagation of the hydrodynamic characteristic waves in the fluid at a given resolution. Similarly, the second term ensures correct treatment of the action of external forces on the fluid. The safety coefficients are set to the following values: $C_{\rm cour} = 0.3$ and $C_{\rm force} = 0.25$.

4.10. SPH dispersion relation

To understand basic properties of SPH,we study the dispersion relation of SPH using a simple SPH model. We present here the case of a soundwave response of a simpler SPH model for the gas (details are presented in an novel extended version for dusty mixture in App.B). To study the dispersion relation of SPH models, let us use the simpler model:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}_a = \mathbf{v}_a \tag{2.128}$$

$$\rho_a = \sum_b m_b W_{ab}(h_a) \tag{2.129}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_a = -\mathcal{P}_a \tag{2.130}$$

where the pressure force is in the following form:

$$\mathcal{P}_a \equiv c_s^2 \sum_b m_b \left(\frac{1}{\rho_a} + \frac{1}{\rho_b}\right) \frac{\partial W_{ab}}{\partial x} \tag{2.131}$$

We then perturb linearly Eq. 2.128–2.129 using the following expressions:

$$a_a = a_{a,0} + \delta a_a \tag{2.132}$$

$$a_i = a_{i,0} + \delta a_i \tag{2.133}$$

$$\delta a_a = \tilde{a}_g e^{i(k x_{a,0} - \omega t)} \tag{2.134}$$

$$\delta a_i = \tilde{a}_d e^{i(k x_{i,0} - \omega t)} \tag{2.135}$$

for $a = (\rho, x, v)$ We note that under the following notation the velocity perturbation is related by a factor $-i\omega$ to the perturbation on the particule position, $\tilde{v}_{g/d} = -i\omega\tilde{x}_{g/d}$.

We introduce the SPH continuous limit as taking the number of particles to infinity, such that they form a continuous distribution. This yields the following dictionary to convert SPH sums into integrals, shown in Tabl. 2.1. The limit of SPH to the continuum equations is performed by first taking the continuous limit, followed by replacing the SPH kernels by Dirac functions. Without providing details, we figured out during this Ph.D. thesis that the procedure to take the continuous limit in SPH deserved further study, as it may lead to inconsistencies. This results in the pressure operator being at first order and using the continuous limit of SPH

$$\mathcal{P} = e^{i(k x_{a,0} - \omega t)} c_s^2 k^2 \tilde{x}_g \left[2\hat{W} - \hat{W}^2\right], \tag{2.136}$$

discrete	continuous
$1^{(W)}_{\mathrm{SPH}} = \frac{1}{\rho}\sum_a m_a W(x_a - b_b, h)$	1
$0^{(W)}_{\mathrm{SPH}} = \frac{1}{\rho}\sum_a m_a \frac{\mathrm{d}W}{\mathrm{d}x}(x_a - b_b, h)$	0
$\sum_a m_a f(x_a)$	$\rho \int f(x)\mathrm{d}x$

Table 2.1: Dictionary of the continuous limit of SPH.

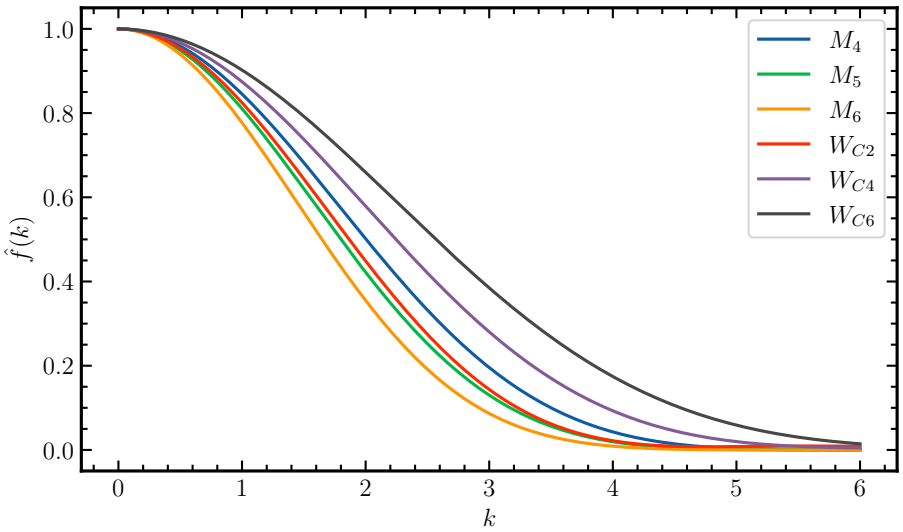


Figure 2.9: Fourier transform of the SPH kernel generators shown in Fig. 2.8.

where $\hat{W}$ is the Fourier transform of the SPH kernel.

Performing the same perturbation on the equation of motion yields

$$0 = -i\omega\tilde{\rho} + ik\rho\hat{W}\tilde{v}, \tag{2.137}$$

$$0 = -i\omega\tilde{v} + \frac{ikc_s^2}{\rho}\left(2 - \hat{W}\right)\tilde{\rho}, \tag{2.138}$$

which gives the following dispersion relation for the SPH soundwave:

$$0 = \omega^2 - c_s^2 k^2 \left[2\hat{W} - \hat{W}^2\right], \tag{2.139}$$

which corresponds to the standard soundwave dispersion relation for a modified wave vector $\tilde{k}^2 = k^2\left[2\hat{W} - \hat{W}^2\right]$. This means that the closer $\hat{W}$ is to unity, the more precise the scheme with regard to the soundwave. In that regard, Wendland kernels are, here, the best choice. However, as mentioned, they are significantly more expensive computationally. Here, M_6 offers one of the best tradeoffs between precision and computational cost. The dispersion relation of the SPH soundwave is shown in Fig. 2.10.

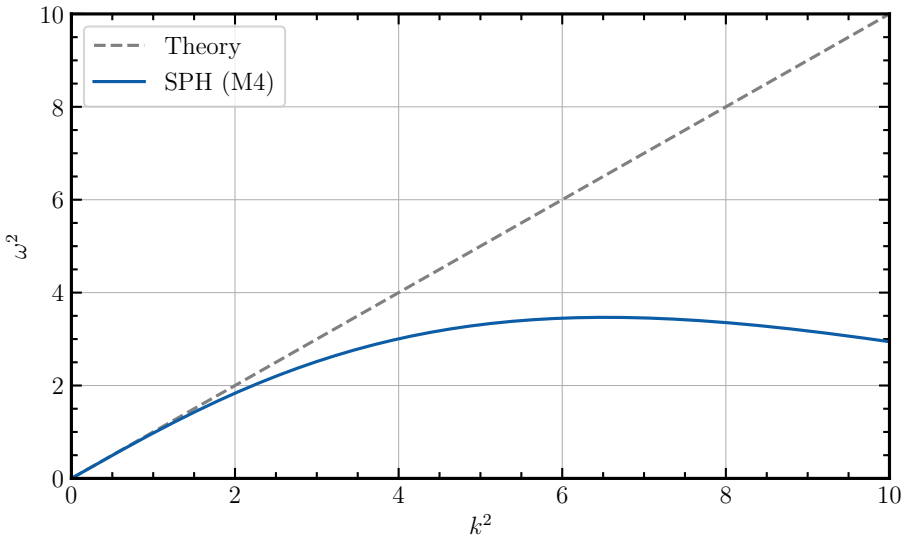


Figure 2.10: SPH soundwave dispersion relation with the M4 kernel, $h = 1$, and $c_s = 1$.

As shown in Fig. 2.10, the SPH scheme can resolve soundwaves up to the wave vector of $k \simeq 2$. However, the radius of the compact support M_4 of the SPH kernel is 2. This means that a soundwave whose frequency is greater than the particle interseparation cannot be resolved as expected. For lower frequencies, soundwaves are resolved with an accuracy related to the second derivative of $\hat{W}$, as the first derivative at $k = 0$ is null.

5. Summary

In summary, we have presented three numerical methods that are widely used in the astrophysics community. We have opted to implement those three methods within Shamrock. For the Zeus scheme, the choice was motivated by the scheme's simplicity and performance. Additionally, it is a good candidate for prototyping grid-based solvers while profiling the performances of the framework. Godunov-based schemes provide a good treatment of hydrodynamical shocks, since the method is specifically tailored for such problems. Furthermore, they are widely used and benchmarked, with several developments and extensions, rendering them ubiquitous within hydrodynamical frameworks. Lastly, SPH-based schemes, provide the possibility of resolving complex dynamical geometries, as they provide good conservation properties for advection effects. In order to implement those three methods in a numerical framework, we need to support particles, grid methods, and potentially AMR and staggered mesh. This highlights the need for abstractions that would be compatible with the three methods (e.g. domain decomposition, load balancing, communications, etc.). In Shamrock, we regularly seek such abstractions to reduce code duplication between the methods.

References

Balsara D. S., 1995, *von Neumann stability analysis of smooth particle hydrodynamics–suggestions for optimal algorithms*, Journal of Computational Physics, 121, 357-372

Benítez-Llambay P., Masset F. S., 2016, *FARGO3D: A New GPU-oriented MHD Code*, ApJS, 223, 11

Chow E., Monaghan J. J., 1997, *Ultrarelativistic SPH*, Journal of Computational Physics, 134, 296-305

Courant R., Friedrichs K., Lewy H., 1928, *Über die partiellen Differenzengleichungen der mathematischen Physik*, Mathematische Annalen, 100, 32-74

Cullen L., Dehnen W., 2010, *Inviscid smoothed particle hydrodynamics*, MNRAS, 408, 669-683

Dehnen W., Aly H., 2012, *Improving convergence in smoothed particle hydrodynamics simulations without pairing instability*, MNRAS, 425, 1068-1082

Gingold R. A., Monaghan J. J., 1977, *Smoothed particle hydrodynamics: theory and application to non-spherical stars.*, MNRAS, 181, 375-389

Godunov S. K., 1959, *A difference scheme for numerical solution of discontinuous solution of hydrodynamic equations*, Math. Sbornik, 271–306

Hairer E., Lubich C., Wanner G., 2003, *Geometric numerical integration illustrated by the Störmer-Verlet method*, Acta Numerica, 12, 399-450

Harten A., 1983, *High Resolution Schemes for Hyperbolic Conservation Laws*, Journal of Computational Physics, 49, 357-393

Inutsuka S.-I., 2002, *Reformulation of Smoothed Particle Hydrodynamics with Riemann Solver*, Journal of Computational Physics, 179, 238-267

Landshoff R., 1955, Technical report, *A numerical method for treating fluid flow in the presence of shocks*. Los Alamos National Lab.(LANL), Los Alamos, NM (United States)

Lattanzio J., Monaghan J., Pongracic H., Schwarz M., 1986, *Controlling penetration*, SIAM Journal on Scientific and Statistical Computing, 591–598

Lodato G., Price D. J., 2010, *On the diffusive propagation of warps in thin accretion discs*, MNRAS, 405, 1212-1226

Lucy L. B., 1977, *A numerical approach to the testing of the fission hypothesis.*, AJ, 82, 1013-1024

Margolin L. G., Lloyd-Ronning N. M., 2022, *Artificial Viscosity – Then and Now*, arXiv e-prints, ADS link, arXiv:2202.11084

Mihalas D., Mihalas B. W., 1984, *Foundations of radiation hydrodynamics*

Monaghan J. J., 1997a, *SPH and Riemann Solvers*, Journal of Computational Physics, 136, 298-307

Monaghan J. J., 1997b, *SPH and Riemann Solvers*, Journal of Computational Physics, 136, 298-307

Monaghan J. J., 2002, *SPH compressible turbulence*, MNRAS, 335, 843-852

Monaghan J. J., Price D. J., 2001, *Variational principles for relativistic smoothed particle hydrodynamics*, MNRAS, 328, 381-392

Morris J. P., 1996, PhD thesis, -

Morris J. P., Monaghan J. J., 1997, *A Switch to Reduce SPH Viscosity*, Journal of Computational Physics, 136, 41-50

Noh W. F., 1987, *Errors for calculations of strong shocks using an artificial viscosity and an artificial heat flux*, Journal of Computational Physics, 78–120

Price D. J., 2008, *Modelling discontinuities and Kelvin Helmholtz instabilities in SPH*, Journal of Computational Physics, 227, 10040-10057

Price D. J., 2012, *Smoothed particle hydrodynamics and magnetohydrodynamics*, Journal of Computational Physics, 231, 759-794

Price D. J., Federrath C., 2010, *A comparison between grid and particle methods on the statistics of driven, supersonic, isothermal turbulence*, MNRAS, 406, 1659-1674

Price D. J., et al., 2018, *Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics*, PASA, 35, e031

Richtmyer R., 1948a, *Proposed numerical method for calculation of shocks*, Report LA-671, 1–18

Richtmyer R., 1948b, *Proposed numerical method for calculation of shocks II*, Report LA-657, 1–33

Richtmyer R. D., Dill E., 1959, *Difference methods for initial-value problems*, Physics Today, 50–50

Schoenberg I. J., 1946, *Contributions to the problem of approximation of equidistant data by analytic functions. Part A. On the problem of smoothing or graduation. A first class of analytic approximation formulae*, Quarterly of Applied Mathematics, 45-99

Seliger R. L., Whitham G. B., 1968, *Variational principles in continuum mechanics*, Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 1–25

Springel V., Hernquist L., 2002, *Cosmological smoothed particle hydrodynamics simulations: the entropy equation*, MNRAS, 333, 649-664

Stone J. M., Norman M. L., 1992a, *ZEUS-2D: A Radiation Magnetohydrodynamics Code for Astrophysical Flows in Two Space Dimensions. I. The Hydrodynamic Algorithms and Tests*, ApJS, 80, 753

Stone J. M., Norman M. L., 1992b, *ZEUS-2D: A Radiation Magnetohydrodynamics Code for Astrophysical Flows in Two Space Dimensions. II. The Magnetohydrodynamic Algorithms and Tests*, ApJS, 80, 791

Sweby P. K., 1984, *High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws*, SIAM Journal on Numerical Analysis, 21, 995-1011

Teyssier R., 2002, *Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES*, A&A, 385, 337-364

Toro E. F., 2013, *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*, Springer Science & Business Media

Van Leer B., in *16th aiaa computational fluid dynamics conference*, booktitle, p. 3559

Verlet L., 1967, *Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules*, Physical Review, 159, 98-103

Von Neumann J., Richtmyer R. D., 1950, *A Method for the Numerical Calculation of Hydrodynamic Shocks*, Journal of Applied Physics, 21, 232-237

Zou L., 2021, *Understand Slope Limiter – Graphically*, arXiv e-prints, ADS link, arXiv:2102.04435

van Leer B., 1974, *Towards the Ultimate Conservation Difference Scheme. II. Monotonicity and Conservation Combined in a Second-Order Scheme*, Journal of Computational Physics, 14, 361-370

van Leer B., 1979, *Towards the Ultimate Conservative Difference Scheme. V. A Second-Order Sequel to Godunov's Method*, Journal of Computational Physics, 32, 101-136

Chapter 3

Challenges of modern computing hardware

Contents

1. Introduction

In this chapter, we will see that modern computing heterogeneous architectures are complex, with many overlapping execution and memory models, as well as featuring complex communication topologies. Following this evolution, numerical codes have to become increasingly complex in order to be able to exploit their full potential. Hence, extensive planning is required to prevent making decisions that would be detrimental to the development of a code. Under that perspective, we will first dive into the inner workings of heterogeneous CPU-GPU architectures to better understand the choices that have been made in this Ph.D. Thesis. In general, we value better knowledge of the underlying hardware in order to produce software that is well tuned for it. To the best of our knowledge, such a review, so far, does not exist in the astrophysical community.

In this chapter, we will start by understanding the principles of how standard homogeneous computing architectures work and the history behind them to better understand the move toward heterogeneous architectures and their design. We will

follow with a deep dive into the inner workings of GPUs to understand their performance and how to optimize for them. Lastly, we will detail how to express code on a GPU and discuss how to adapt to modern multi-GPU architectures.

2. Brief history of HPC supercomputing

2.1. Monolythic supercomputers (1900-80)

The first Turing complete computer designed was ENIAC, which, with its design, laid the foundation of modern computing in many ways. Under the influence of Von Neumann, the computer was transformed into the eponym architecture (Von Neumann, 1993, originally written in 1945). In a Von Neumann architecture, a program is formed by a series of processing instructions that form a computer program. Formally, the computer is divided into four components:

- Multiple ALUs (arithmetic logic unit), which perform basic operations.
- The CU (control unit) is decoding the instructions composing the program to schedule operations in the ALUs.
- The inputs and outputs of the computer.
- The memory stores both the program that is executed, as well as the data that the computer is working on (note that it does not need to be in the same memory).

For example, data is often stored in volatile memory, whereas the program is stored in persistent memory. To date, most computers are still built using a Von Neumann architecture, even if its implementation is more complex. ENIAC was built using vacuum tubes, which were prone to frequent failure. Notably, a common failure cause was when insects burned on the tubes, creating a thermal stress that shattered the tube. This yields the common use of the term 'bug' to refer to an issue in computing.

Later, with the introduction of transistors at Bell Labs in 1947 by John Bardeen, Walter Brattain, and William Shockley, started a transistor revolution mostly in 1960, which quickly replaced vacuum tubes thanks to their greater reliability, smaller size, and lower power consumption.

Using micrometer-sized transistors, Intel created the 4004 microprocessor. It integrates an ALU and the control unit on a single, small chip, making it a central processing unit (CPU). The Intel 4004 is the first CPU to be commercialized. CPUs of this era featured only a small instruction set (46 on the Intel 4004 compared to more than 1000 on modern `x86_64` CPUs with all extensions `http://ref.x86asm.net/geek64.html`).

In order to perform complex calculations, more complex computers were required. An example is the CDC 1604 mainframe computer (mainframe refers to a computer in a single, sometimes big cabinet) built in 1960, which was the first commercially

successful mainframe computer. Only a year later, IBM introduced the 7030 Stretch. This mainframe computer, aside from its 2048 kilobytes and $1.2 \cdot 10^6$ instructions per second, was the precursor of many technologies still used on modern CPUs to this day. The first innovation in the 7030 was instruction pipelining (also called out-of-order execution in modern CPUs), where instructions to be executed are queued in a pipeline, enabling independent instructions to be executed simultaneously. This concept is referred to as instruction-level parallelism. For example, moving data can be performed at the same time as an arithmetic operation if the targeted registers are different. Registers are the memory space internal to the CPU directly connected to ALUs. For example, when performing an addition, the ALU will read from two registers, add the numbers, and return the sum in another register. Additionally, the IBM 7030 Stretch featured CPU caches to keep frequently used data closer to the CPU than it would be in the RAM (random access memory), and was the first computer to implement a prefetcher that would try to load data that would be used preemptively. A typical pattern would be to load the next value after the one currently being loaded, such that in the case of a contiguous read, the data would already be available.

The last computer of this era is the Cray CDC 6600 from 1964, which achieved $2 \cdot 10^6$ instructions per second with 982 kilobytes of memory. It is the first superscalar computer. A superscalar computer refers to a computer having multiple ALUs, where the multiple ALUs are fed using instruction-level parallelism. A superscalar computer is capable of issuing, for example, multiple additions at the same time, even if only a succession of addition instructions were written in the assembly of the program.

So far, all the computers presented are monolithic in the sense that they are a single computer, regardless of its size.

2.2. Distributed supercomputer (1975-Now)

However, such a model does have limits. As expansion progresses, the components become increasingly distant from each other. The distance induces a significant communication latency between the components, limiting the possible performance of a monolithic system. The solution to this problem is to split the supercomputer into several smaller computers connected together using a network. This is called distributed computing and is the basis of modern supercomputers.

The first so-called supercomputer was the S-1 Supercomputer (1975-1988), which initially had 16 nodes (Smotherman, 2023), where a node refers to a single computer connected via a network to the other nodes. The project also independently invented two-bit branch prediction. Taking a branch refers to the moment when a program performs a conditional jump, typically an if or while statement in modern programming languages. If successfully predicted, a computer can still pipeline executions ahead of taking the branch to speed up computation. This is called speculative execution. Interestingly, branch prediction was considered in the development of the

IBM 7030 Stretch but was discarded in the final product because it was detrimental to performance in its implementation.

However, coding for such an architecture was inconvenient because every supercomputer used different communication protocols and implementations. This led to the creation of the Message Passing Interface in 1992 at MPIworkshop (Walker, 1992). In an MPI program, a single instance of the program is running on each CPU, and each program is assigned an integer rank on the supercomputer. All programs are the same, only the rank of each program differs. Additionally, MPI also features collective operations such as reductions, broadcasts, and gatherings across the ranks. The convenience of writing a program once using this standard and running it on any architecture implementing the standard led to wide adoption and is still used today in the scientific community.

The last major change to CPUs that appeared in supercomputers was the creation of multicore CPUs with the IBM POWER4 launched in 2004. In this model, a CPU is made of multiple cores, each capable of running independent tasks. Typically used in conjunction with MPI, a typical usecase was to run an instance of the MPI program per core instead of per node sequentially. In practice, this means that the distributed supercomputer is made of two levels of parallelism, the first being the node network, and additionally, the possibility to parallelize within the node, having multiple CPU cores per node.

3. Recent evolution of computing hardware

CPUs are designed to be able to execute any task. They are not really specialized in a specific type of calculation. It was realized that having dedicated hardware targeting specific types of computation can be both more efficient and powerful than always relying only on CPUs. A first example of such an idea is GRAVITY PIPE featuring some dedicated accelerators dedicated to speeding up N-Body gravity calculations (e.g. Makino & Taiji 1998).

Additionally, GPUs (graphical processing units) initially designed for having large memory bandwidth and exploiting a large number of floating point ALUs in parallel are starting to be used in supercomputers for their greater energy efficiency (Qasaimeh et al., 2019), greater memory bandwidth, and floating point performance while keeping some versatility. In general, the larger the cluster, the more complex its cooling and networking. Using GPUs, in particular, allows for a great increase in the computing density of a node. For example, the Summit supercomputer (which achieved 200PFlops) is made of 2 CPUs with 6 GPUs in each node. An Epyc 7742 64-core CPU from 2019 produces 3.8 TFlops (Dell, 2019) while consuming 225W, whereas a single Tesla V100 from 2017 produces 14 TFlops with only 300W (TechPowerUp, 2024) while having 6 of them per node. This results in better compute density and efficiency. This trend can be seen in Fig. 3.1 where `top500` best CPU based supercomputers are stagnating in performance while GPU based supercomputers are rapidly increasing in performance.

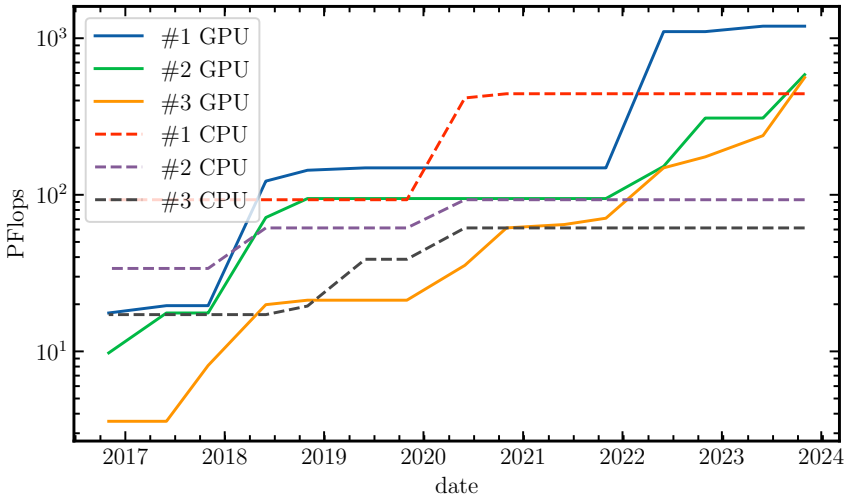


Figure 3.1: Relative evolution of the best GPU and CPU in the top 500. Data from (Top500, 2024)

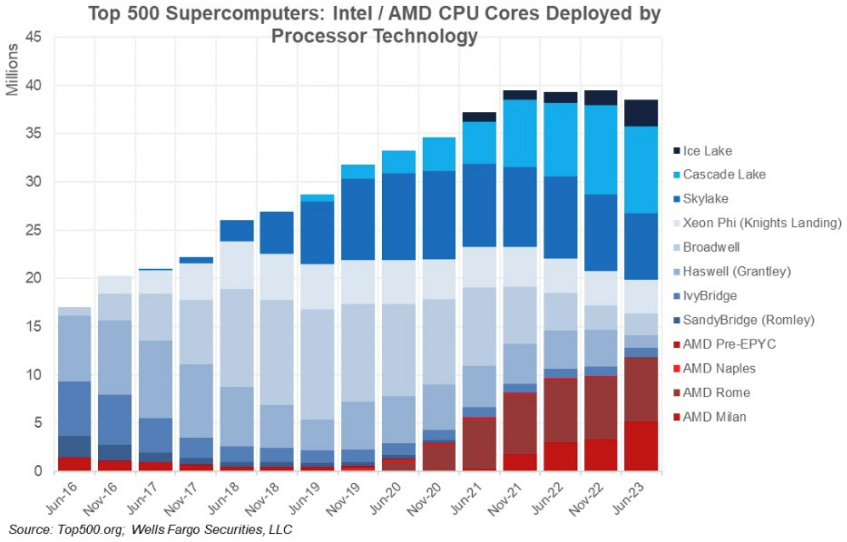


Figure 3.2: Relative evolution of the best GPU and CPU in the top 500 The next platform (2023).

Additionally, the push towards better efficiency is also pressured by climate change and the increase in electricity prices. In general, this means that, in addition to new supercomputers being built with GPUs, pressure is made to use CPUs less. This can be seen in Fig. 3.2, where the number of total CPU cores in the Top500 is decreasing.

This implies that we must adapt current code or develop new ones to target GPUs and newer computing hardware.

4. A deep dive in a GPU

4.1. Topology of a computer

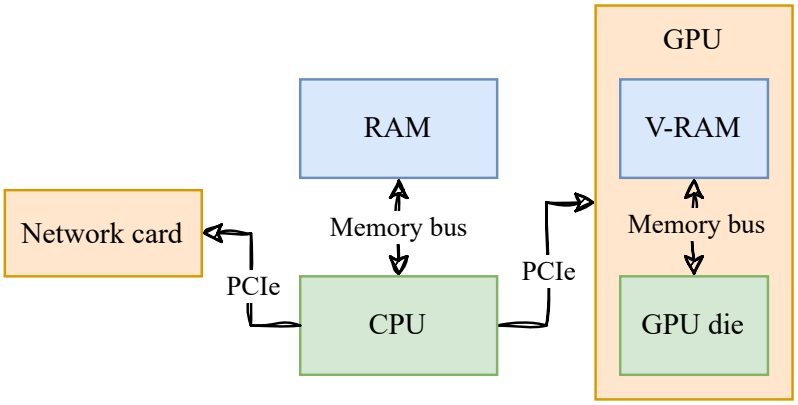


Figure 3.3: Simplified illustration of the internal layout of a computer.

4.1.1. CPU & RAM

To better understand how a GPU is designed and functions in practice, we will first examine the internal design of a standard computer. A particular focus will be made on the analysis of the bandwidth, as it is a primary constraint in simulating hydrodynamics (a memory-bound scenario). As seen in Fig. 3.3, a standard computer is made of a *Central Processing Unit* (CPU) connected to its *Random Access Memory* (RAM). This is where most of the data used by the CPU is stored. The CPU is designed to be able to execute complex workflows involving multiple simultaneous processes, each of which can be complex on its own. Because of the number of processes running on the CPU, latency is a primary concern in its design. In short, a CPU is designed to execute a large number of simultaneous, complex tasks with minimal latency, using potentially complex CPU instructions. The connection with RAM is done using memory channels, where a channel can be thought to be a data bus. The bandwidth between the RAM and the CPU is therefore the product of the number of channels times the bandwidth of each individual one. For reference with RAM, a single channel of DDR5-4800 should be around 30GB/s (see Schlachter & Drake (2019)), where most consumer CPUs have two channels and up to twelve for the latest and largest AMD CPU currently available (AMD EPYC 9754).

4.1.2. PCIe

To the CPU are attached expansion cards, such as a network card for a GPU, using *Peripheral Component Interconnect Express* (PCIe) or equivalent proprietary solutions. PCIe connections are made of multiple lanes, where the total bandwidth

is the product of the number of lanes and the lane bandwidth. Most GPUs are connected using 16 lanes to maximize the available bandwidth.

4.1.3. GPU

In some computers (most desktop computers but not all laptops), a GPU is also connected through PCIe to the CPU. Such GPUs are called discrete GPUs, as opposed to integrated GPUs when the GPU is contained within the CPU die. While GPUs were initially designed to only execute graphical operations, they are currently capable of doing so-called GPGPU (where the GP stands for general purpose), where a GPU will execute a lot of simple tasks. Contrary to CPUs, the focus on GPUs is to be capable to process a lot of similar tasks. This means that a GPU can execute tasks from only a few applications at the same time. A given application will execute thousands of tasks simultaneously on the GPU. In short, GPUs are made to process a lot of simple, similar tasks in parallel. Latency is not an immediate concern. The GPU to VRAM bandwidth is about 768.0 GB/s for a NVIDIA RTX A5000 and up to 3.36 TB/s for a NVIDIA H100 SXM5 96 GB. This shows that, importantly, the memory bandwidth of a consumer-grade GPU is already higher than that of a top-of-the-line CPU, whereas a supercomputer-class GPU will have an order of magnitude higher memory bandwidth than a supercomputer-class CPU. However, PCIe bandwidth is of the order of 63.015GB/s when using 16 lanes, which is an order of magnitude slower than the bandwidth of the GPU. This implies that data transfers between the CPU and the GPU have to be avoided if possible.

4.1.4. Network card

Typical computers also have, a network card capable of a bandwidth of about 1Gbps (10^9 bits per seconds).

4.1.5. Bandwidth

In summary, in modern computers, if a given application is bottlenecked by the memory bandwidth, which is the case in most hydrodynamics applications, it is best to migrate all of the data to the GPU without needing to retrieve it. In other words, using the GPU to accelerate only a small step of an application is not efficient most of the time because of the GPU-to-CPU transfer speed. We note, however, that such requirements could change as some GPU vendors are replacing architectures with a CPU and multiple GPUs with so-called APUs (for example, the AMD Mi300 APU), which combine both a CPU and GPUs on the same die. However, the programming model and behavior of such an architecture are largely unchanged except for the CPU-to-GPU bandwidth, which is non-relevant as the CPU and the GPU share the same memory architecture in such a design. Another experimental approach is to perform computation internally on the memory die; this, in principle, could result in reducing the pressure on memory bandwidth as less data transfer would be

performed. An example of such an architecture is called Samsung-PIM (processing in memory). However, such an architecture is not yet available for testing.

5. GPU execution model

We will now present the execution model of a typical GPU. Assuming that we have a given program executing on the said GPU, we will explore how such a program is made, and how the GPU processes it. Here, the GPU is often referred to as the *device* and the CPU as the *host* as it is scheduling work for the GPU.

5.1. SIMD on CPU

On CPUs, a commonly used model to speed up computation is to use SIMD (Single Instruction Multiple Data). SIMD consists of loading specific, larger registers with data, which can be thought of as vectors of multiple values. An SIMD instruction is a single instruction working on all of those values at the same time. The most basic example of such an instruction is the addition of two 4-dimensional vectors. On the CPU die, they are implemented in the ALU as specific circuits able to perform vectorized operations. In this model, a single instruction is working on multiple data points at the same time, hence the name SIMD. Here, SIMD can be though as a single operation performed on multiple slots at the same time.

5.2. SPMD (Single Program Multiple Data)

On a GPU, the model is almost an extrapolation of the SIMD paradigm. It is called SPMD (Single Program Multiple Data), where instead of single instructions, a small program is run for each slot that would have been SIMD on the CPU. In reality, the model is more complex, as the number of slots can vary and the exact method is not straightforwardly an extension of SIMD.

Contrary to CPUs, GPUs are not equipped with the capability of executing a program by themselves. Instead, they execute work that is scheduled by the host on the device, hence the common use of the term accelerator for GPUs or other types of cards that can perform tasks scheduled by the host faster than the host would. On GPUs, the main difference lies in the scheduling model used internally.

Definition ▶ GPU compute kernel.

A task executed on a GPU is called a *kernel*, where multiple simultaneous instances (or *threads*) of the kernel program will be executed in parallel by the GPU.

This scheduling approach is a model called SPMD (Single Program Multiple Data) Cook (2012), where single programs mean that only a single set of instructions

(the program or kernel in GPU terminology) will be performed, whereas multiple data means that the same set of instructions will be executed several times with different inputs. The main difference with traditional execution is that in SPMD, all instances of the program work collectively on the same global device memory (on the GPU, the VRAM), as shown in Fig. 3.4. Additionally, there is no synchronization across processes in standard SPMD, although we will see how a GPU deviates slightly from this paradigm. In summary, the basic execution of a GPU-accelerated

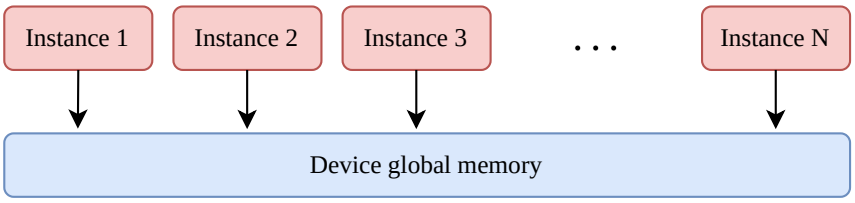


Figure 3.4: Illustration of a SPMD execution model.

program will be a succession of tasks being executed on the host, followed by the enqueuing of a kernel on the GPU, which will potentially *dispatch* millions of instances of the same compute kernel. Once the GPU work is complete, execution will resume on the host as represented on Fig. 3.5.

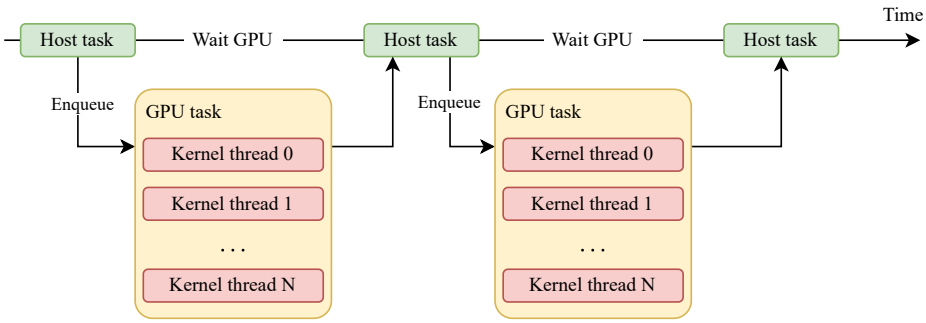


Figure 3.5: Basic workflow of a GPU program.

5.3. SIMT parallelism

Unlike CPUs which execute each thread independently, modern GPUs instead execute groups of coherent threads, where the size of the group times the number of groups gives the total number of threads. Within a group, since all compute kernel threads are executing the same program, GPUs make them coherent. This means that every thread within a group will execute the same instruction of the compute kernel at the same time. They are *synchronous*. This means that a single instruction will be issued at the same time to all the threads within a group. In other words, this execution is called *Single Instruction Multiple Threads*, SIMT parallelism. Contrary to SIMD, which corresponds to single instructions working on a collection of data,

SIMT instructions are single instructions that control multiple threads at the same time. Coherent groups are also grouped together to form a block of threads. On NVIDIA GPUs, the coherent thread group has, in general, 32 threads, whereas the block is made of up to 16 groups of coherent threads. Groups of coherent threads are not coherent together within the thread block, but each thread is within its own coherent group. Nevertheless, threads within the thread block can perform collective operations like exchanges, mutexes, and barriers across all threads of the block. In that sense, GPUs deviate from the standard SPMD model, since threads within a block can be synchronized and can exchange data. However, there are no trivial synchronizations across thread blocks (see Sec. 7.3). This means that a proper representation of a GPU is a SPMD model on the thread blocks and a SIMT model within the coherent groups in a block. No standard naming for the SIMT group and thread block is used in practice. In CUDA, they are referred to as Warps and Blocks, whereas in OpenCL and SYCL they are referred to as wavefronts and workgroups.

5.4. Streaming multi-processor

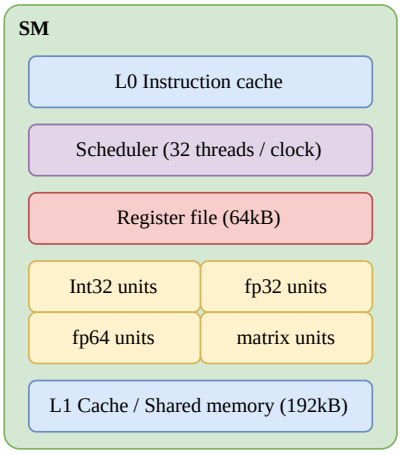


Figure 3.6: Simplified representation of a A100 Nvida GPU Streaming multiprocessor adapted from the information provided in Choquette et al. (2021)

On GPUs, the thread block is executed by a special type of processor called a streaming multiprocessor (SM for short). As explained above, threads are executed in coherent groups, or warps, in CUDA. When the SM is presented with a block of threads, for every instruction in the compute kernel, each warp will run the given instruction using SIMT parallelism. For example, if the next instruction is an addition, the first warp of the block will execute the addition, using SIMT parallelism. This will be followed by the next warp in the block, until all warps have performed the given instructions. After this step, the sequence repeats with the next instruction from the kernel until its completion. This strategy of execution of warps is called

round-robin. Other strategies are possible. However, this goes beyond the scope of the current discussion. Additional details can be found in e.g. Lakshminarayana & Kim (2010).

In order to execute those instructions, multiple compute units are available on the SM. Typically, on a NVIDIA A100, there are 16 single-precision integer units, 16 single-precision floating-point units, 8 double-precision floating-point units, and 1 generalized matrix unit called the tensor core in CUDA. It is important to note that there are almost always fewer floating-point double-precision units than single-precision ones. This means that the peak throughput of double precision will be lower than that of single precision. On consumer GPUs, the ratio is worse, with up to a 64:1 ratio between single and double precision performance.

Similarly to what happens on the CPU, when an instruction is issued, it can either move data to or from a register file or compute with the data in the register file. For example in order to add two numbers, the two numbers are first moved to the register file, in two specific registers, then an addition instruction is called which will write the result in another register from which the result can be stored. However the register file on NVIDIA A100 GPUs has a capacity of 64kB available for the execution of the blocks. Each thread uses a defined at compile time number of registers (counted in number of 32-bits integers, 4 Byte). This means that if threads use too many registers the SM will have to use fewer threads per warp for the register usage of the block to fit in the register file of the SM. This is a situation where the occupancy of the SM is limited by the register usage.

As mentioned above, threads within a block can perform operations together. This is done using the shared SM memory, which by definition is shared across all threads of the block and used to exchange data between them. Like with register files, using too much shared memory will result in lowered occupancy.

5.5. The GPU & Block scheduling

The GPU itself is made of multiple SMs connected to the GPU RAM, where the data that the GPU is collectively working on is stored. Its use is accelerated by the presence of a global memory cache (L2 cache on NVIDIA GPUs), which can read from and write to the global memory, similarly to the global memory accessible to all SMs of the GPU (see Choquette et al. (2021) for more detail about the L2 cache architecture). The GPU global memory in general is either a graphics-DDR type memory or HBM (high bandwidth memory) memory, where the latter type provides the best bandwidth, hence its use in compute-focused GPUs. As of 2024, three types of HBM memory are in use: HBM2, HBM2e, and HBM3, which do not differ in terms of functionality except for the bandwidth that they provide. On compute-dedicated recent GPUs, VRAM bandwidth can range from 1.5TB/s (NVIDIA A100 SXM4-40GB) up to 3.5TB/S (NVIDIA H100 SXM5 96 GB). We will explore the memory bandwidth behavior of GPUs further in Sec. 6.2.

When a kernel is executed by the GPU, the thread blocks are dispatched to the

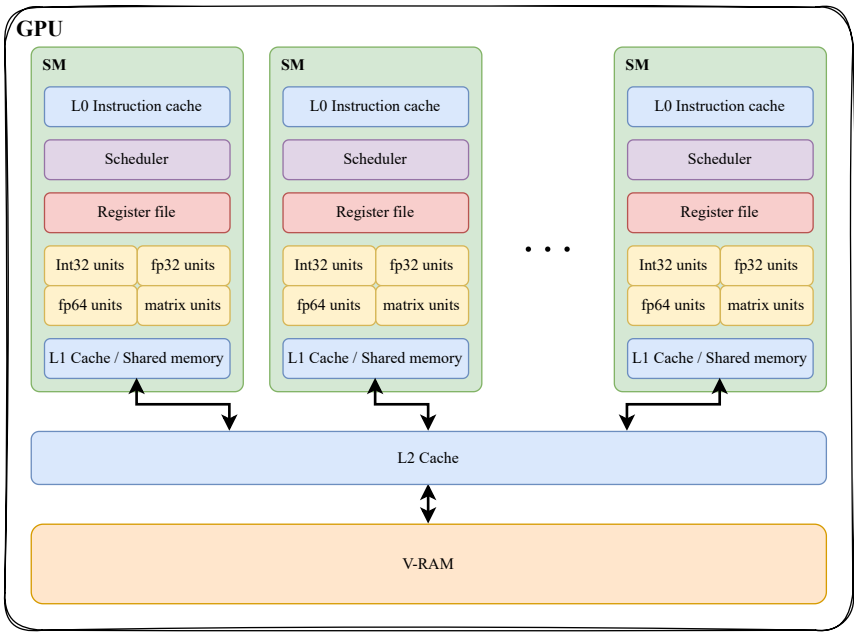


Figure 3.7: Simplified representation of a A100 Nvida GPU adapted from the information provided in Choquette et al. (2021)

SM of the GPU. However, if there are more thread blocks than available SMs, knowing that each block is mapped to a SM means that the GPU must schedule blocks in a specific order. For specific algorithms leveraging block-block synchronizations, it is important to understand the behavior of scheduling and, or manipulate it. We will discuss the fact that it is possible on the GPU for one thread block to wait for another one. However, if a given block is waiting for another block that will never start before the current ones are completed, the program cannot progress and will hang. This condition is called a deadlock.

Definition ▶ Deadlock

Situation where a thread is waiting for the completion of an operation that cannot happen. Typically, when one thread waits for another one to finish and the other thread is also waiting on the first one.

When writing algorithms that involve block-to-block synchronization, it is therefore important to be aware of the guarantee offered by the execution model of the GPU in order to guarantee the correctness and termination of the program.

Definition ▶ Forward progress guarantee

A compute kernel with a forward guarantee refers to the property that the kernel must progress on its execution until completion. In other words, the kernel must be complete because it has a finite number of operations to perform. If the compute kernel has a forward progress guarantee, a deadlock is not possible.

As detailed in Sorensen et al. (2018) in OpenCL, no block execution ordering is guaranteed, as a conforming implementation can have an unfair scheduler where threads are executed sequentially in an order that is not guaranteed by the standard. This means that if the compute kernel features a thread block that is trivially waiting for another one, there are no forward progress guarantees. Neither is the case in CUDA, where, as stated in the programming guide (NVIDIA, 2024a) 'each block of threads can be scheduled on any of the available multiprocessors within a GPU, in any order, concurrently or sequentially, so that a compiled CUDA program can execute on any number of multiprocessors'. Therefore, no assumptions about the order are possible when executing on NVIDIA GPUs.

In practice, it is easy to test if for example the scheduling model executes thread blocks in order with a kernel where a block waits for the previous block to complete. Such kernel will work only if the scheduler schedules thread blocks in order. If that is not the case, using atomic operations (see Sec. 7.3), it is possible to compute an indexing for the blocks that guarantees the ordering of the blocks instead of the provided one. An example would be to use an atomic counter on each block whose result is taken to be the block index.

6. GPU performance

6.1. Rooflines

We have discussed the execution model of a GPU. Now, supposing we have a compute kernel that can run on the GPU, we need to be able to optimize it. A possibility to evaluate the performance of a compute kernel is the number of threads processed per second, which is a useful metric to report the performance of a kernel. However, this metric lacks a comparison with the theoretical performance that one could achieve under ideal conditions. This is why, when optimizing for GPUs, we measure instead the achieved bandwidth and integer/floating point throughput of the kernel. Firstly, GPU vendors report the performance numbers of their GPUs for most primitive types. For example, an Nvidia A100 GPU is capable of producing 19.49 TFLOPS (tera floating point operation per seconds) in fp32 (32-bits floating-point numbers). This means that if our compute kernel achieves peak performance, $19.49 \cdot 10^{12}$ 32-bit floating-point operations should be issued per second. It is therefore a quantitative metric that can be compared against the achieved fp32 throughput.

Some compute kernels can also be limited in performance by the achieved memory bandwidth (memory bound). By comparing the theoretical peak bandwidth, we can see if it achieved peak memory performance or not. For example, an NVIDIA A100 GPU has a bandwidth of 1.56 TB/s, and a fp64 throughput of 9.746 TFLOPS. This means that in an ideal scenario, to achieve peak arithmetic throughput and peak memory bandwidth, we perform 6.24 flop (floating point operations) per transferred byte of data. This number on modern GPUs is very large for hydrodynamics, where one would need to perform almost 50 operations per double precision floating point transferred, or, for example, to calculate the evolution of velocity fields, 150 operations would be required to achieve peak throughput. Except for discontinuous Galerkin-type methods, which can have a larger arithmetic intensity per bytes, most hydrodynamic workflows will be memory-bound. This trend is continuing to worsen with AI workflows pressuring GPUs toward higher computing throughput while not requiring similar bandwidth upgrades.

A useful tool to represent the performance of a kernel is to plot the bandwidth against the flops per byte achieved. This is called a roofline (see Fig. 3.8).

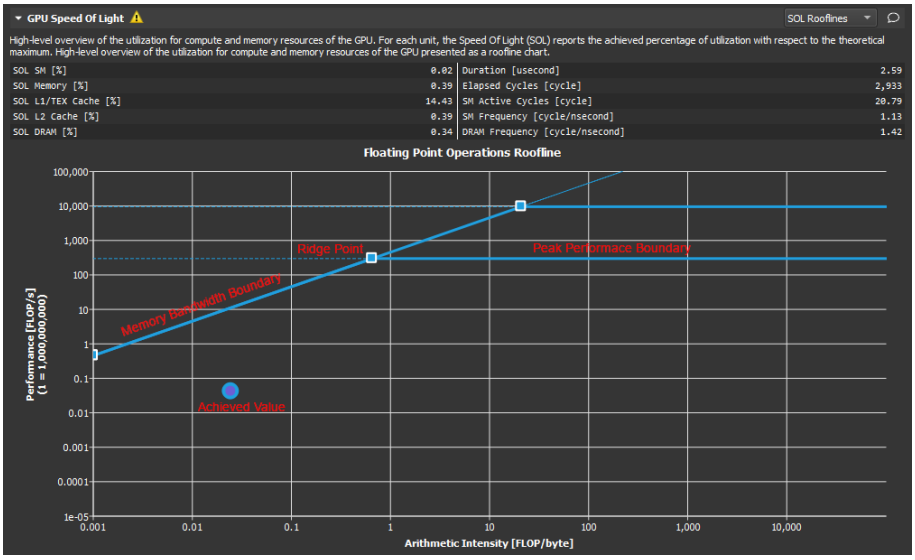


Figure 3.8: Exemple of a roofline produced by Nvidia tools. Source : NVIDIA (2024b)

In practice, achieving peak computing throughput only involves local optimization of the kernel program (see Sec. 6.4) and memory access pattern optimizations. The latter is non-trivial and requires an understanding of the memory architecture of the GPU.

6.2. GPU memory perfomance

To better understand the behavior of GPU memory, we rely on the material in Jones (2022) from which we summarize the memory behavior and reproduce the results.

6.2.1. DRAM cell

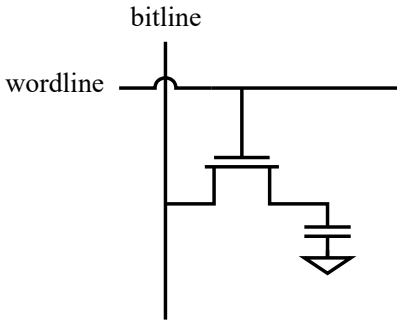


Figure 3.9: DRAM cell

Data in memory is made of bits, which can take the value of one or zero. In order to store a bit in memory in RAM, a so-called DRAM cell is used (Fig. 3.9). It contains a capacitor, which stores electrically the actual value of the bit. In order to either read from or write to memory, a voltage is sent to the wordline. When reading, by enabling current through the transistor the bitline will switch to 1 if the capacitor is holding charges or stay at 0 otherwise. When writing, switching the wordline to ON will store the value of the bitline in the capacitor. Either way, reading from a DRAM cell is therefore *destructive* as one has to take charges from the capacitor to check if it held charges in the first place. This means that in order read data from memory, we have to take the value from the cell and store it back after usage.

6.2.2. DRAM banks

DRAM cells are organized into banks, where a bank is a matrix of DRAM cells. When one wants to read from memory at a given address, it will point to a line and a column of the bank. First, the row decoder will switch the corresponding row's wordline on, which will pull all the data from the line (or page) into the sense amplifiers, also called the row buffer. Row buffers have the role of amplifying the signal recovered from the DRAM cells. When doing so, the data in the line of the bank is destroyed. At the end of the read, the data will have to be written back in the line of the bank. For reference, Jones (2022) gives the values for HBM memory of 16 cycles to load a line into the sense amplifiers, 16 cycles to write back data, and 16 cycles to read column data in the sense amplifiers.

In summary, when performing a single-bit read, the corresponding line is pulled into the sense amplifiers, then the value is sent from the sense amplifiers to the GPU, and lastly, the line is written back into the bank.

In practice, the GPU never reads values individually, as it would be inefficient. Instead, multiple values are pulled at once from the sense amplifier and into the

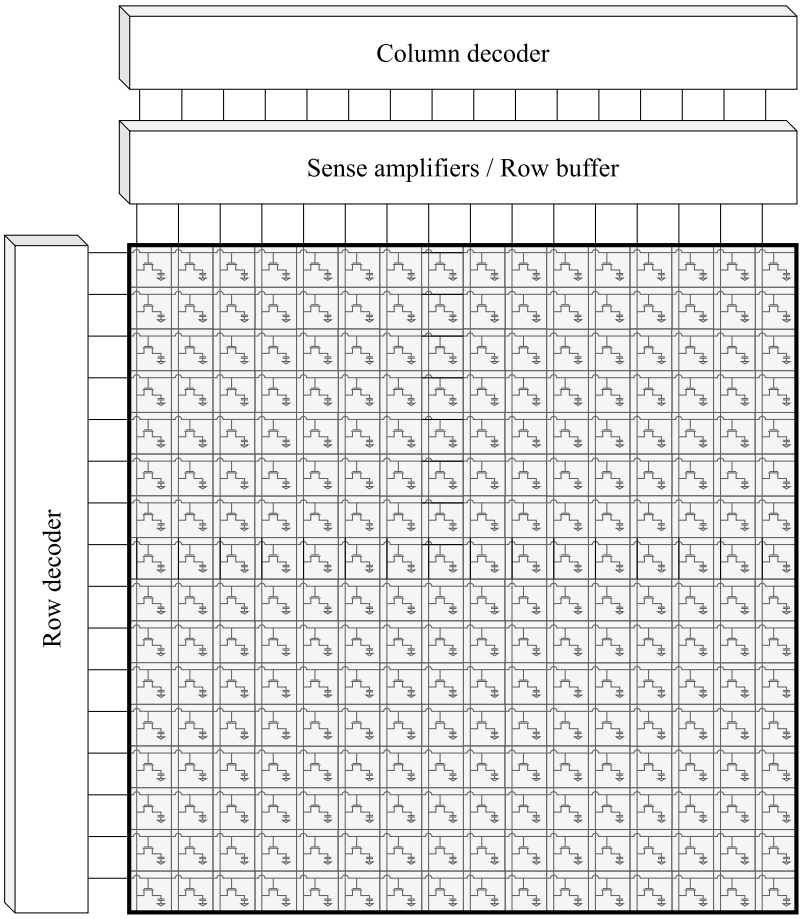


Figure 3.10: DRAM bank

GPU L2 cache. This read type is called a burst read, where the burst refers to the contiguous bits that are pulled from the sense amplifier into the cache. The burst read has the same latency as a standard column read.

6.2.3. Read latency

Due to the way memory functions, accessing addresses that are close together results in better performance compared to accessing addresses that are far apart.

In Jones (2022), the result of a benchmark to understand memory bandwidth performance as a function of the address spacing is sown (the figure is shown Fig. 3.11). However, the details of the benchmark and the explanation of the specific ratios were not provided. This motivated the following benchmark, done in collaboration with F.Lovascio (currently working at ARM). The benchmark described using pseudo code in Alg. 2 was coded in SYCL to try to have a benchmark that is as similar as possible across the device that it was run on. The benchmark consists of reading a double precision floating point number from indexes that are separated by s indexes. We then write the value in a result buffer to avoid the compiler optimiz-

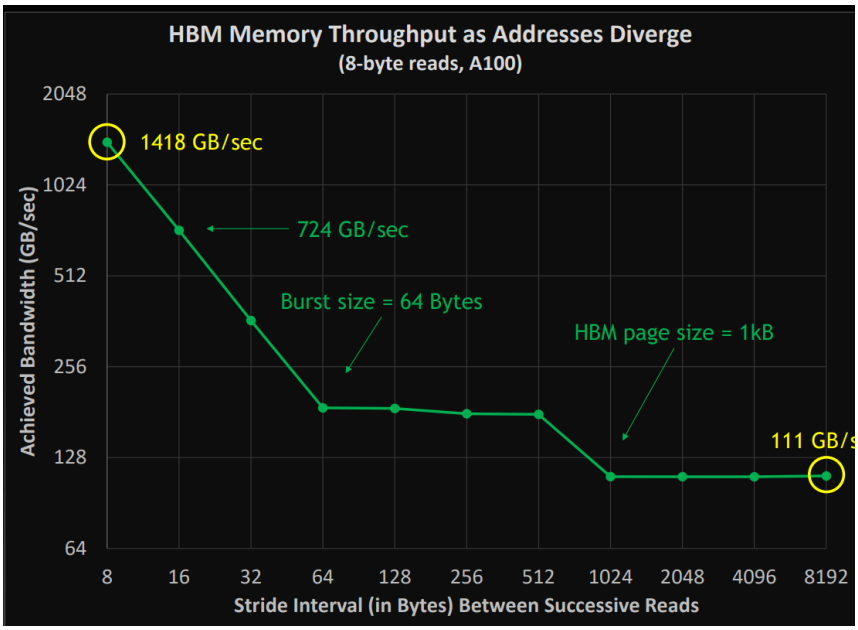


Figure 3.11: Memory bandwidth benchmark showing the memory bandwidth function of the spacing of addresses being read from.(Adapted from Jones (2022))

Algorithm 2: Memory read spacing benchmark

Data: a_n The input dataset (double precision floating point), b_n The output dataset (double precision floating point), s read spacing.

1 **for** i ***in parallel*** **do**
    // index that we read from
2 $iread \leftarrow i * (s+1)\%N$;
3 $b_i \leftarrow a_{iread}$;

ing away the memory read. The benchmark was run with compiler optimizations (`-O3 -march=native`) and compiled with the maximum compute capability available on a given GPU (`sm_86` for a Nvidia RTXA5000). On the CPU we used the OpenMP backend of the AdaptiveCPP compiler, and on the GPU, we used the CUDA backend of the Intel LLVM compiler. For more details about SYCL compilation, please refer to Sec. 8.2. Fig. 3.12 presents the results of the said benchmark. We observe that the initial slope and the two plateaus shown on the NVIDIA benchmark are reproduced.

We interpret the first slope as being the region where consecutive reads are contained in the same burst. Initially, when there is no spacing between reads, the optimal bandwidth is achieved. In that case, a burst of 64 bytes in HBM, corresponding to 8 double-precision floating points, is stored in the L2 cache of the GPU. When we read every even index, only four values remain in the burst. Hence, the bandwidth is halved compared to the peak one. In general, when still in the

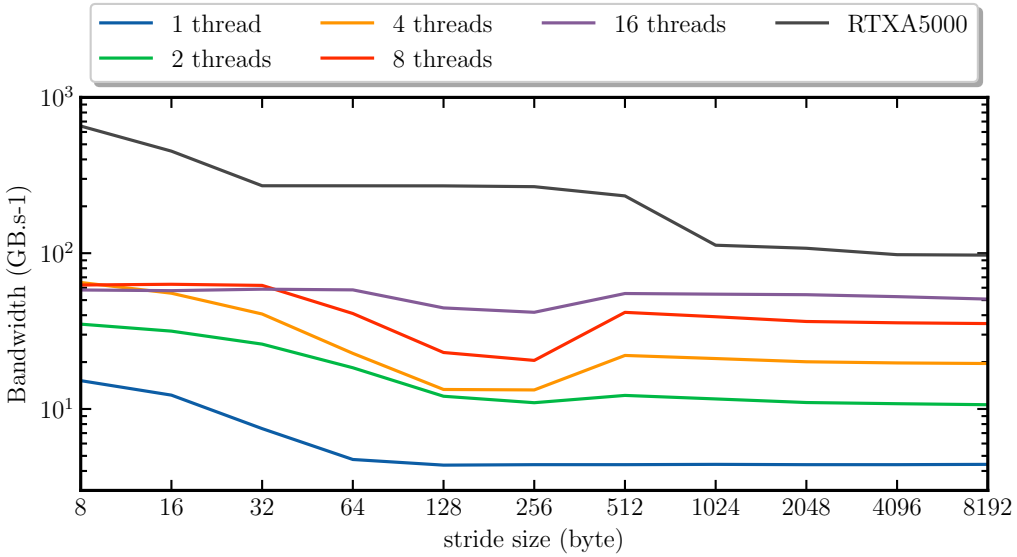


Figure 3.12: Our memory bandwidth benchmark showing the memory bandwidth function of the spacing of addresses being read from to reproduce the results from Fig. 3.11.

same burst, the bandwidth corresponds to the amount of burst that is used divided by the burst size. When we reach a single value used by burst, this leads to the bandwidth being a factor 8 lower compared to the optimal one. A burst read is required for every value. Increasing the spacing further has no additional effect, since one burst is issued per read. Therefore, increasing the spacing has no impact. This is true until reads are spaced by a page size (the size of a memory bank line, not to be confused with a cache line size).

To understand the ratio when the spacing equals or exceeds the memory bank page size, we need to go back to the HBM latencies. When we read multiple values that are on the same page but not in the same burst from memory, we always pay for the line read into the sense amplifiers (16 cycles) and the burst read (16 cycles) for a total of 32 cycles. When the two consecutive reads are from different pages, in order to read two continuous values, we must change the page loaded in the sense apmlifier, adding an additional 16 HBM cycles. This adds up to a total of 48 cycles for this read. In summary, this means that we have an additional one-third reduction in bandwidth when reads exceed the page size on HBM, explaining the last plateau on the Nvidia benchmark.

On a CPU, however, we observe that on multicore execution, the bandwidth is very consistent for any value of the stride size. This behavior was observed on all large server CPUs we tested from any vendor (AMD Epyc CPU (64 cores), Intel Xeon (48 cores), and ARM Neoverse (72 cores)).

In summary, on the GPU, only contiguous reads are able to achieve peak bandwidth, and the layout of the memory has a very large effect on the achieved band-

width, up to a factor of 12 on HBM memory. This means that on a GPU, changing the ordering of the data can have a larger effect than any other optimizations for any memory-bound workflow. However, similar effects are not observed on modern multicore CPUs.

6.3. GPUs and branches

Another important aspect of GPU execution is the handling of branches. Conditional statements, such as jumps, change the control flow of the executed program. However, this seems incompatible with SIMT parallelism, as all threads must execute the same instruction. A typical example is an if-else statement where half of the SIMT threads execute the if branch, and the other half the else branch. In this case, in SIMT parallelism, all threads will execute both branches, but half of the threads will be disabled during the if branch and the others ones during the else branch. This means that both branches are executed sequentially in this case, which can strongly impact performance. However, compilers perform many optimizations to suppress such cases as much as possible, and if the kernel is memory-bound, the additional latency created by the branching will be lower than the memory access latency, and has thus not impact the performance significantly. Performing memory operations within a branch hinders many possible compiler optimizations and has a larger impact.

6.4. SIMD on GPU

The literature on the usage of SIMD on CPUs is extensive, as SIMD has been proven to be a path toward improved performance. On GPUs, the literature on their performance is very sparse. On NVIDIA and AMD GPUs, the assembly emitted when compiling most programs. They exhibit vectorized instructions. NVIDIA typically has only 4-element wide instructions for single precision vectors in its assembly NVIDIA (2024c).

6.5. Execution latency

When the host is launching a kernel on the device, the driver and the API of the device have a submission overhead and latency. For example, in Zhang et al. (2019), it can be seen that the latency and overhead of a kernel submission are of the order of 10μs. This means that for small kernels that only run in a few μs, these latencies cannot be neglected. Therefore, small kernels may under-fill the GPU as well as expose latency, which will result in slow execution even if the kernel is achieving peak bandwidth or compute. Hence, on GPUs, it is preferable to submit large tasks, which will hide such overhead.

Additionally, on recent GPU APIs, it is possible to perform asynchronous execution, meaning that one can queue multiple kernels that will be executed in the

background, and that we will wait for to recover results. Utilizing async CUDA, one can hide the submission latency by enqueuing a large number of kernels.

This effect is becoming more and more important as GPUs get faster and codes get more optimized. For example, as outlined in Páll et al. (2020) Gromacs typically perform a timestep per 1 ms, meaning that in their case, submission latency can be significant and is a major concern.

6.6. GPU internal Load balancing

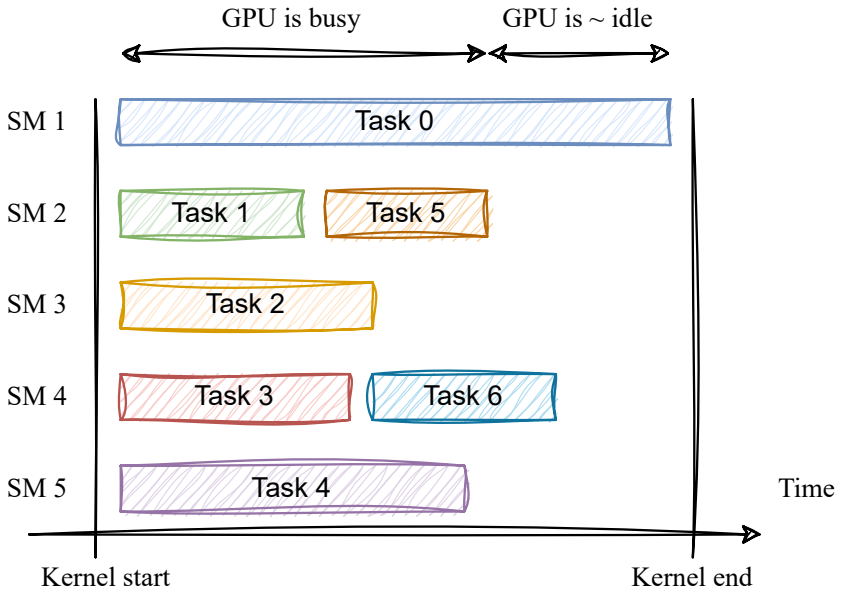


Figure 3.13: Illustration of the execution of a kernel featuring a non-optimal load balancing.

On multi-node supercomputers, load balancing is a primary concern, as poor load balancing can reduce the performance of an application by multiple orders of magnitude, hence the heavy emphasis on it. However, on GPU, even if less taught, the same issue exists. On the GPU, when a kernel starts, the GPU scheduler will start to dispatch the block of threads, which can be thought of as tasks, to the SMs, but the kernel is considered finished only when the last task ends. This means that having a strong imbalance in the task duration will result in the GPU being largely idle while waiting for the last task to finish.

This issue was specifically pointed to by Merrill & Garland (2016b), in which he addresses the issue in the context of sparse matrix vector multiplication.

6.7. Streams

On the GPU, as we have seen, kernels have to launch many tasks in order to fill all the SMs of the GPU. For example, on an NVIDIA A100, 108 SMs are available. In order to fill a SM, at least 32 SIMT threads per SM are required. This means that the bare minimum to fully utilize the compute-units of a GPU is 3456 threads. However, in practice, the threshold tends to lie between 10^4 and 10^6 tasks. This means that by default, small kernel dispatch (less than thousands of threads) would not be able to fully leverage the computing power of a GPU. Additionally, some workflows require copying data to the GPU, running the kernel, and performing the copy back, which would leave the GPU idle during the transfer back to the CPU.

In order to circumvent that limitations the concept of streams was introduced in CUDA 7, and latter was adopted and used in other programming models. A stream can be thought of as a queue, where a kernel is enqueued for execution on the device in this stream. For example, this allows the submission of two independent kernels in two different streams. Ideally, the GPU will run them at the same time in different SMs. This means that for small kernels, in order to fill the GPU, it is possible to queue other kernels alongside the small one. This allows us to fill the GPU with kernels that would be too small individually. Streams also have the potential to be used to hide latency by enqueuing several kernels and retrieving results only later when the latency has already been paid. Also, a more common use of streams is to provide compute communication overlap, where a kernel is computing while the GPU also performs a transfer on another stream.

However, using many streams can also increase individual submission latency because the API and GPU driver will have more complex tasks to schedule.

6.8. Optimization guidelines for GPUs

With a mental model of the way GPUs works and what set its performance, it is now possible to express basic optimization guidelines when coding on GPU architectures. Firstly, in a memory-bound scenario, improving the data layout is a primary focus, as we have seen that changes in the layout can result in a factor 12 speedup on HBM memory. Secondly, we have seen that the streaming multiprocessor has multiple limitations, two of which are the register file size and the shared memory size, and that by exceeding them, the SM is forced to reduce the number of concurrent threads running simultaneously. Therefore, a primary concern in a compute-bound scenario is to target maximum occupancy by reducing register usage and shared memory usage. Additionally, because of the way SIMT works, branching can lead to large performance variations. Additionally, a bad branch pattern may invalidate the branch predictor memory prediction and speculative execution, leading to an additional performance penalty. Lastly, in a latency-bound scenario, it is best to simplify tasks asked of the API, for example, by using kernel fusion patterns (cuda graphs on NVIDIA hardware) or to queue in multiple streams to hide latency with compute.

> ! Note that in a memory-bound case, the latency of memory calls can hide the penalty caused by the presence of branching in the kernel if the memory operation is the same in all SIMT threads.

7. Expressing parallelism on GPU

7.1. Basic parallelism

We can now start to explore how a given problem can be coded on a GPU.

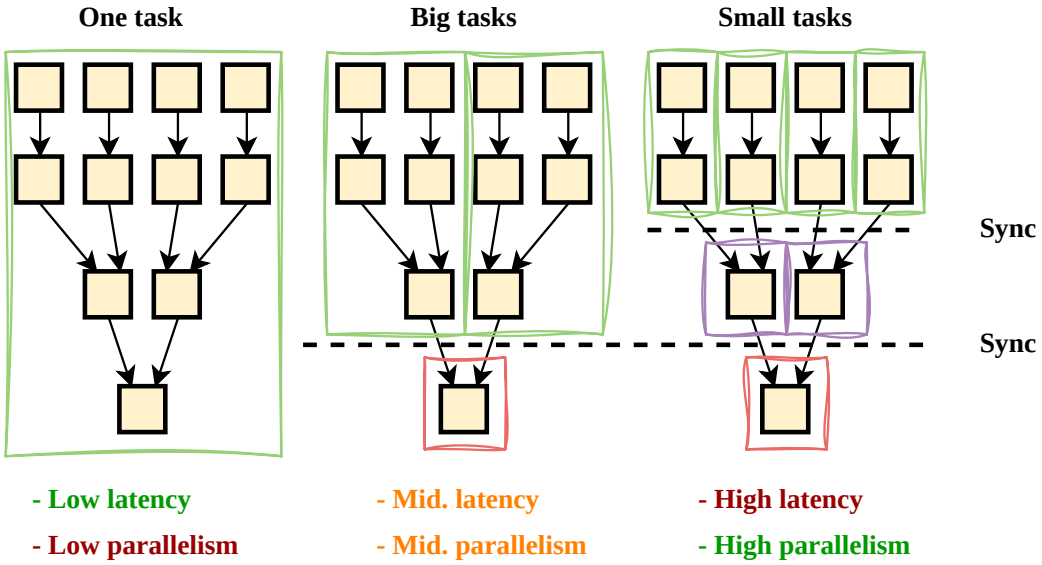


Figure 3.14: Representation of different options to express a succession of operations. Tasks surrounded by the same color are executed together in parallel.

As discussed, parallism on GPUs is expressed through compute kernels. The main difference with CPUs is that only compute kernels can be run on the GPU, and thousands of operations must be run in parallel to fully exploit its potential. This means that one must be careful in the way a given problem is decomposed on the GPU. On a CPU, it is possible to just execute only a single big task, which by definition has no parallelism but exhibits very low latency. This approach can be referred to as the one-task approach as represented on Fig. 3.14. This is typically why really low-latency applications do not parallelize on a GPU or CPU, but rather use a single task with SIMD vectorization (more details in Sec. 5.1). One such example is the WHFAST N-Body dynamics solver (Rein & Tamayo, 2015; Javaheri et al., 2023), whose focus is to maximize the number of timesteps performed with a few objects. In such a case, parallelism is inefficient and adds more latency than any speedup to the algorithm, which makes the one-task approach better in this case. Using a single

thread for WHFAST512, they achieve a staggering number of $2.4 \cdot 10^6$ iterations per second on an Intel Xeon Skylake CPU. The next level of parallelism would be to decompose a problem into a set of tasks that are interdependent. Typically, if one wants to perform a sum of complex functions, it is possible to compute each value in parallel, where each evaluation of the function is a task, and have another task that will perform the sum of the values. The latter task is dependent on all tasks that evaluate each function. Here, the complete procedure can be thought of as a task graph.

It is important to note that during the execution of a task, the GPU memory is not coherent across tasks (see Sec. 7.2), meaning that a given task cannot access in a controlled way the result of another one (except for atomic operations, see Sec. 7.2). The GPU memory is therefore coherent only when the tasks are finished, meaning that a synchronization point must be added after the function evaluations before summing the results. Such an approach adds a first level of parallelism, the trade off being, as mentioned above, that the latency of the routine is increased because of the additional synchronization. In some cases, this first level of parallelism is not enough to use the full GPU, in which case additional decomposition into smaller tasks is required. Having many small tasks will maximize parallelism at the cost of latency.

7.2. Race conditions

On a GPU, thousands of tasks are running simultaneously while working on the same data. As such, one may want to have multiple tasks that edit the same memory, for example, to perform reduction operations. However, tasks on the GPU run in parallel with an order that is not guaranteed. This means that two given tasks can edit the same memory in an undefined order. This leads to a situation called a *race condition* which here would be *undefined behavior*, where the targeted memory location is likely to get corrupted by the simultaneous writes.

Definition ▶ Race condition

A situation with two processes or threads, where the outcome depends on the order of execution.

Definition ▶ Undefined behavior

A situation where the outcome of a section of a program is undefined. Therefore, the outcome cannot be predicted.

However, as on the CPU, on the GPU, so-called *atomic operations* are available, which allows us to solve that issue.

Definition ▶ Atomic operation

An atomic operation is one that cannot leave the system in a partial state (hence the name atomic).

When using atomic operations, basically, the memory space is locked for the duration of the write. Hence, no corruption can happen, and the operation can be completed. However, the order in which the two threads are written is still undefined. This solves the corruption issue, but not the undefined behavior.

In order to resolve the issue of the undefined behavior, algorithms that are implemented should be analyzed carefully. For example, having many threads each incrementing a counter by one will not be an undefined behavior when all the threads are complete, and the count will always be the thread count. Thus, this is not undefined behavior after completion (however, it is during the execution).

! This operation can still be undefined behavior in the case of an integer overflow, which itself is undefined behavior in the C++standard. In general, one must be careful that undefined behavior in an application does not make the complete application undefined, which would likely crash if one were not able to reason about the state of the program.

7.3. Single kernel synchronization

In a context where a task is dependent on another one, instead of adding a synchronization point, one can use an atomic to pass messages across tasks. To better understand the capabilities of such an approach, we can take an example.

Typically, when performing a reduction (situation close to Fig. 3.15), we can think of four elements. A first pair of tasks will perform the addition of two elements, thus returning an array with two elements. After the first task, we must synchronize the result by waiting for the kernel to end. Lastly, we spawn a new task to sum the two remaining elements, giving us the sum of the array. Using atomic operations, we can perform a lock-free synchronization of the result. Firstly, the two tasks perform the additions as previously, but instead of performing synchronization, one of the tasks will receive through an atomic operation the partial sum and perform the last sum. As previously discussed, we synchronized some information. However, we did not issue any additional kernels. Instead, we perform synchronization during the execution of a kernel.

In general, using atomics to pass information across tasks in a kernel is a way to avoid having to enqueue multiple kernels. This means that it is possible to reduce latency using such an approach. However, it means that the tasks, because of the synchronization through atomics, are now unbalanced.

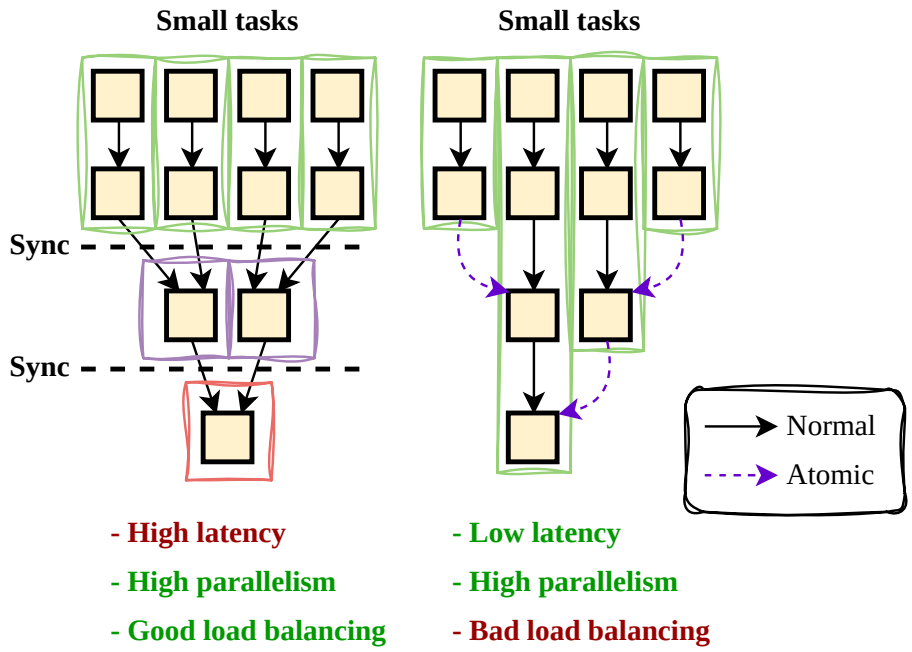


Figure 3.15: Representation of different options to express a succession of operations.

! When using atomic operations, the latency of executing such instructions is not fixed and can vary from call to call depending on where the data is located Schweizer et al. (2020). Hence specific care has to be taken when we want to involve atomic operations otherwise the kernel can be unbalanced, reducing the achieved performance.

This is the main reason why, except in a very specific case, atomics should be issued per thread block rather than by individual threads. Having unbalanced thread blocks can be compensated by using streams, and having unbalanced threads within the block would be more detrimental. One such example where the trade-off is worth taking when implementing the prefix sum algorithm as per Merrill & Garland (2016a), which allows in that specific case to achieve the best performance.

8. Coding on GPU

As stated above, on GPUs, small programs called kernels are launched. However, because GPUs use a combined SPMD and SIMT model, this yields specific needs in the language used to program the specific compute kernels associated.

8.1. History

Initially, GPUs were just designed to be able to render 3-dimensional graphics using polygons efficiently. This was done using dedicated APIs like OpenGL and DirectX with fixed pipelines where GPU vendors were implementing drivers directly. Hence, doing so-called general-purpose programming on a GPU was possible only by using hacks to force specific operations to be performed. Latter OpenGL (in version 2.0 in 2004 with GLSL) and DirectX (with C graphical also named Cg) introduced so-called shaders. Where the shader is used to program the shading of pixels or modify the geometry, it is coded using a C-like DSL (domain-specific language). In this model, the source of the shared is not in the same translation unit as the host application. This is a *multiple source GPU program*.

Definition ▶ Multiple source GPU program

A program where the source for the host CPU and the GPU are not written in the same language or the same file. In such a model, the host and device programs are assembled at compiled or runtime.

They were followed by an effort to make general purpose programming on GPUs possible to ease the development of complex rendering pipelines. Early effort includes McCool et al. (2002), where the shared program was modified to include more complex features, and Buck et al. (2004) with BrookGPU and Tarditi et al. (2006) with LibSh. In both cases, the program is composed of one language that includes both the GPU kernels and the host code. They are the first attempts at a single source-programming model for GPUs.

Definition ▶ Single source GPU program

A program where the source for the host CPU and the GPU are written in the same language or the same file. Or more strictly where both the host and the device code is within the same translation unit.

Even if those programming models have evolved a lot in the past 20 years, OpenGL and DirectX shader models still work in a similar way today. However they still target rendering pipelines more than general-purpose computing. Later, NVIDIA introduced CUDA in 2007, a single-source GPGPU language based on C. The same year AMD/ATI released Close To Metal which did not gain much traction and short-lived since AMD switched fully to OpenCL a few years later.

OpenCL, developed originally by Apple, was introduced in 2009, and backends were developed by most vendors, including NVIDIA. CUDA became popular due to the ease of using a single-source programming model, even if it meant being locked to using NVIDIA GPUs. On the other hand, OpenCL provides versatility as it works on any hardware, including CPU, FPGA, DSP, ... Following the emergence of those programming models, a modified version of OpenMP, called OpenACC Wienke

et al. (2012) was proposed to ease porting codes on GPUs. As of now, only a few hydrodynamics codes were ported to OpenACC (for example, Gadget-3 Ragagnin et al. (2020)). This is due to the current support for OpenACC that is quite sparse and has full support only on the NVIDIA stack, where CUDA is already there. This explains the low adoption rate of OpenACC, even though it can ease porting a legacy CPU codebase to GPUs. To ease porting from NVIDIA to AMD GPUs, AMD released HIP in the ROCm software stack, which, being really close to CUDA, allows porting code easily from CUDA to HIP, which is not used in the community directly as other options aiming at portability are more appealing (e.g. SYCL or Kokkos for exemple). Since version 4.0 of OpenMP, released in 2013, they now support offloading to GPUs in their target extension. Some implementations are starting to become available, with LLVM actively making progress toward natively supporting OpenMP targets. As implementations of this extension are only becoming available, not much feedback is available on its suitability to port legacy codebases. The current status of GPU programming is that many programming models are available to be able to offload code to the GPU. However, not every model supports every GPU vendor, leading to compromises.

To address that issue, multiple programming models were introduced with the common goal of being libraries or compilers written on top of multiple backends targeting existing programming models while keeping the same syntax and code. They are referred to as portable frameworks. OpenCL is one of them as it is the only standard that supports all vendors. However, OpenCL is not a single-source model and is much more complex than, for example, CUDA , hence the low adoption of OpenCL in astrophysics.

As of now, for a codebase aiming at achieving peak performance while being portable across vendors and being single source, there are two main solutions to this portability problem. The first is the Kokkos library introduced by Edwards & Sunderland (2012), which, though macros C++template metaprogramming achieves this goal while making a codebase using Kokkos capable of being compiled directly by a vendor compiler. Examples of its use in astrophysics are detailed in e.g. Lesur et al. (2023); Grete et al. (2022). The other one is the SYCL standard, which is a standard held by Khronos, which already holds OpenGL and OpenCL among many other well-adopted standards in the industry. The main idea of SYCL is that a compiler will split the device from the host code and then pass the two sections independently to the backend and frontend compilers (which can be different). This allows, for example, transpilling SYCL into PTX (the assembly behind CUDA) and then compiled using the nvidia ptx compiler into GPU-offloaded applications. Contrary to Kokkos, SYCL is a programming standard, which means that the specification is free to be implemented by anyone. Allowing for SYCL conformant code to be compiled using any implementations (ideally, since in practice small differences can exist between implementations). Additionally, being a standard means that conformance tests are written for it. We see SYCL as a promising standard being adopted in industry in general (see talks about SYCL at the C++conferences).

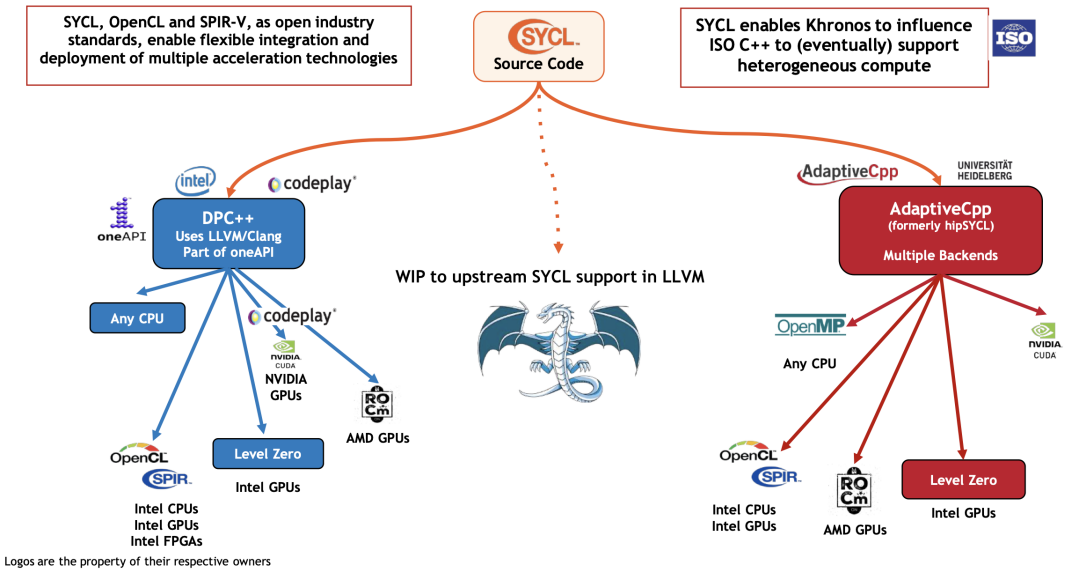


Figure 3.16: Main SYCL implementations with their offloading backends. (Source IOWCL 2024 talk : SYCL union report)

Additionally, Intel is recently working on upstreaming their SYCL compiler to the upstream LLVM clang compiler (see LLVM discourse). Lastly, it is important to mention that Kokkos and SYCL share many members in their design committees, and recently Kokkos released a SYCL backend. In our case, we went with SYCL to develop Shamrock in order to be able to potentially reach a larger pool of users and industry experts on GPU programming.

8.2. SYCL

SYCL was initially announced by the Khronos group in March 2014, with its first version publicly released being SYCL 1.2 in 2015 at the IOWCL conference. Initially, SYCL was designed as a successor or high-level single-source programming model on top of OpenCL. And it was mandating the implementation of OpenCL in the standard, which was a serious limitation of the standard. Alpay & Heuveline (2020) latter showed that it was possible to implement SYCL over CUDA and Hip with the hipSYCL SYCL implementation (now renamed to AdaptiveCPP). This allowed targeting many vendors using multiple backends, whether it was through CUDA , Hip or OpenCL. Codeplay also implemented a CUDA and HIP backend in Intel LLVM around the same timeframe. This led to the standard not mandating OpenCL in subsequent releases and instead being a standard for heterogeneous computing over any backend.

Currently, two SYCL implementations are mainly used (shown in Fig. 3.16). The first one is Intel DPC++, which is a forked and modified version of LLVM that includes SYCL support. It is available in two flavors, either through the

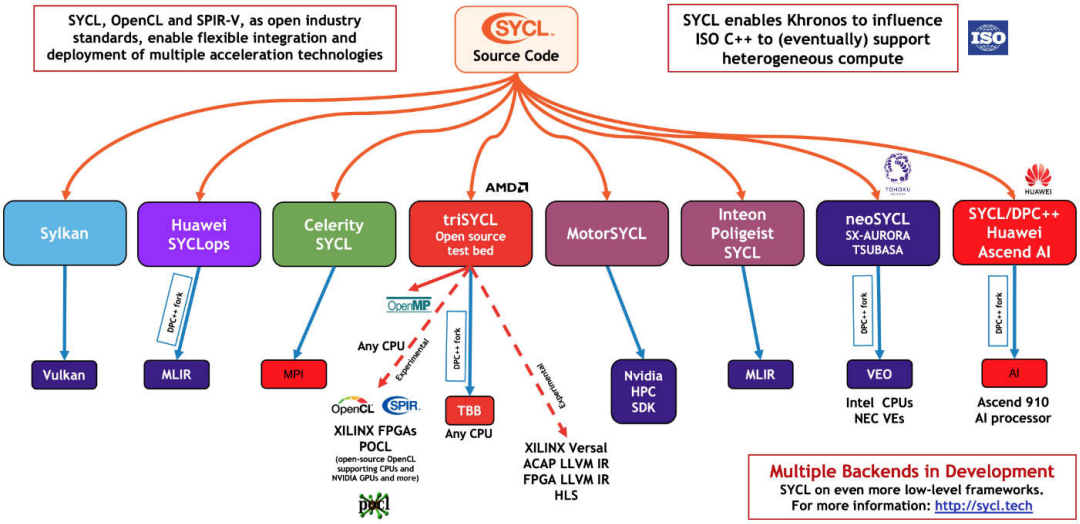


Figure 3.17: SYCL ecosystem with the other SYCL implementations. (Source Khronos website `https://www.khronos.org/sycl/`)

Intel OneAPI framework or directly using the Intel LLVM on GitHub. This implementation is the first to achieve conformance with the SYCL 2020 standard and can be compiled to multiple backends: CUDA for NVIDIA, Rocm for AMD, Level Zero for Intel GPUs, and OpenCL for Intel CPUs and GPUs (even if Intel OpenCL can be used on AMD CPUs). Additionally, they also target CPUs directly using a thread pool implementation (see Intel/llvm Github PR 13176). Intel is planning on upstreaming the SYCL compiler part of DPC++ into LLVM (see LLVM discourse). The second implementation is AdaptiveCPP, developed mainly at Heidelberg University. Instead of modifying LLVM, they instead developed a compiler driver or plugin to add SYCL support (Alpay & Heuveline, 2020) to existing LLVM installations. Currently, they can target all major vendors using CUDA, Rocm, OpenCL, and Level Zero. But they do also support an OpenMP backend and NVC++, the NVIDIA compiler from the HPC SDK, as a plugin. Also, this implementation stands out with its single-pass SYCL compiler, which first compiles the SYCL code with the device code only being lower to LLVM-IR, then during runtime the IR is compiled using just in-time compiling to the kernel code. This means that a single binary can, in theory, be run on any vendor without having to be recompiled for new backends. Lastly, many other SYCL implementations are in development, targeting various hardware, most of which are fairly new and won't be in use in this manuscript, although SHAMROCK may target them in the future for FPGA (Field Programmable Gate Arrays) offloading or to enable other backends for example.

In order to discuss the performance of SYCL implementations, a few examples can be mentioned. Earlier, using SYCL 1.2.1 was compared to Kokkos on a Lattice QCD mini-app Joó et al. (2019). This yields similar performance on NVIDIA

hardware between Kokkos and the Intel LLVM CUDA backend. This result aligns with the findings of the BabelStream project, (Deakin & McIntosh-Smith, 2020), aiming at benchmarking common kernels on multiple programming models. Overall, except for some outliers, using SYCL or a native implementation gives similar performance. Deakin & McIntosh-Smith (2020) also found that SYCL gives mostly a better performance than OpenCL and slightly lower than native CUDA .

However, those projects always test the results on specific benchmarks and not on a complete codebase. Recently, Gromacs did deprecate the OpenCL backend because it required too much effort to be maintained and introduced a SYCL backend to target Intel and AMD GPUs in addition to their heavily optimized native CUDA backend (Gromacs using SYCL on AMD GPUs). This makes Gromacs a good case to compare the performance of SYCL and CUDA. This was done by Apanasevich et al. (2023), which showed similar performance on Gromacs standard benchmarks using SYCL compared to their native CUDA implementation. They also recently performed comparison using AdaptiveCPP's HIP backend recently in Alekseenko et al. (2024), showing similar result on AMD hardware.

In our case, we are aiming towards future proofing and therefore lean toward versatility rather than vendor lock-in. Hence, this eliminates CUDA as an option. Additionally, it is important to note that in 2021 when Shamrock was started, no codes using Kokkos were yet released in astrophysics, Kokkos did not feature a stable backend for Intel GPUs, and multiple SYCL implementations were already working on all major vendors. Additionally, Intel did invest most of their compiler effort in SYCL . Thus, in 2021, we chose to use SYCL as a programming model. Maybe the result could have been different today since both Kokkos and SYCL share similar advantages and inconveniences. In the end, we will probably stick with SYCL to benefit from the efforts made in Gromacs and industry in general. Additionally, being a compiler extension, SYCL can provide potentially more optimizations in the future using MLIR, even if those would be available anyway through the Kokkos SYCL backend.

9. Coding with SYCL

In SYCL, the kernel is run on a device (which can be a GPU, a CPU, a FPGA, or any compute device). The kernel is queued on the device by the host, which is the machine hosting the compute device. The kernel is scheduled and run by a backend using an event queue, where a given action queued in the queue is linked to a corresponding event.

In SYCL, the GPU or CPU that we execute compute kernel on is called *compute device*, and the *host* refers to the CPU hosting the compute device. Note that the compute and host device can be the same CPU.

9.1. Memory model

Two ways of managing memory are available in SYCL. The first one, which was only introduced in SYCL 2020 to ease porting from CUDA or Hip is Unified Shared Memory (USM). In a USM model, the user manages pointers pointing to memory that can be on the *device* and then only accessible on the device. It can also be memory only available on the host, called host memory, or shared memory available on both the host and the device (like CUDA or Hip shared memory, for example, using Xnack for page migration in AMD).

USM allocations

```
#include <sycl/sycl.hpp>

int main() {
  //  Create a default queue
  sycl::queue q {};

  // Allocate on the host
  int* data_host = sycl::malloc_host<int>(1024, q);

  // Allocate on the device
  int* data_device = sycl::malloc_device<int>(1024, q);

  // Allocate on a both using shared memory
  int* data_share = sycl::malloc_shared<int>(1024, q);

  //free the usm pointer
  sycl::free(data_host, q);
  sycl::free(data_device, q);
  sycl::free(data_share, q);

  return 0;
}
```

In a USM application, memory is explicitly managed by the user. The other option to manage memory in SYCL is buffers, where memory is handled implicitly.

A buffer is a memory space that gets accessed when used with an accessor (see latter). The buffer data can be on the host or the device and is migrated implicitly.

Buffer allocations

```
#include <sycl/sycl.hpp>

int main() {
  //  Create a default queue
  sycl::queue q {};

  // Create a buffer with wanted size
  sycl::buffer<int> buf{1024};

  // The buffer destructor automatically free memory
  return 0;
}
```

The buffer accessor model is much simpler to use for a new application, but it can hinder flexibility since memory cannot be handled explicitly by the standard, and there is no interoperability in the SYCL 2020 standard, although it is planned for the next version of the standard. However, most implementations provide extensions for USM-buffer interoperability. In Shamrock, we have opted for the buffer-accessor model, as USM was only introduced later to the standard and was fairly new when the project was started. Additionally, the buffer accessor model provides simpler and bug-free memory management since it is automatically handled by the SYCL implementation. We do, however, plan to switch to the USM codebase, allowing for more flexibility in the management of memory and easing interoperability with vendor libraries such as NVIDA CUDA ccl libraries, fft libraries, blas libraries, etc.

9.2. Execution model

In SYCL, tasks, also called command groups, are queued in a SYCL queue. For example, if we have some input data and want to double it on the GPU, one could bind the host data to a buffer and then perform the doubling task on the GPU.

Exemple of a task with a buffer

```
#include <sycl/sycl.hpp>

int main() {
  std::vector<int> vec = .... // input data

  // Create a default queue
  sycl::queue q;

  // Using a scope force the buffer destructor
  // to write back data in the vector
```

```
  {
    // Vector data is now handled by the buffer
    sycl::buffer<int> buf { vec.data(), sycl::range<1> { vec.size() } };

    // submit a command group to the queue
    q.submit([&](sycl::handler& cgh) {
      // Request read write access to the buffer
      sycl::accessor acc { buf, cgh, sycl::read_write};

      // The command group perform a parralel for
      cgh.parallel_for(vec.size() , [=](sycl::id<1> idx) {
        // Double the value
        acc[idx] = acc[idx]*2;
      }); // End of the kernel
    });   // End of the command group
  }

  return 0;
}
```

Here we bind the data to the buffer. Then within the task that we submit to the command group, we ask for read-write access to the data. This will inform the SYCL queue that data has to be moved to the GPU, which is all implicit. At the end of the buffer lifetime the data will also be implicitely copied back in the `std::vector`.

Now to discuss the advanced capabilities of SYCL queues, let us consider the following example:

Task graph exemple

```
sycl::buffer<int> buf1 = ...; // Input data
sycl::buffer<int> buf2 = ...; // Input data
sycl::buffer<int> buf3 {buf2.size()}; // Output data

sycl::queue q {}; // default queue

//double values in buf1
q.submit([&](sycl::handler& cgh) {
    // Request read write access to buf1
    sycl::accessor acc { buf1, cgh, sycl::read_write};
    cgh.parallel_for(buf1.size() , [=](sycl::id<1> idx) {
        acc[idx] = acc[idx]*2;
    });
});

//double values in buf2
q.submit([&](sycl::handler& cgh) {
    // Request read write access to buf2
    sycl::accessor acc { buf2, cgh, sycl::read_write};
```

```
    cgh.parallel_for(buf2.size() , [=](sycl::id<1> idx) {
        acc[idx] = acc[idx]*2;
    });
});

assert(buf1.size() == buf2.size());

//buf3 = buf1 + buf2
q.submit([&](sycl::handler& cgh) {
    // Request read access to buf1 & buf2
    sycl::accessor acc1 { buf1, cgh, sycl::read_only};
    sycl::accessor acc2 { buf2, cgh, sycl::read_only};
    // Request write only access to buf3
    sycl::accessor acc3 { buf3, cgh, sycl::write_only, sycl::no_init};
    cgh.parallel_for(buf3.size() , [=](sycl::id<1> idx) {
        acc3[idx] = acc1[idx] + acc2[idx];
    });
});
```

Here the following sequence of tasks is implicitly expressing through accessors a task graph, represented on Fig. 3.18. This type of graph is called DAG, for Direct

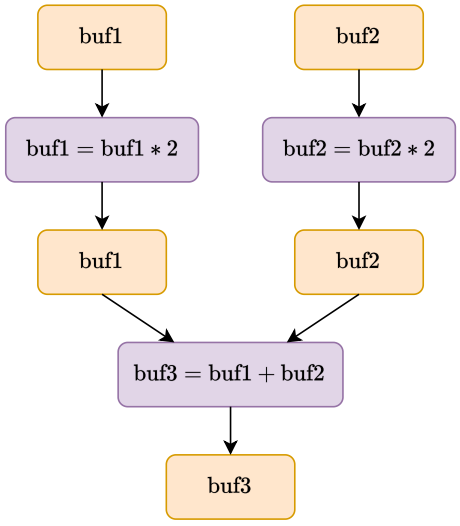


Figure 3.18: Exemple of task graph as represented in SYCL .

Acyclic Graph, as cycles cannot be represented and they must represent actions going forward in time.

9.3. SYCL datatypes

In SYCL , some vector datatypes are provided. They allow expressing multidimensional objects using native SYCL types. For example, one can:

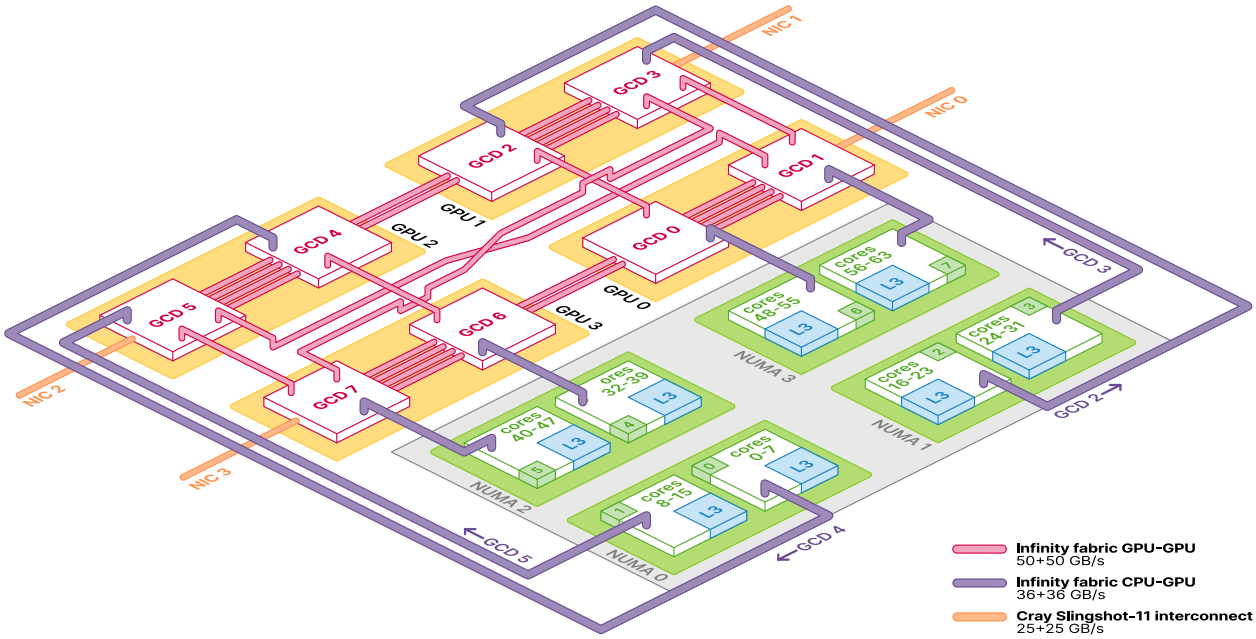


Figure 3.19: Internal block diagram of a HPE Cray EX235a node used in the Lumi, Frontier and Adastra supercomputer (Source: Lumi supercomputer documentation (2024))

Exemple of usage of SYCL vectors

```
sycl::vec<float,3> vec1 {0,1,2};
sycl::vec<float,3> vec2 {0,1,2};
sycl::vec<float,3> vec3 {0,1,2};

float result = sycl::dot(vec1 - vec2, vec3);
```

In this example, SYCL code is compiled in most cases using vectorized operations to perform the operations on SYCL vectors. This allows for the writing of complex operations without the need to write SIMD expressions manually. Additionally, mathematical functions such as cos, sin, exp are also mapped to work on vectors.

10. Multi-GPU architectures

10.1. Hardware

On multiple GPU nodes, the layout is different from a standard computer. On a standard consumer desktop computer, each GPU is connected to the CPU using a PCIe port, but they do not have GPU-to-GPU direct connections. On GPU compute nodes, however, additional GPU-to-GPU connections are provided in conjunction

with a faster-than-PCIe connection to the CPU, to avoid running into PCIe bottlenecks. See, for example, the Infinity Fabric links on a HPE Cray EX235a node shown in Fig. 3.19. Additionally, contrary to common computers, network connections are directly on the GPU to avoid the additional latency of going through the CPU for direct GPU-to-GPU communications. Additionally, to further optimize GPU-to-CPU latency, the CPU is decomposed into blocks of cores called NUMA nodes. The GPU-to-CPU links are directly connected from a GPU to a NUMA node, and in order to reduce the latency, it is best to control and exchange data between those cores and the GPU connected to the NUMA node. Fig. 3.19 shows the node topology of the CPU-to-CPU connections on a node of the LUMI-G supercomputer. Typically, in applications that are sensible on latency (e.g. Gromacs), the GPU-to-core bindings can strongly impact the performance, so taking special care to ensure that the correct block cores are connected to the correct GPU is necessary.

10.2. Usage

In practice, in order to leverage the specific topology of such a node, many approaches are possible. However, for convenience purposes, almost all multi-GPU codes, including SHAMROCK, use one MPI rank per logical GPU (as, for example, for MI250x GPUs, on the node two GPUs available per physical die). AMD calls those logical GPUs GCD for graphics compute die. Doing so, the code is written from the point of view of the GPU, as every instance of the code that is run will correspond to a GPU or GCD. Direct GPU-to-GPU communications are done using MPI GPU-aware communications, which are standard MPI calls made on memory that is owned by the GPU.

Exemple of using MPI GPU-aware with SYCL

```
#include <mpi.h>
#include <sycl/sycl.hpp>

int main() {

  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  //  Create a default queue
  sycl::queue q{};

  // Allocate on the device
  int *data_device = sycl::malloc_device<int>(1024, q);

  if (rank == 0) {
    // Fill datadevice
  }
```

```
  if (rank == 0) {
    // Send data from rank 0 using GPU aware MPI
    MPI_Send(data_device, 1024, MPI_INT, 1, 0, MPI_COMM_WORLD);
  }
  if (rank == 1) {
    // Receive data on rank 1 using GPU aware MPI
    MPI_Recv(data_device, 1024, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
  }

  // free the usm pointer
  sycl::free(data_device, q);

  return 0;
}
```

As shown, it is possible to perform GPU-aware MPI calls using memory managed by SYCL, as `malloc_device` will return a USM pointer that is owned by CUDA or Hip or the backend in general. This means that in SYCL, when using USM MPI-aware, communications can be used normally with GPU pointers instead of host pointers as usual. However, this is not the case with SYCL buffers, as the underlying memory pointer is not exposed.

Lastly, to conclude on the multi-node case, another possibility of using MPI is through so-called stream-aware MPI, which is GPU-aware MPI that is performed instead on GPU streams Zhou et al. (2022), which provides an opportunity of reducing or hindering latencies of communications further by not having to wait for other streams to finish their operations to enqueue the call.

11. Summary

We aim to develop a numerical framework that is efficient on modern computing architectures, and versatile to target all CPU and GPU hardware vendors. As detailed, we have chosen the use of SYCL standard to ensure portability across all architectures as well as performance portability, since the majority of SYCL backends currently achieve close to native performance for major GPU vendors. Aside from portability, we have seen that to develop a GPU-offloading-capable framework, it is imperative to modify the majority of algorithms and associated programming paradigms to align with GPU architectures and their distinct performance requirements. In general, porting an algorithm to GPU hardware requires extensive work. However, most GPU algorithms achieve good performance on the CPU. Therefore, we have chosen to adopt a GPU-centric approach in the development of the framework, as the CPU performance should follow. In summary, on GPUs, we need to be careful with the data ordering as it strongly impacts performance. We also need to maximize parallelism whenever possible while keeping latencies as low as possible. GPU load balancing should be regularly monitored, as it can strongly impact

the resulting performance. Lastly, to minimize performance interference between the CPU and the GPU and minimize data transfers, we try to offload almost all algorithms to the GPU and use the CPU only for task scheduling.

References

Alekseenko A., Páll S., Lindahl E., 2024, *GROMACS on AMD GPU-Based HPC Platforms: Using SYCL for Performance and Portability*, arXiv preprint arXiv:2405.01420

Alpay A., Heuveline V., in *Proceedings of the International Workshop on OpenCL*, booktitle, pp 1–1

Apanasevich L., Kale Y., Sharma H., Sokovic A. M., 2023, *A Comparison of the Performance of the Molecular Dynamics Simulation Package GROMACS Implemented in the SYCL and CUDA Programming Models*

Buck I., Foley T., Horn D., Sugerman J., Fatahalian K., Houston M., Hanrahan P., 2004, *Brook for GPUs: stream computing on graphics hardware*, ACM transactions on graphics (TOG), 777–786

Choquette J., Gandhi W., Giroux O., Stam N., Krashinsky R., 2021, *Nvidia a100 tensor core gpu: Performance and innovation*, IEEE Micro, 29–35

Cook S., 2012, *CUDA programming: a developer's guide to parallel computing with GPUs*, Newnes

Deakin T., McIntosh-Smith S., in *Proceedings of the International Workshop on OpenCL*, booktitle, pp 1–11

Dell 2019, Dell website

Edwards H. C., Sunderland D., in *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores*, booktitle, PMAM '12. Association for Computing Machinery, New York, NY, USA, pp 1–10, doi:10.1145/2141702.2141703, https://doi.org/10.1145/2141702.2141703

Grete P., et al., 2022, *Parthenon – a performance portable block-structured adaptive mesh refinement framework*, arXiv e-prints, ADS link, arXiv:2202.12309

Javaheri P., Rein H., Tamayo D., 2023, *WHFast512: A symplectic N-body integrator for planetary systems optimized with AVX512 instructions*, The Open Journal of Astrophysics, 6, 29

Jones S., 2022, *How CUDA Programming Works*, Talk given at GTC Digital Spring, https://www.nvidia.com/en-us/on-demand/session/gtcfall22-a41101/

Joó B., Kurth T., Clark M. A., Kim J., Trott C. R., Ibanez D., Sunderland D., Deslippe J., in *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, booktitle, pp 14–25

Lakshminarayana N. B., Kim H., in *Workshop on Language, Compiler, and Architecture Support for GPGPU*, booktitle,

Lesur G. R. J., Baghdadi S., Wafflard-Fernandez G., Mauxion J., Robert C. M. T., Van den Bossche M., 2023, *IDEFIX: A versatile performance-portable Godunov code for astrophysical flows*, A&A, 677, A9

Lumi supercomputer documentation 2024, *Lumi supercomputer documentation*

Makino J., Taiji M., 1998, *Scientific simulations with special-purpose computers–the GRAPE systems*

McCool M. D., Qin Z., Popa T., in *SIGGRAPH/EUROGRAPHICS Conference On Graphics Hardware*, booktitle, https://api.semanticscholar.org/CorpusID:15788889

Merrill D., Garland M., 2016a, *Single-pass parallel prefix scan with decoupled look-back*, NVIDIA, Tech. Rep. NVR-2016-002

Merrill D., Garland M., in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, booktitle, pp 678–689, doi:10.1109/SC.2016.57

NVIDIA 2024a, *CUDA C programming guide*

NVIDIA 2024b, *Nsight documentation*

NVIDIA 2024c, *PTX ISA specifications about vectors*, vectors in ptx, https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#vectors

Páll S., Zhmurov A., Bauer P., Abraham M., Lundborg M., Gray A., Hess B., Lindahl E., 2020, *Heterogeneous parallelization and acceleration of molecular dynamics simulations in GROMACS*, J. Chem. Phys., 153, 134110

Qasaimeh M., Denolf K., Lo J., Vissers K., Zambreno J., Jones P. H., in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, booktitle, pp 1–8, doi:10.1109/ICESS.2019.8782524

Ragagnin A., Dolag K., Wagner M., Gheller C., Roffler C., Goz D., Hubber D., Arth A., 2020, *Gadget3 on GPUs with OpenACC*, arXiv e-prints, ADS link, arXiv:2003.10850

Rein H., Tamayo D., 2015, *WHFAST: a fast and unbiased implementation of a symplectic Wisdom-Holman integrator for long-term gravitational simulations*, MNRAS, 452, 376-388

Schlachter S., Drake B., 2019, *Introducing Micron® DDR5 SDRAM: More Than a Generational Update*, XP055844818, May, 6

Schweizer H., Besta M., Hoefler T., 2020, *Evaluating the Cost of Atomic Operations on Modern Architectures*, arXiv e-prints, ADS link, arXiv:2010.09852

Smotherman M., 2023, *S-1 Supercomputer (1975-1988)*, Website

Sorensen T., Evrard H., Donaldson A. F., in *International Conference on Concurrency Theory*, booktitle, https://api.semanticscholar.org/CorpusID:51995301

Tarditi D., Puri S., Oglesby J., in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, booktitle, ASPLOS XII. Association for Computing Machinery, New York, NY, USA, pp 325–335, doi:10.1145/1168857.1168898, https://doi.org/10.1145/1168857.1168898

TechPowerUp 2024, *NVIDIA Tesla V100 PCIe 16 GB Specs*, GPU Database

The next platform 2023, *How AI is going to change supercomputer rankings even more*

Top500 2024, *Top500 ranking*, Top500

Von Neumann J., 1993, *First Draft of a Report on the EDVAC*, IEEE Annals of the History of Computing, 27–75

Walker D. W., 1992, *Standards for message-passing in a distributed memory environment*

Wienke S., Springer P., Terboven C., an Mey D., in *Euro-Par 2012 Parallel Processing:*

18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings 18, booktitle, pp 859–870

Zhang L., Wahib M., Matsuoka S., 2019, *Understanding the overheads of launching CUDA kernels*, ICPP19, 5–8

Zhou H., Raffenetti K., Guo Y., Thakur R., in *Proceedings of the 29th European MPI Users' Group Meeting*, booktitle, pp 1–10

Chapter 4

Shamrock

Contents

Foreword

In this chapter, we present the core work of this Ph.D. Thesis, that is the framework at the core of the Shamrock code. This work corresponds to the majority of the time spent on this Ph.D. Thesis. In a few numbers, in the past 3 years, the development of the code corresponds to

- 441100 lines of code written, and 309466 removed (see Fig. 4.1),
- 310 merged pull requests,
- 2,244 CI workflow run,
- 528 C++ source files,
- Two broken DGX-workstation, and two laptops...

As of now, the code includes 4 numerical solvers:

- SPH: Fully tested up to 256 GPU nodes.
- Zeus: Prototype for the design of the grid in Shamrock, partially tested on basic problems, scaling on multiple GPUs not assessed yet.
- Godunov (RAMSES): Prototype of Godunov AMR solver. The scheme is fully implemented using AMR, but partial tested as of now, the dynamic refinement during simulation has not yet been tested in hydrodynamical simulations.
- N-Body FMM (Appendix E): A prototype implementation of an FMM self-gravity solver in Shamrock, tested on single GPU, ported to multi-GPU, but its performance was not yet fully assessed.

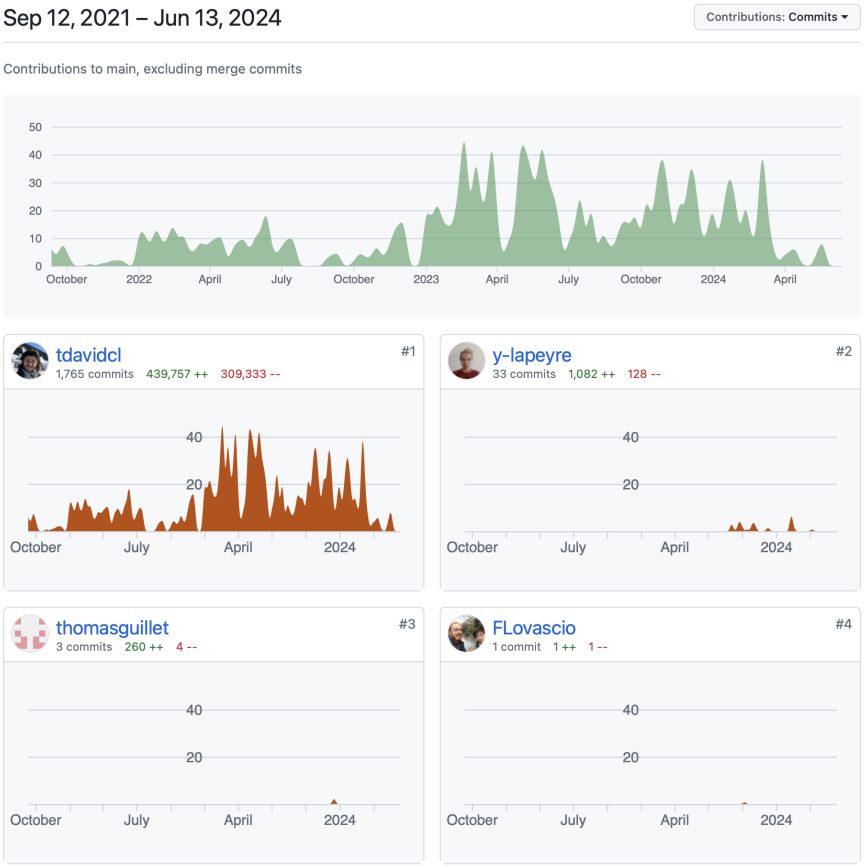


Figure 4.1: Github statistics on the history of contributions to the git repository since its creation.

As of now, the code has a total of 3 contributors aside from myself. The contribution list is as follows:

- Yona Lapeyre (intern, Ph.D. starting in September 2024 with Guillaume Laibe): Collaboration in the Sedov-Taylor blast comparison against PHANTOM and the implementation of artificial conductivity. Implementation of α-disc viscosity in SPH and warp disc simulation setup. Currently working on the extension of the SPH solver to MHD.
- Francesco Lovascio (postdoctoral researcher at CRAL now working at ARM): Collaboration in the design of the ZEUS scheme implementation in SHAMROCK and the shearing box in SPH.
- Thomas Guillet (postdoctoral researcher at Exeter): collaboration in the design of the Godunov scheme implementation in SHAMROCK, and implementation of the Riemann solvers in the code.

The work following this chapter was recently submitted to MNRAS, along with the details of the SPH solver presented in Chapter 5.

1. Introduction

The study of the formation of structures in the Universe is a field in which non-linear, non-equilibrium physical processes interact at many different scales, requiring ever greater computing resources to simulate them, right up to Exascale (one quintillion operations per second). To increase energy efficiency with acceptable CO_2 emissions, recent super computers have been designed with specialised hardware such as ARM central processing units (CPUs) or graphics processing units (GPUs). Those involve multiple computational units that perform the same operation on multiple data simultaneously through specific instructions (Single Instruction Multiple Data, or SIMD parallel processing). This type of hardware differs radically from standard x86 CPUs, requiring a complete rewrite of CPU-based codes.

Considerable efforts have recently been invested into developing codes adapted to the new hybrid architectures aimed at Exascale (e.g. Idefix: Lesur et al. 2023; Parthenon: Grete et al. 2022; Quokka: Wibking & Krumholz 2022). The performance of those codes is conditioned by the rate at which data involved in the solver can be prepared, explaining the efficiency of grid-based Eulerian codes developed to date. For example, the multiphysics Godunov code Idefix uses a fixed grid, so no overhead is required when executing the numerical scheme. On the other hand, simulating moving disordered particles on Exascale architectures is a tremendous challenge, regardless of whether they are tracers for Eulerian methods, super particles for Lagrangian methods or interpolation points for Fast Multiple Moments. The rule of thumb is that performance decreases when the number of neighbours increases and when they are unevenly distributed.

Our code Shamrock is a performance portable framework aiming at hybrid CPU-GPU multi-node Exascale architectures (Sect. 2). The design of Shamrock makes it appealing for with particle-based methods such as Smoothed Particle Hydrodynamics (e.g. Hopkins 2015; Price et al. 2018; Springel et al. 2021), while remaining inherently compatible with any distribution of numerical objects (grids, particles) and numerical schemes (grid-based or Lagrangian). Our strategy in Shamrock is that the tree used for neighbour search is never updated, unlike in existing codes. Instead, we are aiming for a highly efficient fully parallel tree algorithm that allows on-the-fly building and traversal, for any distribution of cells or particles. The specific nature of GPU architectures calls for a different design from the state-of-the-art methods developed for CPUs (e.g. Gafton & Rosswog 2011). The simulation domain undergoes an initial partitioning into sub-domains, fostering communication and interface exchange through an outer layer of MPI parallelism presented in Sect. 3. The core of Shamrock is its inner layer of parallelism, which consists in distributing the operations performed for the hydrodynamical solver on a sub-domain over the GPUs using the SYCL standard. The overall performance of Shamrock hinges on the performance on neighbour finding *on a single GPU*. Hence the need for a tree building and traversal procedure that doesn't bottleneck the hydrodynamical time step. In Sect. 4, we first present a tree algorithm that has

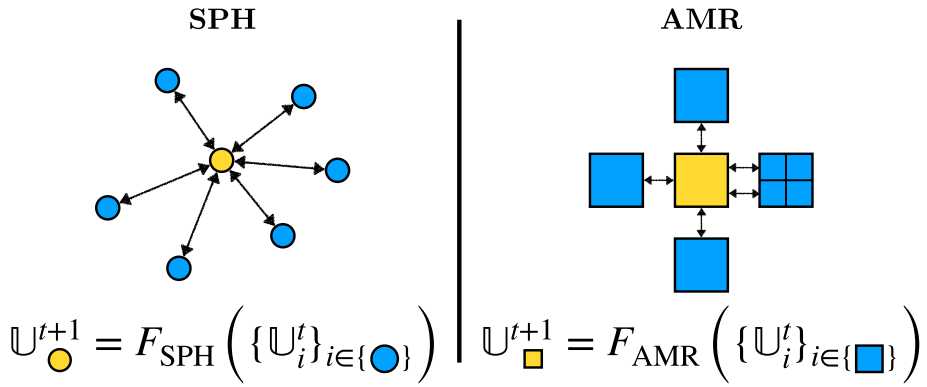


Figure 4.2: Numerical integration of an hydrodynamic quantity $\mathbb{U}$ involves finding neighbours i (particles, cells), then adding their contributions according to the chosen solver F.

the required level of performance for any number of objects compatible with current GPU capabilities. It combines state-of-the-art algorithms on Morton codes (Morton, 1966; Lauterbach et al., 2009) with specific optimisations, a key feature being the Karras algorithm (Karras, 2012). The resulting tree building time is negligible. We note that the subsequent implementation of a Smoothed Particle Hydrodynamics solver (SPH) in Shamrock will be presented chapter 5, in Sect. 1, and present the results obtained on standard astrophysical tests in Sect. 2. Our implementation is almost identical to that of the Phantom code, facilitating performance assessments and comparisons on both one and multiple GPUs (Sect. 3). We discuss potential future directions for Shamrock in Sect. 2.

2. The Shamrock framework

2.1. Modular computational fluid dynamics

Computational fluid dynamics consists in discretising a physical system of partial differential equations, alongside specifying initial and boundary conditions. Deterministic numerical schemes can be viewed as being the combination of neighbour finding and specified arithmetic (see Fig. 4.2), and an algorithm capable of operating on neighbours can provide a generic framework for implementing schemes frequently used in astrophysics, such as Lagrangian Smoothed Particle Hydrodynamics (SPH), Eulerian Adaptive Mesh Refinement (AMR) grid-based methods, or others, within the same structure. This is the very purpose of Shamrock: to abstract optimised neighbour search, in a way that is versatile enough that the user only needs to provide functionality to write new schemes with minimal changes. Fig.4.3 sketches the Shamrock framework: a collection of libraries connected by standardised interfaces, where models (CFD solvers or analysis modules) are implemented atop these libraries.

2.2. Multi-GPUs architectures: choice of languages and standards

Modern computer hardware harnesses graphics processing units (GPUs) as computing accelerators. A typical compute node configuration consists of several GPUs connected to a CPU via a PCI express or other proprietary interconnect. Each GPU is equipped with its network card, enabling direct communications from one GPU to another with the Message Passing Interface (MPI) protocol. The GPUs themselves are specialised hardware capable of exploiting the full bandwidth of their high speed memory in tandem with a high compute throughput using SIMT (Single instruction, Multiple Threads) and SIMD (Single Instruction Multiple Data). This design renders GPUs more potent and energy-efficient than CPUs, especially for processing parallelized tasks involving simple, identical operations. Performing simulations on architectures comprising thousands of GPUs introduces several challenges: evenly distributing the workload among the available GPUs (load balancing problem), communicating data between domains to perform the computation while moving the communication directly to the GPU if possible (communication problem), structuring and organising the workload on the GPU into GPU-executed functions called *compute kernels* to make the best use of hardware capabilities (algorithmic problem). The first two points are common issues associated with MPI, while the third is specific to GPU architecture, raising the question of choosing an appropriate backend.

GPU vendors have developed various standards, languages and libraries to handle GPU programming, the most widely used to date for scientific applications being CUDA and ROCm , which are vendor-specific. To address the issue of portability, libraries and standard have been created to enable the same code to be used on any hardware from any vendors. Current options include Kokkos (Trott et al., 2021), OpenACC and OpenMP (target). The SYCL standard, released by Khronos in 2016, is a domain-specific embedded language compliant with C++17, which is compiled to the native CUDA , ROCm or OpenMP backend. With a single codebase, one can directly target directly any GPUs or CPUs from any vendors, eliminating the need for separate code paths for each supported hardware. To date, the two main SYCL open source compilers are AdaptiveCpp (Alpay & Heuveline, 2020; Alpay et al., 2022), and OpenAPI/DPC++, which is maintained by Intel. Among other heterogeneous parallelisation libraries, we use the SYCL standard to develop Shamrock, since it offers robustness, performance (Markomanolis et al., 2022), portability (Deakin & McIntosh-Smith, 2020; Jin & Vetter, 2022) and potential for durability. SYCL compilers can also generally compile directly to a native language without significant overhead, delivering near-native performance on Nvidia, AMD and Intel platforms (e.g. tests with Gromacs, Alekseenko & Páll 2023; Abraham et al. 2015; Alekseenko et al. 2024). Since C++ code written using SYCL is compiled directly to the underlying backend (CUDA or ROCm or others), we harness direct GPU communication and use vendor libraries directly in the code.

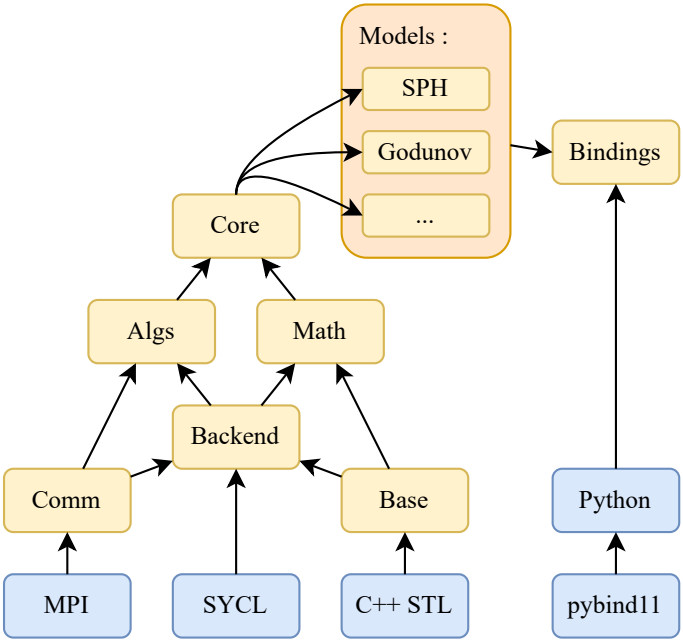


Figure 4.3: Internal structure of SHAMROCK: functionalities for calculating neighbour finding are organised in different layers of abstraction, enabling the independent treatment of any numerical scheme (Models).

2.3. Elements of software design

Software design of SHAMROCK relies on

- A modular organisation of the code structured around interconnected cmake projects,
- Python bindings provided through the use of pybind (Jakob et al., 2024),
- Version control development for forking and branching (Git),
- A comprehensive, automated test library handling multiple configurations of compilers, targets and versions,
- Automated deployment of code across machines by the mean of environment scripts,
- A user-friendly PYTHON frontend for versatility.

Further details are provided in Sect. 4

3. Domain decomposition & MPI

3.1. Simulation box

The three-dimensional volume on which a numerical simulation is performed can be embedded in a cube, whose edges define axes for Cartesian coordinates. This cube is often referred as an Aligned Axis Bounding Box, or $AABB$, particularly within the ray-tracing community. The box is parametrised by two values, r_{min} and r_{max}, which are chosen to represent the minimum and maximum possible coordinates inside the cube in all three dimensions. For convenience, we shall refer to this $AABB$ as the *simulation box* of Shamrock. Within this box, coordinates can be mapped to a grid of integers, by subdividing the simulation box coordinates into N_{g} grid points on each axis, where N_{g} is a power of two. In pratice, we use $N_{\text{g}} = 2^{21}$ or $N_{\text{g}} = 2^{42}$ (see Sect. 3.5).

3.2. Patch decomposition

The first level of parallelisation in Shamrock consists of dividing the simulation box into elementary volumes, or subdomains, which are then distributed across nodes of a computing cluster. For convenience, we shall further refer to these subdomains as *patches*. In Shamrock, patches are constructed following a procedure of recursive refinement. Starting from the simulation box, patches are divided into eight patches of equal sizes by splitting in two equal parts the original patch on each axis. The resulting structure is an octree, where each node is either a leaf or an internal node with eight children. The patches managed by Shamrock are the leaves of this octree. We call this structure the *patch octree* of Shamrock. The patch octree is similar to the structure of a three-dimensional grid that has been adaptively refined (AMR grid). The cells of this AMR grid would correspond to the patches of Shamrock. Similar to an AMR grid, patches can be dynamically subdivided or merged.

To each patch p, we associate an *estimated load* $\mathcal{W}_p$, which is an estimate of the time required to perform the computational load on the patch. The load depends *a priori* on the type of simulation chosen by the user (e.g. fixed or refined grids, particles, see Sect. 3.5). If the estimated load of a patch exceeds a maximum threshold ($\mathcal{W}_p > \mathcal{W}_{\text{max}}$), the patch is subdivided. If the estimated load is below a minimum threshold ($\mathcal{W}_{\text{min}}$), the patch is flagged for a merge operation. In Shamrock, patch merging is performed when all eight patches corresponding to the same node in the patch octree are flagged. To avoid cycling between subdivisions and merges, we enforce $\mathcal{W}_{\text{max}} > 4\mathcal{W}_{\text{min}}$. Hence, the decomposition of the simulation box into patches is only controlled by the values of $\mathcal{W}_{\text{min}}$ and $\mathcal{W}_{\text{max}}$. Shamrock maps several patches to a given MPI rank in a dynamical manner. We call this decomposition an *abstract domain decomposition*. In practice, we find that 10 patches per MPI rank provides a compromise between the level of granularity required for effective load balancing

and the overheads associated with patch management.

3.3. Data Structure

Each patch in SHAMROCK is associated with two types of information. The first type is the *patch metadata*, which encompasses the current status, location and identifier of the patch. The second, called *patch data*, comprises the data pertaining to the fields processed by the patch.

3.3.1. Patch metadata

Within SHAMROCK, metadata is synchronised across all MPI ranks. This synchronisation is made possible by the use of a class of small size (80 bytes when compiled). The metadata of a SHAMROCK patch is represented in the code with the following class

```
template<u32 dim>
struct Patch{
    u64 id_patch;
    u64 pack_node_index;
    u64 load_value;

    std::array<u64,dim> coord_min;
    std::array<u64,dim> coord_max;

    u32 node_owner_id;
};
```

In this class, `u32` denotes 32 bits unsigned integers and `u64` their 64 bits variants, `id_patch` the patch unique identifier, `load_value` the estimated load of a patch (see Sect. 3.2), `coord_min` and `coord_max` represent edges of the AABB patch on the integer grid, `node_owner` the MPI rank owning the current patch. Finally, `pack_node_index` is an additional field used to specify that a patch aims to reside in the same MPI ranks as another one (see section 3.4 for more details). We also provide a dedicated MPI type to facilitate the utilisation of collective operations on patch metadata.

3.3.2. Patch data

The *patch data* of a patch is a list of fields related to a collection of objects (cells or particles). A field can contain one or multiple values per object, as long as the number of values per object is constant. The first field, so-called the *main field* in SHAMROCK, must have one value per object and stores the positions of every object in the patch. Domain decomposition and load balancing are executed based on the

positions stored in the main field. When a patch is moved, split or merged, the corresponding operations are applied to the other fields as well. This ensures that communications are implicitly modified when the layout of the data is changed, eliminating the need for direct user intervention. For efficient implementation of new physics, the fields stored in the patch data can encompass a wide range of types (scalar, vector, or matrices, with float, double, or integer data), arranged in any order. This versatility is enabled by representing the patch data as a `std::vector` of `std::variant` encompassing all possible field types. This aspect is abstracted from the user, as only field identifiers and types are required. One example of such use is

```
PatchData & pdat = ...
// get the layout of the patch data
PatchDataLayout &pdl = pdat.pdl;
// get id of the field (name and type specified)
// f64_3 is a 3 dimensional double precision vector
const u32 ivxyz      = pdl.get_field_idx<f64_3>("vxyz");
// get the field at this id
PatchDataField<f64_3> & vxyz =
                          pdat.get_field<f64_3>(ivxyz);
```

3.3.3. Patch scheduler

In Shamrock, a single class is responsible of managing patches, distributing data to MPI ranks and processing the refinement of the patch grid. This class contains the patch octree, patch metadata, and patch data. It is referred internally as the `PatchScheduler`. This class is only controlled by four parameters: the patch data layout, which specifies the list of fields and the corresponding number of variables, the split criterion $\mathcal{W}_{\text{max}}$ and the merge criterion $\mathcal{W}_{\text{min}}$ that control patch refinement, and the load balancing configuration. The patch scheduler is designed to operate as a black box for the user. The user calls the `scheduler_step` function, which triggers the scheduler to execute merge, split, and load balancing operations. The `scheduler_step` is called at the beginning of every time step in practice. Multiple 'for each' functions are provided in Shamrock as abstractions for iterating over patches. An example of such use is

```
PatchScheduler & scheduler = ...

scheduler.for_each_patchdata(
    // the c++ lambda contain the operation
    // to perform on the patches
    [&](const Patch & p, PatchData &pdat) {
        // do someting on the patch
    }
);
```

These abstractions shield the end user from interactions with the MPI layer. The strategy is as follows: one does not need to be aware of which patches reside on which MPI ranks. Indeed, operations are conducted solely through 'for each' calls to the patches, and the scheduler handles the other tasks.

3.4. Scheduler step

Fig. 4.4 illustrates a single scheduler step in SHAMROCK. During this step, patch data are exclusively processed on their current MPI rank, while patch metadata and the patch tree remain unchanged over all MPI ranks.

3.4.1. Synchronising metadata

The initial operation conducted during a scheduler step consists in synchronising the metadata across the MPI ranks. This operation, named `vector_allgatherv` in SHAMROCK, is implemented as an extension of the MPI primitive `MPI_ALLGATHER_V` (see fig.4.5). Given a `std::vector` in each MPI ranks, `vector_allgatherv` returns on all ranks the same `std::vector` made by concatenating the input vectors in each ranks. We create an MPI type for the patch metadata, and use `vector_allgatherv` to gather all the metadata of all patches on all MPI ranks. This operation returns the list containing the metadata of all patches in the simulation (the step 'Metadata sync' in Fig.4.4).

3.4.2. Listing requests

The operation 'Get requests' depicted in Fig.4.4 provides the list of identifiers for patches requiring merging or splitting. A patch splits when its estimated load exceeds the split criterion (see Sect.3.2). If all children of a node in the patch tree meet the merge criterion, they merge, resulting in the parent node being marked for pending child merge and consequently transitioning into a tree leaf.

3.4.3. Patch splitting

Subsequent split operations on the metadata and the patch tree are carried out in each MPI rank. If the MPI rank holds the patch data associated with the patch being split into eight new patches, the patch data is then subdivided into eight new patch data objects corresponding to the eight newly formed patches.

3.4.4. Collecting information on ranks

The *pack index* is a list containing necessary information indicating whether a given patch a must reside in the same MPI rank as another patch b. After having executed patch splitting, we then go through the list of merge operations along the MPI ranks. We use an identifier that denotes the parent of the eight merging children patches. With the exception of the first child, all the other children patches have their pack

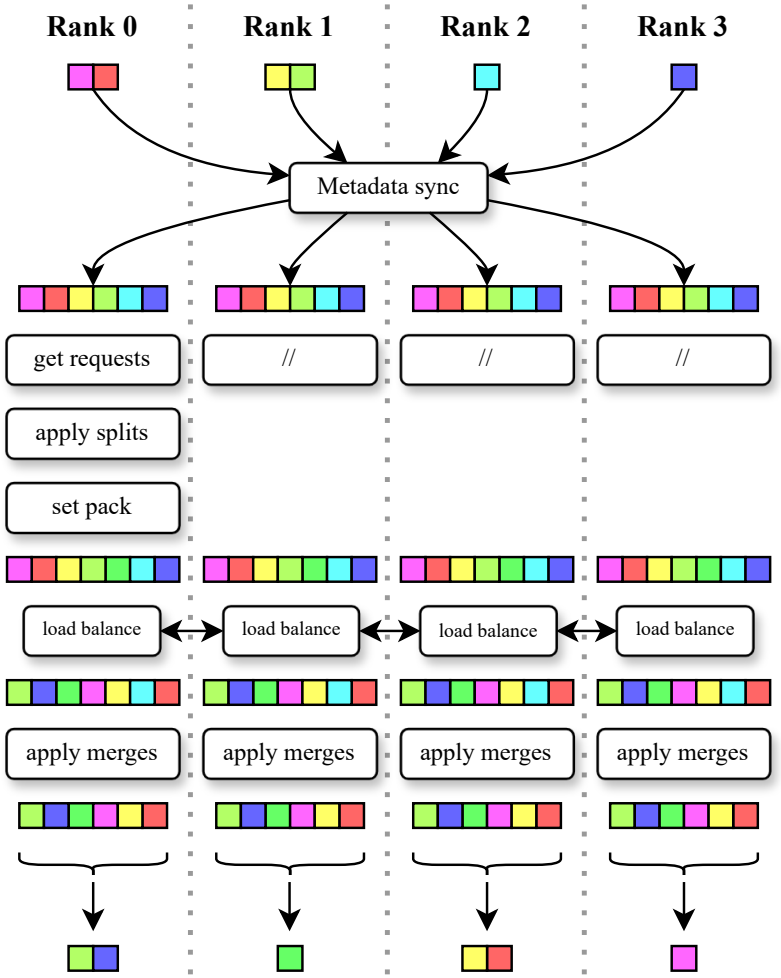


Figure 4.4: Illustration of a scheduler step. Initially, a synchronisation of the patch metadata occurs across all MPI ranks, resulting in each rank possessing an identical list of all patch metadata. Subsequently, each MPI rank generates a list of split and merge requests. Split requests are then executed, followed by setting the packing index. The subsequent operation consists in performing load balancing on all patches. Finally, merge requests are carried out to complete the step.

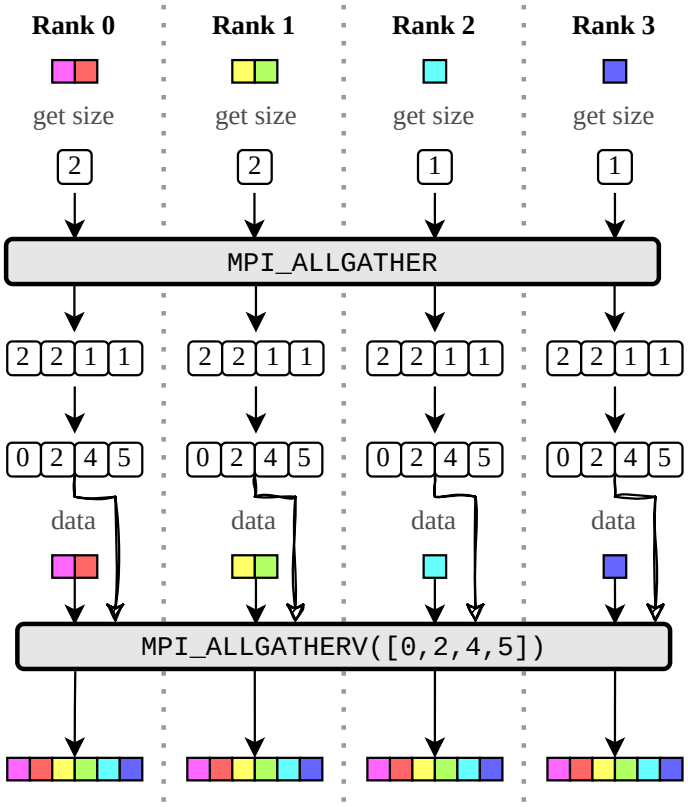


Figure 4.5: Illustration of the steps performed in a `vector_allgatherv` operation. Firstly, the size of each sent vector is retrieved. Secondly, all MPI ranks gather the sizes sent by each other MPI ranks. An exclusive scan is performed on this list to obtain the offset at which the data of each MPI rank will be inserted in the final vector. Finally, a `MPI_ALLGATHER_V` is called using the provided offsets to retrieve the gathered list in each MPI rank.

index set to the index of the first child in the global metadata list. This means that the seven other children patches must be in the same MPI rank as the first one, which enables the merge operation to be conducted at a later stage. The pack index is used during the subsequent load balancing step.

3.4.5. Load balancing

Performing load balancing consists of grouping patches in chunks, and distributing the chunks appropriately over the MPI ranks for their computational charge to be as homogeneous as possible. Load balancing is performed in four sub-steps:

- The load balancing module receives a list of metadata that includes estimated computational loads. Here, a strategy for patch reorganisation of patches is computed (see Sect. 3.5 for the details of the load balancing procedure). The code returns a list specifying the novel MPI rank assignment for each patch. When compared to the current owner of each patch, this list identifies necessary changes, indicating if a patch must move from one MPI rank to another. Additionally, this list incorporates the pack index described above.

- The patch reorganisation encoded in the list of changes is subsequently implemented. We iterate through the list of changes a first time. If the sender MPI rank matches the MPI rank of the current process, it sends the corresponding patch data using a non-blocking send of the serialized data (see 3.7 for details).

- We then go through the list of changes for a second time. If the receiver patch matches the MPI rank of the current process, we execute an MPI non-blocking receive operation to obtain the corresponding patch data in the new rank.

- Finally, we finish by waiting for all MPI operations to complete, thereby concluding the load balancing step.

Generating the list of changes accounts for the pack index. As such, patches intended for merging together are in the same MPI rank after the load balancing operation.

3.4.6. Patch merging

Merge operations require for the eight children patches to be in the same MPI rank, which is garanteed by the packing in the load balancing step. Similarly to split operations, merges are executed across both metadata and the patch tree within all MPI ranks. Merging is also applied to the patch data on the MPI rank that owns the data.

3.5. Load balancing strategies

The load balancing module generates the list of owner of each patch determined by abstract estimates of its required computational load. Load balancing is processed consistently across all MPI ranks for identical inputs. The load balancing module initially utilises the list of all patch metadata, with the estimated load values as input. Patches are then arranged along a Hilbert curve, which is subsequently segmented into contiguous chunks of adjacent patches. The objective of optimal load balancing is to identify a collection of chunks wherein the workload is distributed as evenly as possible across all MPI ranks.

To achieve this, various load balancing strategies are dynamically evaluated in Shamrock (e.g. analytic decomposition, round-robin method), and the one found to be the most effective is selected. The computational overhead involved in assessing the benefits of different load balancing strategies is minimal, since it relies on simple estimations. This process yields a list specifying the new MPI ranks for each patches. This list is compared against the current distribution of patches to generate the change list when load balancing is applied.

3.6. Patch interactions

3.6.1. Interaction criteria

For a given collection of objects (cells or particles), we can establish a condition indicating whether objects i and j interact, and define a Boolean interaction criterion $\gamma_{o/o}(i,j)$ to signify this condition. For example, in the Smoothed Particle Hydrodynamics method, $\gamma_{o/o}$ is defined as true when particle i is within the interaction radius of particle j, or vice versa.

A first generalisation of this object-object criterion is an object-group criterion, which describes if there is interaction between an object and a group of objects. A necessary condition for such a criterion is

$$\gamma_{o/g}(i,\{j\}_j) \Leftarrow \bigvee_j \gamma_{o/o}(i,j)$$

This condition formally expresses the fact that if the interaction criterion is fulfilled for any object in the group, it must also hold true for the entire group. Failure to meet this condition would imply the possibility of interaction with an element of the group without interaction with the group as a whole, which is incorrect. Both the object-object and group-object criteria are used in the tree traversal step (detailed Sec. 4.12). Another extension of the aforementioned criteria is the group-group interaction criterion, which similarly satisfies the following condition

$$\gamma_{g/g}(\{i\}_i,\{j\}_j) \Leftarrow \bigvee_i \gamma_{o/g}(i,\{j\}_j)$$

This latter condition is used to manage ghost zones (see Sect. 3.6.2) and perform two-stages neighbour search (see Sect. 4.14).

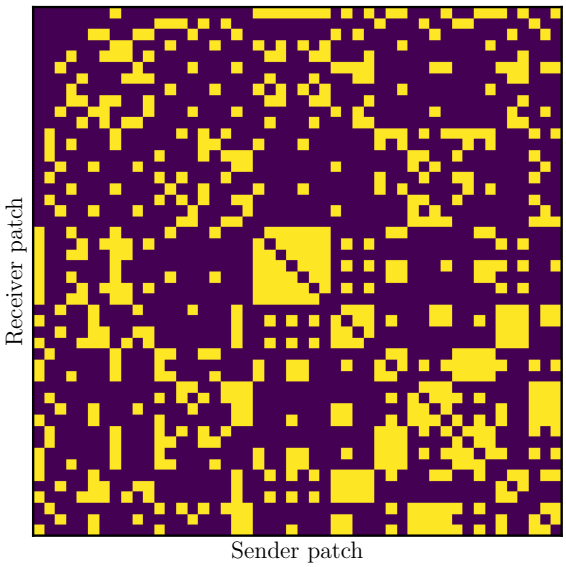


Figure 4.6: Matrix of the interaction graph between patches extracted from an SPH simulation of a protoplanetary disc, involving several hundred patches. Empty patches have been excluded from the graph as they do not meet any interaction criteria due to their emptiness. The resulting interaction matrix is symmetrical and sparse. This visualisation was generated using a debug tool in SHAMROCK, which creates a dot graph representing the ghost zones of the current time step, which can be rendered in its matrix form here showed.

3.6.2. Interaction graph

Patches themselves are objects that can interact. Their interaction is handled using a group-group interaction criterion $\gamma_{g/g}$. The interaction criterion of an empty patch is always false. After assessing the interaction status between all pairs of patches, we define the *interaction graph* of the patches by considering the list of links such that the group-group interaction criterion $\gamma_{g/g}$ is true. Fig. 4.6 shows an example of such a graph of interactions between patches.

3.6.3. Interfaces and ghost zones patch

We define the *interface between two patches* as the smallest set of individual objects for which the group-group interaction criterion between their parent patches is satisfied. To reduce communications between interacting patches, we communicate not the entire patch content to its neighbour, but only their interfaces. These communicated interfaces consequently manifest as ghost extensions for the neighbouring patch and are therefore called *ghosts zones* of patches. The graph of ghost zones between patches is the same as the interaction graph, with links between vertices representing the ghost zone of one patch being sent to another patch.

3.7. Serialisation

In Shamrock, all communications are serialised, i.e. converted into a stream of bytes to reduce the MPI overhead by performing less operations, and shield the user from the MPI layer. To send a patch ghost zone, data are initially packed into a byte buffer. Communication patterns and operations remain therefore unchanged, regardless of the communication content. In particular, the addition of a field simply adds extra data to the serialisation without altering the communication process.

```
// Data to be serialized
std::string str = "exemple";
sycl::buffer<f64_3> buffer = ...;
u32 buf_size = buffer.size();

SerializeHelper ser;

// Compute byte size of header and content
SerializeSize bytelen =
      ser.serialize_byte_size<u32>()
    + ser.serialize_byte_size<f64_3>(buffer.size())
    + ser.serialize_byte_size(test_str);

// Allocate memory
ser.allocate(bytelen);

// Write data
ser.write(buf_size);
ser.write_buf(buffer, n2);
ser.write(test_str);

// Recover the result
sycl::buffer<u8> res = ser.finalize();
```

```
// The byte buffer
sycl::buffer<u8> res = ...;

// Give the buffer to the helper
shamalgs::SerializeHelper ser(std::move(res));

// Recover buffer size
u32 buf_size;
ser.load(buf_size);

// Allocate buffer and load data
sycl::buffer<f64_3> buf (buf_size);
ser.load_buf(buf, buf_size);

// Read the string
std::string str;
ser.load(recv_str);
```

Serialisation in Shamrock relies on a split header data approach. Individual values are stored in the header on the CPU, while buffer data is stored on the device (CPU or GPU). This organisation ensures that individual value reads incurs minimal latency, thus avoiding high GPU load latency. The entire buffer is only assembled on the device at the end of the serialisation procedure. During deserialisation, the header is initially copied to the CPU. To circumvent constraints imposed by the CUDA backend, all reads and writes are adjusted to 8-byte length.

3.8. Sparse MPI communications

In hydrodynamical simulations, interactions among objects are predominantly local, resulting in each patch being connected to only a limited number of other patches in the interaction graph. A crucial element of communication management in Shamrock is to uphold this sparsity. With synchronised metadata, each MPI ranks holds information of the MPI rank to which every patch belongs. We therefore group communication between patches involving the same pair of MPI ranks in a single *patch message* (see Fig. 4.7). The graph corresponding to patch messages to be communicated is also sparse (rank $i \mapsto$ rank j). We therefore apply an MPI operation that extends `MPI_Alltoall` to accommodate a sparse graph structure. The operation, referred to as *sparse all-to-all*, is structured as depicted in Fig. 4.8. Initially, we compile the list of communications to be executed on each node. Subsequently, on each node, we go through the communication list and execute a non-blocking MPI send if the sender's rank matches the current MPI rank. Following this, on each node, we go through the communication list once more and initiate a non-blocking MPI receive if the recipient's rank aligns with the current MPI rank. Finally, we conclude the operation by invoking an MPI wait on all non-blocking communication requests. Exchanges within the patch ghost zone graph are finalised once the sparse all to all operation is completed. If the sender's MPI rank matches that of

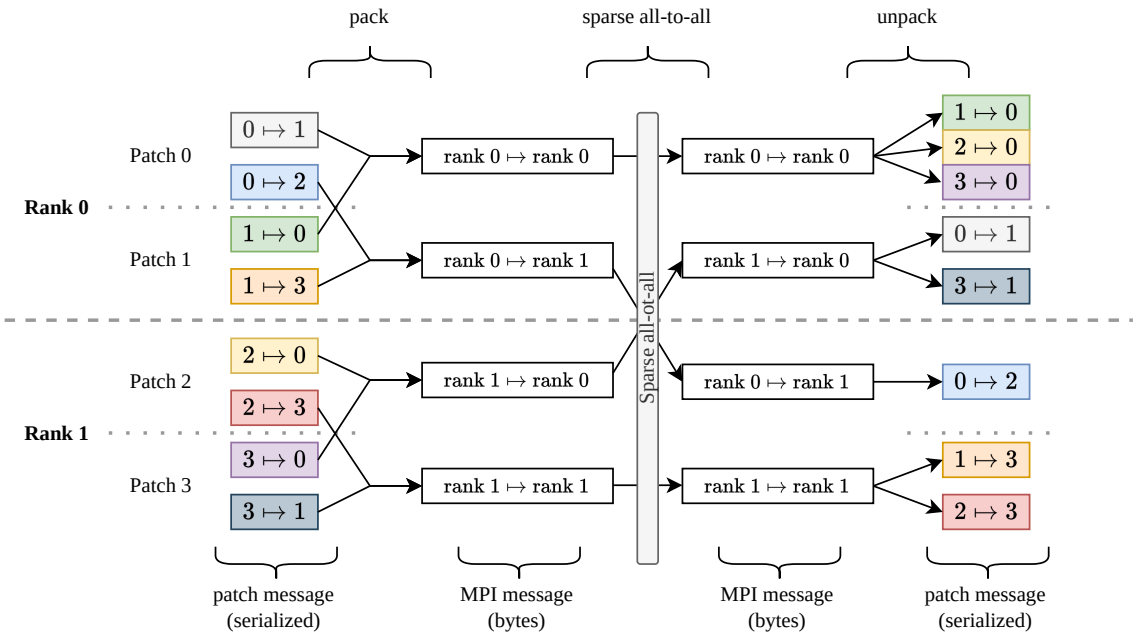


Figure 4.7: Illustration of the behaviour of a MPI sparse communication of patches in SHAMROCK. The first step consists of packing communication between a same pair of MPI ranks together. Subsequently, a sparse all-to-all operation is executed (see Fig.4.8). Finally, the received buffers are unpacked.

the recipient, the communication is disregarded, and an internal memory move is executed instead. Another approach could involve using an MPI reduce operation to count the number of messages received, and trigger the corresponding number of non-blocking receives with `MPI_ANY_SOURCE`. Given the limited number of communications, we observe no practical distinction between the two methods in practice. Moreover, the former approach is easier to debug and optimise, since it eliminates the need for sorting data to ensure determinism in the list of received messages.

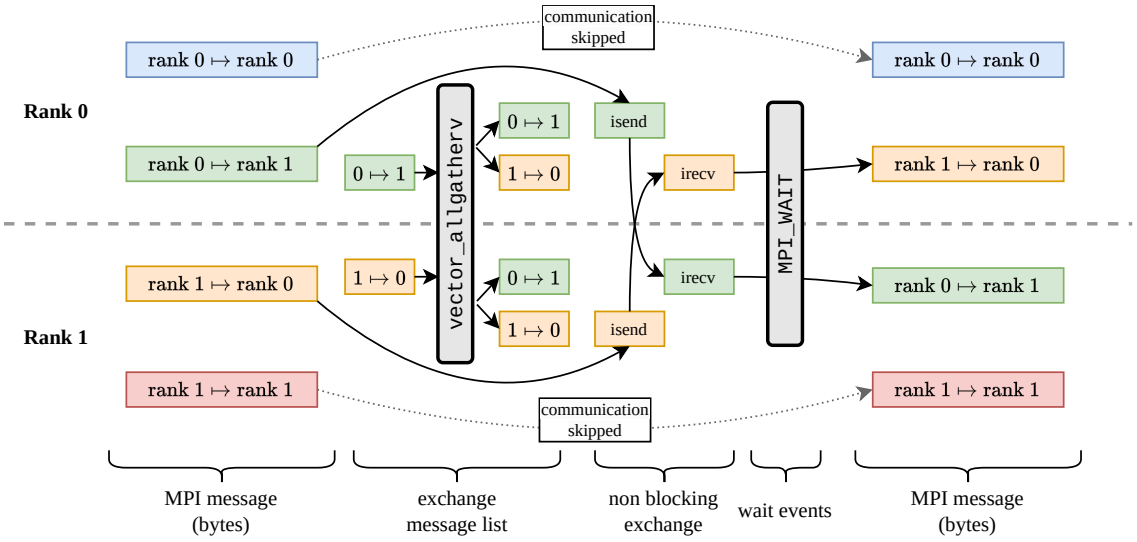


Figure 4.8: Illustration of the behaviour of a MPI sparse all-to-all communication in SHAMROCK. Firstly, a `vector_allgatherv` is performed on the list of communication. Subsequently, each rank executes a non-blocking send of its data. To prepare for receiving, a non-blocking receive is launched for every incoming message. The operation is concluded by waiting for all non-blocking operation to finish. Communications for an MPI rank to itself are skipped by simply relocating the data within the rank.

4. The Shamrock tree

Specific notations used in this Section are given in Table 4.1.

4.1. Morton codes

In hydrodynamic simulations, physical fields are represented on a discrete set of elementary numerical elements such as grid cells or interpolation points (or *numerical objects* for a generic terminology). The positions of these objects are represented by coordinates, usually stored as floating point numbers such as (x, y, z) in three dimensions. These coordinates are usually sampled on a 3D integer grid, which in turn can be mapped onto a 1D integer fractal curve. The Morton space-filling curve, also called *Morton ordering*, is commonly used for this purpose since it has a natural duality with a tree structure (e.g. Samet 2006, see below). In practice, Morton ordering can be constructed from a list of 3D positions as follows. First, the real coordinates in each dimension are remapped over the interval $[0, 1)^3$ (note the exclusion of the value 1) by doing $x \mapsto (x - x_{\min})/(x_{\max} - x_{\min})$, and a similar procedure is applied for y and z respectively. This unit cube is then divided into a 3D grid of $(2^\beta)^3$ elements, where β is the number of bits used to represent integers. Within this grid, the objects possess integer coordinates $(X, Y, Z) \in [0, 2^\beta - 1]^3$.

Table 4.1: List of symbols used in Sect. 4.

Symbol	Definition	Meaning
x, y, z	Sect. 4.1	particle coordinates
X, Y, Z	Sect. 4.1	integer particle coordinates
β	Sect. 4.1	bit count
$\mathtt{X}_0\mathtt{X}_1\cdots\mathtt{X}_{\beta-1}$	Sect. 4.1	binary representation of X
m	$\mathtt{X}_0\mathtt{Y}_0\mathtt{Z}_0\cdots$	generic Morton code
$m_1 \equiv 0101$	Sect. 4.2	Morton code example 1
$m_2 \equiv 0111$	Sect. 4.2	Morton code example 2
$\delta\left(a, b\right)$	eq.4.2	Karras δ operator
$\mathtt{clz}(a)$	Sect. 4.4	count leading zeros
$a \hat{} b$	Sect. 4.4	bitwise XOR operator
a & b	Sect. 4.5	bitwise AND operator
a << b	Sect. 4.5	left bitshift operator
$\mathbf{r}_i$		position of particle i
m_i		Morton code of particle i
$\{\mu_i\}_i$	$\mu_i = m_{\epsilon_i}$	sorted Morton codes
ϵ_i	sort : $\epsilon_i \mapsto i$	sort inverse permutation
ξ_i	Sect. 4.9	Morton-keep mask
$\mathtt{id}_i$		indexes of kept Morton codes
$\mu_{\mathrm{leaf},i}$		tree leaf Morton codes

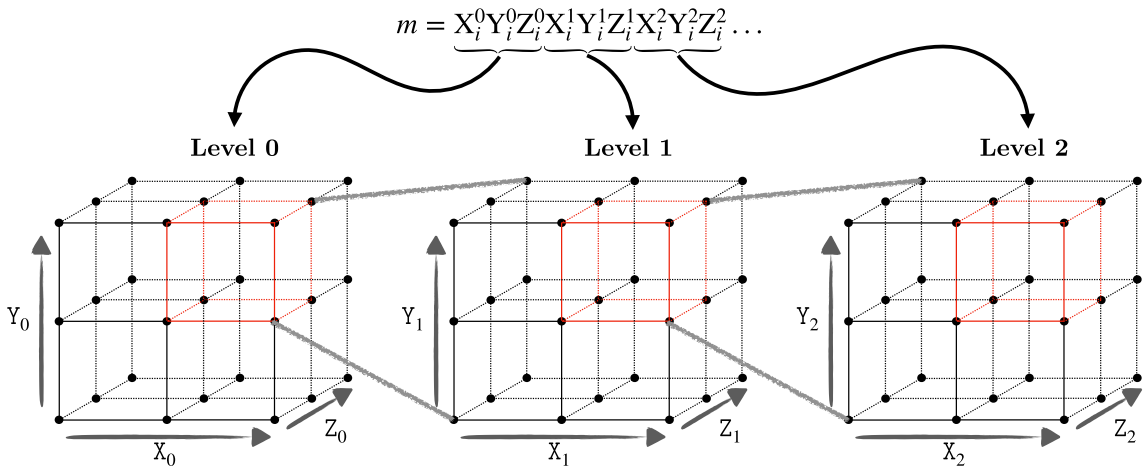


Figure 4.9: Illustration of the duality between Morton codes and the structure of an octree. 3 bits can describe the procedure of dividing a cube into eight smaller cubes. Repeating the procedure with triplets of additional bits produces an octree.

These integer coordinates are noted in their binary representation $X = \mathtt{X}_0\mathtt{X}_1\mathtt{X}_2\cdots$, where $\mathtt{X}_i$ denote the value of the i^{th} bit (the same convention also applies for Y and Z). The Morton space-filling curve comprises a sequence of integers, called *Morton codes* (or Morton numbers), defined through the following construction in a binary basis: the Morton code m of each object is obtained by interleaving the binary representation of each coordinate $m \equiv \mathtt{X}_0\mathtt{Y}_0\mathtt{Z}_0\mathtt{X}_1\mathtt{Y}_1\mathtt{Z}_1\mathtt{X}_2\mathtt{Y}_2\mathtt{Z}_2\cdots\mathtt{X}_{\beta-1}\mathtt{Y}_{\beta-1}\mathtt{Z}_{\beta-1}$.

By default, and unless specified otherwise, Morton codes are presented in binary notation, while other integers are expressed in decimal hereafter. A Morton code can also be interpreted as an ordered position on an octree with $\beta+1$ levels, or alternatively as a position in a binary tree with $3\beta+1$ levels (Fig. 4.9). To illustrate this duality, let us consider the first bit $\mathtt{X}_0$ of a Morton code. If $\mathtt{X}_0 = 0$, the integer coordinate X belongs to the half space where $X < 2^{\beta-1} - 1$. If $\mathtt{X}_0 = 1$, the integer coordinate X belongs to the other half space $X \geq 2^{\beta-1} - 1$. The following bits Y_0 and Z_0 divide the other dimensions in a similar way. The next sequence of bits X_1, Y_1, Z_1 subdivides the subspace characterised by X_0, Y_0, Z_0 in a similar manner, and the construction of a tree follows recursively. After going through all bits and reaching $\mathtt{X}_{\beta-1}\mathtt{Y}_{\beta-1}\mathtt{Z}_{\beta-1}$, one is left with the exact position in the space of integer coordinates. This tree structure consists of nested volumes where each parent volume encompasses all its children, forming as such a Bounded Volume Hierarchy (BVH).

4.2. Prefixes

A *prefix* is the sequence of the first $\gamma \leq \beta$ bits of a Morton code. One defines the *longest common prefix* of two Morton codes a and b as the sequence of matching bits starting from X_0 until two bits differ. As an example, the longest common prefix of $m_1 \equiv \mathbf{01}01$ and $m_2 \equiv \mathbf{01}11$ is $\mathbf{01}$. The longest common prefix of two Morton codes

gives the minimal subspace of the integer 3D grid that contains the two Morton codes. The number of bits used to represent the longest common prefix of a and b, called the *length of the longest common prefix of a and b*, is denoted $\delta(a, b)$.

4.3. Bounding boxes

We use the terminology *prefix class* to refer to a set of Morton codes that have common prefixes. The longest common prefix of any pair of elements in a prefix class is at least of length γ (or equivalently, for any pair of Morton code a, b in the prefix class, $\delta(a, b) \geq \gamma$).

Each prefix class corresponds to an axis aligned bounding box in the space of integer positions, having for generic coordinates $[x_{\min}, x_{\max}) \times [y_{\min}, y_{\max}) \times [z_{\min}, z_{\max})$ (Fig. 4.9). We refer to the set of three integers representing the lengths of the edges of this bounding box as *the size of the bounding box*. Mathematically,

$$\mathbf{s}(\gamma) = \left\{2^{\beta - \lfloor \gamma/3 \rfloor}, 2^{\beta - \lfloor (\gamma-1)/3 \rfloor}, 2^{\beta - \lfloor (\gamma-2)/3 \rfloor}\right\}, \tag{4.1}$$

where $\lfloor \cdot \rfloor$ denotes the floor function of a real number. Indeed, for a given γ, the Morton construction divides the x-axis $\lfloor \gamma/3 \rfloor$ times, the y-axis $\lfloor (\gamma-1)/3 \rfloor$ times and the z-axis, $\lfloor (\gamma-2)/3 \rfloor$ times. The exclusion of the upper bounds in the bounding box ensures that the size on each coordinate axis is a power of 2. Similarly, we define the largest common prefix class between two Morton codes a, b as the prefix class corresponding to the longest common prefix between a and b. The size of the corresponding bounding box is then denoted $\mathbf{s}(a, b) = \mathbf{s}(\delta(a, b))$.

4.4. Longest common prefix length

The length of the longest common prefix of two Morton codes a and b is given by (Karras, 2012)

$$\delta(a, b) \equiv \texttt{clz}(a \hat{} b). \tag{4.2}$$

Eq. 4.2 involves two binary operators. The first one is the bitwise `XOR` $\hat{}$ operator (Exclusive `OR`), that returns the integer formed in binary by zeros where the bits match and ones when they differ. As an example,

$$m_1 \hat{} m_2 = 0010, \tag{4.3}$$

since m_1 and m_2 differ only by their third bit. The second operator is Count Leading Zeros. `clz` operates on a binary integers and returns the numbers of zeros preceding the first 1 in the binary representation. As an example,

$$\texttt{clz}(0010) = 2. \tag{4.4}$$

Following this example, the longest common prefix of $m_1 \equiv \mathbf{01}01$ and $m_2 \equiv \mathbf{01}11$ is **01** and is of length 2. Eqs. 4.3–4.4 allow performance, since instructions `clz` and `XOR` use only one CPU or GPU cycle on modern architectures. Getting the length of the longest common prefix take only 2 cycles with such procedure (e.g. a `xor` followed by `lzcnt` on Intel Skylake architectures).

4.5. Finding common prefixes

To find the longest common prefix between two Morton codes a and b, we first construct a mask c, which is an integer where the first $p = \delta(a, b)$ bits are set to 1 while the remaining bites are set to 0. For example, applying this mask to the two Morton codes m_1 and m_2 from our previous example yields 1100. To generate the mask, we take advantage of the bitwise shift-left operator. The bitwise shift-left operator a << i returns the binary representation of a where the bits are shifted by i^{th} bits to the left, and zeros are introduced in place of non existing bits. Consider u, the integer having only ones in binary representation (i.e $u = 2^\beta - 1$, where β is the size of the binary representation). c is obtained with the following binary operation u << $(\beta - \delta(a, b))$. In our previous example, $\beta = 4$ and $\beta - \delta(m_1, m_2) = 2$ gives 1111 << 2 = 1100. Consider now the bitwise `AND` operator, denoted by &, that returns the integer formed in binary by ones where the bits match and zeros when they differ (& is the bitwise negation of the bitwise `XOR` operator). When applying the bitwise `AND` between m_1 or m_2 and the mask, the result is a binary number where the first bits are the prefix and the subsequent bits are zeros. As an example, applying the bitwise `AND` between m_2 and the mask yields **01**11 & 1100 = **01**00.

4.6. Getting coordinates sizes of bounding boxes

Consider the prefix class formed by Morton codes whose longest common prefix with a (or equivalently b) is $\delta(a, b)$. This prefix class is a set of binary numbers whose smaller and larger values, denoted p_0 and p_1 respectively, are given by

$$p_0(a, b) \equiv \left(2^\beta - 1 \texttt{ << } \beta - \delta(a, b)\right) \texttt{ \& } a, \tag{4.5}$$

$$p_1(a, b) \equiv \left(2^\beta - 1 \texttt{ << } \beta - \delta(a, b)\right) \texttt{ \& } a + \left(2^{\beta - \delta(a,b)} - 1\right). \tag{4.6}$$

These two Morton codes correspond to two integer coordinates, denoted $\mathbf{p}_0$ and $\mathbf{p}_1$, that are the coordinates of the lower and upper edges of the bounding box, respectively. The size of the bounding box corresponding to this prefix class is

$$\mathbf{s}(a, b) = \mathbf{p}_1(a, b) - \mathbf{p}_0(a, b) + (1, 1, 1). \tag{4.7}$$

4.7. Binary radix tree

A binary radix tree is a hierarchical representation of the prefixes of a list of bit strings, corresponding here to the binary representation of integers (e.g. Lauterbach et al. 2009; Karras 2012). The tree is defined by a set of hierarchically connected nodes, where nodes without children are called leaf nodes or *leaves* (light orange circles on Fig. 4.10), and the other ones are called *internal nodes* (blue circles on Fig. 4.10). The binary radix tree is a complete binary tree: every internal node has exactly two children. As such, a tree having n leaves has exactly $n-1$ internal nodes. This property allows us to know lengths of tables in advance, making it particularly

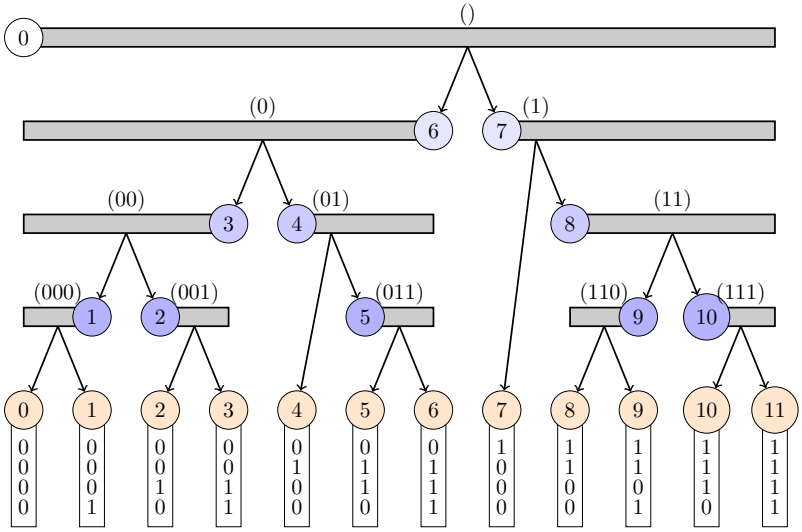


Figure 4.10: The Karras Algorithm associates a structure of radix tree to a sorted list of unduplicated Morton codes (depicted here within rectangles). Within this tree, nodes can be either leaves, denoted by integers in light orange circles, or internal nodes, indicated by integers in blue circles. The grey bars represent the ranges of Morton codes covered by each internal node. Common prefixes of these Morton codes are shown in brackets.

beneficial for GPU programming where dynamic allocation is not feasible. One commonly acknowledged downside of this tree structure is the challenge of efficient hierarchical construction (Lauterbach et al., 2009; Karras, 2012). The deeper one goes down the tree, the longer the length of the common prefix of the Morton codes. The corresponding bounding boxes for each node in the tree are nested and become progressively smaller as one goes down the tree, while the length of the common prefixes increases. The corresponding radix tree forms a Bounding Volume Hierarchy, since each child in the bounding box is contained within the box of its parents.

4.8. Karras algorithm

The Karras algorithm overcomes this difficulty with a fully parallel algorithm that constructs a binary radix tree, in which the list of bit strings is a sorted list of Morton codes without any duplicates Karras (2012). Fig. 4.10 shows a typical binary radix tree constructed by the Karras algorithm. The integers within the light orange circles represent the indices of the Morton codes in the sorted list, and they also are the indices of the corresponding leaves in the tree. The integers within the blue circles denote the indices of the internal nodes, their value come as a by-product of the algorithm. The grey bars denote intervals of leaf indices corresponding to any Morton code contained in the sub-tree beneath an internal node that shares

Internal cell id	0	1	2	3	4	5	6	7	8	9	10
`left-child-id`	6	0	2	1	4	5	3	7	9	8	10
`right-child-id`	7	1	3	2	5	6	4	8	10	9	11
`left-child-flag`	0	1	1	0	1	1	0	1	0	1	1
`right-child-flag`	0	1	1	0	0	1	0	0	0	1	1
`endrange`	11	0	3	0	6	6	0	11	11	8	11

Table 4.2: Tables corresponding to the tree shown on Fig. 4.10 as returned by the Karras algorithm.

the same common prefix. In Fig. 4.10, these prefixes are shown in brackets over the grey bars. The deeper the position in the tree, the longer the prefix. Consider the subset of Morton codes associated to an internal node. The Karras algorithm divides this list in two sub-lists (arrows Fig. 4.10), each with a longer common prefix compared to the original. Such a split is unique. Several tables are associated to the construction of Fig. 4.10. Those are presented on Table 4.2. The split associated to an internal node provides two numbers called the indices of the left and right children respectively, denoted as `left-child-id` and `right-child-id` respectively. *A priori*, these indices can correspond to either an internal node or a leaf. This distinction is encoded by the value of the integers `left-child-flag` and `right-child-flag`, where a 1 means that the corresponding child is a leaf and a 0, an internal node. The grey bar of an internal node has two ends. One corresponds to the index of the node itself, while the other is stored in `endrange`. Although this value is of no use in the construction of the tree itself, it will be important later for calculating the sizes of a bounding box associated to a prefix class and for iterating over objects contained in leafs. The Karras algorithm performs dichotomous searches to compute the values of Table 4.2 in parallel, with no prerequisites other than the Morton codes (we refer to the pseudo-code of the algorithm in Karras 2012 for details). Its efficiency relies firstly on the ability to pre-allocate tables before building the tree, and secondly on the sole use of the δ operator defined in Sect. 4.4, which requires just 2 binary operations on dedicated hardware.

4.9. Removal of duplicated codes

As mentioned in Sect. 4.8, the Karras algorithm requires a sorted list of Morton codes without duplicates (line 1-5 in alg.3). To achieve this, we go through the list of sorted Morton codes and compute a mask to select the Morton codes to retain. The list of Morton codes without duplicates corresponds to the leaves of the tree obtained after applying the Karras algorithm to construct the radix tree.

Algorithm 3: Removal mask initialisation and reduction algorithm

Data: $\{m_i\}_{i\in[0,n)}$ The morton codes.
Result: $\{\xi_i\}_{i\in[0,n)}$ The mask list.

// Flag removal of duplicates
for i ***in parallel*** **do**
  if $i == 0$ **then**
    $\xi_i \leftarrow$ **true**;
  else
    $\xi_i \leftarrow not(m_i = m_{i-1})$;

// Reduction passes
for n_{red} *reduction steps* **do**
  for i ***in parallel*** **do**
    // Get kept morton codes indexes
    $i_{-1} \leftarrow i - 1$;
    while $(\xi_{i_{-1}} =$ ***false*** $\ \&\ \ i_{-1} \geq 0)$ **do**
      $i_{-1} \leftarrow i_{-1} - 1$;
    $i_{-2} \leftarrow i_{-1} - 1$;
    while $(\xi_{i_{-2}} =$ ***false*** $\ \&\ \ i_{-2} \geq 0)$ **do**
      $i_{-2} \leftarrow i_{-2} - 1$;
    $i_{+1} \leftarrow i + 1$;
    while $(\xi_{i_{+1}} =$ ***false*** $\ \&\ \ i_{+1} < N_{morton})$ **do**
      $i_{+1} \leftarrow i_{+1} + 1$;
    // Reduction criterion
    $\delta_0 \leftarrow \delta(\mu_i, \mu_{i_{+1}})$;
    $\delta_{-1} \leftarrow \delta(\mu_{i_{-1}}, \mu_i)$;
    $\delta_{-2} \leftarrow \delta(\mu_{i_{-2}}, \mu_{i_{-1}})$;
    if $not(\delta_0 < \delta_{-1}\ \ \&\ \ \delta_{-2} < \delta_{-1})\ \ \&\ \ \xi_i =$ ***true*** **then**
      $\xi_i \leftarrow$ **true**;
    else
      $\xi_i \leftarrow$ **false**;

Algorithm 4: Leaf object iteration

Data: $\mathtt{id}_i$ The leaf index map.
i the leaf index we want to unpack

for $j \in [id_i, id_{i+1})$ **do**
  $k \leftarrow \epsilon_j$ // index map of the sort
  $\mathcal{F}(k)$

4.10. Reduction

In certain situations, an object may interact with a large number of neighbours, resulting in multiple leaves containing these neighbours for the object. One such situation arises frequently in a Smoothed Particle Hydrodynamics solver, where each particle typically interacts with an average of ~ 60 neighbours. One optimisation strategy to speed up the tree traversal consists in reducing the number of leaves containing these 60 neighbours by grouping some leaves at the lower levels of the tree before applying the Karras algorithm. We have integrated a so-called step of *reduction* to achieve this. The resulting tree mirrors the initial one, but with grouped leaves.

To perform reduction, we require a criterion determining when two leaves, each containing two Morton codes, can be removed to yield the internal cell positioned just above them. This procedure is carried out using Alg.3: if a Morton code constitutes the second leaf of a shared parent, then it is removable. This property is implemented in the radix tree by verifying when $\delta(\mu_{i-2}, \mu_{i-1}) < \delta(\mu_{i-1}, \mu_i) > \delta(\mu_i, \mu_{i+1})$. When this condition is satisfied, the Morton code i is removable. The reduction step modifies the Morton tree list associated to the initial tree built. The tree is therefore already reduced when it is built and has never had any additional nodes.

4.11. Tree building

Fig. 4.11 outlines the tree building algorithm of Shamrock. Initially, Morton codes are generated from coordinates and efficiently sorted while eliminating duplicates. Morton tables are then prepared and pre-processed (a summary of these steps is sketched in Fig. 4.12) before filling the values characterising the tree as in Table 4.2. The lengths associated to the coordinates of the cells are finally calculated. The algorithms described in this section are implemented using `C++` metaprogramming, enabling versatile use of any kind of spatial coordinates in practice.

Compute Morton codes

Morton codes are calculated entirely in parallel (step `"To Morton"` in Fig.4.11). Initially, a buffer storing the positions of the elementary numerical elements is allocated. These positions are mapped to an integer grid following the procedure described in Sect. 4.1. The construction is tested by appropriate sanity checks. The resulting integer coordinates are converted to Morton codes in a Morton code buffer (m_i in Fig.4.11).

Sort by Key

Initially, the list of Morton codes corresponding to the positions of elementary numerical elements is unsorted. A key-value pair sorting algorithm is therefore used

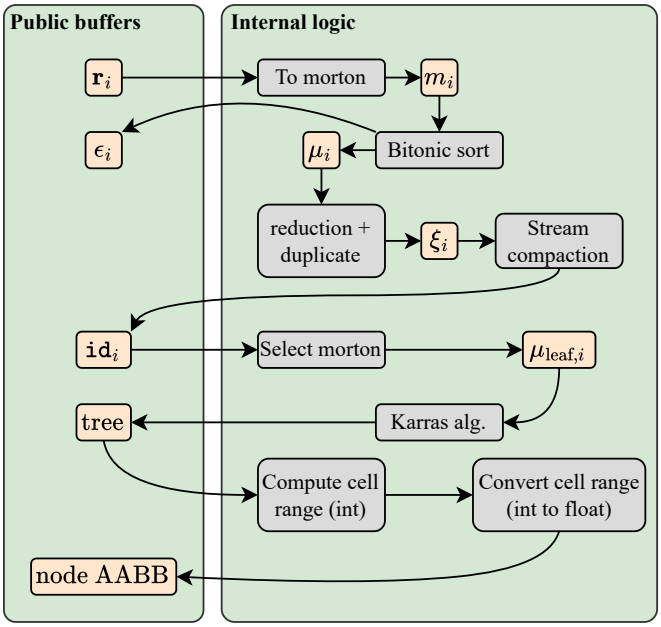


Figure 4.11: Flowchart illustrating the tree-building procedure, indicating the interdependence between each algorithm (grey boxes) and the related buffers (orange boxes). The internal logic box corresponds to the part of the algorithm inaccessible to the user. Buffers depicted outside this box are structures used in other parts of the code.

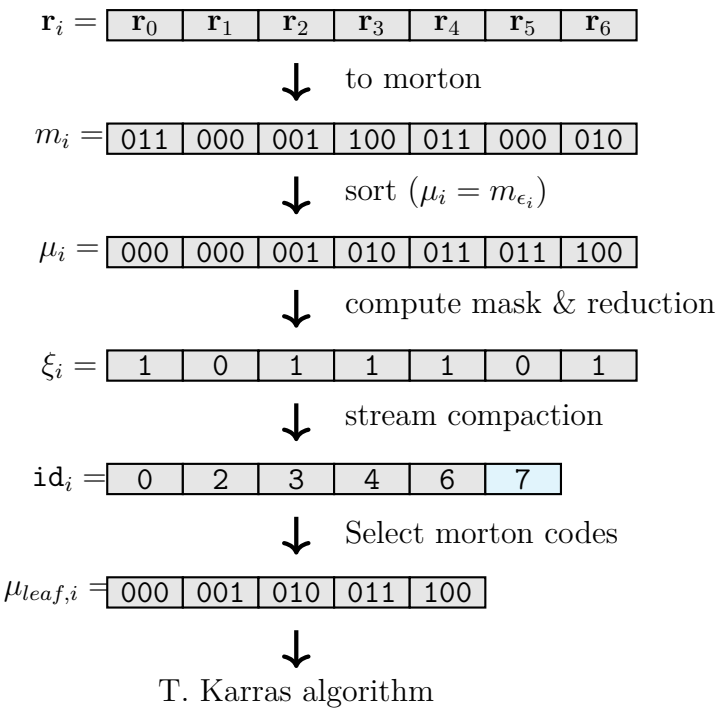


Figure 4.12: The cyan slot in the $\mathtt{id}_i$ row is the total lenght of the input array. ϵ_i is the resulting permutation applied by the sort algorithm.

to sort the Morton codes while keeping track of the original index of the object within the list. For this task, we use a GPU Bitonic sorting algorithm that we have re-implemented using `Sycl`. The Bitonic algorithm is simple and its performance is not heavily reliant on the hardware used (step `"Bitonic sort"` in Fig.4.11, see e.g. Batcher 1968; Nassimi & Sahni 1979). While more efficient alternatives have been suggested in the literature, our observation is that they are more difficult to implement and are not as portable across architectures (e.g. Arkhipov et al. 2017; Adinets & Merrill 2022).

Reduction

From the sorted list of Morton codes, we remove duplicates and apply reduction with a procedure in two steps. In the first step, we generate a buffer of integers where each value is 1 if the Morton code is retained at a given index and 0 otherwise. This information is stored in a buffer called Keep Morton flag buffer (ξ_i in Fig.4.11). In the second step, we use this buffer to perform a stream compaction algorithm (e.g. Blelloch 1990; Horn 2005, see example in Fig. 4.12) to construct simultaneously two lists: a list of Morton codes without duplicates, and the list of the indices of the preserved Morton code prior stream compaction. The stream compaction algorithm heavily depends on an internal exclusive scan algorithm. This algorithm, when applied to the array $\{a_i\}_{i\in[0,n]}$ returns the array $\{\sum_{j=0}^{i-1} a_j\}_{i\in[1,n]}$ and 0 when $i = 0$.

In our case, we implemented the single-pass prefix sum with decoupled look-back algorithm (Merrill & Garland, 2016).

Compute tree tables

At this point, we have a set of Morton codes sorted without duplicates. We then apply the Karras algorithm described in Sect. 4.8 to generate in parallel the tables from which the properties of the tree can be reconstructed (listed on Table 4.2).

Compute tree cell sizes

We define a *tree cell* as the bounding box that corresponds to the Morton codes of the leaves under a given node. This node can either be an internal node or a leaf. Tree cells are therefore the geometric representation of the tree, and needs to be computed for neighbour finding (see Sect. 4.12). In practice, it is sufficient to compute the boundaries of the edges of the cell $[x_{\min}, x_{\max}) \times [y_{\min}, y_{\max}) \times [z_{\min}, z_{\max})$ (Sect. 4.6).

Algorithm 5: Compute tree cell sizes

Data: `morton` The morton code buffer.
Result: `bmin`, `bmax`, the bounds of the cells.

$m_1 = \texttt{morton}[i]$
$m_2 = \texttt{morton}[\texttt{endrange}[i]]$
$\sigma = \delta_{\text{karras}}(m_1, m_2)$
$\mathbf{f}_0 = \mathbf{s}(\sigma)$
$\mathbf{f}_1 = \mathbf{s}(\sigma + 1)$
$\texttt{mask} = \texttt{maxint} << (\texttt{bitlen} - \sigma)$
$\mathbf{p}_0 = (\text{morton} \to \text{real space})\,(\texttt{m}[i]\,\&\,\texttt{mask})$
$\texttt{bmin}[i] = \mathbf{p}_0$
$\texttt{bmax}[i] = \mathbf{p}_0 + \mathbf{f}_0$
if *left child flag*$[i]$ **then**
    $\texttt{bmin}[\texttt{rid}[i] + N_{\text{internal}}] = \mathbf{p}_0$
    $\texttt{bmax}[\texttt{lid}[i] + N_{\text{internal}}] = \mathbf{p}_0 + \mathbf{f}_1$
if *right child flag*$[i]$ **then**
    $\texttt{tmp} = \mathbf{f}_0 - \mathbf{f}_1$
    $\texttt{bmin}[\texttt{rid}[i] + N_{\text{internal}}] = \mathbf{p}_0 + \texttt{tmp}$
    $\texttt{bmax}[\texttt{lid}[i] + N_{\text{internal}}] = \mathbf{p}_0 + \texttt{tmp} + \mathbf{f}_1$

Alg. 5 provides the procedure to compute the size of tree cells, using the vector position $\mathbf{s}$ defined by Eq. 4.7 and the quantities p_0 and p_1 defined by Eqs. 4.5 – 4.6. For internal cells that have leaves as children, the boundary of the edges can be calculated by incrementing the value of $\delta(a, b)$ by one unity and using the new

value in Eqs. 4.5 – 4.6. This gives the expected result for a left child, an extra shift being added for the right child (cf. line 14 of Alg.5).

4.12. Tree traversal

Each cell, leaf or internal of the tree constructed by the procedure described above consists of an axis-aligned bounding boxes and containing several numerical objects. Searching for the neighbours of an object a therefore requires checking the existence of an interaction between a cell of the tree c and the object a, using the object-group interaction criterion $\gamma_{o/g}(a, c)$. Per construction, if the criterion is true for a child cell, it is also true for its parent. Neighbour finding requires therefore starting from the root node and going down the tree, checking at each step whether the interaction criterion is still verified or not. The result is a set of retained tree leaves, that are likely to contain neighbours. The set of neighbours of a given object is then obtained by verifying the object-object interaction criterion on each object in each of the targeted leaves. The algorithmic procedure for these steps is detailed in Alg. 6. It is based on the property that the depth of the tree is shorter than the length of the Morton code representation. This allows a stack of known size to be used to traverse the tree, which can be added at compile time and run on the GPU since there is no dynamic memory allocation. The first step in the algorithm is to push the root node onto the stack. In each subsequent step, we pop the node on top of the stack, and we check whether or not it interacts with the object. If it does, and if it is an internal node, we push its children onto the top of the stack and move on to the next step. Otherwise, if it is a leaf, we iterate through the objects contained in the leaf (Alg. 4), and check the object-object interaction criterion for each object in the given leaf. In the source code of Shamrock, we abstract Alg. 6 under the `rtree_for`. It can be called from within a kernel on the device and can be associated with any interaction criteria. It will then provide an abstract loop over the objects found using the criteria.

4.13. Direct neighbour cache

Using neighbour search directly is technically feasible, but conducting it repeatedly would result in substantial costs due to its intricate logic. Moreover, executing computations within the core of a device kernel with extensive branching would negatively impact performance. To circumvent these issues, we instead build a neighbour cache when traversing the tree, and then reuse this cache for subsequent computations on the particles. The benefits are twofold: firstly, it increases performance for the reasons outlined above, and secondly, it decouples neighbour finding from calculations carried out on the particles, enabling optimisation efforts to be better targeted. Conversely, using such an approach means that we store an integer index for each pair of neighbours, which in SPH is roughly 60 times the number of particles. The memory footprint therefore increases significantly. Taking everything into

Algorithm 6: Tree traversal

Data:
$depth$: The maximal tree depth, N_{inode} : The number of internal nodes in the tree, $\{lchild_{id,j}\}_{j\in[0,N_{inode})}$, $\{rchild_{id,j}\}_{j\in[0,N_{inode})}$, $\{lchild_{flag,j}\}_{j\in[0,N_{inode})}$, $\{rchild_{flag,j}\}_{j\in[0,N_{inode})}$

// Setup index stack
$i \leftarrow depth - 1$;
$s \leftarrow \{err\}_{i\in[0,depth)}$;
// Enqueue the root node
$s_i \leftarrow 0$;
do
  // Pop top of the stack
  $j = s_i$;
  $s_i = err$;
  $i \leftarrow i + 1$;
  // Check if interaction
  $\alpha \leftarrow \gamma_{o/g}(\ldots, j)$;
  if α **then**
    // If the current node is a leaf
    if $j \geq N_{inode}$ **then**
      // Iterate on objects in leaf
      leaf object iteration(j);
    else
      // Push node childs on the stack
      $lid \leftarrow lchild_{id,j} + (N_{inode}) * lchild_{flag,j}$;
      $rid \leftarrow rchild_{id,j} + (N_{inode}) * rchild_{flag,j}$;
      $i \leftarrow i - 1$;
      $s_i = rid$;
      $i \leftarrow i - 1$;
      $s_i = lid$;
  else
    // Gravity
    leaf exclude case(j);
while $i < depth$;

account, we opt for the neighbour caching strategy due to its better performance and extensibility.

Algorithm 7: Neighbour caching

Data: N : The number of objects to build cache for, $\gamma_{o/g}$: the object-group interaction criterion, $\gamma_{o/o}$ the object object interaction criterion.

Result: $\{\xi_i\}_{i\in[0,N+1)}$ The offset map. $\{\Xi_i\}_{i\in[0,N_{neigh})}$ The neighbour id map.

```
1  {c_i ← 0}_{i∈[0,N+1)};
   // First pass to count neighbours
2  for i ∈ [0, N) in parallel do
3      c ← 0;
4      for j ← rtree_for[γ_{o/g}(i, ...)] do
5          if γ_{o/o}(i, j) then
6              c ← c + 1;
7      c_i ← c;
   // c_i contain the neighbours counts
8  {ξ_i}_{i∈[0,N+1)} ← exclusive scan({c_i}_{i∈[0,N+1)});
   // ξ_i contain the neighbour map offset
9  N_neigh ← c_N;
10 {Ξ_i ← 0}_{i∈[0,N_neigh)};
   // Second pass to get neighbours ids
11 for i ∈ [0, N) in parallel do
12     off ← ξ_i;
13     for j ← rtree_for[γ_{o/g}(i, ...)] do
14         if γ_{o/o}(i, j) then
15             Ξ_off ← j;
16             off ← off + 1;
```

We start by allocating a buffer to store the neighbour count for each object. We perform an initial loop over all the objects and do a tree traversal for each of them to obtain the neighbour counts. We then perform an exclusive scan, which gives the offset used to write in the neighbour index map from our neighbour count buffer. The neighbour count buffer has an extra element that is set to zero at its end, this allows us to obtain the total number of neighbours in this slot after the exclusive scan. A final loop writes the indexes of the neighbours to the neighbour index map. Details of this procedure are given in Alg. 7. We can use a procedure similar to the one used for the tree leafs in Alg. 4 to iterate over the neighbours stored in the neighbour cache, as depicted in Alg. 8.

Algorithm 8: Neighbour cache usage

Data: $\{\xi_i \leftarrow 0\}_{i\in[0,N+1)}$ the offset map, $\{\Xi_i\}_{i\in[0,N_{neigh})}$ the neighbour cache

```
for j ∈ [ξ_i, ξ_{i+1}) do
    k ← Ξ_j // index of neighbour
    F(k)
```

4.14. Two-stages neighbour cache

The procedure described in Sect. 4.13 consists of a direct neighbour cache, in the sense that for each object we search directly for its neighbours. A more sophisticated approach, likely to improve performance in most cases, involves splitting the direct case into two stages. In the first step, we search for the neighbours of each tree leaf using the group-group interaction criterion and the group-object criterion. In the second step, we first determine in which leaf the object is, then use the leaf neighbour cache to find the neighbour of the object. The first step only searches for neighbours within the leaves of the tree, while the second step produces the same result as in the direct case. In a two-stages neighbour search, tree traversal is performed once per tree leaf, instead of once per object. When combined with tree reduction, this approach can decrease the number of tree traversals performed by a factor of ten. On the flip side, adopting a two-stage neighbour caching approach increases the number of kernels to be executed on the device and the allocation pressure (temporarily, as the first step is discarded at the end, the memory footprint is unchanged compared to the direct case, but temporary allocation can introduce additional latency). Overall, we observe that two-stages neighbour caching generally improves computational efficiency, and when combined with tree reduction, this strategy ultimately yields the best performance.

5. Summary

In this chapter, we have presented the core of the SHAMROCK framework. The methods detailed here are applicable to any hydrodynamical numerical scheme regardless of whether they are grid-based or particle-based, since patch decomposition, communications, and load balancing are abstracted from the scheme. Given the structure of the MPI layer, we anticipate linear weak scaling, as long as simulation performed is not constrained by communication bandwidth or latency. Furthermore, all the major algorithms of the framework, in particular the construction of the radix tree, are offloaded to GPUs following the guidelines discussed in Chapter 3. We therefore expect a speedup compared to CPU execution. Additionally, the tree-building process has been significantly optimized. This enables almost cost-free recomputation of the tree, thereby eliminating the need for its communication. We will now test the performance of the framework on hydrodynamical simulations.

References

Abraham M. J., Murtola T., Schulz R., Páll S., Smith J. C., Hess B., Lindahl E., 2015, *GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers*, SoftwareX, 19–25

Adinets A., Merrill D., 2022, *Onesweep: A Faster Least Significant Digit Radix Sort for GPUs*, arXiv preprint arXiv:2206.01784

Alekseenko A., Páll S., in *Proceedings of the 2023 International Workshop on OpenCL*, booktitle, IWOCL '23. Association for Computing Machinery, New York, NY, USA, doi:10.1145/3585341.3585350, https://doi.org/10.1145/3585341.3585350

Alekseenko A., Páll S., Lindahl E., 2024, *GROMACS on AMD GPU-Based HPC Platforms: Using SYCL for Performance and Portability*, arXiv preprint arXiv:2405.01420

Alpay A., Heuveline V., in *Proceedings of the International Workshop on OpenCL*, booktitle, IWOCL '20. Association for Computing Machinery, New York, NY, USA, doi:10.1145/3388333.3388658, https://doi.org/10.1145/3388333.3388658

Alpay A., Soproni B., Wünsche H., Heuveline V., in *International Workshop on OpenCL*, booktitle, IWOCL'22. Association for Computing Machinery, New York, NY, USA, doi:10.1145/3529538.3530005, https://doi.org/10.1145/3529538.3530005

Arkhipov D. I., Wu D., Li K., Regan A. C., 2017, *Sorting with GPUs: A Survey*, arXiv e-prints, ADS link, arXiv:1709.02520

Batcher K. E., in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, booktitle, AFIPS '68 (Spring). Association for Computing Machinery, New York, NY, USA, p. 307–314, doi:10.1145/1468075.1468121, https://doi.org/10.1145/1468075.1468121

Blelloch G. E., 1990, *Prefix sums and their applications*, School of Computer Science, Carnegie Mellon University Pittsburgh, PA, USA

Deakin T., McIntosh-Smith S., in *Proceedings of the International Workshop on OpenCL*, booktitle, IWOCL '20. Association for Computing Machinery, New York, NY, USA, doi:10.1145/3388333.3388643, https://doi.org/10.1145/3388333.3388643

Gafton E., Rosswog S., 2011, *A fast recursive coordinate bisection tree for neighbour search and gravity*, MNRAS, 418, 770-781

Grete P., et al., 2022, *Parthenon – a performance portable block-structured adaptive mesh refinement framework*, arXiv e-prints, ADS link, arXiv:2202.12309

Hopkins P. F., 2015, *A new class of accurate, mesh-free hydrodynamic simulation methods*, MNRAS, 450, 53-110

Horn D., 2005, *Stream reduction operations for GPGPU applications*, Gpu gems, 573–589

Jakob W., Rhinelander J., Moldovan D., 2024, *pybind11–Seamless operability between C++ 11 and Python*, URL: https://github. com/pybind/pybind11

Jin Z., Vetter J. S., in *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, booktitle, BCB '22. Association for Computing Machinery, New York, NY, USA, doi:10.1145/3535508.3545591,

https://doi.org/10.1145/3535508.3545591

Karras T., in *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, booktitle, pp 33–37

Lauterbach C., Garland M., Sengupta S., Luebke D. P., Manocha D., 2009, *Fast BVH Construction on GPUs*, Computer Graphics Forum

Lesur G. R. J., Baghdadi S., Wafflard-Fernandez G., Mauxion J., Robert C. M. T., Van den Bossche M., 2023, *IDEFIX: A versatile performance-portable Godunov code for astrophysical flows*, A&A, 677, A9

Markomanolis G. S., et al.,in *Supercomputing Frontiers: 7th Asian Conference, SCFA 2022, Singapore, March 1–3, 2022, Proceedings*, booktitle, Springer-Verlag, Berlin, Heidelberg, p. 79–101, doi:10.1007/978-3-031-10419-0˙6, https://doi.org/10.1007/978-3-031-10419-0_6

Merrill D., Garland M., 2016, *Single-pass parallel prefix scan with decoupled look-back*, NVIDIA, Tech. Rep. NVR-2016-002

Morton G. M., 1966, *A computer oriented geodetic data base and a new technique in file sequencing*, International Business Machines Company New York

Nassimi Sahni 1979, *Bitonic Sort on a Mesh-Connected Parallel Computer*, IEEE Transactions on Computers, 2-7

Price D. J., et al., 2018, *Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics*, PASA, 35, e031

Samet H., 2006, *Foundations of multidimensional and metric data structures*, Morgan Kaufmann

Springel V., Pakmor R., Zier O., Reinecke M., 2021, *Simulating cosmic structure formation with the GADGET-4 code*, MNRAS, 506, 2871-2949

Trott C., et al., 2021, *The Kokkos EcoSystem: Comprehensive Performance Portability for High Performance Computing*, Computing in Science and Engineering, 23, 10-18

Wibking B. D., Krumholz M. R., 2022, *QUOKKA: a code for two-moment AMR radiation hydrodynamics on GPUs*, MNRAS, 512, 1430-1449

Chapter 5

Shamrock SPH solver

Contents

Foreword

We now present the Smoothed Particle Hydrodynamics (SPH) solver of Shamrock. We have chosen to start with SPH because it is numerically the most complex method to implement and establishes a robust foundation for many components needed for other methods. Additionally, to our knowledge, no exascale scalable multi-GPU SPH codes existed in the astrophysics community until now, and efforts were already underway to port other methods to exascale architectures. Finally, our choice of SPH was influenced by the local expertise at CRAL and a month-long collaboration with Professor D. Price at Monash University (Melbourne, Australia).

1. Smoothed Particle Hydrodynamics in Shamrock

1.1. Equations of motion

As detailed in Chapter 1, Sec. 4 in Smoothed Particle Hydrodynamics (SPH), we evolve particles in such a way that their dynamics approximate a fluid statisfying Euler's equation. In SPH, particle densities are defined by

$$\rho_a = \sum_b m_b W_{ab}(h_a), \tag{5.1}$$

$$W_{ab}(h_a) = \frac{C_{\text{norm}}}{h_a^3} f\left(\frac{|\mathbf{r}_a - \mathbf{r}_b|}{h_a}\right), \tag{5.2}$$

The equation of motion for SPH with artificial viscosity are:

$$\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} = \sum_b m_b \left(\frac{P_a + q^a_{ab}}{\rho_a^2 \Omega_a} \nabla_a W_{ab}(h_a) + \frac{P_b + q^b_{ab}}{\rho_b^2 \Omega_b} \nabla_a W_{ab}(h_b) \right), \tag{5.3}$$

$$\Omega_a = 1 - \frac{\partial h_a}{\partial \rho_a} \sum_b m_b \frac{\partial W_{ab}(h_a)}{\partial h_a}, \tag{5.4}$$

where

$$q^a_{ab} = \begin{cases} -\frac{1}{2}\rho_a v_{\mathrm{sig},a} \boldsymbol{v}_{ab} \cdot \hat{\boldsymbol{r}}_{ab}, & \boldsymbol{v}_{ab} \cdot \hat{\boldsymbol{r}}_{ab} < 0 \\ 0 & \text{otherwise,} \end{cases}, \tag{5.5}$$

$$v_{\mathrm{sig},a} = \alpha^{\mathrm{AV}}_a c_{s,a} + \beta |\mathbf{v}_{ab} \cdot \hat{\boldsymbol{r}}_{ab}| \ , \ \alpha^{\mathrm{AV}}_a \in [0,1]. \tag{5.6}$$

The internal energy equation with artificial conductivity is

$$\frac{\mathrm{d}u_a}{\mathrm{d}t} = \frac{P_a + q^a_{ab}}{\rho_a^2 \Omega_a} \sum_b m_b \mathbf{v}_{ab} \cdot \nabla_a W_{ab}(h_a) + \Lambda_{\mathrm{cond}}, \tag{5.7}$$

where $\mathbf{v}_{ab} \equiv \mathbf{v}_a - \mathbf{v}_b$, and the pressure P_a is related to the density ρ_a and other variables through the equation of state. Additionally,

$$\Lambda_{\mathrm{cond}} = \sum_b m_b \beta_u v^u_{\mathrm{sig}} (u_a - u_b) \frac{1}{2} \left[\frac{F_{ab}(h_a)}{\Omega_a \rho_a} + \frac{F_{ab}(h_b)}{\Omega_b \rho_b} \right], \tag{5.8}$$

$$v^u_{\mathrm{sig}} = \sqrt{\frac{|P_a - P_b|}{(\rho_a + \rho_b)/2}}, \tag{5.9}$$

using $F_{ab}(h_a) = \hat{\mathbf{r}}_{ab} \cdot \nabla_a W_{ab}(h_a)$ and β_u to represent the shock conductivity parameter.

In Shamrock, typical functions f with finite compact supports, such as Schoenberg (1946) B-splines like M_4, M_5, M_6 , or Wendland functions like C_2, C_4, C_6 (see e.g., Wendland 1995), are implemented. We define l_a, the interaction radius of SPH particle a, as $l_a = R_{\mathrm{kern}} h_a$, where R_{kern} is the radius of f, the kernel generator function.

The shock detection is performed by Cullen & Dehnen (2010) viscosity switch. The value of the shock viscosity parameter α_a is evolved using

$$\frac{\mathrm{d}\alpha_a}{\mathrm{d}t} = -\frac{(\alpha_a - \alpha_{\mathrm{loc},a})}{\tau_a}. \tag{5.10}$$

The targeted value of the shock viscosity parameter $\alpha_{\mathrm{loc},a}$ is defined using

$$\alpha_{\mathrm{loc},a} \equiv \min \left(10 A_a \frac{h_a^2}{c^2_{\mathrm{s},a}}, \alpha_{\max} \right), \tag{5.11}$$

where

$$A_a \equiv \xi_a \max\left[-\frac{\mathrm{d}}{\mathrm{d}t}\left(\nabla\cdot\boldsymbol{v}_a\right), 0\right], \tag{5.12}$$

is the shock indicator and ξ_a is the corrective factor (Balsara, 1995)

$$\xi \equiv \frac{|\nabla\cdot\boldsymbol{v}|^2}{|\nabla\cdot\boldsymbol{v}|^2 + |\nabla\times\boldsymbol{v}|^2}. \tag{5.13}$$

The rising time $\tau_a \equiv h_a/(c_{s,a}\sigma_d)$ is parameterised by the decay parameter $\sigma_d = 0.1$, a typical value for practical cases. In practice, $\alpha_a(t)$ is set directly to $\alpha_{\mathrm{loc},a}$ if $\alpha_{\mathrm{loc},a} > \alpha_a(t)$. The SPH derivatives exact to the linear order are used to compute

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\nabla\cdot\boldsymbol{v}_a\right) = \sum_i \frac{\partial a_a^i}{\partial x_a^i} - \sum_{i,j}\frac{\partial v_a^i}{\partial x_a^j}\frac{\partial v_a^j}{\partial x_a^i}, \tag{5.14}$$

where for a given field ϕ, this accurate SPH derivative is

$$R_a^{ij}\frac{\partial \phi_a^k}{\partial x_a^j} = \sum_b m_b\left(\phi_b^k - \phi_a^k\right)\frac{\partial W_{ab}\left(h_a\right)}{\partial x^i}, \tag{5.15}$$

where,

$$R_a^{ij} = \sum_b m_b\left(x_b^i - x_a^i\right)\frac{\partial W_{ab}\left(h_a\right)}{\partial x^j} \approx \delta^{ij}. \tag{5.16}$$

Inverting R_a^{ij} and applying it to Eq. 5.15 provides the desired derivative.

1.2. SPH interaction criterion

Eq. 5.3 shows that two SPH particles interact when their relative distance is inferior to the maximum of their interaction radius. Formally, the object-object interaction criterion between two particles a and b is

$$\gamma_{o/o}(a,b) \equiv \left\{|\mathbf{r}_a - \mathbf{r}_b| < \max(l_a, l_b)\right\}. \tag{5.17}$$

Consider now a group of SPH particles, and let us embed them in an axis-aligned bounding box (AABB). Consider another SPH particle. A necessary condition for the latter particle to interact with the AABB is: it resides within the volume formed by extending the AABB in all directions by the maximum of all interaction radii of particles inside the AABB, or, a ball centered on the particle, with a radius equal to its interaction radius, intersects the AABB. Formally, the interaction criterion between the particle and the AABB of particles is therefore

$$\begin{aligned}\gamma^1_{o/g}(a, \{b\}_{b\in AABB}) \equiv &\left(r_a \in AABB \oplus l_{AABB,b}\right)\\ &\vee \Big(B(\mathbf{r}_a, l_a)\cap AABB \neq \emptyset\Big),\end{aligned} \tag{5.18}$$

where $B(\mathbf{r}_a, l_a)$ is a ball centred on $\mathbf{r}_a$ and of diameter $2l_a$, $l_{AABB,b}$ is the maximum interaction radius of the particles in the AABB, $\max_{b\in AABB}(l_b)$, $AABB \oplus l$ is the operation that extends the AABB in every direction by a distance l and $\vee$ is the boolean `or` operator. Consider now the ball centred on $\mathbf{r}_a$ with a diameter of $2l_a$. We denote $AABB(\mathbf{r}_a, l_a)$ the square AABB with a side length of $2l_a$, centred at $\mathbf{r}_a$, ensuring that it encompasses the ball. Replacing the original ball by this AABB in Eq. 5.18 yields the following group-object criterion

$$\gamma^2_{o/g}(a, \{b\}_{b\in AABB}) \equiv (r_a \in AABB \oplus l_{AABB,b}) \\ \vee \Big(AABB(\mathbf{r}_a, l_a) \cap AABB \neq \emptyset\Big). \tag{5.19}$$

Though less stringent than that of Eq. 5.18, this criterion is easier to handle in practice. Indeed, one can show that (App. 6)

$$AABB_1 \oplus h \cap AABB_2 \neq \emptyset \Leftrightarrow AABB_1 \cap AABB_2 \oplus h \neq \emptyset. \tag{5.20}$$

Let $\text{AABB}_{1\text{e}}$ and $\text{AABB}_{2\text{e}}$ denote the extended version of AABB_1 and AABB_2, extended by the distance h in all three directions respectively. Eq. 5.20 asserts that if $\text{AABB}_{1\text{e}}$ intersects AABB_1, it is equivalent for AABB_1 to intersect $\text{AABB}_{2\text{e}}$. Applied on Eq. 5.19, Eq. 5.20 guarantees that the object-group interaction criterion can be rewritten by moving the contribution of the interaction radius of the particle a to the term corresponding to the AABB in the second brackets, as follows

$$\gamma^2_{o/g}(a, \{b\}_{b\in AABB}) \equiv (r_a \in AABB \oplus l_{AABB,b}) \vee \tag{5.21}$$
$$\vee \Big(AABB(\mathbf{r}_a, 0) \cap AABB \oplus l_a \neq \emptyset\Big),$$
$$\equiv (r_a \in AABB \oplus l_{AABB,b}) \tag{5.22}$$
$$\vee (r_a \in AABB \oplus l_a)$$
$$\equiv [r_a \in AABB \oplus \max(l_{AABB,b}, l_a)]. \tag{5.23}$$

The three criteria discussed above satisfy the hierarchy

$$\gamma^2_{o/g}(a, \{b\}_{b\in AABB}) \Leftarrow \gamma^1_{o/g}(a, \{b\}_{b\in AABB}) \\ \Leftarrow \bigvee_b \gamma_{o/o}(a, b). \tag{5.24}$$

Finally, one can extend the first form of $\gamma^2_{o/g}$ to the following group-group interaction criterion

$$\gamma_{g/g}(AABB_1, AABB_2) \equiv \Big([AABB_1 \oplus l_{AABB_1,a}] \cap AABB_2 \neq \emptyset\Big) \\ \vee \Big(AABB_1 \cap [AABB_2 \oplus l_{AABB_2,b}] \neq \emptyset\Big). \tag{5.25}$$

Using Eq. 5.20 similarly as for Eq.5.19 we obtain the form of the group-group interaction criterion used in SHAMROCK,

$$\gamma_{g/g}(AABB_1, AABB_2) \equiv \Big(AABB_1 \cap [AABB_2 \oplus \max(l_{AABB_1,a}, l_{AABB_2,a})] \neq \emptyset\Big). \tag{5.26}$$

In summary, the interaction criteria used for SPH in SHAMROCK are:

- Object-object criterion :

$$\gamma_{o/o}(a, b) = |\mathbf{r}_a - \mathbf{r}_b| < R_{\text{kern}} \max(h_a, h_b)$$

- Object-group criterion :

$$\gamma^2_{o/g}(a, \{b\}_{b\in AABB}) = [r_a \in AABB \oplus R_{\text{kern}} \max(h_{AABB,b}, h_a)]$$

- Group-group criterion :

$$\gamma_{g/g}(AABB_1, AABB_2) = \Big(AABB_1 \cap [AABB_2 \oplus R_{\text{kern}} \max(h_{AABB_1,a}, h_{AABB_2,a})] \neq \emptyset\Big)$$

1.3. Adaptive smoothing length

As mentionned Chapter 1, Sec. 4 in astrophysics, a typical choice consists in choosing h_a in a way that the resolution follows the density

$$\rho(h) = m\left(\frac{h_{\text{fact}}}{h}\right)^3, \tag{5.27}$$

where h_{fact} is a tabulated dimensionless constant that depends on the kernel (e.g. $h_{\text{fact}} = 1.2$ for the M_4 cubic kernel). This specific form also implies that the averaged number of neighbours within the compact support of a given SPH particle is roughly constant throughout the simulation and conveniently ensure proper GPU load balancing in our case. Eq. 5.27 must itself be consistent with the definition of density Eq. 5.2, since h depends on ρ and vice versa. Achieving this requires for density and smoothing length to be calculated simultaneously, by minimising the function

$$\delta\rho = \rho_a - \rho(h_a). \tag{5.28}$$

This approach allows an accurate use of $\rho(h_a)$ in the algorithms rather than calculating the SPH sum. In practice, the iterative procedure is conducted with a Newton-Raphson algorithm. The steps outlined in Alg. 9 describe the iterative procedure used to update the smoothing length. A technicality related to ghost zones

Algorithm 9: Smoothing length update

Data: h_a^n The smoothing lengths at timestep n, χ The ghost zone size tolerance.

Result: h_a^{n+1} The smoothing lengths at timestep $n+1$.

$\{\epsilon_a \leftarrow -1\}_a$;
// Use a copy of h_a^n to do iterations
$\{h_a \leftarrow h_a^n\}_a$;
// Outer loop for ghost exchange
while $\min_a(\epsilon_a) = -1$ **do**
  ... exchange ghosts positions with tolerance χ ...;
  // Inner loop for Newton-Rahpson
  while $\max_a(\epsilon_a) > \epsilon_c$ **do**
    for a ***in parallel*** **do**
      // Compute the SPH sum
      $\rho_a \leftarrow \sum_b m_b W_{ab}(h_a)$;
      // Newton-Rahpson
      $\delta\rho \leftarrow \rho_a - \rho(h_a)$;
      $d\delta\rho \leftarrow \sum_b m_b \frac{\partial W_{ab}(h_a)}{\partial h_a} + \frac{3\rho_a}{h_a}$;
      $h_a^{n+1} \leftarrow h_a - \delta\rho / d\delta\rho$;
      // Avoid over/under-shooting
      if $h_a^{n+1} > h_a \lambda$ **then**
        $h_a^{n+1} \leftarrow h_a \lambda$;
      else if $h_a^{n+1} < h_a / \lambda$ **then**
        $h_a^{n+1} \leftarrow h_a / \lambda$;
      $\epsilon_a \leftarrow |h_a^{n+1} - h_a| / h_a^n$;
      // Exceed ghost size
      if $h_a^{n+1} > h_a^n \chi$ **then**
        $h_a^{n+1} \leftarrow h_a^n \chi$;
        $\epsilon_a \leftarrow -1$;

arises during this procedure. The size γ_{12} of the ghost zone separating two adjacent patches, P_1 and P_2, is determined by the group-group interaction criterion between these patches

$$\gamma_{12} = \max\left(\max_{\{a\}} h_a, \max_{\{b\}} h_b\right). \tag{5.29}$$

where a and b stem for indices of particles in P_1 and P_2 respectively. In SHAMROCK, the size γ_{12} is increased by a safety factor χ, termed as the *ghost zone size tolerance*. This factor acknowledges that ghost zone structures should withstand fluctuations in smoothing lengths throughout the iterative process. With this tolerance, the smoothing length can fluctuate by a factor of χ during density iterations without necessitating SHAMROCK to regenerate the ghost zones. In practice, we first exchange the ghost zones using a tolerance $\chi = 1.1$, then iterate until all particles converge to the consistent smoothing length or exceed the ghost zone size tolerance. If the latter occurs, we restart the process from the beginning with the updated smoothing length. We find that this almost rarely arises, except during the initial time step when the smoothing length is converged for the first time. Alg. 9 shows that in SHAMROCK, we use an additional safety factor, denoted as λ, to prevent over- and undershooting throughout the iterations. Without this correction, the iterative procedure may yield unstable negative smoothing lengths. In practice, we use $\lambda = 1.2$.

1.4. Time stepping

1.4.1. Leapfrog integration

In SHAMROCK, we employ a symplectic second-order leapfrog integrator, or 'Kick-drift-kick' (e.g. Verlet 1967; Hairer et al. 2003), which achieves second-order accuracy in space in smooth flows:

$$\boldsymbol{v}^{n+\frac{1}{2}} = \boldsymbol{v}^n + \frac{1}{2}\Delta t \boldsymbol{a}^n, \tag{5.30}$$

$$\boldsymbol{r}^{n+1} = \boldsymbol{r}^n + \Delta t \boldsymbol{v}^{n+\frac{1}{2}}, \tag{5.31}$$

$$\boldsymbol{v}^{*} = \boldsymbol{v}^{n+\frac{1}{2}} + \frac{1}{2}\Delta t \boldsymbol{a}^n, \tag{5.32}$$

$$\boldsymbol{a}^{n+1} = \boldsymbol{a}\left(\boldsymbol{r}^{n+1}, \boldsymbol{v}^{*}\right), \tag{5.33}$$

$$\boldsymbol{v}^{n+1} = \boldsymbol{v}^{*} + \frac{1}{2}\Delta t \left[\boldsymbol{a}^{n+1} - \boldsymbol{a}^n\right], \tag{5.34}$$

where $\mathbf{r}^n$, $\mathbf{v}^n$ and $\mathbf{a}^n$ denote positions, velocities and acceleration at the $n^{-\text{th}}$ time step Δt. In the scheme presented in Price et al. 2018, a combined iteration is used to calculate the acceleration $\boldsymbol{a}^{n+1}$ and update the smoothing length at the same time. To minimise the amount of data communicated, we separate the acceleration and the smoothing length update. In SHAMROCK, the smoothing length is calculated after applying Eq. 5.32. Only positions are required for the smoothing length iteration.

Once these iterations are complete, we first calculate Ω_a using Eq. 5.4, then exchange the ghost zones with the required fields, including Ω_a subsequently used in derivative computations. Similar to the approach used in PHANTOM, we use the correction applied to the velocity, calculated during the correction step of the leapfrog, as a reference to check that the resulting solution is reversible over time. The correction applied at the end of the leapfrog scheme is as follows

$$\Delta \mathbf{v}_i = \frac{1}{2}\Delta t \left[\boldsymbol{a}_i^{n+1} - \boldsymbol{a}_i^n\right]. \tag{5.35}$$

We use the result of Eq. 5.35 to verify that the maximum correction does not exceed a fraction ϵ_{v} of the mean square correction

$$\max_i \left(|\Delta \mathbf{v}_i| / \sqrt{\frac{1}{N}\sum_j |\Delta \mathbf{v}_j|^2|} \right) < \varepsilon_{\mathrm{v}}. \tag{5.36}$$

In practice, we set the value $\varepsilon_{\mathrm{v}} = 10^{-2}$. If any particles fail to meet this criterion, we recalculate the acceleration and apply the correction step again with $\boldsymbol{v}^* \leftarrow \boldsymbol{v}^{n+1}$ instead.

1.4.2. Choice of the timestep

The value of the explicit time step is governed by the Courant-Friedrich-Levy stability condition (Courant et al., 1928). Following Price et al. (2018) from Lattanzio et al. (1986); Monaghan (1997),

$$\Delta t \equiv \min(C_{\mathrm{cour}} \frac{h_a}{v_{\mathrm{sig},a}^{\mathrm{dt}}}, C_{\mathrm{force}} \sqrt{\frac{h_a}{|\boldsymbol{a}_a|}}). \tag{5.37}$$

The first term allows to correctly capture the propagation of the hydrodynamic characteristic waves in the fluid at a given resolution. Similarly, the second term ensures correct treatment of the action of external forces on the fluid. The safety coefficients are set to the following values $C_{\mathrm{cour}} = 0.3$ and $C_{\mathrm{force}} = 0.25$.

1.4.3. CFL multiplier

To minimize the cost associated with executing the correction cycles of the leapfrog scheme, we reduce the time step for a few iterations when Eq. 5.36 is not satisfied, similar to the approach taken in PHANTOM. To do this in SHAMROCK, we introduce a so-called *CFL multiplier* λ_{CFL}, which consists of an additional variable factor applied to the CFL condition. Therefore, the effective C_{cour} and C_{force} used in SHAMROCK SPH solver are

$$C_{\mathrm{cour}} = \lambda_{\mathrm{CFL}} \tilde{C}_{\mathrm{cour}}, \quad C_{\mathrm{force}} = \lambda_{\mathrm{CFL}} \tilde{C}_{\mathrm{force}}, \tag{5.38}$$

where $\tilde{C}_{\text{cour}}$ and $\tilde{C}_{\text{force}}$ are the safety coefficients chosen by default by the user. If Eq. 5.36 is not satisfied, we divide λ_{CFL} by a factor of 2. Otherwise, at each time step,

$$\lambda_{\text{CFL}}^{n+1} = \frac{1 + \lambda_{\text{stiff}} \lambda_{\text{CFL}}^{n}}{1 + \lambda_{\text{stiff}}}, \tag{5.39}$$

where λ_{stiff} is a coefficient that parameters the stiffness of the evolution of the CFL multiplier. This numerical strategy allows to handle shocks in the simulation, automatically cycling leapfrog iterations over the CFL condition, thereby reducing the time step to enhance energy conservation. This procedure is particularly effective during the first time steps of the Sedov-Taylor blast problem.

1.4.4. Shock detection

The shock viscosity parameter α is evolved according to Eq.5.10. After the leafprog prediction step, an implicit time step is used for this integration

$$\alpha_{\text{loc},a}^{n+1} = \alpha_{\text{loc},a}(\boldsymbol{v}^*, \nabla \boldsymbol{v}^*, \nabla \boldsymbol{a}^n), \tag{5.40}$$

$$\alpha_a^{n+1} = \max\left(\frac{\alpha_a^n + \alpha_{\text{loc},a}^{n+1} \Delta t/\tau_a}{1 + \Delta t/\tau_a}, \alpha_{\text{loc},a}^{n+1}\right). \tag{5.41}$$

1.4.5. Summary

We have implemented in Shamrock an SPH hydrodynamical solver with self-consistent smoothing length that handles shock though the combined used of shock viscosity and conductivity with state-of-the-art shock detector. Fig. 5.1 shows a comprehensive overview of one SPH time step in Shamrock.

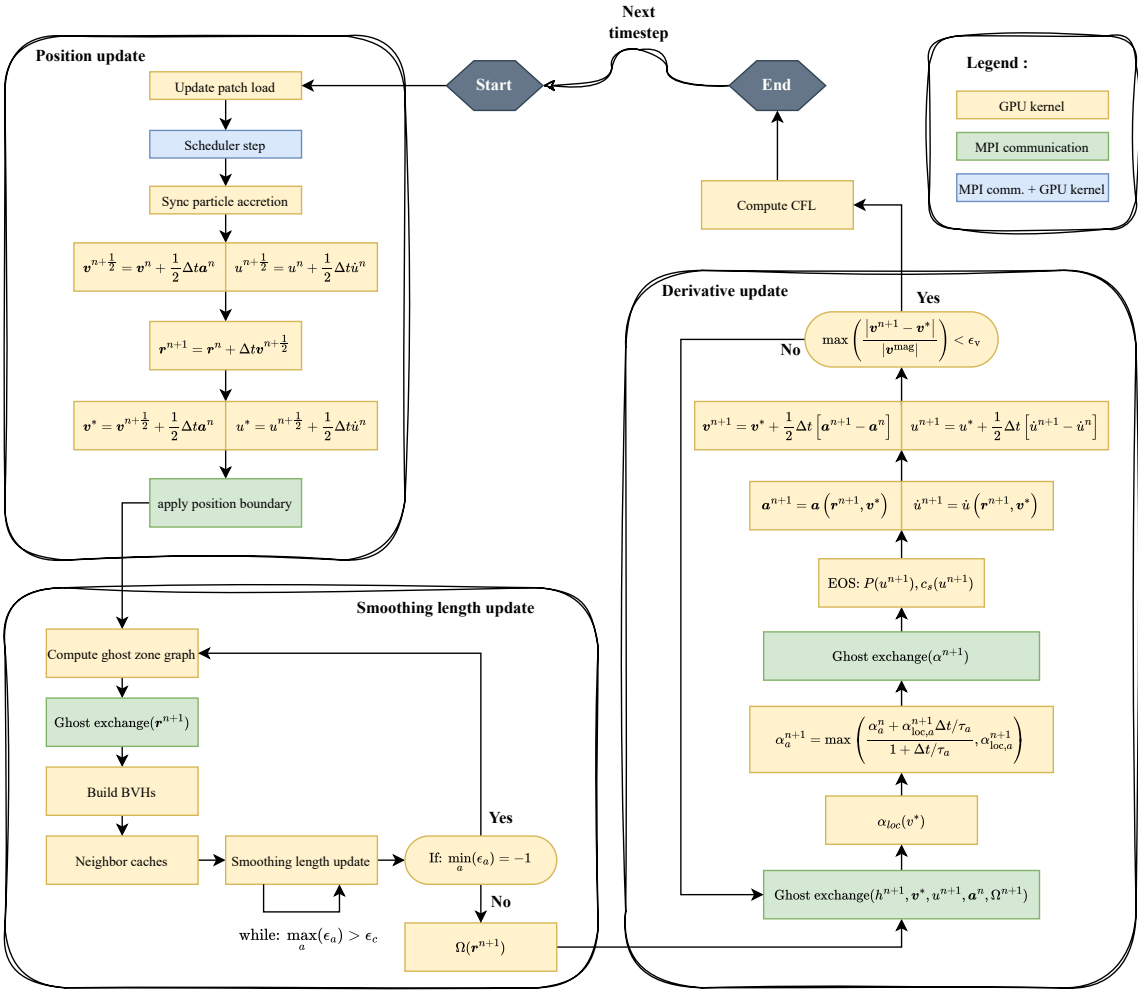


Figure 5.1: Illustration of an SPH time step through an organisational diagram representing one time step of the SPH scheme, the process being divided into three main sub-steps. Firstly, position updates (scheduling step for patch decomposition, leapfrog prediction, and application of position boundaries if necessary). Secondly, smoothing length updates (generation of ghost zone graph, construction of BVHs, creation of neighbour caches, smoothing length adjustment, computation of Ω). Thirdly, derivative updates (field exchange, viscosity and derivative updates, application of leapfrog corrector). The step concludes with updating the CFL condition. The corresponding equations showed on this flowchart correspond to the position and derivative update to the equations shown Sect. 1.4.1 and 1.4.4. For the smoothing length the corresponding equations are detailed in Sect. 1.3.

2. Physical tests

2.1. Generalities

First, we validate the SPH solver by performing convergence tests against classical problems having analytical solutions, such as the Sod tube and the Sedov-Taylor blast test. The hydrodynamic tests presented in this section are performed using the M_6 kernel with $h_{fact} = 1.0$, with an average number of neighbours of 113 neighbours for each SPH particle (almost no difference is expected in the results when using other spline kernels, e.g. Price et al. 2018). In all tests, momentum is conserved to machine precision. The choice of the CFL condition result in energy deviations that do not exceed 10^{-6} relative error with respect to the initial value.

Secondly, we examine the residuals obtained from comparing the results generated by Shamrock and Phantom. Since both codes use the same SPH algorithm, such an analysis is required for conducting further performance comparisons. L2 errors are estimated using the norm

$$\mathcal{L}_2(A_{\rm sim}, A_{\rm ref}) = \sqrt{\frac{1}{N_{\rm part}} \sum_i |A_{i,\rm sim} - A_{i,\rm ref}|^2}, \tag{5.42}$$

where $A_{i,\rm ref}$ represents the reference quantities, while $A_{i,\rm sim}$ denotes the quantities computed in the simulation.

2.2. Advection

We first perform an advection test in a periodic box of length unity to verify the correct treatment of the periodic boundaries by Shamrock. Three lattices of $(16 \times 12 \times 12)$, $(64 \times 24 \times 24)$ and $(16 \times 12 \times 12)$ particles having velocities $v_x = 1.0$ are initially juxtaposed, such that $\rho = 1.0$ if $0.25 \leq x \leq 0.75$, and $\rho = 0.1$ elsewhere. We let the simulation evolve until the step has crossed several times the boundaries of the box (we choose $t = 11$, although any other time, even very large, yields the same outcome since SPH is Galilean invariant). The result obtained at the end of the simulation is identical to the initial setup to machine precision.

2.3. Sod tube

We perform a Sod-tube test (Sod, 1978) by setting up a box with a discontinuity between a left state and a right state initially positioned at $x = 0.5$. In the left state $x < 0.5$, the density and the pressure are set to $\rho_{\rm l} = 1$ and $P_{\rm l} = 1$, while in the right state $x > 0.5$, they are set to $\rho_{\rm r} = 0.125$ and $P_{\rm r} = 0.1$ respectively. To initialise the density profile, we use a periodic box in which we setup a $24 \times 24 \times 256$ hexagonal close packed lattice in $x \in [-0.5, 0.5]$ and $12 \times 12 \times 128$ in $x \in [0.5, 1.5]$. The initial velocity is uniformly set to zero throughout the simulation. The size of the simulation box size is adjusted such that we ensure periodicity across the y and z boundaries.

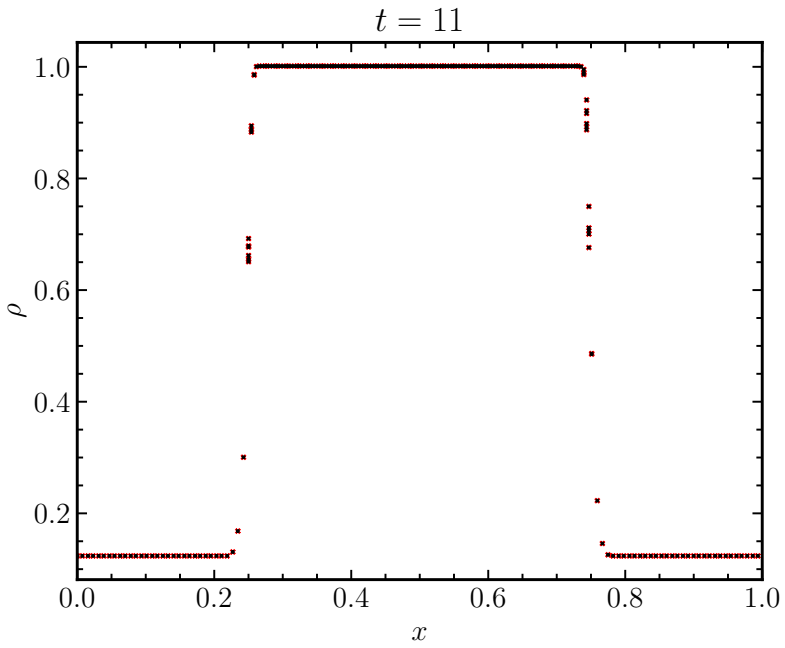


Figure 5.2: Advection of a density step across several traversal of a periodic box, in code units. SPH being Galilean invariant, the results (black dots) precisely match the initial setup (red crosses) down to machine precision, thus validating the boundary treatment in Shamrock.

We use $\gamma = 1.4$ to align our test with the Sod tube test commonly performed in grid codes. No particle relaxation step is used in this test, since the initial distribution of SPH particles closely resembles a relaxed distribution akin to a crystal lattice. We use periodic boundaries in the x direction. Shock viscosity is setup with the default parameters of Shamrock, namely $\sigma_d = 0.1$, $\beta = 2$, $\beta_u = 1$. The setup presented above is then evolved until $t = 0.245$. Fig. 5.3 shows results obtained for velocity, density, and pressure, displaying additionally the shock-capturing parameter α. For $N_x = 128$ particles $\mathcal{L}_2$ errors are $\sim 10^{-3}$ in v and $\sim 10^{-4}$ in ρ and P, similarly to what is obtained with other SPH codes. Similar setups are used to perform convergence analysis, except for the lattice, for which we use $24 \times 24 \times N_x$, and $12 \times 12 \times (N_x/2)$ particles instead respectively. Results obtained when varying the value of N_x are reported in Fig. 5.4. We observe second-order convergence on the pressure, first-order convergence in density, and in-between convergence in velocity. The scattering observed in the velocity field behind the shock corresponds to particle having to reorganise the crystal lattice, a typical feature of SPH (e.g. Price et al. 2018). Letting the shock evolve further and interact with the periodic boundary, we verify that we obtain a second symmetric shock, as expected.

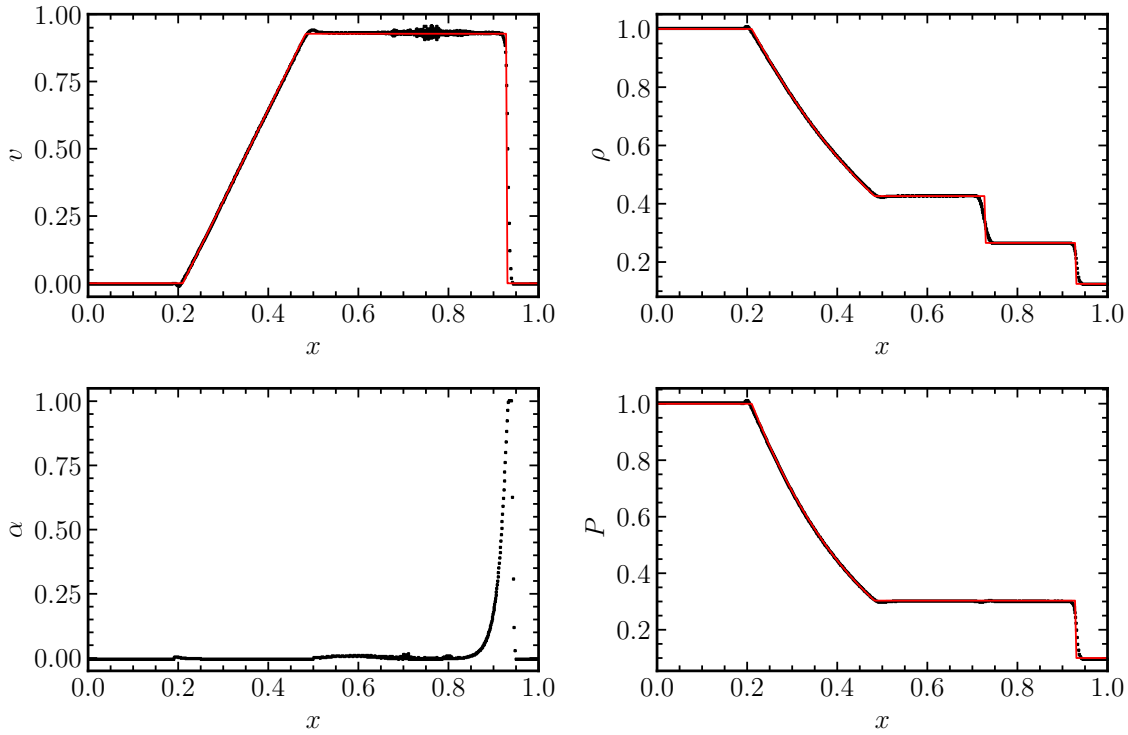


Figure 5.3: Result obtained for the Sod-tube test by juxtaposing two tubes of $24 \times 24 \times 512$ particles in $x \in [-0.5, 0.5]$ and $12 \times 12 \times 256$ particles in $x \in [0.5, 1.5]$ organised in hexagonal compact packing lattices. The density is set to $\rho = 1$ in $x \in [-0.5, 0.5]$ and $\rho = 0.125$ in $x \in [0.5, 1.5]$. Initial pressure is $P = 1$ for $x \in [-0.5, 0.5]$ and $P = 0.1$ for $x \in [0.5, 1.5]$, with zero initial velocities. An adiabatic equation of states with $\gamma = 1.4$ is used. Boundaries are periodic, and only half of the simulation is displayed. The simulation is performed until $t = 0.245$, and numerical results are compared against the analytic solution. We additionally show the values of the shock viscosity parameter α.

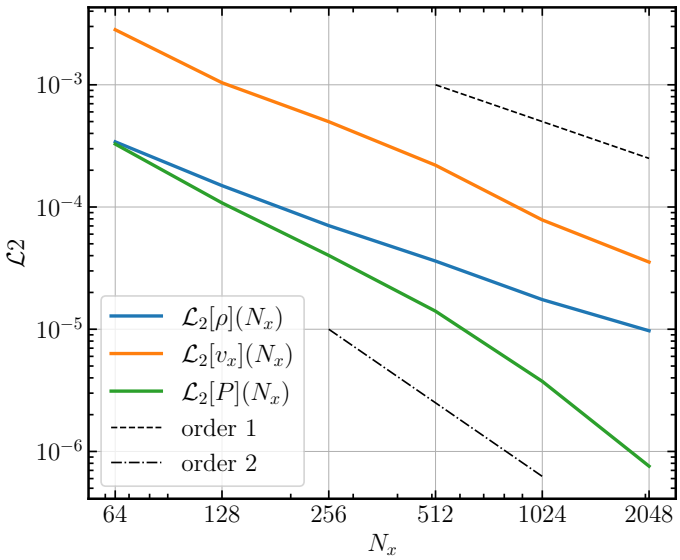


Figure 5.4: $\mathcal{L}_2$ errors obtained for the Sod shock tube test presented on Fig.5.3 as a function of the number N_x of particles used on the x axis for $x \in [-0.5, 0.5]$. We observe second-order convergence on the pressure, first-order convergence in density, and in-between convergence in velocity, as found in other SPH codes.

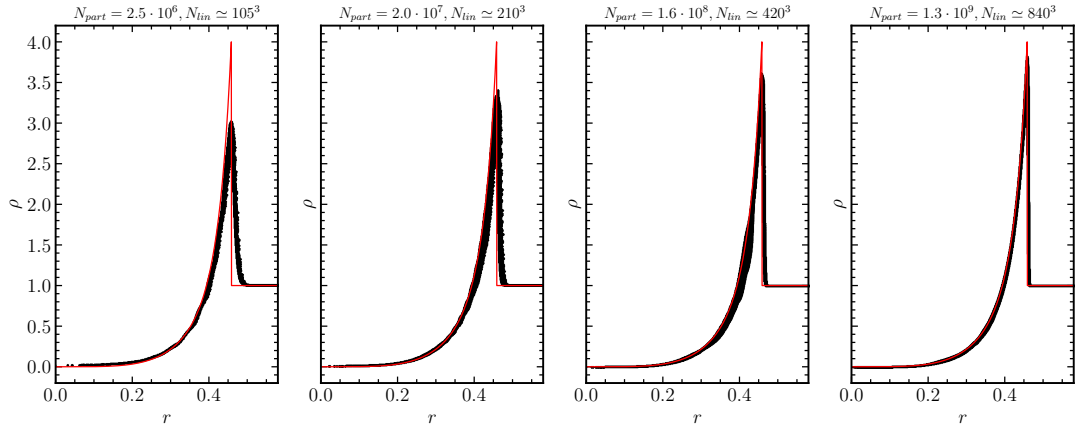


Figure 5.5: Result of the densities (black dots) obtained for the Sedov-Taylor blast test described in Sect. 2.4 at $t = 0.1$ for N_{part} particles, with $N_{\text{part}} = 2.5 \cdot 10^6, 2.0 \cdot 10^7, 1.6 \cdot 10^8, 1.3 \cdot 10^9$ SPH particles (global time-stepping), corresponding to an inter particle spacing on the HCP lattice of respectively $10^{-2}/4, 10^{-2}/8, 10^{-2}/16, 10^{-2}/32$. Results are represented against the analytical solution (solid red line). The legend provides the effective linear resolution N_{lin} corresponding the cubic root of the number of particle displayed on each graphs which are truncated at $r = 0.58$.

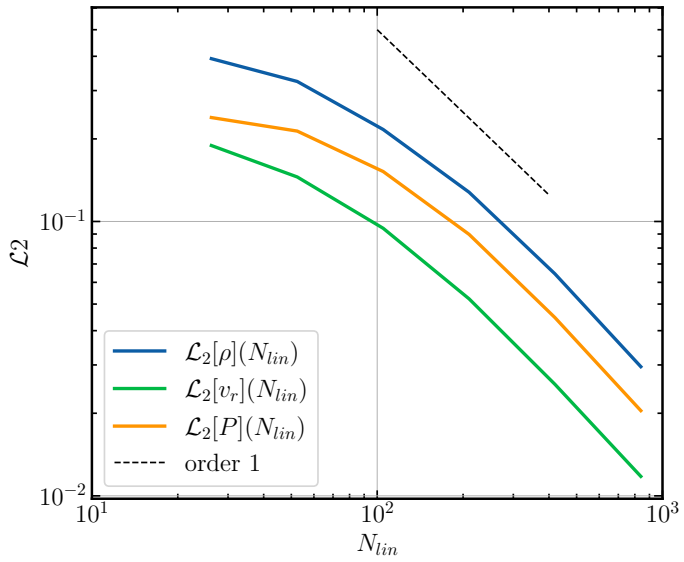


Figure 5.6: Convergence study of the Sedov-Taylor blast test presented in Fig. 5.5. Order one convergence is achieved for v, ρ and P, similarly to what is obtained with other SPH codes.

2.4. Sedov-Taylor blast

We perform a Sedov-Taylor blast wave test (Sedov, 1959; Taylor, 1950a,b) by first setting up a medium of uniform density $\rho = 1$ with $u = 0$ and $\gamma = 5/3$, in a 3D box of dimensions $[-0.6, 0.6]^3$. The particles are arranged locally on a compact hexagonal lattice. The smoothing length is initially converged by iterating a white time step. Internal energy is then injected in the centre of the box. This energy peak is smoothed by the SPH kernel according to $u_a = W(\mathbf{r}, 2h_0) \times E_0$, where the total amount of energy injected is fixed at $E_0 = 1$ and h_0 is the smoothing length of the particles after relaxation. For this test, the CFL condition is lowered to $\tilde{C}_{\text{cour}} = \tilde{C}_{\text{force}} = 0.1$ to prevent leapfrog corrector sub-cycling caused by the strong shock. This result in an enhanced energy conservation, with a maximum relative error of 10^{-6} observed across all tests. The simulation is then evolved up to $t = 0.1$. Simulations with $N = 26, 52, 105, 210$ have been performed on a single A100 GPU of an NVidia DGX workstation. Simulations with $N = 420$ and $N = 840$ were executed on the Adastra supercomputer (see Sect. 3) on 4 and 32 nodes respectively. The highest resolution blast test involves 1.255 Gpart, including ghost particles. The simulation consists in 14979 iterations performed in 14 hours, including setup and dumps, on 32 nodes corresponding to 128 Mi250X or equivalently 256 GCDs (see Sect. 3.1.1 for details). The total energy consumed for this test is 1.94 GJ, as reported by Slurm. The power consumption per node is 1195 W, which equates to slightly over half of the peak consumption of a single node (2240 W). Numerical results are compared against analytical solutions. Fig.5.5 shows results obtained for the density for N^3 particles, with $N = 105, 210, 420, 840$. For the latter case, $\mathcal{L}_2$ errors are of

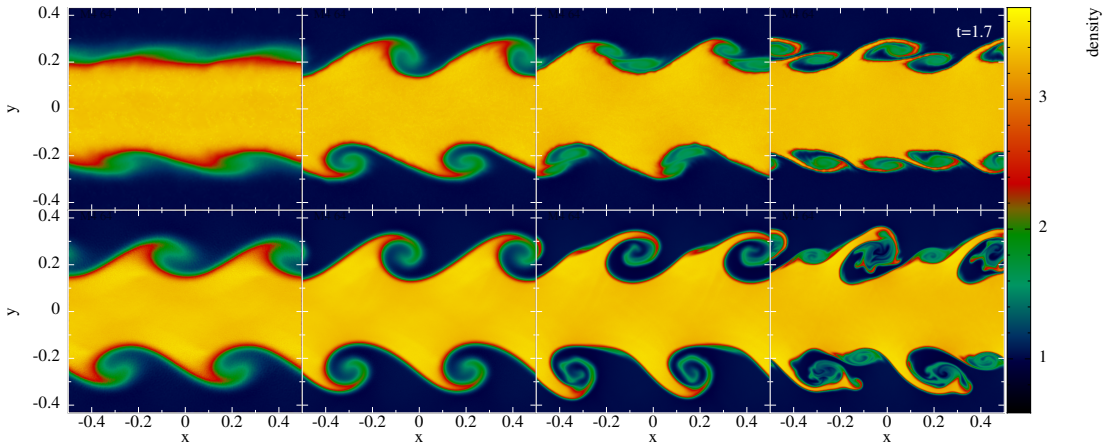


Figure 5.7: Density profiles obtained in the low-resolution 3D Kelvin-Helmholtz test described in Sect. 2.5 at $t = 1.7$ with the M_4 kernel (top panel) and the M_6 quintic kernel (bottom panel) respectively in code units. The instability is correctly captured with the M_6 kernel, similarly to the findings of Tricco (2019). From left to right, the numbers of particles N_l and N_r used along the x axis for the low-density and the high-density regions are: $N_\mathrm{l} = 128$ and $N_\mathrm{r} = 192$, $N_\mathrm{l} = 256$ and $N_\mathrm{r} = 384$, $N_\mathrm{l} = 512$ and $N_\mathrm{r} = 758$, $N_\mathrm{l} = 1024$ and $N_\mathrm{r} = 1536$, which corresponds to 215×10^3, 860×10^3, 3.4×10^6 and 13.7×10^6 particles respectively.

order $\sim 10^{-1}$, which is similar to what is obtained with other SPH codes with this particular setup. Figure 5.6 shows that order one convergence is achieved for v, ρ and P, similarly to what is obtained with other SPH codes.

2.5. Kelvin-Helmholtz instability

We test the ability of Shamrock to capture instabilities related to discontinuities on internal energy by performing a Kelvin-Helmholtz instability test (Price, 2008). We adopt a setup close to the one proposed by Schaal et al. (2015), that gives rise to secondary instabilities fostering additional turbulence mixing. The initial pressure, density and velocity profiles are initialised according to

$$P = 3.5 \tag{5.43}$$

$$\rho = \begin{cases} 1, & \text{if } |y| > y_s/4, \\ (3/2)^3, & \text{if } |y| \leq y_s/4, \end{cases} \tag{5.44}$$

$$v_x = \begin{cases} \xi/2, & \text{if } |y| > y_s/4, \\ -\xi/2, & \text{if } |y| \leq y_s/4, \end{cases} \tag{5.45}$$

$$v_y = \varepsilon \sin(4\pi x) \left\{ \exp\left(-\frac{(y - y_s/4)^2}{2\sigma^2} \right) + \exp\left(-\frac{(y + y_s/4)^2}{2\sigma^2} \right) \right\}. \tag{5.46}$$

The test is performed in 2.5D, restricting the z axis to a thin layer comprising only 6 SPH particles in the low density region, and 9 particles in the high density region. We opt for a density ratio of $(3/2)^3$ between the two regions to simplify the particle setup process and circumvent unnecessary complexities associated with arranging particles on closed-packed lattices. We use $\gamma = 1.4$. The slip velocity and the perturbation parameters are $\xi = 1$, $\epsilon = 10^{-2}$, $\sigma = 0.05/\sqrt{2}$, similarly to the values used in Schaal et al. (2015). Simulations are performed on a single A100-40GB GPU. This GPU can accommodate a maximum of approximately $\sim 40 \cdot 10^6$ particles for the M_4 kernel and $\sim 20 \cdot 10^6$ particles for the M_6 kernel. Fig. 5.7 shows results obtained at increasing resolutions for the M_4 kernel (top panel) and the M_6 kernel (bottom panel). Similarly to the findings of Tricco (2019), we first observe that the M_4 fails to accurately capture the instability, even at high resolutions, as vortices appear flattened and overly diffused. Conversely, we observe that all these features are effectively captured when employing the M_6 kernel. The further Sect. 2.6 shows that our results align almost perfectly with those obtained with Phantom. The growth rate observed for the instability matches therefore the findings reported in Tricco (2019).

2.6. Conformance with Phantom

We aim to benchmark the performance of Shamrock against a state-of-the-art, robust, optimised and extensively tested SPH code running on CPUs. Several SPH codes are in use in the community (e.g. Bonsai-SPH Bedorf & Portegies Zwart 2020, Gadget Springel et al. 2021, Gasoline Wadsley et al. 2004, Gizmo Hopkins 2014, Seren Hubber et al. 2011, Swift Schaller et al. 2018). We choose Phantom, since it is optimised for hydrodynamics, well-used by the astrophysical community and extensively tested and documented (Price et al., 2018). Before conducting comparisons, one has to ensure that the two solvers are rigorously identical, up to identified insignificant errors. This is the purpose of the next two tests, that are uncommonly designed to reveal discrepancies by amplifying errors using lower resolution or less regular kernels than achievable. For this, we generate the initial conditions with Phantom, then start an identical simulation from the same dump using a fixed time step.

2.6.1. Residuals: Low res Sedov-Taylor blast wave test

We first measure the residual discrepancies between Shamrock and Phantom by comparing results obtained on two identical Sedov-Taylor blast wave tests described in Sect. 2.4, fixing the time step to $dt = 10^{-5}$. This three-dimensional test is highly sensitive to rounding errors, primarily due to the presence of a low-density, zero-energy region surrounding the blast wave. In particular, aligning the behaviours of the shock viscosity parameter $\alpha^{\rm AV}$ proves being particularly challenging. Finally, Fig. 5.8 shows that discrepancies between Shamrock and Phantom are imperceptible to the naked eye. Quantitatively, the $\mathcal{L}_2$ errors are

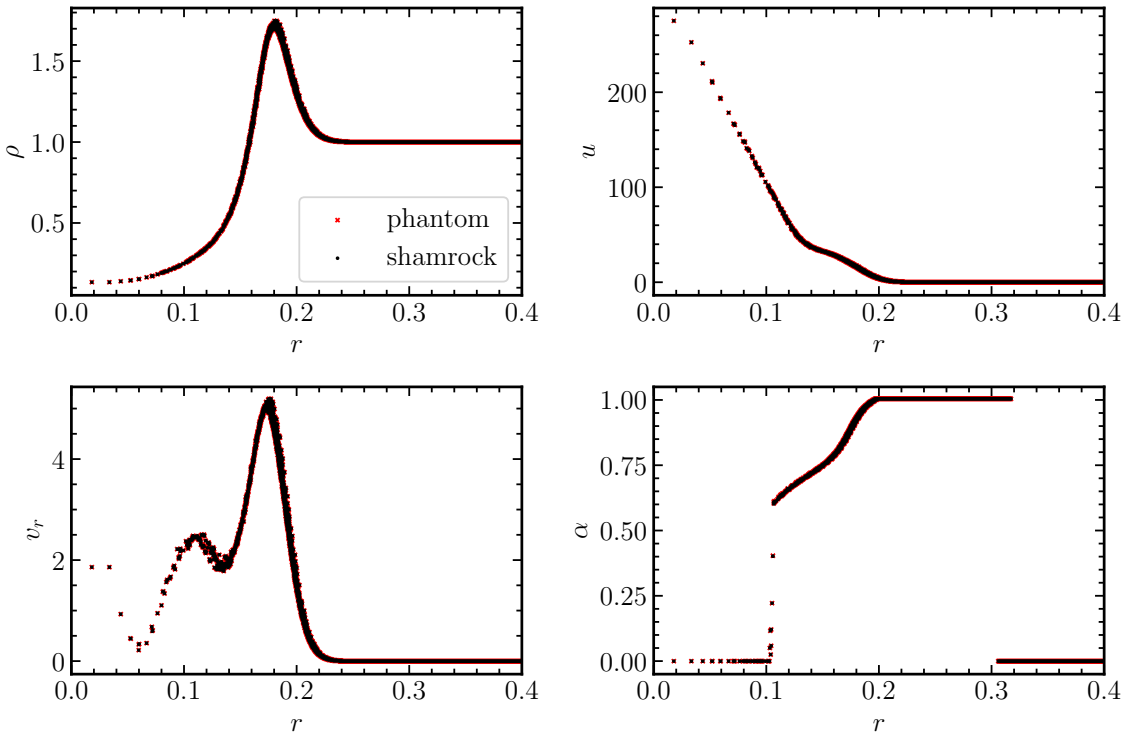


Figure 5.8: A comparison is made between the densities ρ, internal energies u, velocities v_r, and shock detection parameters α obtained at $t = 1$ from two identical low-resolution Sedov-Taylor blast wave tests conducted by PHANTOM (red dots) and SHAMROCK (black dots). Initially, PHANTOM is used to generate the same setup file for the two simulations. Runs are then conducted using a fixed time-step of $dt = 10^{-5}$. The dots are indistinguishable by eye (e.g. the $\mathcal{L}_2$ error on the velocity field is of order $5 \cdot 10^{-4}$): the implementations of the SPH solver are identical in the two codes.

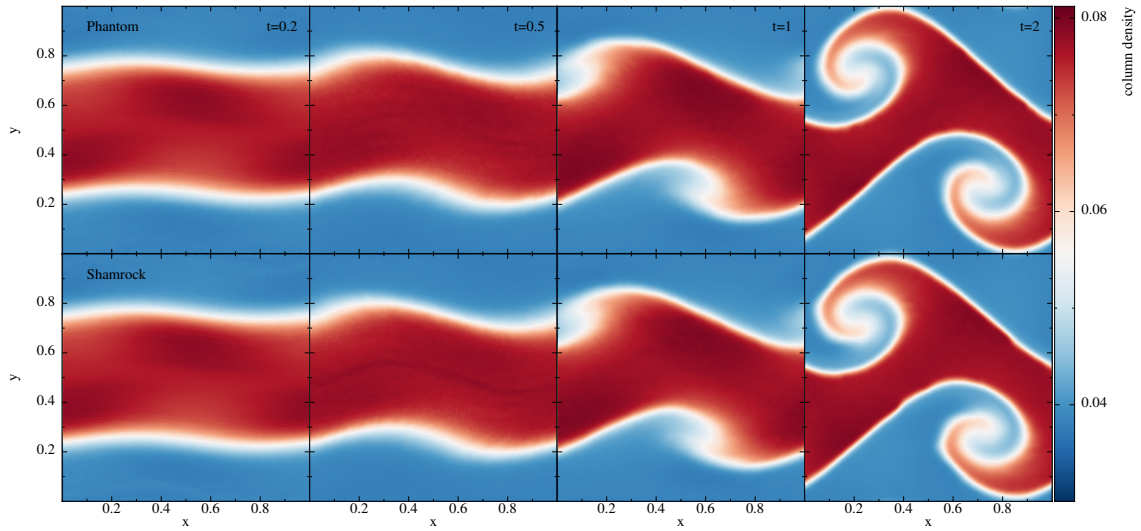


Figure 5.9: Density profiles obtained at $t = 0.2, 0.5, 1, 2$ on a low resolution Kelvin-Helmholtz test by PHANTOM (top panel) and SHAMROCK (bottom panel). Results are almost identical. The test is voluntarily performed with an unsuited M4 kernel at low-resolution to accentuate residual discrepancies between the two solvers. Those stem from the use of single precision in one and double precision in the other for shock detection variables.

- relative $\mathcal{L}_2$ distance r : $2.0869658802024003e-07$,
- relative $\mathcal{L}_2$ distance h : $3.952645327403623e-05$,
- relative $\mathcal{L}_2$ distance v_r : 0.0005418229957181854,
- relative $\mathcal{L}_2$ distance u : $3.6622341394801246e-05$.

Following an in-depth examination, the sole identified distinctions between the two solvers are as follows: in PHANTOM, the shock parameter $\alpha^{\rm AV}$ and the estimate of $\nabla\cdot\mathbf{v}$ are stored as single-precision floating-point numbers, while in SHAMROCK, they are double-precision.

2.6.2. Residuals: Low res Kelvin-Helmholtz instability test

We measure the residuals between SHAMROCK and PHANTOM by performing the Kelvin-Helmholtz instability test implemented in PHANTOM at commit number `e01f76c3`, at low resolution. Simulations are evolved to $t = 2$, while dumps are produced every $\Delta t = 0.1$ to sample the development of the instability. We choose the M4 kernel and a low number of particles to reveal the differences between the codes. Fig. 5.9 and Fig. 5.10 show the compared evolutions of the density and of the shock parameter respectively. At $t = 0.2$, no difference is observed in the density field. For shock viscosity, we observe for PHANTOM a small noise of relative

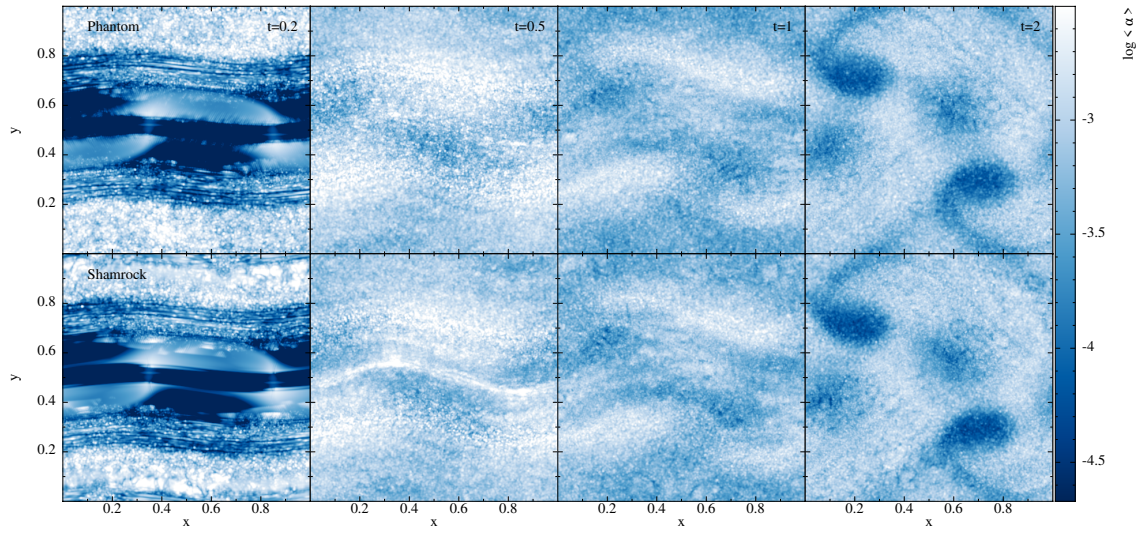


Figure 5.10: Same plot as in Fig.5.9, revealing the amplitude of truncature errors in the shock viscosity parameter. To our understanding, these errors represent the sole source of discrepancies between the implementations of SPH in PHANTOM and SHAMROCK.

amplitude $\lesssim 0.1\%$ along the line $y = 0.5$, which we attribute to $\alpha, h, \nabla \cdot \mathbf{v}$ being stored as single precision fields in PHANTOM. These fluctuations are not present in SHAMROCK, since these quantities are calculated in double precision. At $t = 0.5$ we can distinguish at this low resolution a small line of higher shock viscosity and density in SHAMROCK, which is not present in PHANTOM. We attribute these residuals to the fact that the PHANTOM simulation may have increased numerical viscosity due to single precision errors, while the SPH lattice in the SHAMROCK simulation is still reorganising. At $t = 1$, no significant difference is observed between the two simulations. Finally, at $t = 2$, tiny differences can be observed at the edges of the instability, for the same reasons as at $t = 1$.

2.7. Summary

The hydrodynamic SPH solver implemented in SHAMROCK passes successfully the standard tests (advection, Sod tube, Sedov-Taylor blast, Kelvin-Helmholtz instability). The implementation of the SPH solver in SHAMROCK is identical to that of PHANTOM. Results obtained with the two codes are almost indistinguishable, the residuals being attributed to the choice of floating-point precision for the quantities $h, \alpha, \nabla \cdot \mathbf{v}$. This sets the basis for rigorous performance comparison.

3. Performance

3.1. Characteristics of the benchmarks

3.1.1. Hardware specificities

The tests performed to estimate performance with Shamrock were conducted on two systems. Single GPU and CPU tests were performed on an Nvidia A100-SXM4-40GB GPU of an Nvidia DGX workstation. This workstation is equipped with 4 Nvidia A100 40GB GPUs paired with an Epyc7742 64-core CPU, and are exploited via SIDUS (Quemener & Corvellec, 2013) by the Centre Blaise Pascal at ENS de Lyon . CPU tests were carried out on the CPU of the same DGX workstation using AdaptiveCPP OpenMP backend. For those Shamrock was compiled using `-O3 -march=native`. For single GPU tests of Shamrock compilation was performed using the Intel fork of the llvm/clang-19 compiler, also referenced as ONEAPI/DPC++ with optimizations `-O3 -march=native`. We voluntarily didn't used fast math optimisations as they would not be used in production. We use the CUDA/PTX backend of Intel llvm targetting the CUDA architecture `sm_s70` corresponding to the compute capability of A100 GPUs. The Intel LLVM CUDA/PTX backend generates code using PTX ISA, the assembly language used to represent CUDA kernels. This result in a program that actually that is first lowered from C++ to PTX, then compiled using Nvidia `ptxas` tool, which enable the code to be profiled using Nvidia's CUDA tools. The CUDA version used is 12.0.

Multi-GPU and multi node tests were performed on Adastra Supercomputer at CINES in France, using up to 256 compute nodes, each compute node is a HPE Cray EX235a each equipped with 4 Mi250X GPUs paired with a 64 Cores AMD Epyc Trento CPU. On this platform we used ROCm/HIP backend of Intel llvm targetting the AMD GPU architecture `gfx90a` corresponding to the compute capability of Mi250X GPUs. The Intel llvm compiler was compiled using in the same module environment as Shamrock. We used Cray CPE 23.12 with acceleration on `gfx90a` and Trento on the host, in conjunction with the provided `PrgEnv-intel`. MPI with GPU aware support support was provided by the cray MPIch 8.1.26 module. ROCm support was provided by both `amd-mixed` 5.7.1 and `rocm` 5.7.1. Although the Mi250X GPU is a single chip, it is made up of two GCDs, which appear as separate instances on the compute node, where one MPI rank is assigned per GCDs.

3.1.2. Setups

We present the performances of the SPH hydrodynamical solver of Shamrock on the Sedov Taylor blast wave, since it involves contributions of all the different terms in the hydrodynamical solver, and it is neither specific to astrophysics nor SPH. We compare the results with the one obtained with the hydro dynamical solver of Phantom with an almost identical implementation (see Sect. 2.6), on a computing units having similar power consumption.

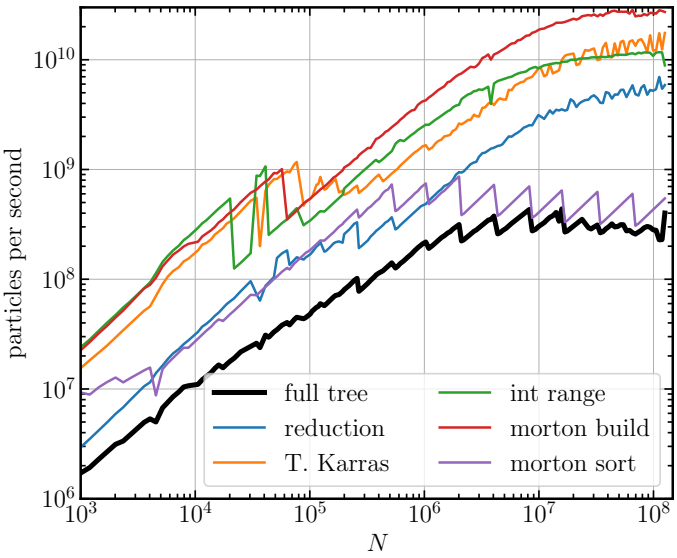


Figure 5.11: Benchmark of the performance of the tree building in SHAMROCK. Each curve represents the number of particles processed per second for various segments of the algorithm. The thick solid black curve shows the total time to build the tree. The other curves show the performance of the main algorithms involved in the tree building procedure. Those correspond to benchmarks of the isolated algorithms, which break the asynchronous nature of SYCL. As such, the sum of the individual times do not rigorously add up to the exact time of the entire algorithm. This benchmark used a dataset of input positions generated from an hexagonal closed packing lattice, with variations in lattice spacing. Varying the initial distribution of particles will not affect total performance of the tree, since overall, the building time is dominated by the bitonic sort. In this test, we used single precision Morton codes.

We use the M_4 kernel and set $h_{fact} = 1.2$. To setup the lattice, we first consider a box of size $[-0.6, 0.6]^3$. For a desired number of particle N, the volume per particle is cV/N, where c is compacity of a close-packing of equal spheres. As such, the spacing dr between particles is $dr = (3cV/4\pi N)^{1/3}$. We then adapt the boundaries of the simulation volume to ensure periodicity in all directions for the initial close-packed lattice of particles.

3.2. Performance of tree building

Fig. 5.11 shows the performance of the SHAMROCK tree building algorithm described in Sect. 4.11, by presenting results of tests carried out over 10^3 to 10^8 objects distributed on a regular cubic lattice. The results are presented in figures showing the number of object integrated to the tree per second, as a function of the total number

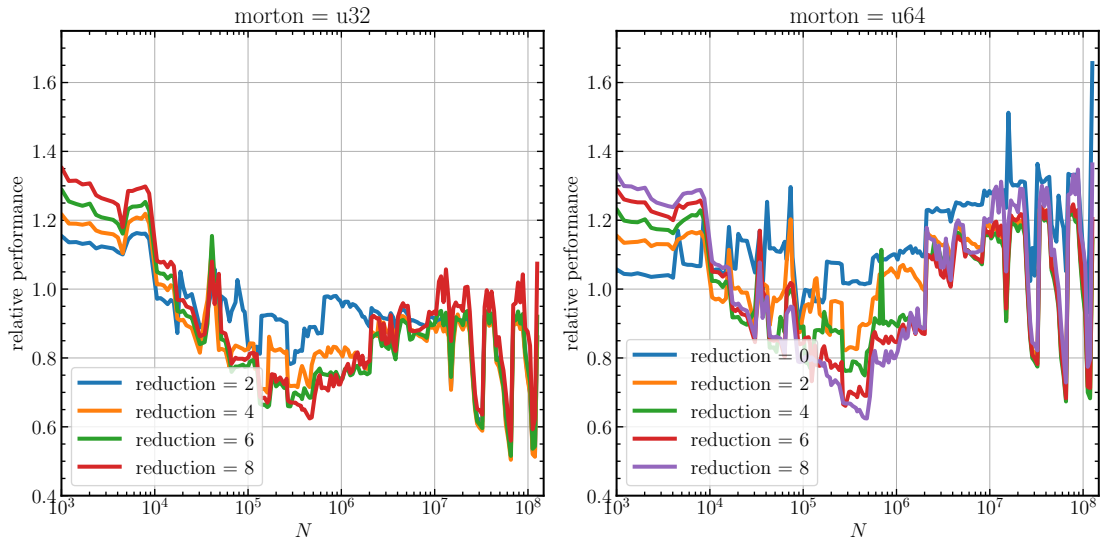


Figure 5.12: Relative performance of the complete tree building procedure for two different types of Morton codes (left: u32, right: u64), for different levels of reduction. The setup for this test is identical to the one presented in Fig. 5.11.

of objects. This metric highlights the efficiency threshold of the GPU, where the computation time is shorter than the actual GPU programming overhead. It also highlights any deviation from a linear computation time as a function of the size of the input.

For a small to moderate number of objects $N \lesssim N_c$ where $N_c \sim 10^6$, the overhead of launching a GPU kernel leaves a significant inprint compared to the computational charge. A few million objects per GPU is the typical number of objects above which the algorithm can be used efficiently. For any $N \geq N_c$ tested, the tree is built at a typical constant rate of 5×10^{-9} s per object. Equivalently, 200 millions of objects per second are processed for Morton codes and the associated positions in double-precision.

For $N \geq N_c$, the algorithm achieves an almost constant performance, as long as it could be tested on current hardware. Fluctuations of up to 30% are observed for specific values of N. These peaks are consistent across several executions, and therefore probably due to the hardware scheduler on the GPU. Tree construction is dominated by the bitonic sorting algorithm (see Fig. 5.11). Since this algorithm does not depend on the values stored in the buffer, its performance is not affected by the spatial distribution of objects, and regular or randomly arranged points deliver the same performance. The performance of tree building of Shamrock is therefore independent of the distribution of objects considered.

Fig. 5.12 shows that performance is almost unaffected by the type (single, double, float or integer) used for the positions ($\sim 5-10\%$, spikes being probably due to the hardware scheduler). Performance is increased by a factor $\sim 30\%$ when the Morton

code representation is reduced to single precision.

3.3. Performance of neighbour cache building

To measure performance of cache build and SPH time stepping, we first setup the particles as discussed in Sect. 3.2. Additionally, the smoothing length has been converged, resulting in 60 neighbours for the M4 kernel. After this setup, we perform a single time step. Fig. 5.13 reports the time spent during this time step to build the cache and perform the iteration. We compare the results obtained for the two strategies presented in Sect. 4.13 –4.14, along with different levels of reduction. We find that enhancing the reduction level results in better overall performance, particularly up to reduction level 6. For higher levels of reduction, performance drops as a consequence of the too large number of particles per leaf. Optimal configuration corresponds to $\sim$ 10 particles per leaf (reduction level of 4), which is similar to the number of particles per leaf in Phantom (Price et al., 2018). Additionally, integrating a two-stage neighbour cache alongside the reduction algorithm can double performance. To sum up, the combined use of reduction and a two-stage cache enhances cache building performance by tenfold, while doubling time-stepping performance.

3.4. Performance of time stepping

3.4.1. One GPU

We evolve setup a Sedov blast using Phantom git pulled at commit number `e01f76c3`, with compile flags `IND_TIMESTEPS=no MAXP=50000000` and evolve it with Phantom for a five timesteps and lower both CFL to 10^{-3} to avoid leapfrog corrector sub-cycling in both codes and produce a restart file. We then start both Shamrock and Phantom on the same restart for 5 iterations, to avoid result being affected by cache warm up. IO has carefully been subtracted from the Phantom measured time. The performance of Shamrock is first tested on a single A100-SXM4-40GB GPU, of total power 275 W. Fig. 5.14 shows the number of particles per second iterated as a function as the total number of particles N_{part} in the simulation. As expected, performance increase as the computational pressure on the GPU increases, up to the point where the solver becomes memory-bound ($\sim 10^6$ particles). Beyond this threshold, a typical speed of 12×10^6 particles per second is achieved. For a comparison, we perform a similar test with Phantom on an AMD Epyc7742 CPU. On this architecture, Phantom fully exploit its OpenMP parallelisation across the 64 cores (128 threads). The power consumption is also similar to the one of the A100-SXM4-40GB used for Shamrock ($\sim$ 275W). For the test described above, one obtains $\gtrsim 2 \times 10^6$ particles per second in most cases. Note that on this Epyc7742 CPU architecture, Shamrock (compiled using AdaptiveCPP OpenMP backend) achieves slightly higher performance. Despite the limitations inherent in such a comparison, we estimate that Shamrock attains approximately

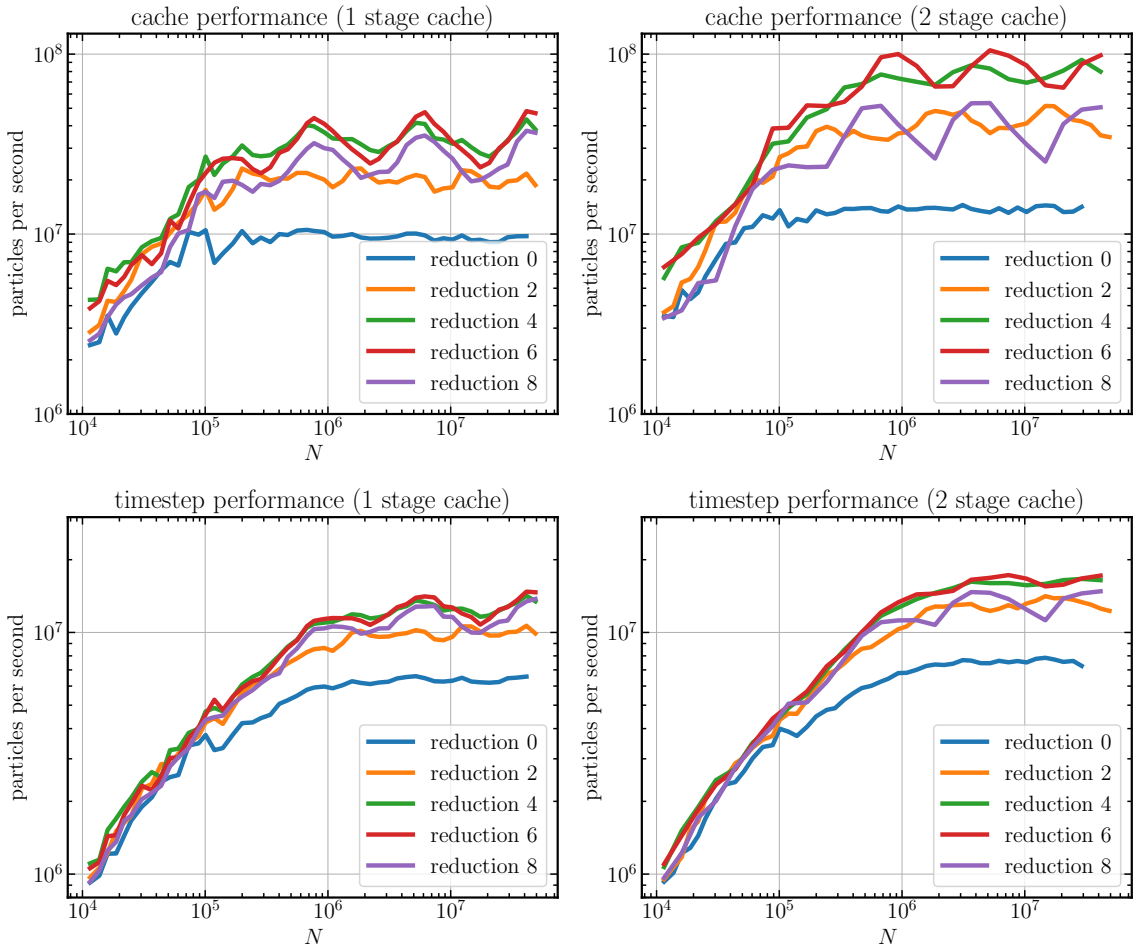


Figure 5.13: Performances of cache building and time stepping measured on one time step of the Sedov-Taylor blast wave test presented in Fig. 5.5. Increasing the level of reduction yields improved performance overall up to reduction level 6. Additionally, using two-stages neighbour caching improves performance up to a factor of two when used in conjunction with the reduction algorithm. In total, employing both reduction and a two-stage cache enhances cache building performance by a factor of ten, while doubling time-stepping performance.

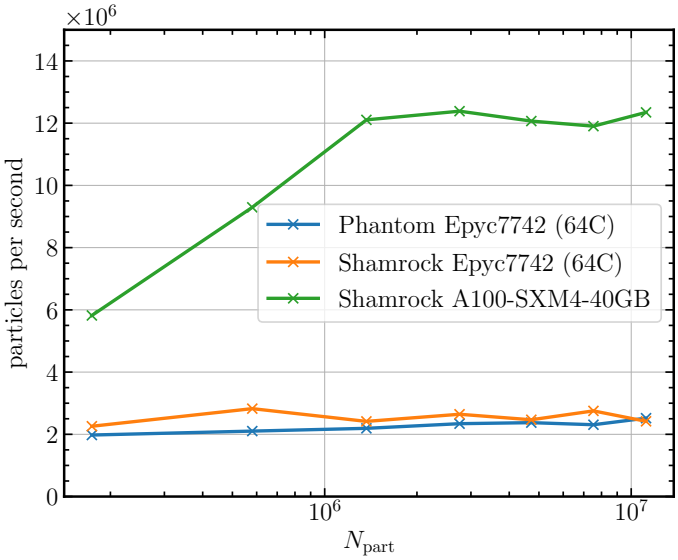


Figure 5.14: Comparative benchmark of Shamrock and Phantom ran on a same restart file of a Sedov-Taylor blast at multiple resolutions produced by Phantom. Shamrock does a achieve a slightly higher performance on CPU compared to Phantom, when run a on a single NVIDIA A100-SXM4-40GB GPU the performance is around 5 times higher for large datasets. Small datasets are not large enough to saturate the GPU explaining the lowered performance on GPU below 10^6 particles.

a ~ 5 factor gain in performance when executed on a single GPU compared to a state-of-the-art SPH CPU code with equivalent power consumption.

3.4.2. Multiple GPUs

We perform the multi-GPU test of Shamrock on the Adastra supercomputer of the French CINES, in its early February 2024 configuration. In this multi-GPU test, the split criterion of patches is set at one-sixth of the number of particles per GPU, guaranteeing a minimum of 8 patches per MPI process. We evolve over 5 time steps of the Sedov test and report the performance obtained on the last time step. Unlike in the case of a single GPU, we do not evolve the simulation prior to measurement to limit computational expenses. The scale at which internal energy is injected remains consistent across all tests.

Fig. 5.15 shows that for 65×10^9 particles distributed over 1000 GPUs (64×10^6 particles per GPU), Shamrock achieves 9×10^9 particles iterated per second. Consequently, iterating one time step over the entire 65×10^9 particles requires 7 seconds on this cluster. These results correspond to around 1.5 times the performance achieved by a single A100 on the same test at the same commit, a value close to what is expected given the hardware specifications. This demonstrates no significant deviation in behaviour attributable to the choice of GPUs. To achieve good load balancing, we find that we need around 10 patches per MPI process. On Adastra, this translates to 20 patches per GPU, amounting to a bit over 1 million particles per Mi250x Graphics Compute Die. Fig. 5.14 shows that for small number of particles, the GPU execution units are not loaded efficiently. A correct load corresponds to a typical 2 millions of particles per GPU. Fig. 5.15 also reveals saw tooth shape as the number of particles increases. This feature can be interpreted by noting that every multiple of 8 GPUs, the patches are divided to avoid becoming too large, causing performance to drop below the efficiency threshold. Efficiency then increases again with the number of particles, until another factor of 8 in the resolution is reached, necessitating further patch refinement. Additionally on Fig. 5.15 we also report the energy efficiency of the weak scaling tests. We measure on every single nodes tests the power consumption related to an iteration of the solver using the hardware counters of the HPE Cray EX235a node. The reported value for multiple nodes is extrapolated from the single node case assuming a total power consumption being the product of the number of nodes times the single node power consumption time the parallel efficiency. Finally we report the power efficiency measured in particles per second per Watt which is also the number of particles processes with a single Joule. The total power consumption of a node in those tests is not very sensible to the number of particles per GPUs. However the GPU performance can be significantly reduced when GPUs do not have enough particle to process, and having a larger number of particle per GPUs result in the highest efficiency. Maximising the number of particles per GPU maximises efficiency in most cases.

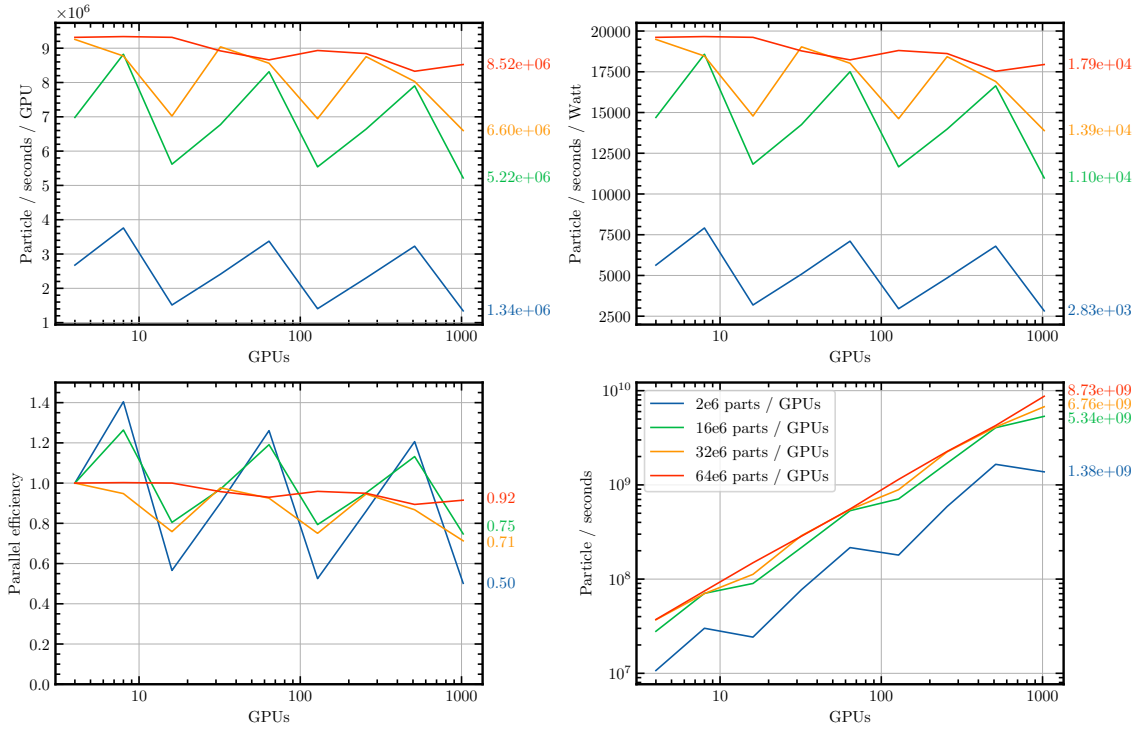


Figure 5.15: Weak scaling tests conducted on the CINES Adastra supercomputer. These tests are performed for multiple resolutions, from 1 node to 256 nodes, corresponding to using 4 GPUs with 8 MPI ranks to 1024 GPUs with 2048 MPI ranks. In these tests, we use the setup of a Sedov-Taylor blast and report the number of particles per GPU. The patch decomposition is set to have at least 8 patches per MPI ranks. We observe that for large simulations, the scaling test results in 92% parallel efficiency on 65 billion particles at 9 billion particle per seconds. Lowering the base resolution reduce the per-GPU performance since GPUs start to be under-utilised. Additionally the variation of the number of particle per patch result in variation in per-GPU performance, resulting in a saw-tooth pattern. We also report the energy efficiency of the tests, were the power consumption used was measured in the single node case and extrapolated as being the product of the number of nodes times the single node consumption.

3.5. Summary

The larger the simulation, the higher the performance per GPU. Multi-GPU architectures therefore require large simulations to scale and be energy-efficient. With Shamrock, this effect can be mitigated by reducing the number of patches, albeit with the potential drawback of rising load balancing issues. Similar benchmarks would be required for further implementations in Shamrock of additional physical processes or setups, possibly involving different particle distributions.

4. Software design

4.1. Development

4.1.1. Codebase organisation

The Shamrock project aims to be fully modular, in the sense that it is made up of several cmake projects which are connected using standardised interfaces. For example, the algorithmic library of Shamrock is a cmake sub-project that depends on the backend library. This allows the shamrock sub-projects to be as independent as possible, avoiding merge conflicts and enabling development efforts to be better focused. To date, the project comprises 12 sub-projects. This number is very likely to change in the future, with future additions and refactoring.

4.1.2. Git

The Shamrock project is hosted on GitHub. We adopt a methodology akin to the one employed by the LLVM project (Lattner & Adve, 2004), where the main branch is protected and can only be modified by pull requests from the feature/fix branches from contributors forks of the project. Releases are performed by branching from the main branch, facilitating the implementation of fixes to existing versions of the code. The CI test pipeline is routinely executed on GitHub, assessing both the main branch and all incoming pull requests. Successful completion of all tests is mandatory for changes to be merged into the main branch.

4.2. Testing

Numerous unit testing and validation options are available for C++. However, none of the standard solutions available match our specific requirements, the main one being that tests are integrated with MPI . Because of this constraint, we have developed our own in-house test library, designed to provide the main features of gtest, while retaining the ability to specify the number of MPI ranks for a particular test. The current test library is capable of performing unittest, validation tests, and benchmarks. On GitHub, we use self-hosted runners to perform the tests with multiple configurations of compilers, targets and versions.

4.3. Environment scripts

Compiling Shamrock on different machines entails dealing with a wide range of diversity. Typical technical aspects involve setting up LLVM, MPI and SYCL, which may involve numerous steps on a machine with missing libraries or having complex configuration. To ensure consistency in Shamrock configuration across machines, we have designed environment scripts. These scripts aim to produce a build directory with all the requirements for building the code, as well as to provide an 'activate' script in this folder, which configures the environment variable and loads the correct modules by sourcing them. In addition, these scripts offer utility functions such as

- setupcompiler: Setup the SYCL compiler
- updatecompiler: Update the environment
- shamconfigure: Configure Shamrock
- shammake: Build Shamrock

This functionality is provided by a 'new-env' script that configures the build directory with all requirements, including the compiler SYCL, automatically. In summary, only 5 commands are needed to build a working version of Shamrock, an example would be

```
# Setup the environment
./env/new-env \
    --builddir build \
    --machine debian-generic.acpp \
    -- \
    --backend cuda \
    --arch sm_70

# Now move in the build directory
cd build
# Activate the workspace, which will
# define some utility functions
source activate
# Configure Shamrock
shamconfigure
# Build Shamrock
shammake
```

4.4. Runscripts

In Shamrock, our aim is to handle setup files and configuration files that would allow great versatility in the use of the code, as well as on-the-fly analysis. Handling such a complexity through configuration files alone is both difficult and non-standard. Moreover, a user should not be required to know C++ to be able to use

```python
import shamrock

# Create a Shamrock context
ctx = shamrock.Context()
ctx.pdata_layout_new()

# Get the SPH model
model = shamrock.get_SPHModel(
    context = ctx,
    vector_type = "f64_3",
    sph_kernel = "M6")

# configure the solver
cfg = model.gen_default_config()
cfg.set_artif_viscosity_VaryingCD10(
    alpha_min = 0.0,
    alpha_max = 1,
    sigma_decay = 0.1,
    alpha_u = 1,
    beta_AV = 2)
cfg.set_boundary_periodic()
cfg.set_eos_adiabatic(gamma = 5./3.)
cfg.print_status()

model.set_solver_config(cfg)
model.set_cfl_cour(0.3)
model.set_cfl_force(0.25)

# Initialise the patch scheduler
model.init_scheduler(
    split_crit = 1e6,
    merge_crit = 1e4)

#  .... Do setup ....

# Run the simulation until t=1 and dump
t_end = 1.0
model.evolve_until(t_end)
dump = model.make_phantom_dump()
dump.save_dump("output")
```

Figure 5.16: Example of a simplified SHAMROCK runscript

the code. Using a PYTHON frontend offers a suitable solution to ensure both code versatility and ease of use. To do this, we use pybind11 (Jakob et al., 2024), which allows to map C++ functions or classes from the C++ source code to a '`shamrock`' python library. In the current version of SHAMROCK, two uses are possible. The first is to use SHAMROCK as a python interpreter that will go through and execute the content of a runscript (the script of a SHAMROCK run), which can include, if desired, configuration, simulation and post-processing in a single run and script (see Fig. 5.16 for an example of a runscript).

The other use is to compile SHAMROCK as a Python library and install it through pip, enabling the code to be used in Jupyter notebooks. Using SHAMROCK as a Python library is ideal for local machine prototyping, while on a cluster, employing SHAMROCK as a Python interpreter is highly recommended.

4.5. Units

In SHAMROCK we have chosen to use code units which are a rescaling of base SI units, where the factor is chosen at runtime in the runscript.

5. Conclusion

We introduced SHAMROCK, a modular and versatile framework designed to run efficiently on multi-GPUs architectures, towards Exascale simulations. The efficiency of SHAMROCK is due to its tree, based on a fully parallel binary logic (Karras algorithm). On a single GPU of an A100, the algorithm builds a tree for 200 million particles in one second. The tree traversal speed, while summing over approximately 60 neighbors, reaches 12 million particles per second per GPU. This property makes it possible to build a Smoothed Particle Hydrodynamics (SPH) solver where neighbors are not stored but recalculated on the fly, reconstructing a tree almost instantaneously.

To exploit the efficiency of this framework, we have implemented and tested an hydrodynamic SPH solver in SHAMROCK. For a Sedov-Taylor blast test performed with 10^6 particles on a single A100 GPU, a SHAMROCK simulation is around $\sim$6 times faster than an identical simulation performed with PHANTOM on an Epyc 7742 multicore CPU architecture of equivalent power. The parallelization of SHAMROCK on several nodes relies on an MPI protocol with hollow communications between the interfaces of a patch system that groups calculations performed on different GPUs. SHAMROCK 's scaling has been tested on the ADASTRA supercomputer (2000 mi250x GPUs). As expected, the higher the computational load on the GPU, the better the efficiency of the code. For 32×10^6 particles per GPU and 65 billions of particles in total, we achieve 92% efficiency at low scaling, in a simulation where 9×10^9 particles are iterated per second. Iterating one time step over the 65×10^9 particles takes therefore 7 seconds on this architecture. SHAMROCK is therefore a promising

framework that will soon be extended to other grid-based algorithms with adaptive refinement.

Appendix

6. AABB extension/intersection permutation

We prove the following theorem:

$$AABB_1 \oplus h \cap AABB_2 \neq \emptyset \Leftrightarrow AABB_1 \cap AABB_2 \oplus h \neq \emptyset,$$

where $AABB \oplus l$ is the operation that extends the AABB in every direction by a distance l. One initial observation is that an AABB is equivalent to a ball defined using the infinity norm $||\cdot||_\infty$. Consequently, the intersection of two AABBs is the result of intersecting along each axis independently. Formally, define a first AABB 1 as the Cartesian product of three intervals $AABB_1 = I_{1,x} \times I_{1,y} \times I_{1,z}$, and a second AABB as $AABB_2 = I_{2,x} \times I_{2,y} \times I_{2,z}$. Their intersection is $AABB_1 \cap AABB_2 = (I_{1,x} \cap I_{2,x}) \times (I_{1,y} \cap I_{2,y}) \times (I_{1,z} \cap I_{2,z})$. Hence, proving the theorem in one dimensions directly extends to three dimensions. Consider now two one-dimensional intervals $I_1 = [\alpha_1, A_1], I_2 = [\beta_1, B_1]$. With $d(a, b)$ the distance in one dimension between a point a and b, and $B(r, h)$ being a ball in one dimension of position r and radius h, we have

$$\begin{aligned}
&\emptyset \neq I_1 \oplus h \cap I_2 \\
\Leftrightarrow\ &\emptyset \neq [\alpha_1 - h, A_1 + h] \cap [\beta_1, B_1] \\
\Leftrightarrow\ &\emptyset \neq B\left(\frac{A_1 + \alpha_1}{2}, \frac{A_1 - \alpha_1}{2} + h\right) \cap B\left(\frac{B_1 + \beta_1}{2}, \frac{B_1 - \beta_1}{2}\right) \\
\Leftrightarrow\ &d\left(\frac{A_1 + \alpha_1}{2}, \frac{B_1 + \beta_1}{2}\right) \leq \frac{A_1 - \alpha_1}{2} + h + \frac{B_1 - \beta_1}{2} \\
\Leftrightarrow\ &\emptyset \neq B\left(\frac{A_1 + \alpha_1}{2}, \frac{A_1 - \alpha_1}{2}\right) \cap B\left(\frac{B_1 + \beta_1}{2}, \frac{B_1 - \beta_1}{2} + h\right) \\
\Leftrightarrow\ &\emptyset \neq [\alpha_1, A_1] \cap [\beta_1 - h, B_1 + h] \\
\Leftrightarrow\ &\emptyset \neq I_1 \cap I_2 \oplus h,
\end{aligned}$$

which completes the proof.

References

Balsara D. S., 1995, *von Neumann stability analysis of smooth particle hydrodynamics–suggestions for optimal algorithms*, Journal of Computational Physics, 121, 357-372

Bedorf J., Portegies Zwart S., 2020, *Bonsai-SPH: A GPU accelerated astrophysical Smoothed Particle Hydrodynamics code*, SciPost Astronomy, 1, 001

Courant R., Friedrichs K., Lewy H., 1928, *Über die partiellen Differenzengleichungen der mathematischen Physik*, Mathematische Annalen, 100, 32-74

Cullen L., Dehnen W., 2010, *Inviscid smoothed particle hydrodynamics*, MNRAS, 408, 669-683

Hairer E., Lubich C., Wanner G., 2003, *Geometric numerical integration illustrated by the Störmer-Verlet method*, Acta Numerica, 12, 399-450

Hopkins P. F., 2014, *GIZMO: Multi-method magneto-hydrodynamics+gravity code*, Oct., ADS link, Astrophysics Source Code Library, record ascl:1410.003

Hubber D. A., Batty C. P., McLeod A., Whitworth A. P., 2011, *SEREN - a new SPH code for star and planet formation simulations. Algorithms and tests*, A&A, 529, A27

Jakob W., Rhinelander J., Moldovan D., 2024, *pybind11–Seamless operability between C++ 11 and Python*, URL: https://github. com/pybind/pybind11

Lattanzio J., Monaghan J., Pongracic H., Schwarz M., 1986, *Controlling penetration*, SIAM Journal on Scientific and Statistical Computing, 591–598

Lattner C., Adve V., in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, booktitle, pp 75–86, doi:10.1109/CGO.2004.1281665

Monaghan J. J., 1997, *SPH and Riemann Solvers*, Journal of Computational Physics, 136, 298-307

Price D. J., 2008, *Modelling discontinuities and Kelvin Helmholtz instabilities in SPH*, Journal of Computational Physics, 227, 10040-10057

Price D. J., et al., 2018, *Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics*, PASA, 35, e031

Quemener E., Corvellec M., 2013, *SIDUS—the solution for extreme deduplication of an operating system*, Linux Journal, 3

Schaal K., Bauer A., Chandrashekar P., Pakmor R., Klingenberg C., Springel V., 2015, *Astrophysical hydrodynamics with a high-order discontinuous Galerkin scheme and adaptive mesh refinement*, MNRAS, 453, 4278-4300

Schaller M., Gonnet P., Draper P. W., Chalk A. B. G., Bower R. G., Willis J., Hausammann L., 2018, *SWIFT: SPH With Inter-dependent Fine-grained Tasking*, May, ADS link, Astrophysics Source Code Library, record ascl:1805.020

Schoenberg I. J., 1946, *Contributions to the problem of approximation of equidistant data by analytic functions. Part A. On the problem of smoothing or graduation. A first class of analytic approximation formulae*, Quarterly of Applied Mathematics, 45-99

Sedov L. I., 1959, *Similarity and Dimensional Methods in Mechanics*

Sod G. A., 1978, *Review. A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws*, Journal of Computational Physics, 27, 1-31

Springel V., Pakmor R., Zier O., Reinecke M., 2021, *Simulating cosmic structure formation with the GADGET-4 code*, MNRAS, 506, 2871-2949

Taylor G., 1950a, *The Formation of a Blast Wave by a Very Intense Explosion. I. Theoretical Discussion*, Proceedings of the Royal Society of London Series A, 201, 159-174

Taylor G., 1950b, *The Formation of a Blast Wave by a Very Intense Explosion. II. The Atomic Explosion of 1945*, Proceedings of the Royal Society of London Series A, 201, 175-186

Tricco T. S., 2019, *The Kelvin-Helmholtz instability and smoothed particle hydrodynamics*, MNRAS, 488, 5210-5224

Verlet L., 1967, *Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules*, Physical Review, 159, 98-103

Wadsley J. W., Stadel J., Quinn T., 2004, *Gasoline: a flexible, parallel implementation of TreeSPH*, New A, 9, 137-158

Wendland H., 1995, *Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree*, Advances in computational Mathematics, 389–396

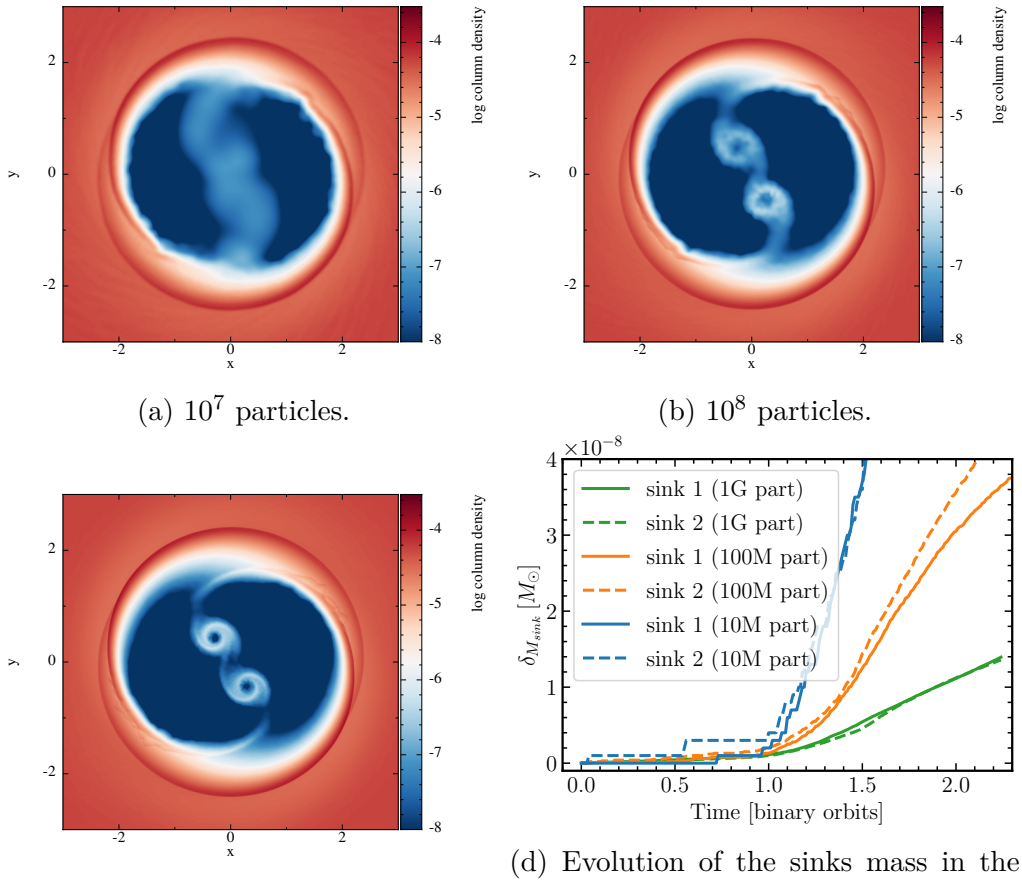
Chapter 6

Conclusion

Contents

1. A first astrophysical application

In order to assess Shamrock's potential to tackle systems of interest in the context of protoplanetary discs we test the code on the numerical setup closed to the one proposed by Duffell et al. (2024) as it is numerically challenging in three dimensions due its large resolution requirement. The numerical setup consist of two stars of equal mass $M_{1,2} = 0.5 M_\odot$ orbiting with a constant binary separation of $d = 1$au (i.e. with a null eccentricity). The units are chosen in order to have $GM_\odot = 1$ such that a binary orbit time correspond to unity in time code units. Around this binary star system a disc is set up with a surface density profile $\Sigma(r) = \Sigma_0 \left[(1-\delta_0)\, e^{-(R_{\rm cav}/r)^{12}} + \delta_0\right]$, where the inner cavity radius is set to be $R_{\rm cav} = 2.5$au, and the background density to be $\delta_0 = 10^{-5}$. The simulation is three-dimensional, we adopt a Gaussian vertical density profile $\rho(r,z) = \frac{\Sigma(r)}{H}\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2H^2}\right)$ where the scale height of the discs $H(r)/r$ is set to be constant equal to 0.1. Following Duffell et al. (2024) we use a locally isothermal equation of state with the sound speed $c_{\rm s}(\mathbf{r}) = (\frac{H}{r})\sqrt{-\Phi_1(\mathbf{r}) - \Phi_2(\mathbf{r})}$, where $\Phi_{1,2}(\mathbf{r})$ are the gravitational potential of the two stars. For the velocity profile, we use a Keplerian velocity profile with sub-Keplerian correction due to the radial pressure gradient. The gas is modelled using SPH with the constant α-viscosity disc model to have an equivalent $\alpha_{\rm SS} = 10^{-3}$. The disc density is set up using a Monte Carlo method such as described in Price et al. (2018), then the velocity is set based on the particles positions. Lastly, we use sink particles to model the binary stars with an accretion radius of $r_{\rm acc} = 0.05$au, where particles get accreted whenever their distance to a sink is below $r_{\rm acc}$. Upon accretion the mass, linear and angular momentum of the accreted particle is added to the sink particle. The simulations were carried up to 5 binary orbits except for the 10^9 particle simulations that is stopped after 2.4 orbits as we used remaining hours in the time allocation to perform it on the Adastra supercomputer. The 10^9 particle simulation was performed on Adastra supercomputer for 2 days (at a rate of about one orbit per day), on 8 GPU nodes, accounting for about 1500 GPU hours.



(a) 10^7 particles.

(b) 10^8 particles.

(c) 10^9 particles.

(d) Evolution of the sinks mass in the simulations.

Figure 6.1: SPH simulation of a circumbinary discs according to the setup proposed by Duffell et al. (2024) performed with SHAMROCK SPH solver. Panels (a), (b) and (c) shows face-on view of the column integrated density after 1.88 orbits of the discs at different resolutions, respectively 10^7, 10^8 and 10^9 particles. The bottom right panel (d) shows the evolution of the mass accreted by the sink particles δM_{sink} in the simulations shown in panels (a), (b) and (c) as a function of the time.

On Fig. 6.1 we observe that for the simulation at 10^7 particles the inner cavity is under-resolved. This result in direct accretion of the gas falling into the cavity to be accreted immediately onto the sink, this yield a large accretion rate. When the resolution is increased to 10^8 particles, the gas entering on the cavity falls on circumstellar discs (sometimes also called mini-discs) before being accreted by the star. This result in a buffering of the accretion, which lowers the resulting accretion rate, as detailed in Farris et al. (2014). Indeed, at this resolution we observe a reduced accretion rate compared to 10^7 particles, however, the circumstellar discs are under-resolved which lead to over-diffusivity, and enhanced accretion compared to a resolved case. For the largest resolution simulation we performed, at 10^9 particles the circumstellar discs having a significant resolution further reduce the accretion

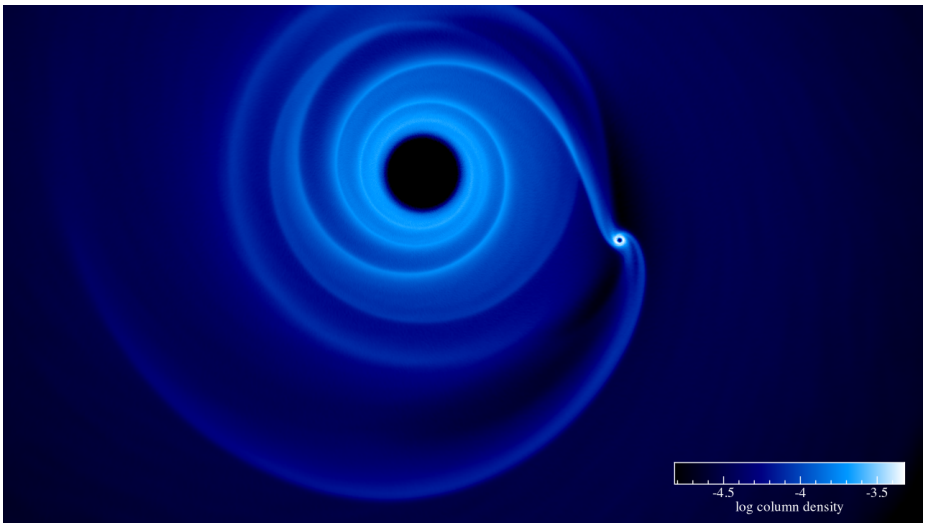
rate compared to the 10^8 particle case as the circumstellar discs are more resolved. Here the circumstellar discs are buffering the accretion as expected.

We capture a drastic reduction of the accretion rate due to the resolution of the circumstellar discs, the accretion buffering behaviour of circumstellar discs in this three-dimensional simulation, we can not conclude yet on the question of convergence. Indeed, further increasing the resolution may further reduce the accretion rate. Qualitatively, one do not expect the accretion rate to vary significantly as the resolution increase once the circumbinary discs are resolved. This should further be confirmed with simulations at higher resolution. One can not discard the hypothesis of a novel physical accretion regimes triggered by physics at very small scales. Also, independently of the convergence of such simulation, this question must be tackled independently of the question of consistence as the absence of realistic thermal structure of the circumstellar and circumbinary discs here may be a reasonable approximation at lower resolution but should be accounted at larger resolutions, for example when the cooling timescales can not be neglected any more, typically in large resolution simulations of circumplanetary discs. Nevertheless, those simulations pave the way toward the study of resolved accretion rates in double or triple star systems which is a currently active subject in the community (see Ceppi et al. 2022).

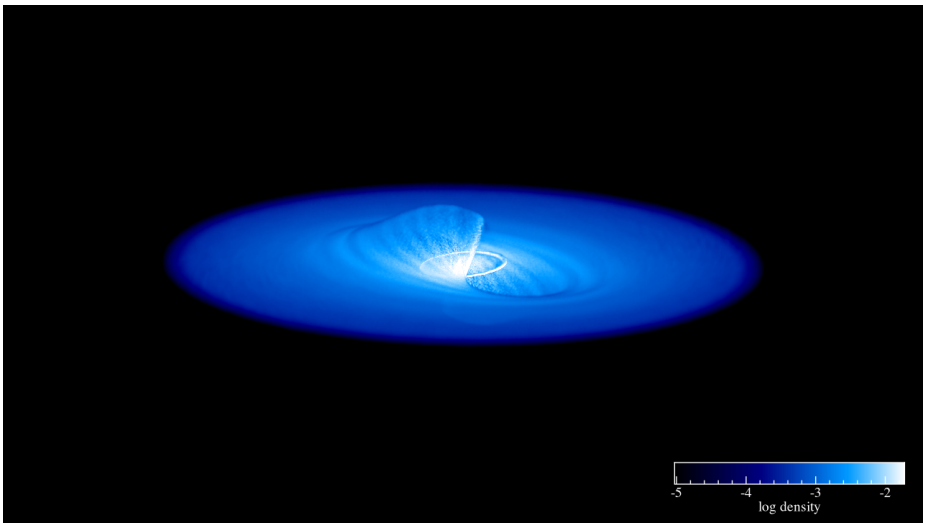
2. Perspectives

2.1. Multi-physics

Multi-scale astrophysical problems are often multi-physical. To be consistent, very high-resolution simulations must include realistic physics. The current version of Shamrock focuses on a purely hydrodynamic SPH solver. Some examples of astrophysical simulation already possible with the Shamrock SPH solver are shown in Fig. 6.2. Next steps of development consist of implementing local algorithms to address the radiative dynamics of magnetized and dusty fluids. In principle, the modular format of Shamrock facilitates the assembly of a new set of known numerical equations into a solver. The biggest challenge is the implementation of gravity, a non-local interaction that requires a new algorithmic layer based on group-group interactions. Two main algorithms are used to handle gravity by the various astrophysical codes. SPH codes mainly use the method of Fast Multipole Moments (FMM), which takes advantage of the tree structure inherent in particle methods. The technical hurdle lies in achieving numerical efficiency, even for high opening angles involving summation over hundreds or even thousands of neighbors. An alternative approach is to employ a multigrid method, but to our knowledge this has not yet been implemented in an SPH code. These additional physical elements require the implementation of an individual time step to maximize performance. It will consequently be also possible to take advantage of Shamrock's efficiency to benchmark individual time stepping against fixed time stepping in simulations that



(a) SPH simulation of a protoplanetary disc with a planet embedded inside performed with SHAMROCK (10 millions of particles on the DGX machine).



(b) SPH simulation of Lense-Thirring precession in a disc around a Kerr black hole performed with SHAMROCK.

Figure 6.2: Example of astrophysical simulations performed with SHAMROCK SPH solver.

were previously not tractable.

2.2. Multi-methods

The SHAMROCK framework relies on the efficient construction and traversal of its tree, irrespective of the numerical object considered. In this work, we have considered particles to develop an SPH solver, but these can be replaced by Eulerian cells in an agnostic way. Since the tree algorithm of SHAMROCK scales almost perfectly

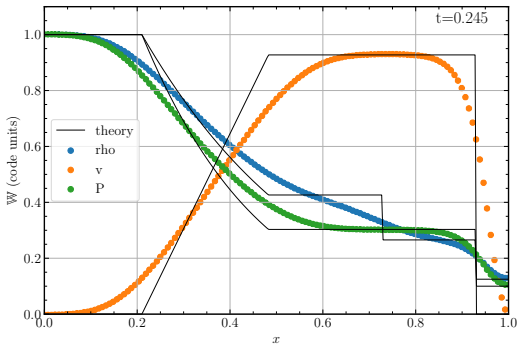
to any disordered particle distribution, we can expect similar performance, even on an AMR (Adaptive Mesh Refinement) grid. In principle, various algorithms can be implemented on this grid (such as finite differences or finite volumes), with the moderate cost of incorporating a few specific modules tailored to these solvers (like the accumulation of flows on faces for finite volumes).

The advantages of such a unified framework are twofold. Firstly, to validate an astrophysical model by achieving consistent results with inherently different methods, while the physical model used is rigorously identical (e.g. opacities, cooling rates, resistivities, chemical networks, equations of state). Secondly, to enable rigorous evaluation of numerical methods in terms of accuracy and computational efficiency for specific numerical problems. To our knowledge, no such framework currently exists.

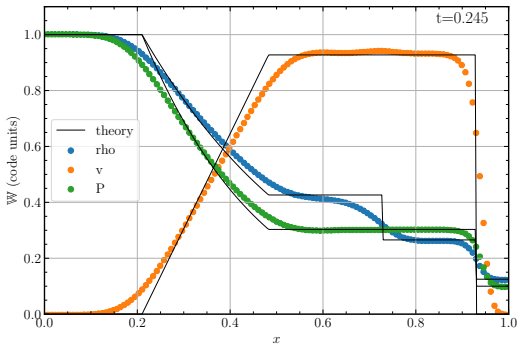
So far, aside from the SPH solver, two prototypes of implementation of the Zeus scheme and Godunov are actively in development in Shamrock. The Zeus scheme is implemented as described in Chapter 2 and is yet to be fully tested. The Godunov scheme is in a similar state. The only missing piece from the scheme described in Chapter 2 is that the scheme does not feature time midpoint interpolation as of now, making it first-order in time but second-order in space. We present a few examples of the Sod-tube test performed with both the Zeus and the Godunov solvers with multiple configurations in Fig. 6.3. The implementation of both schemes is performed by representing grid cells on an integer grid, replacing the interaction criterion of SPH with a criterion where cells are neighbors if they share a common face. Aside from those modifications and the scheme itself, the Zeus and the Godunov schemes in Shamrock involve the same modules as the SPH solver (e.g. domain decomposition, load balancing) and most of the algorithms (e.g. exchanging ghost zones, building trees, finding neighbors, applying the scheme). The implementation of the Godunov scheme, to date, is more performant than the one of the Zeus scheme. Experience on the implementation of the Zeus scheme helped us to optimize the subsequent implementation of the Godunov solver. Current performances are limited by memory allocation pressure and kernel submission latency. We recently tested our Godunov implementation on the Adastra supercomputer, as shown in Fig. 6.4. The solver currently achieves 248 million cells per second on a single Adastra node. We expect both the Godunov and Zeus solvers to have massively improved performance in the future with the addition of memory pooling in Shamrock to reduce memory allocation pressure.

2.3. Data analysis

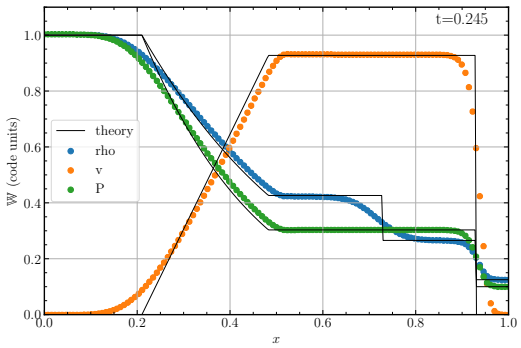
The efficiency of the Shamrock tree means that data analyses can be carried out very efficiently, whether on particles, cells or both. We plan to develop a native library that will enable these analyses to be performed efficiently on multi-GPU parallel architectures.



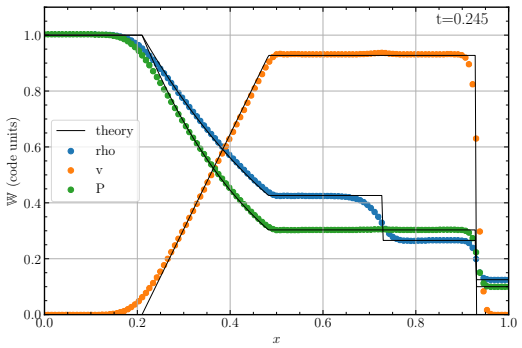
(a) Godunov, no slope limiter, Rusanov Riemann solver.



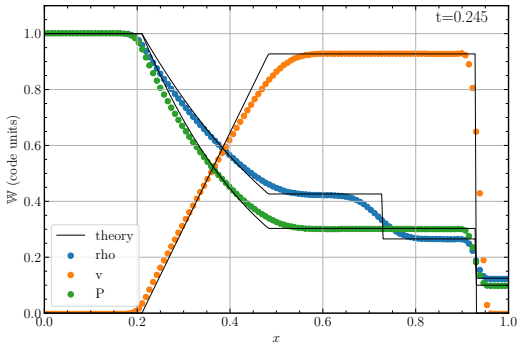
(b) Godunov, minmod slope limiter, Rusanov Riemann solver.



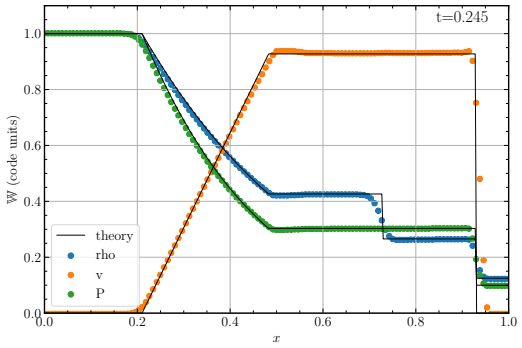
(c) Godunov, no slope limiter, HLL Riemann solver.



(d) Godunov, minmod slope limiter, HLL Riemann solver.



(e) Zeus, no slope limiter.



(f) Zeus, van leer slope limiter.

Figure 6.3: Results of the Sod tube test in SHAMROCK with different configurations of the grid solvers. We present the results using Godunov scheme in the plot a-d and using Zeus scheme in e-f. For the Godunov scheme we have performed the test using either the minmod slope limiter or no slope reconstruction, for the Riemann solvers we used either the Rusanov or the HLL solvers (see Toro 2013).

2.4. Optimization of latencies

The SHAMROCK framework has been designed to optimize performance on multi-node architectures. As discussed in the previous section, specificities of modern

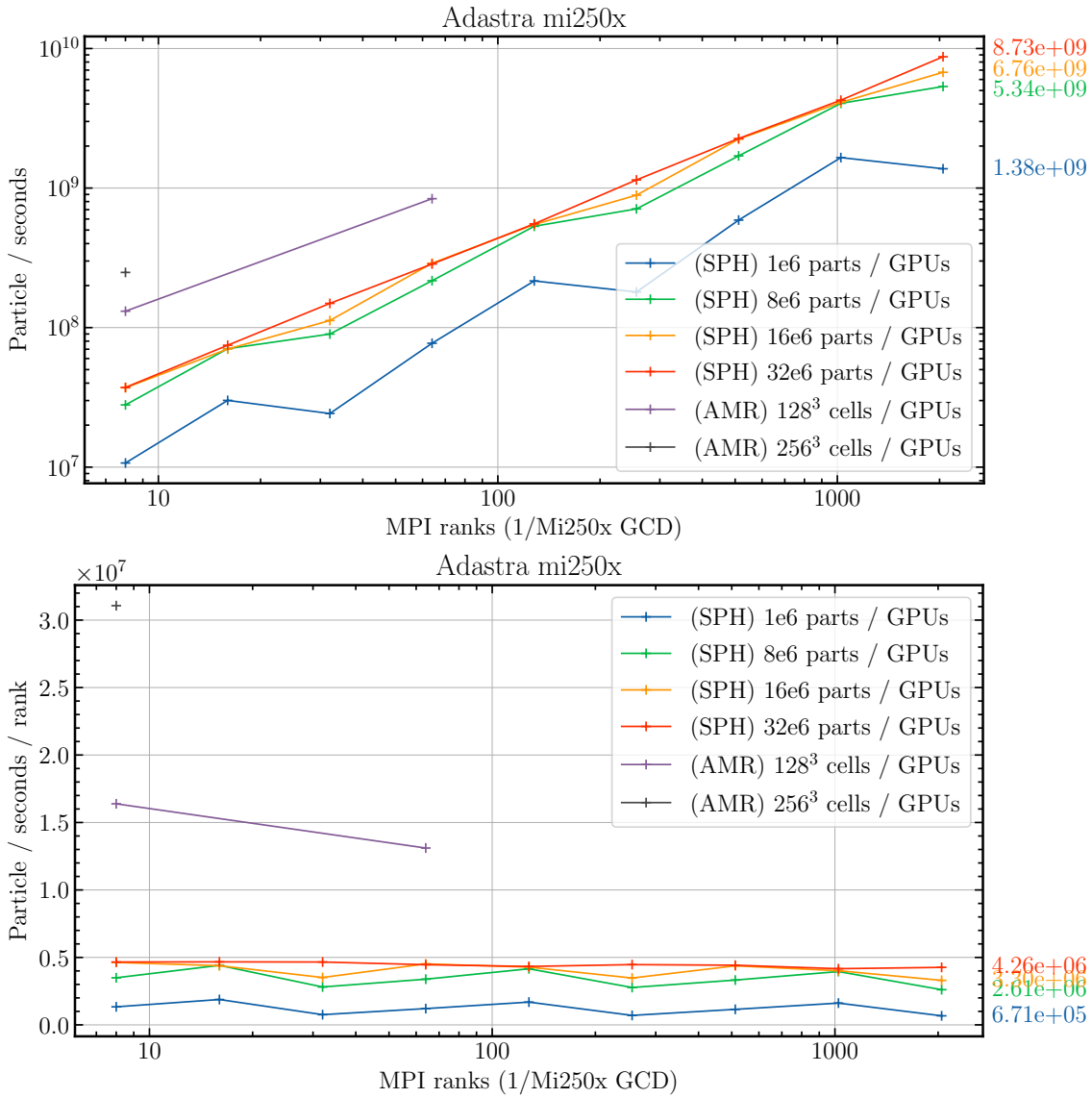


Figure 6.4: Performance of the Godunov scheme prototype in SHAMROCK compared to the SPH scalling curves. The Godunov solver achieves around 248 million cells per second on a single Adastra node. This performance is not definitive, as significant optimizations can be performed by reducing allocations and optimizing latencies. On those scalling curves, the SPH is tested with a Sedov-Taylor blast and the Godunov on a Sod-tube test on a three-dimensional cube.

hardware imply that performance increases with the computational load demanded of the GPUs. Maybe counter-intuitively, Shamrock is therefore not designed by default to run small simulations with a large number of iterations. Since efficient execution of such simulations remains a complementary challenge to that of Exascale, a layer of optimizations regarding remnant latencies remains to be implemented. This would enable users to employ Shamrock for both small and large-scale simulations. We will benefit from the knowledge of the work done by the Gromacs team towards optimizing latencies in SYCL (Alekseenko et al., 2024).

3. Conclusion

This Ph.D. thesis was initially motivated by the study of planet formation in the early phase of the protostellar collapse. However, during an initial study of dusty protostellar collapses, SPH was found to be limiting due to the absence of monofluid formalism for the large grains, as well as its global performance. This led to the study of extensions of dusty SPH schemes (monofluid and two-fluid) and the examination of the viability of SPH on GPUs. Trying to port SPH on a GPU quickly suggested the need for a tree algorithm. In our case, we realized that the Bounding Volume Hierarchies (BVH) used in ray-tracing was a promising option. The feasibility of such an approach was confirmed by the Karras (2012) algorithm, which can be adapted to SPH to build the tree in negligible time compared to an SPH timestep. This resulted in a single GPU SPH code, named at the time Sphive. Internally at CRAL, this tree algorithm suggested the feasibility of an AMR grid where the AMR Octree's cells would be handled by such a tree algorithm. After initial prototyping of the AMR grid, we finally decided to implement both the AMR and the SPH in a single code to avoid duplication. This led to the construction of Shamrock, where the AMR and SPH components are abstracted away in a common framework. At this point, the code was capable of running only on a single GPU. The generalization to multiple GPUs came from the possibility of exploiting the speed of the tree to recompute it, instead of having to communicate it. At the same time, other multi-GPU codes began to appear within the community. From our initial draft of a single GPU SPH code, the code swiftly evolved into a multi-method multi-GPU one developed during the majority of the Ph.D. thesis and was finally tested on an exascale architecture near the end of the Ph.D., leading to better than expected performance.

Due to the abstractions required to construct the Shamrock framework, the code now abstracts various numerical schemes, algorithmics, communications, and physics into independent modules connected through standardized interfaces. These abstractions make it possible to work independently on different components of the code. In the near future, we plan to enhance SHAMROCK by collaborating with mathematicians on the schemes, computer scientists on the algorithms, as well as physicists, geophysicists, hydrodynamicists, and engineers of new physical processes to solve interesting problems.

References

Alekseenko A., Páll S., Lindahl E., 2024, *GROMACS on AMD GPU-Based HPC Platforms: Using SYCL for Performance and Portability*, arXiv preprint arXiv:2405.01420

Ceppi S., Cuello N., Lodato G., Clarke C., Toci C., Price D. J., 2022, *Accretion rates in hierarchical triple systems with discs*, MNRAS, 514, 906-919

Duffell P. C., et al., 2024, *The Santa Barbara Binary-disk Code Comparison*, ApJ, 970, 156

Farris B. D., Duffell P., MacFadyen A. I., Haiman Z., 2014, *Binary Black Hole Accretion from a Circumbinary Disk: Gas Dynamics inside the Central Cavity*, ApJ, 783, 134

Karras T., in *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, booktitle, pp 33–37

Price D. J., et al., 2018, *Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics*, PASA, 35, e031

Toro E. F., 2013, *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*, Springer Science & Business Media

Appendix A

Polydisperse Magnetised SI

Contents

Foreword

The following study was conducted in collaboration with F. Lovascio during his postdoctoral position at CRAL over a period of one month in the summer of 2023. This was motivated by the discovery that the polydisperse streaming instability could be extended to MHD simply with minimal change to the community code PSITOOLS. These two instabilities play a key role in planet formation: can SI quench or favor MRI and vice-versa? Can an instability develop under the combined action of magnetic fields and dust? We conducted the analytical work necessary for this study and utilized PSITOOLS to produce the results presented hereafter. However, we did not complete the physical interpretation of the results yet, and further validation of the method would be required to ensure the correctness of the results.

1. Context

The linear SI is driven by the gas pressure gradient in discs. The pressure gradient drives an azimuthal velocity drift between dust and gas, which subsequently induces dust radial drift due to gas drag acting on the dust leading to a loss of angular momentum and gas outwards radial drift due to dust back-reaction. The SI provides a powerful mechanism for planet formation, and appears to well predict several observables, like Kuiper-Belt object rotation distributions. The linear instability itself, however, has been shown to not be as robust as previously thought. It has been shown by Krapp et al. (2019); Paardekooper et al. (2020) that the SI considerably changes character for multiple dust sizes and dust with a grain size distribution (polydisperse).

When going polydisperse, the usual dust density becomes an integral over the dust distribution of mass:

$$\rho_d = \int_{a_0}^{a_{\max}} \sigma_a \mathrm{d}a, \tag{A.1}$$

where ρ_d denotes the total dust density and σ_a the dust density distribution on the dust size space. Similarly, the integrated dust fraction can be defined as such:

$$\mu = \int_{a_0}^{a_{\max}} \epsilon_a \mathrm{d}a, \tag{A.2}$$

where $\mu = \rho_\mathrm{d}/(\rho_\mathrm{d} + \rho_\mathrm{g})$ is the fraction of dust mass in the fluid integrated in dust sizes, and ϵ_a denotes the dust fraction distribution in sizes. From Paardekooper et al. (2020, 2021), we have the following equations governing the polydisperse dust-gas evolution:

$$\partial_t \rho_\mathrm{g} + \nabla \cdot (\rho_\mathrm{g} \mathbf{v}_\mathrm{g}) = 0\,, \tag{A.3}$$
$$\partial_t \sigma_a + \nabla \cdot (\sigma_a \mathbf{v}_{\mathrm{d},a}) = 0, \tag{A.4}$$
$$\partial_t \mathbf{v}_\mathrm{g} + (\mathbf{v}_\mathrm{g} \cdot \nabla)\mathbf{v}_\mathrm{g} = -\frac{\nabla p}{\rho_\mathrm{g}} + \mathbf{f}_\mathrm{g} - \frac{1}{\rho_\mathrm{g}} \int \sigma_a \mathbf{f}_{\mathrm{drag,d,a}} \mathrm{d}a\,, \tag{A.5}$$
$$\partial_t \mathbf{v}_{\mathrm{d},a} + (\mathbf{v}_{\mathrm{d},a} \cdot \nabla)\mathbf{v}_{\mathrm{d},a} = \mathbf{f}_\mathrm{d} + \mathbf{f}_{\mathrm{drag,d,a}}, \tag{A.6}$$

where $\rho_\mathrm{g}, \mathbf{v}_\mathrm{g}$ are the gas density and velocity, and with drag

$$\mathbf{f}_{\mathrm{drag,d,a}} = -\frac{\mathbf{v}_{\mathrm{d},a} - \mathbf{v}_\mathrm{g}}{\tau_\mathrm{s}(a)}, \tag{A.7}$$

and stopping time

$$\tau_\mathrm{s} = \sqrt{\frac{\pi}{8}} \frac{a \rho_\mathrm{b}}{\rho_\mathrm{g} c}. \tag{A.8}$$

2. Non-ideal MHD with polydisperse dust in shearing box

2.1. Basic equations

We consider the extension of the polydisperse dust-gas equations of motion in an ionized gas medium with neutral grains, resulting in the coupling of the non-ideal MHD equations for the gas and the magnetic field and polydisperse dust coupled to

the gas via the drag force.

$$\partial_t \rho_{\rm g} + \nabla \cdot (\rho_{\rm g} \mathbf{v}_{\rm g}) = 0\,, \tag{A.9}$$

$$\partial_t \sigma_a + \nabla \cdot (\sigma_a \mathbf{v}_{{\rm d},a}) = 0, \tag{A.10}$$

$$\partial_t \mathbf{v}_{\rm g} + (\mathbf{v}_{\rm g} \cdot \nabla)\mathbf{v}_{\rm g} = -\frac{\nabla p}{\rho_{\rm g}} + \mathbf{f}_{\rm g} + \frac{1}{\mu_0 \rho_{\rm g}}(\boldsymbol{\nabla} \times \mathbf{B}) \times \mathbf{B} - \frac{1}{\rho_{\rm g}} \int \sigma_a \mathbf{f}_{\rm drag,d,a} {\rm d}a\,, \tag{A.11}$$

$$\partial_t \mathbf{v}_{{\rm d},a} + (\mathbf{v}_{{\rm d},a} \cdot \nabla)\mathbf{v}_{{\rm d},a} = \mathbf{f}_{\rm d} + \mathbf{f}_{\rm drag,d,a}. \tag{A.12}$$

The induction equation in non-ideal MHD, including ohmic diffusion, Hall effect, and ambipolar diffusion, is (Mouschovias et al., 2009; Tsukamoto et al., 2023):

$$\frac{\partial \mathbf{B}}{\partial t} = \boldsymbol{\nabla} \times (\mathbf{v} \times \mathbf{B}) - \boldsymbol{\nabla} \times \{\eta_{\rm Ohm} \boldsymbol{\nabla} \times \mathbf{B} + \eta_{\rm Hall} (\boldsymbol{\nabla} \times \mathbf{B}) \times \frac{\mathbf{B}}{B} + \eta_{\rm AD} \frac{\mathbf{B}}{B} \times \left[(\boldsymbol{\nabla} \times \mathbf{B}) \times \frac{\mathbf{B}}{B} \right] \} \tag{A.13}$$

We also have the soleinoidal condition:

$$\boldsymbol{\nabla} \cdot \mathbf{B} = 0 \tag{A.14}$$

2.2. Background magnetic field

In order to perform an analysis of the linear instability, we must first choose a background solution that will be perturbed. To achieve this, we adapt the procedure from Lin & Hsu (2022) to our case. For the steady state background, we assume an axysymmetric ($\partial_\phi = 0$) and an unstratified case ($\partial_z = 0$). The solenoidal condition (eq.A.14) gives here $\frac{1}{r}\frac{\partial}{\partial r}(rB_r) = 0$, hence

$$B_r = \left(\frac{R_0}{R}\right) B_{r,0} \tag{A.15}$$

All components of the magnetic field $\mathbf{B}$ must exclusively be functions of r due to the soleinoidal condition. Assuming that $\mathbf{v}_{\rm g} = r\Omega(r)\hat{\boldsymbol{\phi}}$, we can use the induction equation to compute a corresponding valid steady-state magnetic field. If we only account for ohmic diffusion (e.g. Lin & Hsu (2022)), a valid solution for the $\mathbf{B}$ field is a radial and azimuthal field varying in radius. But with the hall effect and ambipolar diffusion, such a state is too complex to be solved analytically, and the background solution from Lin & Hsu (2022) is incompatible with the induction equation. A valid background solution is to take $\mathbf{B} = B_z(r)\hat{\mathbf{z}}$. For simplicity in this study, we chose to stick with a constant vertical magnetic field $\mathbf{B} = B_{z,0}\hat{\mathbf{z}}$.

For a magnetic field, only function of the radial coordinate, the Lorentz force can be written to be

$$F_r = -(B_{\phi,0}^2/r) - B_{\phi,0}B_{\phi,0}^{'} - B_{z,0}B_{z,0}^{'} \tag{A.16}$$

$$F_\phi = (B_{r,0}R_0B_{\phi,0})/r^2 + (B_{r,0}R_0B_{\phi,0}^{'})/r \tag{A.17}$$

$$F_z = (B_{r,0}R_0B_{z,0}^{'})/r \tag{A.18}$$

which result in a null Lorentz force in the steady state background, allowing us to use the background solutions from Paardekooper et al. (2020, 2021). Adding back a variation of the vertical magnetic field as a function of radius would result in a radial Lorentz force, which can be treated as in Lin & Hsu (2022) in the form of an additional drift term.

2.3. Polydisperse non-ideal MHD in shearing box

As per Goldreich & Lynden-Bell (1965); Lin (2021); Lin & Hsu (2022), we use a local expansion around $(r_0, \phi_0, 0)$ in cylindrical The radial, azimuthal, and vertical are renamed x, y, z. On top of that, we assume axisymetry $\partial_y = 0$. The box is corotating at velocity $v_k = -\frac{3}{2}x\Omega_0\hat{\mathbf{y}}$. The velocities are modified using the following mapping:

$$\mathbf{v} \mapsto \mathbf{v} + (r_0 - \frac{3x}{2})\Omega_0\hat{\mathbf{y}} \tag{A.19}$$

using the transformation on velocities self-advection becomes:

$$\partial_t\mathbf{v} + \mathbf{v}\cdot\boldsymbol{\nabla}\mathbf{v} \;\mapsto\; \partial_t\mathbf{v} + \mathbf{v}\cdot\boldsymbol{\nabla}\mathbf{v} - 2\Omega_0 v_y\hat{\mathbf{x}} + \frac{\Omega_0}{2}v_x\hat{\mathbf{y}} \tag{A.20}$$

$$= \partial_t\mathbf{v} + \mathbf{v}\cdot\boldsymbol{\nabla}\mathbf{v} + 2\boldsymbol{\Omega}\times\mathbf{v} \tag{A.21}$$

The equations of conservation of mass are unchanged when moving to the shearing box. The induction equation gets an additional shear term

$$\frac{\partial\mathbf{B}}{\partial t} = \nabla\times(\mathbf{v}\times\mathbf{B}) - \nabla\times\{\eta_{\text{Ohm}}\nabla\times\mathbf{B} \tag{A.22}$$

$$+\eta_{\text{Hall}}\,(\nabla\times\mathbf{B})\times\frac{\mathbf{B}}{B} + \eta_{\text{AD}}\frac{\mathbf{B}}{B}\times\left[(\nabla\times\mathbf{B})\times\frac{\mathbf{B}}{B}\right]\Big\} - \hat{\mathbf{y}}\frac{3}{2}\Omega_0 B_x, \tag{A.23}$$

In the shearing box approximation, treat the background pressure gradient as perturbative external force that sustains radial drift as per Lin (2021); Lin & Hsu (2022) on the gas in the form of a force

$$\mathbf{f}_{\text{g}} = 2\eta\hat{\mathbf{x}} - \nabla\Phi, \tag{A.24}$$

where

$$\eta = \frac{1}{2\rho_{\text{g}}}\frac{\partial P}{\partial r} \sim \frac{c^2}{r_0}, \tag{A.25}$$

This results in the following set of equations for the non-ideal MHD with polydisperse dust in the shearing box approximation:

$$\partial_t \rho_\mathrm{g} + \nabla \cdot (\rho_\mathrm{g} \mathbf{v}_\mathrm{g}) = 0 \tag{A.26}$$

$$\partial_t \mathbf{v}_\mathrm{g} + (\mathbf{v}_\mathrm{g} \cdot \nabla)\mathbf{v}_\mathrm{g} = 2\eta\hat{\mathbf{x}} - \frac{\nabla p}{\rho_\mathrm{g}} - 2\boldsymbol{\Omega} \times \mathbf{v}_\mathrm{g} - \nabla\Phi + \frac{1}{\mu_0 \rho_\mathrm{g}}(\boldsymbol{\nabla} \times \mathbf{B}) \times \mathbf{B} + \frac{1}{\rho_\mathrm{g}} \int \sigma \frac{\mathbf{v}_{\mathrm{d},a} - \mathbf{v}_\mathrm{g}}{\tau_\mathrm{s}(a)} \mathrm{d}a. \tag{A.27}$$

$$\partial_t \sigma_a + \nabla \cdot (\sigma_a \mathbf{v}_{\mathrm{d},a}) = 0, \tag{A.28}$$

$$\partial_t \mathbf{v}_{\mathrm{d},a} + (\mathbf{v}_{\mathrm{d},a} \cdot \nabla)\mathbf{v}_{\mathrm{d},a} = -2\boldsymbol{\Omega} \times \mathbf{v}_{\mathrm{d},a} - \nabla\Phi - \frac{\mathbf{v}_{\mathrm{d},a} - \mathbf{v}_\mathrm{g}}{\tau_\mathrm{s}(a)}. \tag{A.29}$$

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{v} \times \mathbf{B}) - \nabla \times \left\{ \eta_\mathrm{Ohm} \nabla \times \mathbf{B} + \eta_\mathrm{Hall} (\nabla \times \mathbf{B}) \times \frac{\mathbf{B}}{B} + \eta_\mathrm{AD} \frac{\mathbf{B}}{B} \times \left[(\nabla \times \mathbf{B}) \times \frac{\mathbf{B}}{B} \right] \right\} - \hat{\mathbf{y}} \frac{3}{2} \Omega_0 B_x, \tag{A.30}$$

$$\boldsymbol{\nabla} \cdot \mathbf{B} = 0 \tag{A.31}$$

2.4. Steady state solutions

As detailed in Chapter 2.3, the dust drifts inward due to the drag. When taken in the polydisperse dust case, Paardekooper et al. (2020) details the background solution to be

$$v_{gx} = \frac{2\eta}{\kappa} \frac{\mathcal{J}_1}{(1 + \mathcal{J}_0)^2 + \mathcal{J}_1^2}, \tag{A.32}$$

$$v_{gy} = -Sx - \frac{\eta}{\Omega} \frac{1 + \mathcal{J}_0}{(1 + \mathcal{J}_0)^2 + \mathcal{J}_1^2}, \tag{A.33}$$

$$v_{dx} = \frac{2\eta}{\kappa} \frac{\mathcal{J}_1 - \kappa\tau_\mathrm{s}(a)(1 + \mathcal{J}_0)}{(1 + \kappa^2\tau_\mathrm{s}(a)^2)((1 + \mathcal{J}_0)^2 + \mathcal{J}_1^2)}, \tag{A.34}$$

$$v_{dy} = -Sx - \frac{\eta}{\Omega} \frac{1 + \mathcal{J}_0 + \kappa\tau_\mathrm{s}(a)\mathcal{J}_1}{(1 + \kappa^2\tau_\mathrm{s}(a)^2)((1 + \mathcal{J}_0)^2 + \mathcal{J}_1^2)}, \tag{A.35}$$

with integrals

$$\mathcal{J}_m = \frac{1}{\rho_\mathrm{g}} \int \frac{\sigma(\kappa\tau_\mathrm{s}(a))^m}{1 + \kappa^2\tau_\mathrm{s}(a)^2} \mathrm{d}a, \tag{A.36}$$

with the addition in this study of the vertical magnetic field. We note that taking a varying magnetic field function of r results in a radial Lorentz force. In such a case, the radial force applied to the gas force would be

$$\mathbf{f}_\mathrm{g} = (2\eta + F_{L,x})\hat{\mathbf{x}} - \nabla\Phi, \tag{A.37}$$

which, as in Lin & Hsu (2022), can be interpreted as an additional drift pressure by taking η to be $\eta_\mathrm{tot} = \eta + \frac{F_R}{2}$.

2.5. Plasma parameter

As detailed in Sec. 3.3, the relative importance of MHD can be related to a non-dimensional parameter, the plasma beta, which is defined here as

$$\beta \equiv \frac{P_{\text{gas}}}{P_{\text{mag}}} = \frac{2\mu_0 \rho c_s^2}{\gamma B^2} = \frac{2c_s^2}{\gamma c_a^2}, \tag{A.38}$$

where the alfven speed is

$$c_a \equiv \frac{|B_z|}{\sqrt{\rho \mu_0}}. \tag{A.39}$$

Alternatively we can defines the magnetic field B using β and the soundspeed c_s to be

$$B = \sqrt{\frac{2\mu_0 \rho}{\gamma}} \frac{c_s}{\sqrt{\beta}}, \tag{A.40}$$

$$c_a = \sqrt{\frac{2}{\gamma \beta}} c_s. \tag{A.41}$$

3. Reducing the problem to standard PSI

3.1. Solenoidal condition

The solenoidal condition becomes here

$$\nabla \cdot \mathbf{B} = 0 \mapsto \nabla \cdot \bar{\mathbf{B}} + \nabla \cdot \delta \mathbf{B}. \tag{A.42}$$

However, this solenoidal condition being already verified for the background implies that

$$\nabla \cdot \delta \mathbf{B} = 0 \quad \Leftrightarrow \quad \mathbf{k} \cdot \tilde{\mathbf{B}} = 0 \tag{A.43}$$

The perturbed magnetic field must be orthogonal to the wavevector. In practice, to impose such a condition, we can replace every occurence of, for example, $\tilde{B}_y$ by $-(\tilde{B}_x k_x + \tilde{B}_z k_z)/k_y$ for non-null k_y.

3.2. Lorentz force

In this section, the background state and the linear pertubation will be denoted by the superscript 0. Additionally, axisymetric perturbed quantites are expanded as

Fourier modes: $a^1 = \tilde{a}e^{i(\mathbf{k}\cdot\mathbf{x}-\omega t)}$, with $\partial_y = 0$. The pertubed Lorentz force is

$$f^1_{\mathrm{g,x}} = -\frac{1}{\rho_\mathrm{g}}\partial_x\left(\frac{\mathbf{B}^0\cdot\mathbf{B}^1}{4\pi}\right) + \frac{1}{4\pi\rho_\mathrm{g}}\left(\mathbf{B}^0\cdot\nabla\right)B^1_x,$$
$$f^1_{\mathrm{g,y}} = -\frac{1}{\rho_\mathrm{g}}\partial_y\left(\frac{\mathbf{B}^0\cdot\mathbf{B}^1}{4\pi}\right) + \frac{1}{4\pi\rho_\mathrm{g}}\left(\mathbf{B}^0\cdot\nabla\right)B^1_y,$$
$$f^1_{\mathrm{g,z}} = -\frac{1}{\rho_\mathrm{g}}\partial_z\left(\frac{\mathbf{B}^0\cdot\mathbf{B}^1}{4\pi}\right) + \frac{1}{4\pi\rho_\mathrm{g}}\left(\mathbf{B}^0\cdot\nabla\right)B^1_z,$$

where, using the vertical magnetic field background, the force can be written as

$$f^1_{\mathrm{g,x}} = -\frac{1}{4\pi\rho_\mathrm{g}}\left(B^0_z\partial_x\right)B^1_z + \frac{1}{4\pi\rho_\mathrm{g}}\left(B^0_z\partial_z\right)B^1_x,$$
$$f^1_{\mathrm{g,y}} = \frac{1}{4\pi\rho_\mathrm{g}}\left(B^0_z\partial_z\right)B^1_y,$$
$$f^1_{\mathrm{g,z}} = 0,$$

Therefor, the Lorentz force pertubation can be written as $\tilde{\mathbf{F}}_L = -i\chi\tilde{\mathbf{B}}$ with

$$\chi = \frac{1}{4\pi\rho_g}\begin{pmatrix} -B_{0,z}k_z & 0 & B_{0,z}k_x \\ 0 & -B_{0,z}k_z & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{A.44}$$

3.3. Induction equation

We now aim at rewriting the magnetic field as a function of the other parameters in order to rewrite the Lorentz as an additional source term to the existing polydisperse problem without magnetic fields described in Paardekooper et al. (2020, 2021); McNally et al. (2021). In this paragraph, we denote the gas velocity simply by $\mathbf{v}$. Firstly, since the background magnetic field is constant,

$$\frac{\partial\delta\mathbf{B}}{\partial t} = \nabla\times(\delta\mathbf{v}\times\mathbf{B}_0) + \nabla\times(\mathbf{v}_0\times\delta\mathbf{B}) - \nabla\times\{\eta_\mathrm{Ohm}\nabla\times\delta\mathbf{B}$$
$$+\eta_\mathrm{Hall}\left(\nabla\times\delta\mathbf{B}\right)\times\frac{\mathbf{B}_0}{B_0} + \eta_\mathrm{AD}\frac{\mathbf{B}_0}{B_0}\times\left[\left(\nabla\times\delta\mathbf{B}\right)\times\frac{\mathbf{B}_0}{B_0}\right]\Big\} - \hat{\mathbf{y}}\frac{3}{2}\Omega_0\delta B_x, \tag{A.45}$$

Replacing by the Fourier components yield,

$$-i\omega\tilde{\mathbf{B}} = i\mathbf{k}\times(\tilde{\mathbf{v}}\times\mathbf{B}_0) + i\mathbf{k}\times(\mathbf{v}_0\times\tilde{\mathbf{B}}) - i\mathbf{k}\times\Big\{\eta_\mathrm{Ohm}i\mathbf{k}\times\tilde{\mathbf{B}}$$
$$+\eta_\mathrm{Hall}\left(i\mathbf{k}\times\tilde{\mathbf{B}}\right)\times\frac{\mathbf{B}_0}{B_0} + \eta_\mathrm{AD}\frac{\mathbf{B}_0}{B_0}\times\left[\left(i\mathbf{k}\times\tilde{\mathbf{B}}\right)\times\frac{\mathbf{B}_0}{B_0}\right]\Big\} - \hat{\mathbf{y}}\frac{3}{2}\Omega_0\tilde{B}_x, \tag{A.46}$$

which can be rewritten in the form

$$0 = i\mathbf{k}\times(\tilde{\mathbf{v}}\times\mathbf{B}_0) + i\omega\tilde{\mathbf{B}} + i\mathbf{k}\times(\mathbf{v}_0\times\tilde{\mathbf{B}}) - i\mathbf{k}\times\Big\{\eta_\mathrm{Ohm}i\mathbf{k}\times\tilde{\mathbf{B}}$$
$$+\eta_\mathrm{Hall}\left(i\mathbf{k}\times\tilde{\mathbf{B}}\right)\times\frac{\mathbf{B}_0}{B_0} + \eta_\mathrm{AD}\frac{\mathbf{B}_0}{B_0}\times\left[\left(i\mathbf{k}\times\tilde{\mathbf{B}}\right)\times\frac{\mathbf{B}_0}{B_0}\right]\Big\} - \hat{\mathbf{y}}\frac{3}{2}\Omega_0\tilde{B}_x, \tag{A.47}$$

Where all terms are either factors of $\tilde{\mathbf{v}}$ or $\tilde{\mathbf{B}}$, allowing us to rewrite it as the following vector equality:

$$0 = N \cdot \tilde{\mathbf{v}} + K \cdot \tilde{\mathbf{B}}, \tag{A.48}$$

where N and K are matrices. Lastly we can rewrite the magnetic field perturbation $\tilde{\mathbf{B}}$ to be a function of $\tilde{\mathbf{v}}$

$$\tilde{\mathbf{B}} = -K^{-1} \cdot N \cdot \tilde{\mathbf{v}}, \tag{A.49}$$

provided that K^{-1}, the invert of K exist. Using that result, we can rewrite the Lorentz force in the gas velocity equation to be

$$\tilde{\mathbf{F}}_L = i\chi \cdot K^{-1} \cdot N \cdot \tilde{\mathbf{v}}_g \tag{A.50}$$

Thus, the complete pertubation equation for the polydisperse dust gas mixture in a shearing box with non-ideal MHD is the standard problem without a magnetic field, with an additional source term on the velocity perturbation.

4. Numerical method

In order to study that system, we use the same method (implemented in the community code PSITOOLS) as described in Paardekooper et al. (2020, 2021); McNally et al. (2021), where the Lorentz force is a source term for velocity perturbations. It allows us to find the largest eigenvalues whose imaginary part is positive in the continuous eigenvalue problem of the polydisperse streaming instability. Aside from floating-point errors, the method is, by design, guaranteed to find the most unstable eigenvalue. However, the procedure can take up to a minute per value of k. To be able to carry out this study, we parralelized using MPI the computation of the eigenvalues. The strategy goes as follows: a main process is charged with receiving any found eigenvalue. That process performs the following loop: perform a MPI receive operation from any source; when a message is received, write the result to the file storing the list of found eigenvalues. Then repeat until shutdown of the program. The other processes perform the following loop: generate a random non-dimensional wavenumber $\mathbf{K} = \mathbf{k}\eta/\Omega^2$ value in log space, find the corresponding eigenvalue with the largest imaginary part, send the eigenvalue with the generated $\mathbf{K}$ to the main process, then repeat until shutdown of the program. The results that will be presented each consist of a few hours of computation on 256 cores.

5. Results

We first reproduced the standard SI for monodisperse grains in Fig. A.1. The low K_x region ($K_x < 1$) and high K_z region ($K_z > 1$) correspond roughly to the so-called epicyclic modes of streaming instability. The unstable band for higher K_x corresponds to the so-called secular modes of streaming instability.

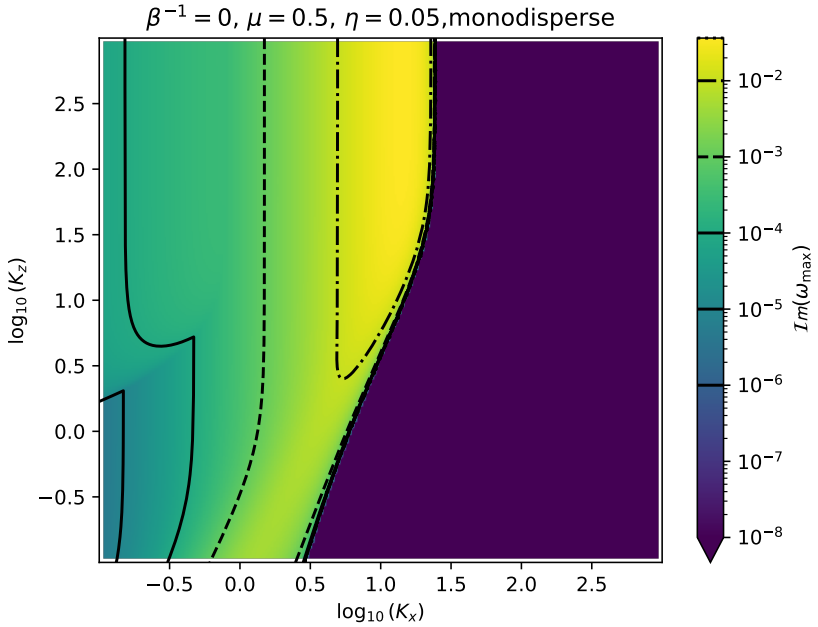


Figure A.1: Standard SI

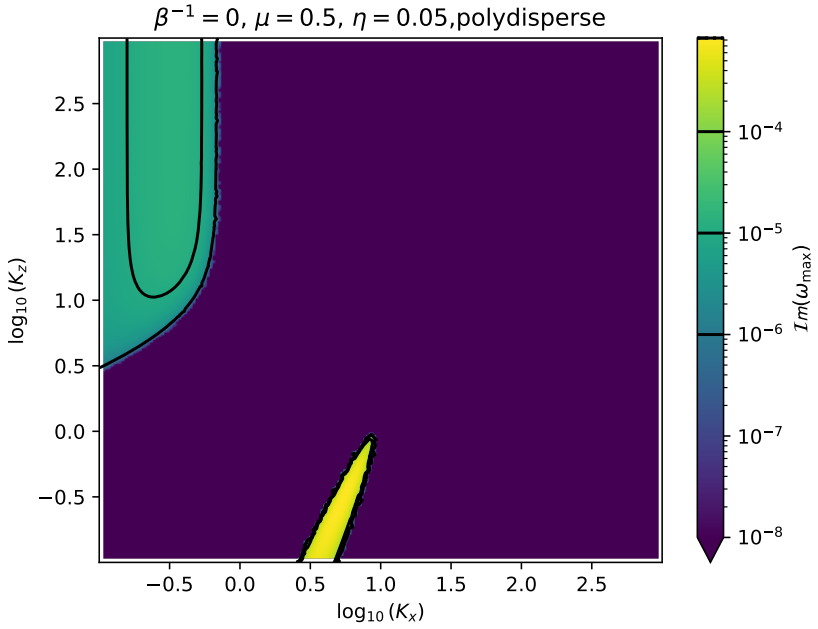


Figure A.2: Standard PSI

When moving to polydisperse dust distribution, both the secular and epyciclic streaming instability are much less unstable ($Im(\omega_{\max})$ is more than 2 orders of

magnitude lower). Additionally, the unstable regions are narrower compared to the standard streaming instability. As discussed in Paardekooper et al. (2021), the addition of turbulence in this case can make the polydisperse streaming instability inoperable.

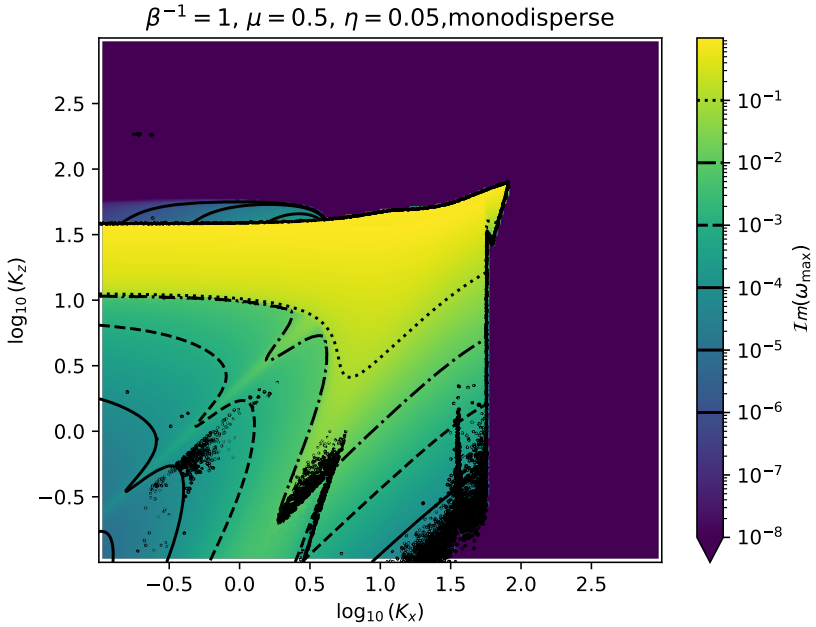


Figure A.3: Magnetized SI

We now present the case of monodisperse dust in the presence of a magnetic field, as shown in Fig. A.3. This situation should reproduce the results of Lin & Hsu (2022). The result are qualitatively correct, we need now to perform a quantitative analysis to verify that we fully reproduced them. The presence of the magnetic field results in multiple changes to the streaming instability. First, most of the epyciclic unstable modes are removed in presence of a magnetic field. Second, the secular mode of streaming instability is still present. Last, an additional Alfven wave-supported streaming instability is present along the $K_x \simeq K_z$ axis (see Lin & Hsu 2022 for discussion about this specific mode). In addition to streaming instability the magnetorotational instability is present on top of the streaming instability in the $1 \lesssim K_z \lesssim 1.5$ region on Fig. A.3.

Lastly, when using a polydisperse dust distribution, we now observe that most of the features in the magnetized monodisperse streaming instability remain with a comparable growth rate.

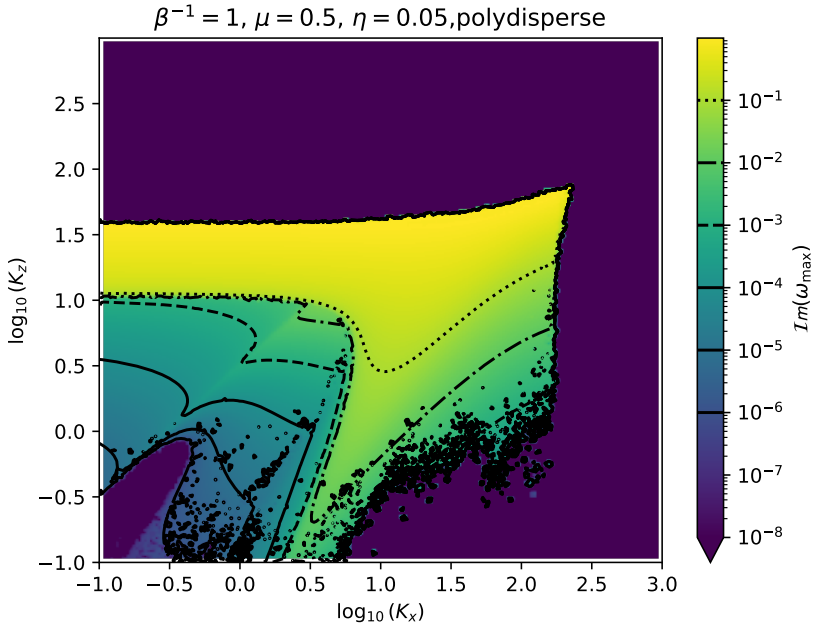


Figure A.4: Magnetized PSI

6. Discussion and future prospects

We have shown in this study that the presence of a magnetic field can result in the presence of modes that are attributed to magnetized streaming instability in the polydisperse case, whereas the polydispersity results in the non-magnetized case in quenching the instability. This suggests that the presence of a magnetic field makes streaming instability more robust to polydisperse quenching. However, we note that the modes attributed to the MRI have a larger growth rate and therefore could dominate SI growth. However, the non-linear evolution of magnetized polydisperse streaming instability would require numerical codes able to resolve polydisperse dust distribution, which is a yet-to-be-solved numerical challenge. We, however, did not assessed if the eigenvectors can still concentrate dust which is the characteristic feature of streaming instability. We also did not fully validated that the monodisperse magnetised case reproducing exactly results from Lin & Hsu (2022). This study shows the possibility of resusing the community code PSITOOLS and extending it to the magnetised case without significant changes to the code. Future prospects include getting a better understanding of the physical processes underlying the magnetised polydisperse streaming instability and verifying its conformance with the literature in the monodisperse case. Additionally the development of numerical schemes able to capture polydisperse effects would be required to study its non-linear evolution. In SHAMROCK, we aim at optimizing the future dust solver to resolve simulations with large numbers of dust species and MHD to attempt to

reproduce the linear results as well as exploring the non-linear phase.

References

Goldreich P., Lynden-Bell D., 1965, *II. Spiral arms as sheared gravitational instabilities*, MNRAS, 130, 125

Krapp L., Benítez-Llambay P., Gressel O., Pessah M. E., 2019, *Streaming Instability for Particle-size Distributions*, ApJ, 878, L30

Lin M.-K., 2021, *Stratified and Vertically Shearing Streaming Instabilities in Protoplanetary Disks*, ApJ, 907, 64

Lin M.-K., Hsu C.-Y., 2022, *Streaming Instabilities in Accreting and Magnetized Laminar Protoplanetary Disks*, ApJ, 926, 14

McNally C. P., Lovascio F., Paardekooper S.-J., 2021, *Polydisperse streaming instability - III. Dust evolution encourages fast instability*, MNRAS, 502, 1469-1486

Mouschovias T. C., Kunz M. W., Christie D. A., 2009, *Formation of interstellar clouds: Parker instability with phase transitions*, MNRAS, 397, 14-23

Paardekooper S.-J., McNally C. P., Lovascio F., 2020, *Polydisperse streaming instability - I. Tightly coupled particles and the terminal velocity approximation*, MNRAS, 499, 4223-4238

Paardekooper S.-J., McNally C. P., Lovascio F., 2021, *Polydisperse streaming instability - II. Methods for solving the linear stability problem*, MNRAS, 502, 1579-1595

Tsukamoto Y., et al.,in *Astronomical Society of the Pacific Conference Series*, booktitle, edited by Inutsuka, S. and Aikawa, Y. and Muto, T. and Tomida, K. and Tamura, M., p. 317

Appendix B

Precision of 2-fluid SPH methods

Contents

Foreword

This work was performed at the beginning of the Ph.D. thesis to understand the precision of the dust-gas mixture in the SPH relative to the analytical model. It was motivated as a possible extension of the analysis of the SPH soundwave presented in Chapter 1,Sec. 4.10 and to deeply understand the behavior of two-fluid dusty SPH. This work derives the analytical dispersion relation for the two-fluid SPH dust gas model and finds an analytical criterion for the reconstruction parameter to maximize the precision of the scheme.

1. The Dustywave problem

As presented in David-Cléris & Laibe (2021), we consider the following equations of motion for the evolution of a one-dimensional astrophysical

$$\partial_t \rho_{\rm g} + v_{\rm g}\partial_x \rho_{\rm g} = -\rho_{\rm g}\,\partial_x v_{\rm g}, \tag{B.1}$$
$$\partial_t \rho_{\rm d} + v_{\rm d}\partial_x \rho_{\rm d} = -\rho_{\rm d}\,\partial_x v_{\rm d}, \tag{B.2}$$
$$\partial_t v_{\rm g} + v_{\rm g}\partial_x v_{\rm g} = +\frac{K}{\rho_{\rm g}}\left(v_{\rm d}-v_{\rm g}\right) - \frac{\partial_x P}{\rho_{\rm g}}, \tag{B.3}$$
$$\partial_t v_{\rm d} + v_{\rm d}\partial_x v_{\rm d} = -\frac{K}{\rho_{\rm d}}\left(v_{\rm d}-v_{\rm g}\right), \tag{B.4}$$

where g and d stand for gas and dust, respectively (e.g. Garaud et al. 2004) and K denotes the drag coefficient. Assuming isothermal gas $P = c_{\rm s}^2\rho_{\rm g}$, we expand linearly

Eqs. B.1 – B.4 under the generic form $a = a_0 + \delta a$, with $v_{\rm d0} = v_{\rm g0} = 0$. One obtains

$$\partial_t \delta\rho_{\rm g} = -\rho_{\rm g,0}\,\partial_x \delta v_{\rm g}, \tag{B.5}$$
$$\partial_t \delta\rho_{\rm d} = -\rho_{\rm d,0}\,\partial_x \delta v_{\rm d}, \tag{B.6}$$
$$\partial_t \delta v_{\rm g} = +\frac{K}{\rho_{\rm g,0}}\left(\delta v_{\rm d} - \delta v_{\rm g}\right) - c_{\rm s}^2 \frac{\partial_x \delta\rho_{\rm g}}{\rho_{\rm g,0}}, \tag{B.7}$$
$$\partial_t \delta v_{\rm d} = -\frac{K}{\rho_{\rm d,0}}\left(\delta v_{\rm d} - \delta v_{\rm g}\right). \tag{B.8}$$

We decompose the perturbation on Fourier space under the form $\delta a = \tilde{a} e^{i(kx-\omega t)}$ for each perturbed field, giving the condition

$$\begin{vmatrix} -i\omega & 0 & \frac{ic_{\rm s}k\rho_{\rm g,0}}{\rho_{\rm g,0}+\rho_{\rm d,0}} & 0 \\ 0 & -i\omega & 0 & \frac{ic_{\rm s}k\rho_{\rm d,0}}{\rho_{\rm g,0}+\rho_{\rm d,0}} \\ \frac{ic_{\rm s}k\left(\rho_{\rm g,0}+\rho_{\rm d,0}\right)}{\rho_{\rm g,0}} & 0 & -i\omega + \frac{1}{t_{\rm g}} & -\frac{1}{t_{\rm g}} \\ 0 & 0 & -\frac{1}{t_{\rm d}} & -i\omega + \frac{1}{t_{\rm d}} \end{vmatrix} = 0, \tag{B.9}$$

where $t_{\rm g} \equiv \frac{\rho_{\rm g,0}}{K}$ and $t_{\rm d} \equiv \frac{\rho_{\rm d,0}}{K}$. We obtain the following dispersion relation

$$\omega^4 + \frac{i}{t_{\rm s}}\omega^3 - c_{\rm s}^2 k^2 \omega^2 - \frac{i}{t_{\rm s}} c_{\rm s}^2 k^2 (1-\epsilon)\omega = 0, \tag{B.10}$$

where the barycentric stopping time is $t_{\rm s} \equiv \frac{\rho_{\rm g,0}\rho_{\rm d,0}}{K\left(\rho_{\rm g,0}+\rho_{\rm d,0}\right)}$ and $\epsilon \equiv \frac{\rho_{\rm d,0}}{\rho_{\rm g,0}+\rho_{\rm d,0}}$ is the total dust fraction. Rescaling time and space by $t_{\rm s}$ and $c_{\rm s}t_{\rm s}$ respectively gives in its dimensionless form

$$\omega^4 + i\omega^3 - k^2\omega^2 - ik^2(1-\epsilon)\omega = 0, \tag{B.11}$$

where we preserved the notations ω and k for further readability. We disregard the solution $\omega_{\rm null} = 0$ on the null space. On this space, Eq. B.11 reduces to

$$\omega^3 + i\omega^2 - \omega k^2 - ik^2(1-\epsilon) = 0, \tag{B.12}$$

which can alternatively be written under the convenient form.

$$\omega^2 - k^2 + \frac{i}{\omega}\left(\omega^2 - k^2(1-\epsilon)\right) = 0. \tag{B.13}$$

2. SPH dustywave

2.1. Equation of motions

We want to study the behavior of a dustywave in SPH. In particular, we want to study the response of the scheme described in Price & Laibe (2020) to linear perturbation. As introduced in Chapter 4.10, we use a simplified model to study its

linear response. For simplicity, we neglect the smoothing length evolution due to the density perturbation and assume an isothermal equation of state $P = c_{\rm s}^2\rho$. Using index $(a,b),(i,j)$, respectively, for the gas and the dust, the equations of motion of the SPH model detailed in Price & Laibe (2020) can be written as follows:

$$\frac{{\rm d}\mathbf{x}_a}{{\rm d}t} = \mathbf{v}_a, \tag{B.14}$$

$$\frac{{\rm d}\mathbf{x}_i}{{\rm d}t} = \mathbf{v}_i, \tag{B.15}$$

$$\rho_a = \sum_b m_b W_{ab}(h_a), \tag{B.16}$$

$$\rho_i = \sum_j m_j W_{ij}(h_i), \tag{B.17}$$

$$\frac{{\rm d}\mathbf{v}_a}{{\rm d}t} = -\mathcal{D}_{a,{\rm g}} - \mathcal{P}_a, \tag{B.18}$$

$$\frac{{\rm d}\mathbf{v}_i}{{\rm d}t} = +\mathcal{D}_{i,{\rm d}}, \tag{B.19}$$

where the pressure and drag terms are expressed in the following form:

$$\mathcal{P}_a \equiv c_s^2 \sum_b m_b \left(\frac{1}{\rho_a} + \frac{1}{\rho_b}\right) \frac{\partial W_{ab}}{\partial x}, \tag{B.20}$$

$$\mathcal{D}_{a,{\rm g}} \equiv \sum_i m_i \frac{(\mathbf{v}^*_{ai}\cdot\hat{\mathbf{r}}_{ai})\,\hat{\mathbf{r}}_{ai}}{(\rho_a+\rho_i)\,t^{\rm s}_{ai}} D_{ai}(h), \tag{B.21}$$

$$\mathcal{D}_{i,{\rm d}} \equiv \sum_a m_a \frac{(\mathbf{v}^*_{ai}\cdot\hat{\mathbf{r}}_{ai})\,\hat{\mathbf{r}}_{ai}}{(\rho_i+\rho_a)\,t^{\rm s}_{ai}} D_{ai}(h), \tag{B.22}$$

where index $(a,b),(i,j)$ stand respectively for gas and dust, $t^{\rm s}_{aj} = \frac{\rho_a\rho_j}{K(\rho_a+\rho_j)}$. $h = \max(h_a,h_i)$. $\mathbf{v}^*$ refers to the reconstruction used on the velocities from Price & Laibe (2020), defined as follows:

$$\mathbf{v}^*_{ai}\cdot\hat{\mathbf{r}}_{ai} = \mathbf{v}_{ai}\cdot\hat{\mathbf{r}}_{ai} - \mu_{ai}|\mathbf{r}_{ai}|\left(S_{ai}+S_{ia}\right), \tag{B.23}$$

where $\mu_{ai} \equiv m_a/(m_a+m_i)$ is a so-called reconstruction parameter, which allows selecting where the drag force is evaluated on the line of sight joining two SPH particles of different types.

2.2. Linear perturbation

We expand linearly the*numerical equations* Eq. B.14 – B.22 using the following form

$$a_a = a_{a,0} + \delta a_a \tag{B.24}$$

$$a_i = a_{i,0} + \delta a_i \tag{B.25}$$

$$\delta a_a = \tilde{a}_g e^{i(kx_{a,0}-\omega t)} \tag{B.26}$$

$$\delta a_i = \tilde{a}_d e^{i(kx_{i,0}-\omega t)} \tag{B.27}$$

discrete SPH sum	Cnntinuous limit
$1^{(W)}_{\mathrm{SPH}} = \frac{1}{\rho}\sum_a m_a W(x_a - b_b, h)$	1
$0^{(W)}_{\mathrm{SPH}} = \frac{1}{\rho}\sum_a m_a \frac{\mathrm{d}W}{\mathrm{d}x}(x_a - b_b, h)$	0
$\sum_a m_a f(x_a)$	$\rho \int \mathrm{d}x\, f(x)$

Table B.1: Continuous limit of the SPH terms involved in the derivation.

for $a = (\rho, x, v)$. We note that under the following notation the velocity perturbation is related by a factor $-i\omega$ to the perturbation on the particle position, $\tilde{v}_{g/d} = -i\omega\tilde{x}_{g/d}$. The details of the computation is presented Sec. 4.

2.3. Continuous limit

We now perform the continuous limit (infinite number of particle) on the results of Sec. 4. To do so we use the dictionary presented Table B.1. The mass conservation equation in the continuous limit yields

$$-i\omega\tilde{\rho}_g + ik\rho_g\hat{W}_{\mathrm{g}}\tilde{v}_g = 0, \tag{B.28}$$

$$-i\omega\tilde{\rho}_d + ik\rho_g\hat{W}_{\mathrm{d}}\tilde{v}_d = 0. \tag{B.29}$$

And the pressure and drag operators are

$$\mathcal{P} = e^{i(kx_{a,0}-\omega t)}c_s^2k^2\tilde{x}_g\left[2\hat{W} - \hat{W}^2\right], \tag{B.30}$$

$$\mathcal{D}_{a,\mathrm{g}} = e^{i(kx_{a,0}-\omega t)}\left[\frac{1}{t_g}\tilde{v}_g - \frac{1}{t_g}\left(\hat{D}_h - k\mu\hat{W}_d\frac{\partial\hat{D}_h}{\partial k}\right)\tilde{v}_d\right], \tag{B.31}$$

$$\mathcal{D}_{i,\mathrm{d}} = e^{i(kx_{i,0}-\omega t)}\left[\frac{1}{t_d}\left(\hat{D}_h - k\mu\hat{W}_g\frac{\partial\hat{D}_h}{\partial k}\right)\tilde{v}_g - \frac{1}{t_d}\tilde{v}_d\right], \tag{B.32}$$

where $\hat{W}_{\mathrm{g}}$, $\hat{W}_{\mathrm{d}}$, and $\hat{D}$ denotes, respectively the Fourier transform of the SPH kernels W for the gas, W for the dust, and drag SPH kernel D. Replacing the operators formulas in the equation of motion Eq. B.14–B.19 provides the following set of equation for the linear SPH dustywave problem.

$$0 = -i\omega\tilde{\rho}_{g/d} + ik\rho_g\hat{W}\tilde{v}_{g/d}, \tag{B.33}$$

$$0 = -i\omega\tilde{v}_g + \frac{ikc_s^2}{\rho_g}\left(2 - \hat{W}\right)\tilde{\rho}_g + \frac{1}{t_g}\left(1 - Z_{\mathrm{g}}\right)\tilde{v}_g, \tag{B.34}$$

$$0 = -i\omega\tilde{v}_d + \frac{1}{t_d}\left(1 - Z_{\mathrm{d}}\right)\tilde{v}_d, \tag{B.35}$$

were we defined,

$$Z_{\rm g} \equiv \hat{D}_h - k\mu\hat{W}_d\frac{\partial\hat{D}_h}{\partial k}, \tag{B.36}$$

$$Z_{\rm d} \equiv \hat{D}_h - k\mu\hat{W}_g\frac{\partial\hat{D}_h}{\partial k}. \tag{B.37}$$

The determinant condition for the dispersion relation can be written in a similar form as eq.B.9

$$\begin{vmatrix} -i\omega & 0 & ik\hat{W}_g\rho_g & 0 \\ 0 & -i\omega & 0 & ik\hat{W}_d\rho_d \\ \frac{ic_s^2k(2-\hat{W}_g)}{\rho_g} & 0 & \frac{1}{t_g}-i\omega & -\frac{1}{t_g}Z_g \\ 0 & 0 & -\frac{1}{t_d}Z_d & \frac{1}{t_d}-i\omega \end{vmatrix} = 0 \tag{B.38}$$

The corresponding dispersion relation for the complete perturbation, not accounting for the continuous limit, is presented in Sec. 5.

2.3.1. With reconstruction

By computing the determinant Eq. B.38, we get the following dispersion relation:

$$0 = \omega^2 - c_s^2k^2\mathcal{I}_W + \frac{i}{\omega t_s}\left(\omega^2 - c_s^2k^2(1-\epsilon)\mathcal{I}_W\right) + \frac{(1-\epsilon)\epsilon}{t_s^2}\mathcal{O}_D^*, \tag{B.39}$$

$$\mathcal{I}_W = 2\hat{W}_g - \hat{W}_g^2, \tag{B.40}$$

$$\mathcal{O}_D^* = \left(\hat{D}_h - k\mu\hat{W}_d\frac{\partial\hat{D}_h}{\partial k}\right)\left(\hat{D}_h - k\mu\hat{W}_g\frac{\partial\hat{D}_h}{\partial k}\right) - 1, \tag{B.41}$$

where $\mathcal{I}_W$ is the resulting numerical correction to the wavevector k. We recover the physical dispersion relation in the $h \to 0$ limit for any $\mu \in [0,1]$ since $\mathcal{I}_W \to 1$ and $\mathcal{O}_D^* \to 0$. This shows the validity of the method in the continuum equation limit (where SPH kernels converge to Dirac distributions).

2.3.2. Without reconstruction

The absence of reconstruction is equivalent to setting $\mu = 0$. In absence of reconstruction the SPH scheme is the usual two-fluid model as used in Price et al. (2018). In this context the dispersion relation is

$$0 = \omega^2 - c_s^2k^2\mathcal{I}_W + \frac{i}{\omega t_s}\left(\omega^2 - c_s^2k^2(1-\epsilon)\mathcal{I}_W\right) + \frac{(1-\epsilon)\epsilon}{t_s^2}\mathcal{O}_D, \tag{B.42}$$

$$\mathcal{I}_W = 2\hat{W}_g - \hat{W}_g^2, \tag{B.43}$$

$$\mathcal{O}_D = \hat{D}_h^2 - 1. \tag{B.44}$$

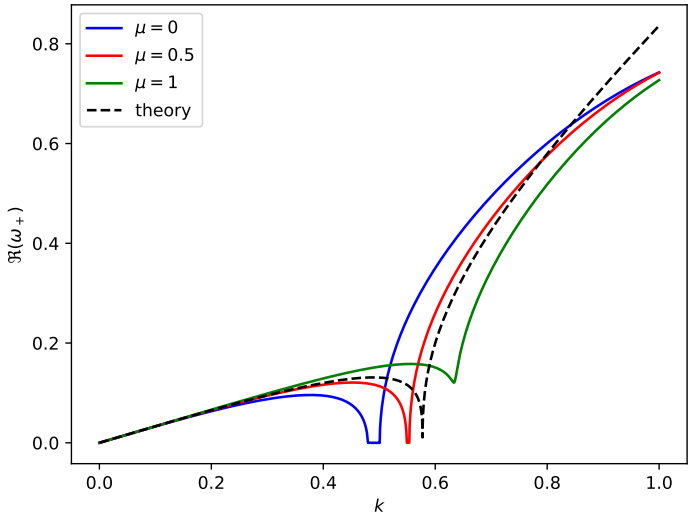


Figure B.1: Real part of the root ω_+ for different values of μ with $\epsilon = 8/9$ for the M_4 kernel. The black dashed line provides the analytic solution of the physical dustywave problem. The Blue, green and red solid lines provide the analytic solution of the smoothed equation in the continuous limit for different values of the reconstruction parameter.

2.4. Analytic spatial resolution criterion

From eq.B.39, we see that the error in the numerical case comes from two sources: the resolution of the drag and the gas dynamic. Since $h = \max(h_i, h_a)$, the drag resolution is equal to the lowest resolution between gas and dust. The resolution in the dust contributes mainly to the resolution of dust-specific features such as mixture modes, bifurcation of the dustywave problem between non-propagative and soundwave like mode, and the gas's sound waves. Therefore, the CFL can still be taken to be $\Delta t < C\frac{h}{c_s}$, but in order to resolve the dust accurately, we need to have $h < c_s t_s$ to resolve the mixture mode and the transition to sound wave.

2.5. Optimal reconstruction

We now seek an optimal value for the reconstruction parameter μ from the dispersion relation. We expand linearly $Z_{g/d}$ in orders of k. This yields

$$\begin{aligned} Z_{g/d} &= \hat{D}_h - k\mu\hat{W}_{g/d}\frac{\partial\hat{D}_h}{\partial k} \\ &= 1 + \frac{k^2}{2}\frac{\partial^2\hat{D}_h}{\partial k^2} - k\mu\hat{W}_{g/d}(0)k\frac{\partial^2\hat{D}_h}{\partial k^2} + o(k^3) \\ &= 1 + \frac{k^2}{2}\frac{\partial^2\hat{D}_h}{\partial k^2}\left[1-2\mu\right] + o(k^3), \end{aligned}$$

which give the result

$$\mathcal{O}_D^* = k^2\frac{\partial^2\hat{D}_h}{\partial k^2}\left[1-2\mu\right] + o(k^3). \tag{B.45}$$

When using $\mu = 0.5$ (ie $m_{\rm g} = m_{\rm d}$) the $D''(0)$ term is canceled. This results in the error created by the drag term being of order 4 in k. It is therefore suitable to always use $\mu = 0.5$. We additionally show the real part of the root ω^+ for different values of the reconstruction parameter μ in Fig. B.1, where $\mu = 0.5$ yields the best result. This is the first time that the value of a reconstruction parameter is constrained analytically in SPH.

2.6. Test in simulations

We have run the same tests as described in Price & Laibe (2020), for a dustfraction of $\epsilon = 8/9$, for multiple values of the reconstruction parameter μ. We present the result of the L1 error of the DUSTYWAVE in Fig. B.2 and DUSTYSHOCK test in Fig. B.3. Both tests also suggest $\mu = 0.5$ as the best choice of the reconstruction parameter as suggested by the performed analytical study.

3. Conclusion

We have derived the analytical dispersion relation of a two-fluid SPH dust gas model with reconstruction developed in Price & Laibe (2020). From this dispersion relation, the criterion $\mu = 0.5$ was found to maximize the precision of the scheme in the linear perturbation regime. Additionally, the two-fluid dust-gas SPH scheme reproduces the analytical results of the dustywave problem detailed in David-Cléris & Laibe (2021). We note, however, that this work was performed in the continuous limit, and the effect of discrete errors should be ideally accounted for using the dispersion relation detailed in Sec. 5, which could be an extension of this work. This analysis may be helpful to improve the viscosity term in SPH with an optimal reconstruction.

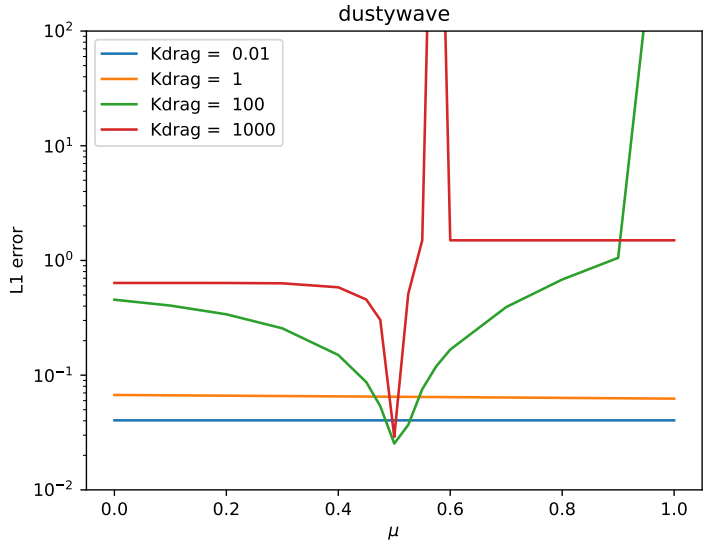


Figure B.2: Real part of the root ω_+ for different values of μ with $\epsilon = 8/9$ for the M_4 kernel.

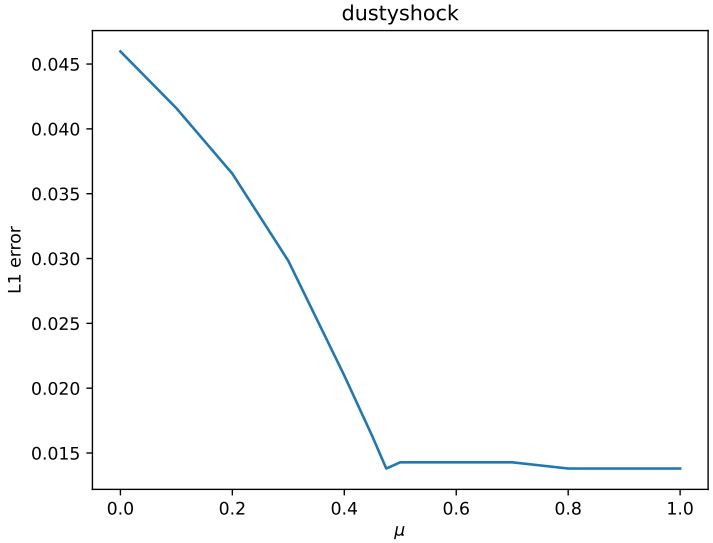


Figure B.3: Real part of the root ω_+ for different values of μ with $\epsilon = 8/9$ for the M_4 kernel.

Appendix

4. Linear expansion of the SPH equations

4.1. Mass conservation

We start by expanding linearly the mass conservation equation Eq. B.16:

$$
\begin{aligned}
\rho_a &= \sum_b m_b W_{ab}(h_a) \\
\delta\rho_a &= -\rho_{\mathrm{g},0} + \underbrace{\sum_b m_b W(x_{a,0} - x_{b,0})}_{\rho_{\mathrm{g},0}\mathbb{1}_{\mathrm{SPH}}} \\
&\quad + \sum_b m_b(\delta x_a - \delta x_b)\frac{\partial W}{\partial x}(x_{a,0} - x_{b,0}) \\
\tilde{\rho}_{\mathrm{g}} &= \rho_{\mathrm{g},0}\left(\mathbb{1}_{\mathrm{SPH}} - 1\right) e^{-i(kx_{a,0}-\omega t)} \\
&\quad + \tilde{x}_{\mathrm{g}} \sum_b m_b(1 - e^{-ik(x_{a,0}-x_{b,0})})\frac{\mathrm{d}W}{\mathrm{d}x}(x_{a,0} - x_{b,0}, h_a) \\
-i\omega\tilde{\rho}_{\mathrm{g}} &= -i\omega\rho_{\mathrm{g},0}\left(\mathbb{1}_{\mathrm{SPH}} - 1\right) e^{-i(kx_{a,0}-\omega t)} \\
&\quad + \tilde{v}_{\mathrm{g}} \sum_b m_b(1 - e^{-ik(x_{a,0}-x_{b,0})})\frac{\mathrm{d}W}{\mathrm{d}x}(x_{a,0} - x_{b,0}, h_a) \\
-i\omega\tilde{\rho}_{\mathrm{g}} &= \underbrace{-i\omega\rho_{\mathrm{g},0}\left(\mathbb{1}_{\mathrm{SPH}} - 1\right) e^{-i(kx_{a,0}-\omega t)}}_{0_{\mathrm{disc,g}}} \\
&\quad + \tilde{v}_{\mathrm{g}}\Bigg[\underbrace{\sum_b m_b \frac{\mathrm{d}W}{\mathrm{d}x}(x_{a,0} - x_{b,0}, h_a)}_{\rho_{\mathrm{g},0}0_{\mathrm{SPH}}^{(W_{\mathrm{g}})}} \\
&\quad \underbrace{- \sum_b m_b e^{-ik(x_{a,0}-x_{b,0})}\frac{\mathrm{d}W}{\mathrm{d}x}(x_{a,0} - x_{b,0}, h_a)}_{\rho_{\mathrm{g},0}\mathcal{F}_{\mathrm{SPH}}^{(W'_{\mathrm{g}})}}\Bigg] \\
-i\omega\tilde{\rho}_{\mathrm{g}} &= 0_{\mathrm{disc,g}} + \tilde{v}_{\mathrm{g}}\rho_{\mathrm{g},0}\left[0_{\mathrm{SPH}}^{(W_{\mathrm{g}})} - \mathcal{F}_{\mathrm{SPH}}^{(W'_{\mathrm{g}})}\right].
\end{aligned}
$$

The same computation can be performed for the dust by replacing subscripts g by d. Therefor

$$
-i\omega\tilde{\rho}_{\mathrm{g/d}} = 0_{\mathrm{disc,g/d}} + \tilde{v}_{\mathrm{g/d}}\rho_{\mathrm{g/d},0}\left[0_{\mathrm{SPH}}^{(W_{\mathrm{g/d}})} - \mathcal{F}_{\mathrm{SPH}}^{(W'_{\mathrm{g/d}})}\right].
$$

4.2. Pressure term

$$
\begin{aligned}
\mathcal{P} =& -c_s^2 \sum_b m_b \left(\frac{1}{\rho_a} + \frac{1}{\rho_b}\right) \frac{\partial W_{ab}}{\partial x} \\
=& -c_s^2 \sum_b m_b \left[\left(\frac{1}{\rho_{a,0}} + \frac{1}{\rho_{b,0}}\right) - \left(\frac{\delta\rho_a}{\rho_{a,0}^2} + \frac{\delta\rho_b}{\rho_{b,0}^2}\right)\right] \left[\frac{\partial W}{\partial x}(x_{a,0} - x_{b,0}) + (\delta x_a - \delta x_b)\frac{\partial^2 W}{\partial x^2}(x_{a,0} - x_{b,0})\right] \\
=& -c_s^2 \frac{2}{\rho_{\mathrm{g},0}} \underbrace{\sum_b m_b \frac{\partial W}{\partial x}(x_{a,0} - x_{b,0})}_{\rho_{\mathrm{g},0} 0_{\mathrm{SPH}}^{(W_\mathrm{g})}} - c_s^2 \frac{2}{\rho_{\mathrm{g},0}} \sum_b m_b (\delta x_a - \delta x_b) \frac{\partial^2 W}{\partial x^2}(x_{a,0} - x_{b,0}) \\
&\quad + c_s^2 \frac{1}{\rho_{\mathrm{g},0}^2} \sum_b m_b \left(\delta\rho_a + \delta\rho_b\right) \frac{\partial W}{\partial x}(x_{a,0} - x_{b,0}) \\
=& -2c_s^2 0_{\mathrm{SPH}}^{(W_\mathrm{g})} - c_s^2 \frac{2}{\rho_{\mathrm{g},0}} \tilde{x}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \left[\underbrace{\sum_b m_b \frac{\partial^2 W}{\partial x^2}(x_{a,0} - x_{b,0})}_{\rho_{\mathrm{g},0} 0_{\mathrm{SPH}}^{(W'_\mathrm{g})}} - \underbrace{\sum_b m_b e^{-ik(x_{a,0} - x_{b,0})} \frac{\partial^2 W}{\partial x^2}(x_{a,0} - x_{b,0})}_{\rho_{\mathrm{g},0} \mathcal{F}_{\mathrm{SPH}}^{(W''_\mathrm{g})}}\right] \\
&\quad + c_s^2 \frac{1}{\rho_{\mathrm{g},0}^2} \tilde{\rho}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \left[\underbrace{\sum_b m_b \frac{\partial W}{\partial x}(x_{a,0} - x_{b,0})}_{\rho_{\mathrm{g},0} 0_{\mathrm{SPH}}^{(W_\mathrm{g})}} + \underbrace{\sum_b m_b e^{-ik(x_{a,0} - x_{b,0})} \frac{\partial W}{\partial x}(x_{a,0} - x_{b,0})}_{\rho_{\mathrm{g},0} \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}}\right] \\
=& -2c_s^2 0_{\mathrm{SPH}}^{(W_\mathrm{g})} - 2c_s^2 \tilde{x}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \left[0_{\mathrm{SPH}}^{(W'_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W''_\mathrm{g})}\right] + c_s^2 \frac{1}{\rho_{\mathrm{g},0}} \tilde{\rho}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \left[0_{\mathrm{SPH}}^{(W_\mathrm{g})} + \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}\right] \\
=& -2c_s^2 0_{\mathrm{SPH}}^{(W_\mathrm{g})} - 2c_s^2 \frac{1}{\rho_{\mathrm{g},0}\left[0_{\mathrm{SPH}}^{(W_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}\right]} \left[\tilde{\rho}_\mathrm{g} + \frac{0_{\mathrm{disc,g}}}{i\omega}\right] e^{i(kx_{a,0} - \omega t)} \left[0_{\mathrm{SPH}}^{(W'_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W''_\mathrm{g})}\right] \\
&\quad + c_s^2 \frac{1}{\rho_{\mathrm{g},0}} \tilde{\rho}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \left[0_{\mathrm{SPH}}^{(W_\mathrm{g})} + \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}\right] \\
=& -2c_s^2 0_{\mathrm{SPH}}^{(W_\mathrm{g})} - 2c_s^2 \tilde{x}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \left[0_{\mathrm{SPH}}^{(W'_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W''_\mathrm{g})}\right] + c_s^2 \frac{1}{\rho_{\mathrm{g},0}} \tilde{\rho}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \left[0_{\mathrm{SPH}}^{(W_\mathrm{g})} + \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}\right] \\
=& \underbrace{-2c_s^2 0_{\mathrm{SPH}}^{(W_\mathrm{g})} - \frac{2c_s^2}{\rho_{\mathrm{g},0}} \frac{0_{\mathrm{SPH}}^{(W'_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W''_\mathrm{g})}}{0_{\mathrm{SPH}}^{(W_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}} \frac{0_{\mathrm{disc,g}}}{i\omega} e^{i(kx_{a,0} - \omega t)}}_{0_{\mathrm{disc},\mathcal{P},\mathrm{g}}} - \frac{2c_s^2}{\rho_{\mathrm{g},0}} \frac{0_{\mathrm{SPH}}^{(W'_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W''_\mathrm{g})}}{0_{\mathrm{SPH}}^{(W_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}} \tilde{\rho}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \\
&\quad + c_s^2 \frac{1}{\rho_{\mathrm{g},0}} \tilde{\rho}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \left[0_{\mathrm{SPH}}^{(W_\mathrm{g})} + \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}\right] \\
=& 0_{\mathrm{disc},\mathcal{P},\mathrm{g}} + \frac{c_s^2}{\rho_{\mathrm{g},0}} \left[0_{\mathrm{SPH}}^{(W_\mathrm{g})} + \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}\right] \tilde{\rho}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} - \frac{2c_s^2}{\rho_{\mathrm{g},0}} \frac{0_{\mathrm{SPH}}^{(W'_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W''_\mathrm{g})}}{0_{\mathrm{SPH}}^{(W_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}} \tilde{\rho}_\mathrm{g} e^{i(kx_{a,0} - \omega t)} \\
=& 0_{\mathrm{disc},\mathcal{P},\mathrm{g}} + \frac{c_s^2}{\rho_{\mathrm{g},0}} \left(\left[0_{\mathrm{SPH}}^{(W_\mathrm{g})} + \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}\right] - 2\frac{0_{\mathrm{SPH}}^{(W'_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W''_\mathrm{g})}}{0_{\mathrm{SPH}}^{(W_\mathrm{g})} - \mathcal{F}_{\mathrm{SPH}}^{(W'_\mathrm{g})}}\right) \tilde{\rho}_\mathrm{g} e^{i(kx_{a,0} - \omega t)}
\end{aligned}
$$

4.3. Drag term

$$
\begin{aligned}
S_{a,i} = \frac{\partial v_a}{\partial x_a} &= \frac{-\sum_b m_b v_{ab} \frac{\partial W_{ab}}{\partial x}(h_a)}{\sum_b m_b (r_b - r_a) \frac{\partial W_{ab}}{\partial x}(h_a)} \\
&= -\frac{1}{\mathbb{1}_{\mathrm{SPH}}^{(W_a')} \rho_{a,0}} \sum_b m_b \left(\delta v_a - \delta v_b\right) \frac{\partial W_{ab,0}}{\partial x} \\
&= -\frac{e^{i(k x_{a,0} - \omega t)}}{\mathbb{1}_{\mathrm{SPH}}^{(W_a')} \rho_{a,0}} \tilde{v}_a \sum_b m_b \left(1 - e^{-ik(x_{a,0} - x_{b,0})}\right) \frac{\partial W_{ab,0}}{\partial x}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{D}_{a,\mathrm{g}} =& \sum_i m_i \frac{\left(\mathbf{v}_{ai}^* \cdot \hat{\mathbf{r}}_{ai}\right) \hat{\mathbf{r}}_{ai}}{\left(\rho_a + \rho_i\right) t_{ai}^{\mathrm{s}}} D_{ai}(h) \\
=& \sum_i m_i \frac{1}{\left(\rho_a + \rho_i\right) t_{ai}^{\mathrm{s}}} v_{ai} D_{ai}(h) - \sum_i \frac{m_i \mu}{\left(\rho_a + \rho_i\right) t_{\mathrm{s}}} \left(S_{ai} + S_{ia}\right) |r_{ai}| \hat{r}_{ai} D_{ai}(h) \\
=& \frac{1}{t_{\mathrm{s}} \left(\rho_{\mathrm{g},0} + \rho_{\mathrm{d},0}\right)} e^{i(k x_{a,0} - \omega t)} \sum_i m_i \left(\tilde{v}_{\mathrm{g}} - \tilde{v}_{\mathrm{d}} e^{-ik(x_{a,0} - x_{i,0})}\right) D_{ai,0}(h) + \frac{\mu}{\left(\rho_{\mathrm{g},0} + \rho_{\mathrm{d},0}\right) t^{\mathrm{s}}} \sum_i m_i r_{ai,0} D_{ai,0}(h) \times \\
& \Bigg[\frac{1}{1_{\mathrm{SPH}}^{(W_{\mathrm{g}})} \rho_{\mathrm{g},0}} e^{i(k x_{a,0} - \omega t)} \tilde{v}_{\mathrm{g}} \sum_b m_b \left(1 - e^{-ik(x_{a,0} - x_{b,0})}\right) \frac{\mathrm{d}W_{ab}}{\mathrm{d}x}(h_a) \\
& \qquad + \frac{1}{1_{\mathrm{SPH}}^{(W_{\mathrm{d}})} \rho_{\mathrm{d},0}} e^{i(k x_{i,0} - \omega t)} \tilde{v}_{\mathrm{d}} \sum_j m_j \left(1 - e^{-ik(x_{i,0} - x_{j,0})}\right) \frac{\mathrm{d}W_{ij}}{\mathrm{d}x}(h_i) \Bigg] \\
=& \frac{1}{t_{\mathrm{s}} \left(\rho_{\mathrm{g},0} + \rho_{\mathrm{d},0}\right)} e^{i(k x_{a,0} - \omega t)} \Bigg(\tilde{v}_{\mathrm{g}} \underbrace{\sum_i m_i D_{ai,0}(h)}_{\rho_{\mathrm{d},0} \mathbb{1}_{\mathrm{SPH,d}}^{(D)}} - \tilde{v}_{\mathrm{d}} \underbrace{\sum_i m_i e^{-ik(x_{a,0} - x_{i,0})} D_{ai,0}(h)}_{\rho_{\mathrm{d},0} \mathcal{F}_{\mathrm{SPH,d}}^{(D)}} \Bigg) \\
& \qquad + \frac{\mu}{\left(\rho_{\mathrm{g},0} + \rho_{\mathrm{d},0}\right) t^{\mathrm{s}}} \sum_i m_i r_{ai,0} D_{ai,0}(h) \times \\
& \qquad \Bigg[\frac{1}{1_{\mathrm{SPH}}^{(W_{\mathrm{g}})} \rho_{\mathrm{g},0}} e^{i(k x_{a,0} - \omega t)} \tilde{v}_{\mathrm{g}} \Bigg(\underbrace{\sum_b m_b \frac{\mathrm{d}W_{ab}}{\mathrm{d}x}(h_a)}_{\rho_{\mathrm{g},0} 0_{\mathrm{SPH}}^{(W_{\mathrm{g}})}} - \underbrace{\sum_b m_b e^{-ik(x_{a,0} - x_{b,0})} \frac{\mathrm{d}W_{ab}}{\mathrm{d}x}(h_a)}_{\rho_{\mathrm{g},0} \mathcal{F}_{\mathrm{SPH}}^{(W_{\mathrm{g}}')}} \Bigg) \\
& \qquad + \frac{1}{1_{\mathrm{SPH}}^{(W_{\mathrm{d}})} \rho_{\mathrm{d},0}} e^{i(k x_{i,0} - \omega t)} \tilde{v}_{\mathrm{d}} \Bigg(\underbrace{\sum_j m_j \frac{\mathrm{d}W_{ij}}{\mathrm{d}x}(h_i)}_{\rho_{\mathrm{d},0} 0_{\mathrm{SPH}}^{(W_{\mathrm{d}})}} - \underbrace{\sum_j m_j e^{-ik(x_{i,0} - x_{j,0})} \frac{\mathrm{d}W_{ij}}{\mathrm{d}x}(h_i)}_{\rho_{\mathrm{d},0} \mathcal{F}_{\mathrm{SPH}}^{(W_{\mathrm{d}}')}} \Bigg) \Bigg] \\
=& \frac{\rho_{\mathrm{d},0}}{t_{\mathrm{s}} \left(\rho_{\mathrm{g},0} + \rho_{\mathrm{d},0}\right)} e^{i(k x_{a,0} - \omega t)} \left(\tilde{v}_{\mathrm{g}} \mathbb{1}_{\mathrm{SPH,d}}^{(D)} - \tilde{v}_{\mathrm{d}} \mathcal{F}_{\mathrm{SPH,d}}^{(D)} \right) + \frac{\mu}{\left(\rho_{\mathrm{g},0} + \rho_{\mathrm{d},0}\right) t^{\mathrm{s}}} \sum_i m_i r_{ai,0} D_{ai,0}(h) \times \\
& \left[\frac{1}{1_{\mathrm{SPH}}^{(W_{\mathrm{g}})}} e^{i(k x_{a,0} - \omega t)} \tilde{v}_{\mathrm{g}} \left(0_{\mathrm{SPH}}^{(W_{\mathrm{g}})} - \mathcal{F}_{\mathrm{SPH}}^{(W_{\mathrm{g}}')} \right) + \frac{1}{1_{\mathrm{SPH}}^{(W_{\mathrm{d}})}} e^{i(k x_{i,0} - \omega t)} \tilde{v}_{\mathrm{d}} \left(0_{\mathrm{SPH}}^{(W_{\mathrm{d}})} - \mathcal{F}_{\mathrm{SPH}}^{(W_{\mathrm{d}}')} \right) \right]
\end{aligned}
$$

$$
\begin{aligned}
=&\frac{\rho_{\mathrm{d},0}}{t_{\mathrm{s}}\left(\rho_{\mathrm{g},0}+\rho_{\mathrm{d},0}\right)}e^{i(kx_{a,0}-\omega t)}\left(\tilde{v}_{\mathrm{g}}\mathbb{1}^{(D)}_{\mathrm{SPH,d}}-\tilde{v}_{\mathrm{d}}\mathcal{F}^{(D)}_{\mathrm{SPH,d}}\right)+\\
&\left[\frac{\mu}{\left(\rho_{\mathrm{g},0}+\rho_{\mathrm{d},0}\right)t^{\mathrm{s}}}\frac{1}{1^{(W_{\mathrm{g}})}_{\mathrm{SPH}}}e^{i(kx_{a,0}-\omega t)}\tilde{v}_{\mathrm{g}}\left(0^{(W_{\mathrm{g}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}\right)\underbrace{\sum_i m_i r_{ai,0}D_{ai,0}(h)}_{\rho_{\mathrm{d},0}0^{(D)}_{1,\mathrm{SPH,d}}}\right.\\
&\left.+\frac{\mu}{\left(\rho_{\mathrm{g},0}+\rho_{\mathrm{d},0}\right)t^{\mathrm{s}}}\frac{1}{1^{(W_{\mathrm{d}})}_{\mathrm{SPH}}}\tilde{v}_{\mathrm{d}}\left(0^{(W_{\mathrm{d}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}}\right)\sum_i m_i r_{ai,0}D_{ai,0}(h)e^{i(kx_{i,0}-\omega t)}\right]\\
=&\frac{\rho_{\mathrm{d},0}}{t_{\mathrm{s}}\left(\rho_{\mathrm{g},0}+\rho_{\mathrm{d},0}\right)}e^{i(kx_{a,0}-\omega t)}\left(\tilde{v}_{\mathrm{g}}\mathbb{1}^{(D)}_{\mathrm{SPH,d}}-\tilde{v}_{\mathrm{d}}\mathcal{F}^{(D)}_{\mathrm{SPH,d}}\right)+\\
&\left[\frac{\mu\rho_{\mathrm{d},0}}{\left(\rho_{\mathrm{g},0}+\rho_{\mathrm{d},0}\right)t^{\mathrm{s}}}\frac{1}{1^{(W_{\mathrm{g}})}_{\mathrm{SPH}}}e^{i(kx_{a,0}-\omega t)}\tilde{v}_{\mathrm{g}}\left(0^{(W_{\mathrm{g}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}\right)0^{(D)}_{1,\mathrm{SPH,d}}\right.\\
&\left.+\frac{\mu}{\left(\rho_{\mathrm{g},0}+\rho_{\mathrm{d},0}\right)t^{\mathrm{s}}}\frac{1}{1^{(W_{\mathrm{d}})}_{\mathrm{SPH}}}\tilde{v}_{\mathrm{d}}\left(0^{(W_{\mathrm{d}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}}\right)e^{i(kx_{a,0}-\omega t)}\underbrace{\sum_i m_i r_{ai,0}D_{ai,0}(h)e^{-ik(x_{a,0}-x_{i,0})}}_{i\rho_{\mathrm{d},0}\partial_k\mathcal{F}^{(D)}_{\mathrm{SPH,d}}}\right]\\
=&\frac{\rho_{\mathrm{d},0}}{t_{\mathrm{s}}\left(\rho_{\mathrm{g},0}+\rho_{\mathrm{d},0}\right)}e^{i(kx_{a,0}-\omega t)}\left(\tilde{v}_{\mathrm{g}}\mathbb{1}^{(D)}_{\mathrm{SPH,d}}-\tilde{v}_{\mathrm{d}}\mathcal{F}^{(D)}_{\mathrm{SPH,d}}\right)+\\
&\left[\frac{\mu\rho_{\mathrm{d},0}}{\left(\rho_{\mathrm{g},0}+\rho_{\mathrm{d},0}\right)t^{\mathrm{s}}}\frac{1}{1^{(W_{\mathrm{g}})}_{\mathrm{SPH}}}e^{i(kx_{a,0}-\omega t)}\tilde{v}_{\mathrm{g}}\left(0^{(W_{\mathrm{g}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}\right)0^{(D)}_{1,\mathrm{SPH,d}}+\frac{\mu\rho_{\mathrm{d},0}}{\left(\rho_{\mathrm{g},0}+\rho_{\mathrm{d},0}\right)t^{\mathrm{s}}}\frac{1}{1^{(W_{\mathrm{d}})}_{\mathrm{SPH}}}\tilde{v}_{\mathrm{d}}\left(0^{(W_{\mathrm{d}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}}\right)e^{i(kx_{a,0}-\omega t)}i\partial_k\mathcal{F}^{(D}_{\mathrm{SP}}\right.\\
=&e^{i(kx_{a,0}-\omega t)}\left[\frac{1}{t_{\mathrm{g}}}\tilde{v}_{\mathrm{g}}\underbrace{\left(\mathbb{1}^{(D)}_{\mathrm{SPH,d}}+\mu\frac{1}{1^{(W_{\mathrm{g}})}_{\mathrm{SPH}}}\left(0^{(W_{\mathrm{g}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}\right)0^{(D)}_{1,\mathrm{SPH,d}}\right)}_{1_{\mathrm{d}\to\mathrm{g,g}}}-\frac{1}{t_{\mathrm{g}}}\tilde{v}_{\mathrm{d}}\underbrace{\left(\mathcal{F}^{(D)}_{\mathrm{SPH,d}}-\mu\frac{1}{1^{(W_{\mathrm{d}})}_{\mathrm{SPH}}}\left(0^{(W_{\mathrm{d}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}}\right)i\partial_k\mathcal{F}^{(D)}_{\mathrm{SPH,d}}\right)}_{1_{\mathrm{d}\to\mathrm{g,d}}}\right]\\
=&e^{i(kx_{a,0}-\omega t)}\left[\frac{1}{t_{\mathrm{g}}}\tilde{v}_{\mathrm{g}}1_{\mathrm{d}\to\mathrm{g,g}}-\frac{1}{t_{\mathrm{g}}}\tilde{v}_{\mathrm{d}}1_{\mathrm{d}\to\mathrm{g,d}}\right]
\end{aligned}
$$

4.4. Discrete sph equations

$$\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t}=-\mathcal{D}_{a,\mathrm{g}}-\mathcal{P}_a \tag{B.46}$$

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t}=+\mathcal{D}_{i,\mathrm{d}} \tag{B.47}$$

$$-i\omega\tilde{\rho}_{\mathrm{g/d}}=0_{\mathrm{disc,g/d}}+\tilde{v}_{\mathrm{g/d}}\rho_{\mathrm{g/d},0}\left[0^{(W_{\mathrm{g/d}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{g/d}})}_{\mathrm{SPH}}\right] \tag{B.48}$$

$$\mathcal{P}=0_{\mathrm{disc},\mathcal{P},\mathrm{g}}+\frac{c_s^2}{\rho_{\mathrm{g},0}}\left(\left[0^{(W_{\mathrm{g}})}_{\mathrm{SPH}}+\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}\right]-2\frac{0^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}-\mathcal{F}^{(W''_{\mathrm{g}})}_{\mathrm{SPH}}}{0^{(W_{\mathrm{g}})}_{\mathrm{SPH}}-\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}}\right)\tilde{\rho}_{\mathrm{g}}e^{i(kx_{a,0}-\omega t)} \tag{B.49}$$

$$\mathcal{D}_{a,\mathrm{g}}=e^{i(kx_{a,0}-\omega t)}\left[\frac{1}{t_{\mathrm{g}}}\tilde{v}_{\mathrm{g}}1_{\mathrm{d}\to\mathrm{g,g}}-\frac{1}{t_{\mathrm{g}}}\tilde{v}_{\mathrm{d}}1_{\mathrm{d}\to\mathrm{g,d}}\right] \tag{B.50}$$

$$\mathcal{D}_{i,\mathrm{g}}=e^{i(kx_{i,0}-\omega t)}\left[\frac{1}{t_{\mathrm{d}}}\tilde{v}_{\mathrm{g}}1_{\mathrm{g}\to\mathrm{d,g}}-\frac{1}{t_{\mathrm{d}}}\tilde{v}_{\mathrm{d}}1_{\mathrm{g}\to\mathrm{d,d}}\right] \tag{B.51}$$

5. Discrete dispersion relation

Neglecting the offset term $0_{\mathrm{disc},\mathcal{P},\mathrm{g}}$ we get the following matrix to compute the dispersion relation

$$\begin{vmatrix} -i\omega & 0 & \rho_{\mathrm{g},0}\left[\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}} - 0^{(W_{\mathrm{g}})}_{\mathrm{SPH}}\right] & 0 \\ 0 & -i\omega & 0 & \rho_{\mathrm{d},0}\left[\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}} - 0^{(W_{\mathrm{d}})}_{\mathrm{SPH}}\right] \\ c_s^2 \frac{1}{\rho_{\mathrm{g},0}}\left[0^{(W_{\mathrm{g}})}_{\mathrm{SPH}} + \mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}\right] & 0 & -i\omega + \frac{1_{\mathrm{d}\to\mathrm{g},\mathrm{g}}}{t_{\mathrm{g}}} - 2c_s^2 \frac{1}{-i\omega}\left[0^{(W'_{\mathrm{g}})}_{\mathrm{SPH}} - \mathcal{F}^{(W''_{\mathrm{g}})}_{\mathrm{SPH}}\right] & -\frac{1_{\mathrm{d}\to\mathrm{g},\mathrm{d}}}{t_{\mathrm{g}}} \\ 0 & 0 & -\frac{1_{\mathrm{g}\to\mathrm{d},\mathrm{g}}}{t_{\mathrm{d}}} & -i\omega + \frac{1_{\mathrm{g}\to\mathrm{d},\mathrm{g}}}{t_{\mathrm{d}}} \end{vmatrix} = 0 \tag{B.52}$$

This yield the following dispersion relation for the SPH dustywave in the discrete case

$$\begin{aligned} 0 = & -i\left(\mathcal{F}^{(W'_{\mathrm{g}})^2}_{\mathrm{SPH}} - 2\mathcal{F}^{(W''_{\mathrm{g}})}_{\mathrm{SPH}}\right)\left(\mathbb{1}^{(D)}_{\mathrm{SPH,g}} - \mu\frac{\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}}}{1^{(W_{\mathrm{d}})}_{\mathrm{SPH}}} 0^{(D)}_{1,\mathrm{SPH,g}}\right)(-1+\epsilon) \\ & + \left(\mathcal{F}^{(W'_{\mathrm{g}})^2}_{\mathrm{SPH}} - 2\mathcal{F}^{(W''_{\mathrm{g}})}_{\mathrm{SPH}} + \left[\left(\mathbb{1}^{(D)}_{\mathrm{SPH,d}} - \mu\frac{\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}}{1^{(W_{\mathrm{g}})}_{\mathrm{SPH}}} 0^{(D)}_{1,\mathrm{SPH,d}}\right)\left(\mathbb{1}^{(D)}_{\mathrm{SPH,g}} - \mu\frac{\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}}}{1^{(W_{\mathrm{d}})}_{\mathrm{SPH}}} 0^{(D)}_{1,\mathrm{SPH,g}}\right)\right. \right. \\ & \left. \left. - \left(\mathcal{F}^{(D)}_{\mathrm{SPH,d}} + \mu\frac{\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}}}{1^{(W_{\mathrm{d}})}_{\mathrm{SPH}}} i\partial_k \mathcal{F}^{(D)}_{\mathrm{SPH,d}}\right)\left(\mathcal{F}^{(D)}_{\mathrm{SPH,g}} + \mu\frac{\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}}{1^{(W_{\mathrm{g}})}_{\mathrm{SPH}}} i\partial_k \mathcal{F}^{(D)}_{\mathrm{SPH,g}}\right)\right](-1+\epsilon)\epsilon\right)\omega \\ & + i\left[\mathbb{1}^{(D)}_{\mathrm{SPH,g}} - \mu\frac{\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}}}{1^{(W_{\mathrm{d}})}_{\mathrm{SPH}}} 0^{(D)}_{1,\mathrm{SPH,g}} + \left(\frac{\mathcal{F}^{(W'_{\mathrm{d}})}_{\mathrm{SPH}}}{1^{(W_{\mathrm{d}})}_{\mathrm{SPH}}} 0^{(D)}_{1,\mathrm{SPH,g}} - \frac{\mathcal{F}^{(W'_{\mathrm{g}})}_{\mathrm{SPH}}}{1^{(W_{\mathrm{g}})}_{\mathrm{SPH}}} 0^{(D)}_{1,\mathrm{SPH,d}}\right)\mu\epsilon + \epsilon\left(\mathbb{1}^{(D)}_{\mathrm{SPH,d}} - \mathbb{1}^{(D)}_{\mathrm{SPH,g}}\right)\right]\omega^2 \\ & + \omega^3 \end{aligned} \tag{B.53}$$

References

David-Cléris T., Laibe G., 2021, *Large dust fractions can prevent the propagation of soundwaves*, MNRAS, 504, 2889-2894

Garaud P., Barrière-Fouchet L., Lin D. N. C., 2004, *Individual and Average Behavior of Particles in a Protoplanetary Nebula*, ApJ, 603, 292-306

Price D. J., Laibe G., 2020, *A solution to the overdamping problem when simulating dust-gas mixtures with smoothed particle hydrodynamics*, MNRAS, 495, 3929-3934

Price D. J., et al., 2018, *Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics*, PASA, 35, e031

Appendix C

Full-Monofluid formalism

Contents

Foreword

This project, which was started at the beginning of the Ph.D. thesis, aims at generalizing the dust-gas monofluid approach to large grains (St $\gtrsim 1$), for which the terminal velocity approximation cannot be used. Indeed, the current formulation proposed by Hutchison (2017) has been found to conserve energy to machine precision as it should, but not dust momentum. Previously, the scheme was built on the energy conservation equation. In this work, we derive the full-monofluid formalism from dust momentum conservation to ensure its conservation in the resulting scheme. We first present the monofluid formalism without terminal velocity approximation. We then proceed with developing the SPH operators for the full monofluid formalism.

1. Monofluid formalism

In order to derive the monofluid equation of motion, we first need to rewrite the dust or gas quantities in terms of mixture quantities.

$$\mathbf{v}_\mathrm{g} = \mathbf{v} - \frac{\rho_\mathrm{d}}{\rho}\Delta\mathbf{v} = \mathbf{v} - \epsilon\Delta\mathbf{v}, \tag{C.1}$$

$$\mathbf{v}_\mathrm{d} = \mathbf{v} + \frac{\rho_\mathrm{g}}{\rho}\Delta\mathbf{v} = \mathbf{v} + (1-\epsilon)\Delta\mathbf{v}, \tag{C.2}$$

$$\rho_\mathrm{g} = (1-\epsilon)\rho, \tag{C.3}$$

$$\rho_\mathrm{d} = \epsilon\rho. \tag{C.4}$$

The idea is to rewrite the Euler equations in terms of those mixed quantities to reformulate the dust-gas mixture as one fluid with internal quantities.

Proposition ▶ Monofluid conservation of total mass

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

Proof. We start by reformulating the two-fluid Euler equations (Eq. 1.18-1.19) in the one fluid formalism.

$$0 = \frac{\partial \rho_{\mathrm{g}}}{\partial t} + \nabla \cdot (\rho_{\mathrm{g}} \mathbf{v}_{\mathrm{g}}) \tag{C.5}$$

$$= \frac{\partial (1-\epsilon)\rho}{\partial t} + \nabla \cdot \left[(1-\epsilon)\,\rho\,(\mathbf{v} - \epsilon \Delta \mathbf{v})\right] \tag{C.6}$$

$$= -\frac{\partial \epsilon}{\partial t}\rho + (1-\epsilon)\frac{\partial \rho}{\partial t} + \nabla \cdot \left[(1-\epsilon)\rho \mathbf{v}\right] - \nabla \cdot \left[(1-\epsilon)\rho\epsilon\Delta \mathbf{v}\right], \tag{C.7}$$

$$0 = \frac{\partial \rho_{\mathrm{d}}}{\partial t} + \nabla \cdot (\rho_{\mathrm{d}} \mathbf{v}_{\mathrm{d}}) \tag{C.8}$$

$$= \frac{\partial \epsilon\rho}{\partial t} + \nabla \cdot \left[\epsilon\rho\,(\mathbf{v} + (1-\epsilon)\Delta \mathbf{v})\right] \tag{C.9}$$

$$= \frac{\partial \epsilon}{\partial t}\rho + \epsilon\frac{\partial \rho}{\partial t} + \nabla \cdot \left[\epsilon\rho \mathbf{v}\right] + \nabla \cdot \left[(1-\epsilon)\rho\epsilon\Delta \mathbf{v}\right]. \tag{C.10}$$

Taking the sum (C.7) + (C.10), give the total mass conservation equation for the monofluid formalism :

$$\boxed{0 = \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v})}$$

□

Proposition ▶ Monofluid conservation of dust mass

$$\left(\frac{\partial}{\partial t} + \mathbf{v} \cdot \nabla\right)\epsilon + \frac{1}{\rho}\nabla \cdot \left[(1-\epsilon)\epsilon\rho\Delta \mathbf{v}\right] = 0$$

Proof. Let us take the difference (C.10) - (C.7) :

$$0 = 2\frac{\partial \epsilon}{\partial t}\rho + 2\epsilon\frac{\partial \rho}{\partial t} + 2\nabla\cdot[\epsilon\rho\mathbf{v}] + 2\nabla\cdot[(1-\epsilon)\epsilon\rho\Delta\mathbf{v}] \tag{C.11}$$

$$0 = \frac{\partial \epsilon}{\partial t}\rho + \epsilon\frac{\partial \rho}{\partial t} + \nabla\cdot[\epsilon\rho\mathbf{v}] + \nabla\cdot[(1-\epsilon)\epsilon\rho\Delta\mathbf{v}] \tag{C.12}$$

$$0 = \frac{\partial \epsilon}{\partial t}\rho + \underbrace{\epsilon\frac{\partial \rho}{\partial t} + \epsilon\nabla\cdot[\rho\mathbf{v}]}_{\text{mass conservation}=0} + \rho\mathbf{v}\cdot\nabla\epsilon + \nabla\cdot[(1-\epsilon)\epsilon\rho\Delta\mathbf{v}] \tag{C.13}$$

$$0 = \frac{\partial \epsilon}{\partial t} + \mathbf{v}\cdot\nabla\epsilon + \frac{1}{\rho}\nabla\cdot[(1-\epsilon)\epsilon\rho\Delta\mathbf{v}] \tag{C.14}$$

$$0 = \frac{\partial \epsilon}{\partial t} + \mathbf{v}\cdot\nabla\epsilon + \frac{1}{\rho}\nabla\cdot[(1-\epsilon)\epsilon\rho\Delta\mathbf{v}]\,. \tag{C.15}$$

Hence, the advection equation for the dust fraction is

$$\boxed{\left(\frac{\partial}{\partial t} + \mathbf{v}\cdot\nabla\right)\epsilon + \frac{1}{\rho}\nabla\cdot[(1-\epsilon)\epsilon\rho\Delta\mathbf{v}] = 0.}$$

□

Definition ▶ Barycentric forces

$$\mathbf{f} = \frac{\rho_\mathrm{g}\mathbf{f}_\mathrm{g} + \rho_\mathrm{d}\mathbf{f}_\mathrm{d}}{\rho}$$

Definition ▶ Differential forces

$$\Delta\mathbf{f} = \mathbf{f}_\mathrm{d} - \mathbf{f}_\mathrm{g}$$

Proposition ▶ Monofluid barycentric velocity

$$\left(\frac{\partial}{\partial t} + \mathbf{v}\cdot\nabla\right)\mathbf{v} = \mathbf{f} - \frac{\nabla P}{\rho} - \frac{1}{\rho}\nabla\cdot[\rho(1-\epsilon)\epsilon\Delta\mathbf{v}\Delta\mathbf{v}]$$

Proof. For the momentum equations of dust and gas, the computation of their monofluid counterpart is easier if we start from the conservative form of the momentum conservation equations of gas and dust.

$$\frac{\partial \rho_\mathrm{g}\mathbf{v}_\mathrm{g}}{\partial t} + \nabla\cdot[\rho_\mathrm{g}\mathbf{v}_\mathrm{g}\mathbf{v}_\mathrm{g}] = \rho_\mathrm{g}\mathbf{f}_\mathrm{g} - \nabla P + K\left(\mathbf{v}_\mathrm{d} - \mathbf{v}_\mathrm{g}\right), \tag{C.16}$$

$$\frac{\partial \rho_\mathrm{d}\mathbf{v}_\mathrm{d}}{\partial t} + \nabla\cdot[\rho_\mathrm{d}\mathbf{v}_\mathrm{d}\mathbf{v}_\mathrm{d}] = \rho_\mathrm{d}\mathbf{f}_\mathrm{d} - K\left(\mathbf{v}_\mathrm{d} - \mathbf{v}_\mathrm{g}\right). \tag{C.17}$$

We can find the equation for the velocity of the mixture by summing C.16 and C.17:

$$\frac{\partial}{\partial t}\left(\rho_{\mathrm{g}}\mathbf{v}_{\mathrm{g}}+\rho_{\mathrm{d}}\mathbf{v}_{\mathrm{d}}\right)+\nabla\cdot\left[\rho_{\mathrm{g}}\mathbf{v}_{\mathrm{g}}\mathbf{v}_{\mathrm{g}}+\rho_{\mathrm{d}}\mathbf{v}_{\mathrm{d}}\mathbf{v}_{\mathrm{d}}\right]=\rho\mathbf{f}-\nabla P. \tag{C.18}$$

Now we can replace every quantity with their monofluid counterpart

$$\frac{\partial}{\partial t}\left(\rho\mathbf{v}\right)+\nabla\cdot\left[\rho_{\mathrm{g}}\mathbf{v}_{\mathrm{g}}\mathbf{v}_{\mathrm{g}}+\rho_{\mathrm{d}}\mathbf{v}_{\mathrm{d}}\mathbf{v}_{\mathrm{d}}\right]=\rho\mathbf{f}-\nabla P, \tag{C.19}$$

$$\frac{\partial}{\partial t}\left(\rho\mathbf{v}\right)+\nabla\cdot\left[(1-\epsilon)\rho\left(\mathbf{v}-\epsilon\Delta\mathbf{v}\right)\left(\mathbf{v}-\epsilon\Delta\mathbf{v}\right)+\epsilon\rho\left(\mathbf{v}+(1-\epsilon)\Delta\mathbf{v}\right)\left(\mathbf{v}+(1-\epsilon)\Delta\mathbf{v}\right)\right]=\rho\mathbf{f}-\nabla P, \tag{C.20}$$

$$\frac{\partial}{\partial t}\left(\rho\mathbf{v}\right)+\nabla\cdot\left[\rho\mathbf{v}\mathbf{v}\right]+\nabla\cdot\left[\rho(1-\epsilon)\epsilon^{2}\Delta\mathbf{v}\Delta\mathbf{v}+\rho(1-\epsilon)^{2}\epsilon\Delta\mathbf{v}\Delta\mathbf{v}\right]=\rho\mathbf{f}-\nabla P, \tag{C.21}$$

$$\frac{\partial}{\partial t}\left(\rho\mathbf{v}\right)+\nabla\cdot\left[\rho\mathbf{v}\mathbf{v}\right]=\rho\mathbf{f}-\nabla P-\nabla\cdot\left[\rho(1-\epsilon)\epsilon\Delta\mathbf{v}\Delta\mathbf{v}\right], \tag{C.22}$$

$$\left(\frac{\partial}{\partial t}+\mathbf{v}\cdot\nabla\right)\mathbf{v}=\mathbf{f}-\frac{\nabla P}{\rho}-\frac{1}{\rho}\nabla\cdot\left[\rho(1-\epsilon)\epsilon\Delta\mathbf{v}\Delta\mathbf{v}\right], \tag{C.23}$$

giving us the equation of motion for the velocity of the mixture

$$\boxed{\left(\frac{\partial}{\partial t}+\mathbf{v}\cdot\nabla\right)\mathbf{v}=\mathbf{f}-\frac{\nabla P}{\rho}-\frac{1}{\rho}\nabla\cdot\left[\rho(1-\epsilon)\epsilon\Delta\mathbf{v}\Delta\mathbf{v}\right].}$$

□

Proposition ▶ Monofluid differential velocity

$$\left(\frac{\partial}{\partial t}+\mathbf{v}\cdot\nabla\right)\Delta\mathbf{v}=-\frac{\Delta\mathbf{v}}{t_{\mathrm{s}}}+\Delta\mathbf{f}-\left(\Delta\mathbf{v}\cdot\nabla\right)\mathbf{v}+\left(2\epsilon-1\right)\left(\Delta\mathbf{v}\cdot\nabla\right)\Delta\mathbf{v}+\Delta\mathbf{v}\left(\Delta\mathbf{v}\cdot\nabla\right)\epsilon$$

Proof. For the $\Delta\mathbf{v}$ equation, we can use the difference $\frac{\text{C.17}}{\rho_{\mathrm{d}}}-\frac{\text{C.16}}{\rho_{\mathrm{g}}}$,

$$\frac{1}{\rho_{\mathrm{d}}}\partial_t\left(\rho_{\mathrm{d}}\mathbf{v}_{\mathrm{d}}\right)-\frac{1}{\rho_{\mathrm{g}}}\partial_t\left(\rho_{\mathrm{g}}\mathbf{v}_{\mathrm{g}}\right)+\frac{1}{\rho_{\mathrm{d}}}\nabla\cdot\left(\rho_{\mathrm{d}}\mathbf{v}_{\mathrm{d}}\mathbf{v}_{\mathrm{d}}\right)-\frac{1}{\rho_{\mathrm{g}}}\nabla\cdot\left(\rho_{\mathrm{g}}\mathbf{v}_{\mathrm{g}}\mathbf{v}_{\mathrm{g}}\right)=\Delta\mathbf{f}-K\Delta v\left(\frac{1}{\rho_{\mathrm{g}}}+\frac{1}{\rho_{\mathrm{d}}}\right)+\frac{\nabla P_{\mathrm{g}}}{\rho_{\mathrm{g}}},$$

$$\frac{\partial}{\partial t}\Delta\mathbf{v}=\Delta\mathbf{f}-\frac{\Delta\mathbf{v}}{t_{\mathrm{s}}}+\frac{\nabla P_{\mathrm{g}}}{\rho_{\mathrm{g}}}+\left(\mathbf{v}_{\mathrm{g}}\cdot\nabla\right)\mathbf{v}_{\mathrm{g}}-\left(\mathbf{v}_{\mathrm{d}}\cdot\nabla\right)\mathbf{v}_{\mathrm{d}}.$$

We can now develop the advection terms for gas and dust,

$$\begin{aligned}\left(\mathbf{v}_{\mathrm{d}}\cdot\nabla\right)\mathbf{v}_{\mathrm{d}}&=\mathbf{v}\cdot\nabla\mathbf{v}+(1-\epsilon)\Delta\mathbf{v}\cdot\nabla\left[(1-\epsilon)\Delta\mathbf{v}\right]\ +\\&\quad\ \mathbf{v}\cdot\nabla\left[(1-\epsilon)\Delta\mathbf{v}\right]\ +(1-\epsilon)\Delta\mathbf{v}\cdot\nabla\mathbf{v},\\\left(\mathbf{v}_{\mathrm{g}}\cdot\nabla\right)\mathbf{v}_{\mathrm{g}}&=\mathbf{v}\cdot\nabla\mathbf{v}+\ \ \epsilon\Delta\mathbf{v}\cdot\nabla\left[\ \ \epsilon\ \ \Delta\mathbf{v}\right]\ -\\&\quad\ \mathbf{v}\cdot\nabla\left[\ \ \epsilon\ \ \Delta\mathbf{v}\right]\ -\ \ \epsilon\ \ \Delta\mathbf{v}\cdot\nabla\mathbf{v}.\end{aligned}$$

Taking the difference yield,

$$(\mathbf{v}_\mathrm{g}\cdot\nabla)\,\mathbf{v}_\mathrm{g}-(\mathbf{v}_\mathrm{d}\cdot\nabla)\,\mathbf{v}_\mathrm{d}=\epsilon\Delta\mathbf{v}\cdot\nabla\left[\epsilon\Delta\mathbf{v}\right]-(1-\epsilon)\Delta\mathbf{v}\cdot\nabla\left[(1-\epsilon)\Delta\mathbf{v}\right]\\-(\mathbf{v}\cdot\nabla)\,\Delta\mathbf{v}-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}.$$

Hence, the result,

$$\left(\frac{\partial}{\partial t}+\mathbf{v}\cdot\nabla\right)\Delta\mathbf{v}=-\frac{\Delta\mathbf{v}}{t_\mathrm{s}}+\Delta\mathbf{f}-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}\\+\epsilon\Delta\mathbf{v}\cdot\nabla\left[\epsilon\Delta\mathbf{v}\right]-(1-\epsilon)\Delta\mathbf{v}\cdot\nabla\left[(1-\epsilon)\Delta\mathbf{v}\right].$$

We can further transform this expression as follows:

$$\begin{aligned}
f_{I_\Delta v}&=-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}+\epsilon\Delta\mathbf{v}\cdot\nabla\left[\epsilon\Delta\mathbf{v}\right]-(1-\epsilon)\Delta\mathbf{v}\cdot\nabla\left[(1-\epsilon)\Delta\mathbf{v}\right]\\
&=-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}+\epsilon\Delta\mathbf{v}\cdot\nabla\left[\epsilon\Delta\mathbf{v}\right]-\Delta\mathbf{v}\cdot\nabla\left[(1-\epsilon)\Delta\mathbf{v}\right]+\epsilon\Delta\mathbf{v}\cdot\nabla\left[(1-\epsilon)\Delta\mathbf{v}\right]\\
&=-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}-\Delta\mathbf{v}\cdot\nabla\left[\Delta\mathbf{v}\right]+\Delta\mathbf{v}\cdot\nabla\left[\epsilon\Delta\mathbf{v}\right]+\epsilon\Delta\mathbf{v}\cdot\nabla\left[\Delta\mathbf{v}\right]\\
&=-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}-\Delta\mathbf{v}\cdot\nabla\left[\Delta\mathbf{v}\right]+\epsilon\Delta\mathbf{v}\cdot\nabla\left[\Delta\mathbf{v}\right]+\Delta\mathbf{v}\cdot\nabla\left[\epsilon\right]\Delta\mathbf{v}+\epsilon\Delta\mathbf{v}\cdot\nabla\left[\Delta\mathbf{v}\right]\\
&=-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}-\Delta\mathbf{v}\cdot\nabla\left[\Delta\mathbf{v}\right]+2\epsilon\Delta\mathbf{v}\cdot\nabla\left[\Delta\mathbf{v}\right]+\Delta\mathbf{v}\cdot\nabla\left[\epsilon\right]\Delta\mathbf{v}\\
&=-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}+(2\epsilon-1)\Delta\mathbf{v}\cdot\nabla\left[\Delta\mathbf{v}\right]+\Delta\mathbf{v}\cdot\nabla\left[\epsilon\right]\Delta\mathbf{v}
\end{aligned}$$

$$\left(\frac{\partial}{\partial t}+\mathbf{v}\cdot\nabla\right)\Delta\mathbf{v}=-\frac{\Delta\mathbf{v}}{t_\mathrm{s}}+\Delta\mathbf{f}-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}\\+(2\epsilon-1)\,(\Delta\mathbf{v}\cdot\nabla)\,\Delta\mathbf{v}+\Delta\mathbf{v}\,(\Delta\mathbf{v}\cdot\nabla)\,\epsilon$$

It is also possible to use the vector analysis identity $\nabla(\mathbf{X}\cdot\mathbf{X})=2(\mathbf{X}\cdot\nabla)\mathbf{X}+2\mathbf{X}\times(\nabla\times\mathbf{X})$ to transform the expression.

$$\begin{aligned}
\left(\frac{\partial}{\partial t}+\mathbf{v}\cdot\nabla\right)\Delta\mathbf{v}=&-\frac{\Delta\mathbf{v}}{t_\mathrm{s}}+\Delta\mathbf{f}-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}\\
&+\frac{1}{2}\nabla\left(\epsilon^2\Delta\mathbf{v}\cdot\Delta\mathbf{v}\right)-\epsilon\Delta\mathbf{v}\times(\nabla\times\epsilon\Delta\mathbf{v})\\
&-\frac{1}{2}\nabla\left((1-\epsilon)^2\Delta\mathbf{v}\cdot\Delta\mathbf{v}\right)+(1-\epsilon)\Delta\mathbf{v}\times(\nabla\times(1-\epsilon)\Delta\mathbf{v})
\end{aligned}\tag{C.24}$$

This lead to the equation on the differential velocity in the same form as presented in Lebreuilly et al. (2019),

$$\left(\frac{\partial}{\partial t}+\mathbf{v}\cdot\nabla\right)\Delta\mathbf{v}=-\frac{\Delta\mathbf{v}}{t_\mathrm{s}}+\Delta\mathbf{f}-(\Delta\mathbf{v}\cdot\nabla)\,\mathbf{v}+\frac{1}{2}\nabla\left((2\epsilon-1)\Delta\mathbf{v}\cdot\Delta\mathbf{v}\right)\\-\epsilon\Delta\mathbf{v}\times(\nabla\times\epsilon\Delta\mathbf{v})+(1-\epsilon)\Delta\mathbf{v}\times(\nabla\times(1-\epsilon)\Delta\mathbf{v}).\tag{C.25}$$

□

Proposition 1.1 ▶ Monofluid conservation of energy

$$\left(\frac{\partial}{\partial t}+\mathbf{v}\cdot\nabla\right)u_{\mathrm{g}}=-\frac{P}{(1-\epsilon)\rho}\nabla\cdot(\mathbf{v}-\epsilon\Delta\mathbf{v})+\epsilon\Delta\mathbf{v}\cdot\nabla u_{\mathrm{g}}+\frac{K\Delta\mathbf{v}^2}{\rho_{\mathrm{g}}}$$

Proof. Starting from the energy conservation equation in two-fluid formalism, the monofluid variant is given by

$$\begin{aligned}\frac{\partial u_{\mathrm{g}}}{\partial t}+(\mathbf{v}_{\mathrm{g}}\cdot\nabla)\,u_{\mathrm{g}}&=-\frac{P}{\rho_{\mathrm{g}}}(\nabla\cdot\mathbf{v}_{\mathrm{g}})+\frac{K\left(\mathbf{v}_{\mathrm{d}}-\mathbf{v}_{\mathrm{g}}\right)^2}{\rho_{\mathrm{g}}}\\ \frac{\partial u_{\mathrm{g}}}{\partial t}+((\mathbf{v}-\epsilon\Delta\mathbf{v})\cdot\nabla)\,u_{\mathrm{g}}&=-\frac{P}{(1-\epsilon)\rho}(\nabla\cdot\mathbf{v}_{\mathrm{g}})+\frac{K\Delta\mathbf{v}^2}{\rho_{\mathrm{g}}}\\ \frac{\partial u_{\mathrm{g}}}{\partial t}+(\mathbf{v}\cdot\nabla)\,u_{\mathrm{g}}&=\epsilon\Delta\mathbf{v}\cdot\nabla u_{\mathrm{g}}-\frac{P}{(1-\epsilon)\rho}(\nabla\cdot(\mathbf{v}-\epsilon\Delta\mathbf{v}))+\frac{K\Delta\mathbf{v}^2}{\rho_{\mathrm{g}}}.\end{aligned}$$

□

2. SPH identities

Definition ▶ SPH Symetric/Antisymetric notations

For sake of clarity, we adopt notations inspired from the one used in general relativity. Using the unit vector $\mathbf{1}$, defined by $\mathbf{1}_a=1,\forall a$, we define

$$\begin{aligned}(T_a+T_b)&=T_{(a}\mathbf{1}_{b)}=T_{(a\mathbf{1}b)},\\ (T_a-T_b)&=T_{[a}\mathbf{1}_{b]}=T_{[a\mathbf{1}b]}.\end{aligned}$$

In SPH, we often use the symmetry or antisymmetry of the operators to prove the conservation equation. In order to do so, two formulas are useful to permute a sum from its symmetry form to its antisymmetric form, or inversely, in a double sum.

Proposition ▶ Symmetry permutation in SPH

$$\begin{aligned}\sum_a\sum_b m_a m_b\ X_a T_{(a\mathbf{1}b)}\nabla_a W_{ab}&=\sum_a\sum_b m_a m_b\ T_a X_{[a\mathbf{1}b]}\nabla_a W_{ab},\\ \sum_a\sum_b m_a m_b\ X_a T_{(a\mathbf{1}b)}\cdot\nabla_a W_{ab}&=\sum_a\sum_b m_a m_b\ T_a^\nu X^\mu_{[a\mathbf{1}b]}\nabla_a W^\nu_{ab}.\end{aligned}$$

Proof.

$$\begin{aligned}
&\sum_a \sum_b m_a m_b \; X_a (T_a + T_b) \nabla W_{ab} \\
&\quad = \sum_a \sum_b m_a m_b \; X_a T_a \nabla W_{ab} + \sum_a \sum_b m_a m_b \; X_a T_b \nabla W_{ab} \\
&\quad = \sum_a \sum_b m_a m_b \; X_a T_a \nabla W_{ab} - \sum_a \sum_b m_a m_b \; X_b T_a \nabla W_{ab} \\
&\quad = \sum_a \sum_b m_a m_b \; T_a (X_a - X_b) \nabla W_{ab}
\end{aligned}$$

$$\begin{aligned}
&\sum_a \sum_b m_a m_b \; X_a \left[(T_a + T_b) \cdot \nabla W_{ab} \right] \\
&\quad = \sum_a \sum_b m_a m_b \; X_a \left[T_a \cdot \nabla W_{ab} \right] + \sum_a \sum_b m_a m_b \; X_a \left[T_b \cdot \nabla W_{ab} \right] \\
&\quad = \sum_a \sum_b m_a m_b \; X_a \left[T_a \cdot \nabla W_{ab} \right] - \sum_a \sum_b m_a m_b \; X_b \left[T_a \cdot \nabla W_{ab} \right] \\
&\quad = \sum_a \sum_b m_a m_b \; T_a^\nu (X_a - X_b)^\mu \nabla W_{ab}^\nu
\end{aligned}$$

□

3. Derivation from conservation equations

3.1. Conservation of dust mass

In SPH, the pressure operator is given in its antisymmetric form by the following discretization:

$$-\frac{1}{\rho} \nabla(\chi) \xrightarrow[\text{SPH}]{} \sum_b m_b \left(\frac{\chi}{\rho^2} \right)_{(a^1 b)} \nabla W_{ab}. \tag{C.26}$$

As mentioned in Chapter 1, Sec. 4.3, using the antisymmetric form of the pressure ensures the conservation of momentum. We use a similar guess for the conservation of dust mass to build the SPH operator. In order to check the symmetry property of the operator, we can write the conservation of the mass of the dust

$$0 = \frac{\mathrm{d}\mathbf{M}_\mathrm{d}}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t} \sum_a m_a \epsilon_a = \sum_a m_a \frac{\mathrm{d}\epsilon_a}{\mathrm{d}t}. \tag{C.27}$$

This shows that the operator must be antisymmetric, which is the case using the antisymmetric canonical SPH derivative. This causes the conservation of dust mass in SPH to be

Proposition ▶ SPH equation of the conservation of dust mass

$$\frac{\mathrm{d}\epsilon_a}{\mathrm{d}t} = \sum_b m_b \left(\frac{\epsilon(1-\epsilon)\Delta\mathbf{v}}{\rho}\right)_{(a^{\mathbf{1}}b)} \nabla W_{ab}$$

3.2. Conservation of dust momentum

To build operators for the other equations, we start with the conservation of dust momentum, which is conserved in the absence of drag. Firstly, the dust momentum in SPH is

$$\mathbf{P}_\mathrm{d} = \sum_a m_a(\epsilon_a \mathbf{v}_a + \epsilon_a(1-\epsilon_a)\Delta\mathbf{v}_a). \tag{C.28}$$

Its time derivative should be null in the absence of dust

$$\begin{aligned}
0 = \frac{\mathrm{d}\mathbf{P}_\mathrm{d}}{\mathrm{d}t} &= \sum_a m_a \left(\epsilon_a \frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} + \frac{\mathrm{d}\epsilon_a}{\mathrm{d}t}\mathbf{v}_a + \epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t} + (1-2\epsilon_a)\frac{\mathrm{d}\epsilon_a}{\mathrm{d}t}\Delta\mathbf{v}_a\right) \\
&= + \sum_a m_a \left(\epsilon_a \frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t}|_{\mathbf{f}_\mathrm{g}} + \epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{\mathbf{f}_\mathrm{g}}\right) && \text{(C.29)} \\
&\quad + \sum_a m_a \left(\frac{\mathrm{d}\epsilon_a}{\mathrm{d}t}\mathbf{v}_a + \epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{-(\Delta\mathbf{v}\cdot\nabla)\mathbf{v}}\right) && \text{(C.30)} \\
&\quad + \sum_a m_a \left((1-2\epsilon_a)\frac{\mathrm{d}\epsilon_a}{\mathrm{d}t}\Delta\mathbf{v}_a + \epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_1\right) && \text{(C.31)} \\
&\quad + \sum_a m_a \left(\epsilon_a \frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t}|_{-\frac{1}{\rho}\nabla(\ldots)} + \epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_2\right). && \text{(C.32)}
\end{aligned}$$

We will now construct the SPH operators for the terms of the equation on barycentric and differential velocity so that we can ensure conservation of dust momentum. To do so, we will group terms together and identify potential operators based on the conservation of dust momentum. In most of the computation, we rely heavily on the SPH double-sum permutation identities. See SPH identities for the following formulas shown in Sec. 2. From here, we will work out the cancellation lines per line, assuming that each line of the momentum equation (Eq. C.29-C.32) must be equal to zero.

Equation C.29:

$$\begin{aligned}
(\text{C.29}) &= \sum_a m_a \left(\epsilon_a \frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t}|_{\mathbf{f}_\mathrm{g}} + \epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{\mathbf{f}_\mathrm{g}}\right) \\
&= \sum_a m_a \left(\epsilon_a(1-\epsilon_a)\mathbf{f}_\mathrm{g} + \epsilon_a(1-\epsilon_a)(-\mathbf{f}_\mathrm{g})\right) = 0.
\end{aligned}$$

This expression can be solved by setting:

$$\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t}|_{\mathbf{f}_\mathrm{g}} = (1-\epsilon_a)\mathbf{f}_\mathrm{g},$$
$$\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{\mathbf{f}_\mathrm{g}} = -\mathbf{f}_\mathrm{g}.$$

Equation C.30:

$$\begin{aligned}
(\text{C.30}) &= \sum_a m_a \left(\frac{\mathrm{d}\epsilon_a}{\mathrm{d}t}\mathbf{v}_a + \epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{-(\Delta\mathbf{v}\cdot\nabla)\mathbf{v}} \right) \\
&= \sum_a m_a \frac{\mathrm{d}\epsilon_a}{\mathrm{d}t}\mathbf{v}_a + \sum_a m_a\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{-(\Delta\mathbf{v}\cdot\nabla)\mathbf{v}} \\
&= \sum_a m_a\mathbf{v}_a \sum_b m_b \left(\frac{\epsilon(1-\epsilon)\Delta\mathbf{v}}{\rho} \right)_{(a\mathbf{1}b)} \cdot \nabla W_{ab} \\
&\quad + \sum_a m_a\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{-(\Delta\mathbf{v}\cdot\nabla)\mathbf{v}} \\
&= \sum_a m_a \frac{\epsilon_a(1-\epsilon_a)}{\rho_a} \sum_b m_b\mathbf{v}_{[a\mathbf{1}b]} \left[\Delta\mathbf{v}_a \cdot \nabla W_{ab}\right] \\
&\quad + \sum_a m_a\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{-(\Delta\mathbf{v}\cdot\nabla)\mathbf{v}} \\
&= \sum_a m_a\epsilon_a(1-\epsilon_a) \left[\frac{1}{\rho_a} \sum_b m_b\mathbf{v}_{[a\mathbf{1}b]} \left[\Delta\mathbf{v}_a \cdot \nabla W_{ab}\right] + \frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{-(\Delta\mathbf{v}\cdot\nabla)\mathbf{v}} \right].
\end{aligned}$$

This equation yields a possible choice for this term to be

$$\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_{-(\Delta\mathbf{v}\cdot\nabla)\mathbf{v}} = -\frac{1}{\rho_a}\sum_b m_b\mathbf{v}_{[a\mathbf{1}b]} \left[\Delta\mathbf{v}_a \cdot \nabla W_{ab}\right].$$

Equation C.31:

$$
\begin{aligned}
(\text{C.31}) &= \sum_a m_a \left((1-2\epsilon_a)\frac{\mathrm{d}\epsilon_a}{\mathrm{d}t}\Delta\mathbf{v}_a + \epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_1 \right) \\
&= \sum_a m_a \left((1-2\epsilon_a) \left[\sum_b m_b \left(\frac{\epsilon(1-\epsilon)\Delta\mathbf{v}}{\rho} \right)_{(a^1b)} \cdot \nabla W_{ab} \right] \Delta\mathbf{v}_a \right) \\
&\quad + \sum_a m_a \left(\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_1 \right) \\
&= \sum_a \sum_b m_a m_b \left[(1-2\epsilon)\Delta v^\mu \right]_a \left(\frac{\epsilon(1-\epsilon)\Delta v^\nu}{\rho} \right)_{(a^1b)} \nabla^\nu W_{ab} \\
&\quad + \sum_a m_a \left(\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta v_a^\mu}{\mathrm{d}t}|_1 \right) \\
&= \sum_a \sum_b m_a m_b \left(\frac{\epsilon(1-\epsilon)\Delta v^\nu}{\rho} \right)_a \left[(1-2\epsilon)\Delta\mathbf{v} \right]_{[a^1b]} \nabla^\nu W_{ab} \\
&\quad + \sum_a m_a \left(\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta v_a^\mu}{\mathrm{d}t}|_1 \right) \\
&= \sum_a m_a \left(\frac{\epsilon(1-\epsilon)\Delta v^\nu}{\rho} \right)_a \sum_b m_b \left[(1-2\epsilon)\Delta\mathbf{v} \right]_{[a^1b]} \nabla^\nu W_{ab} \\
&\quad + \sum_a m_a \left(\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta v_a^\mu}{\mathrm{d}t}|_1 \right) \\
&= \sum_a m_a \epsilon_a(1-\epsilon_a) \left(\frac{\mathrm{d}\Delta v_a^\mu}{\mathrm{d}t}|_1 + \frac{\Delta v_a^\nu}{\rho_a} \sum_b m_b \left[(1-2\epsilon)\Delta\mathbf{v} \right]_{[a^1b]} \nabla^\nu W_{ab} \right).
\end{aligned}
$$

Therefore, we can take

$$
\frac{\mathrm{d}\Delta v_a^\mu}{\mathrm{d}t}|_1 = -\frac{\Delta v_a^\nu}{\rho_a} \sum_b m_b \left[(1-2\epsilon)\Delta\mathbf{v} \right]_{[a^1b]} \nabla^\nu W_{ab},
$$

which can be rewritten as

$$
\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_1 = -\frac{1}{\rho_a} \sum_b m_b \left[(1-2\epsilon)\Delta\mathbf{v} \right]_{[a^1b]} \left[\Delta\mathbf{v}_a \cdot \nabla W_{ab} \right].
$$

Equation C.32: For the last term, in order to find an expression, we must use an ansatz to close the system of equations. We use the following ansatz:

$$
\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t}|_{-\frac{1}{\rho}\nabla(\ldots)} = \sum_b m_b \left[\frac{\epsilon(1-\epsilon)}{\rho} \Delta\mathbf{v}\Delta\mathbf{v} \right]_{(a^1b)} \cdot \nabla W_{ab}. \tag{C.33}
$$

By replacing this ansatz in C.32, we can compute the last terms.

$$
\begin{aligned}
(\text{C.32}) &= \sum_a m_a \left(\epsilon_a \frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t}|_{-\frac{1}{\rho}\nabla(...)} + \epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_2 \right) = 0 \\
&= \sum_a m_a\epsilon_a \sum_b m_b \left[\frac{\epsilon(1-\epsilon)}{\rho}\Delta\mathbf{v}\Delta\mathbf{v}\right]_{(a^1b)} \cdot \nabla W_{ab} + \sum_a m_a\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_2 \\
&= \sum_a \sum_b m_a m_b \epsilon_a \left[\frac{\epsilon(1-\epsilon)}{\rho}\Delta\mathbf{v}\Delta\mathbf{v}\right]_{(a^1b)} \cdot \nabla W_{ab} + \sum_a m_a\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_2 \\
&= \sum_a \sum_b m_a m_b \epsilon_{[a^1b]} \left[\frac{\epsilon(1-\epsilon)}{\rho}\Delta\mathbf{v}\Delta\mathbf{v}\right]_{a} \cdot \nabla W_{ab} + \sum_a m_a\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_2 \\
&= \sum_a m_a \left[\frac{\epsilon(1-\epsilon)}{\rho}\Delta\mathbf{v}\Delta\mathbf{v}\right]_{a} \sum_b m_b \epsilon_{[a^1b]} \cdot \nabla W_{ab} + \sum_a m_a\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_2 \\
&= \sum_a m_a\epsilon_a(1-\epsilon_a)\left[\frac{\Delta\mathbf{v}_a}{\rho_a}\Delta\mathbf{v}_a \cdot \sum_b m_b\epsilon_{[a^1b]}\nabla W_{ab}\right] \\
&\quad + \sum_a m_a\epsilon_a(1-\epsilon_a)\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_2,
\end{aligned}
$$

where the last SPH operator is

$$
\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t}|_2 = -\frac{\Delta\mathbf{v}_a}{\rho_a}\Delta\mathbf{v}_a \cdot \sum_b m_b\epsilon_{[a^1b]}\nabla W_{ab}.
$$

Summary The complete set of equations for the full-monofluid formalism in SPH is:

Proposition ▶ SPH full-monofluid formalism

$$
\begin{aligned}
\frac{\mathrm{d}\epsilon_a}{\mathrm{d}t} &= \sum_b m_b \left(\frac{\epsilon(1-\epsilon)\Delta\mathbf{v}}{\rho}\right)_{(a^1b)} \nabla W_{ab}, \\
\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} &= (1-\epsilon_a)\mathbf{f}_{\mathrm{g}} + \sum_b m_b \left[\frac{\epsilon(1-\epsilon)}{\rho}\Delta\mathbf{v}\Delta\mathbf{v}\right]_{(a^1b)} \cdot \nabla W_{ab}, \\
\frac{\mathrm{d}\Delta\mathbf{v}_a}{\mathrm{d}t} &= -\mathbf{f}_{\mathrm{g}} - \frac{1}{\rho_a}\sum_b m_b \mathbf{v}_{[a^1b]} \left[\Delta\mathbf{v}_a \cdot \nabla W_{ab}\right] \\
&\quad - \frac{1}{\rho_a}\sum_b m_b \left[(1-2\epsilon)\Delta\mathbf{v}\right]_{[a^1b]} \left[\Delta\mathbf{v}_a \cdot \nabla W_{ab}\right] - \frac{\Delta\mathbf{v}_a}{\rho_a}\Delta\mathbf{v}_a \cdot \sum_b m_b\epsilon_{[a^1b]}\nabla W_{ab}.
\end{aligned}
$$

4. Summary

We have derived in this appendix a full monofluid formalism without terminal diffusion approximation from the dust momentum conservation equation. This ensures

that the current scheme conserves the momentum of the dust. We, however, have not yet included the modified derivatives from Price (2012) (with Ω_a terms in the denominator of gradient operators). As for the artificial viscosity, the only required change to implement it is to replace the occurrence of the gas density in the solvers by $\rho(1-\epsilon)$ (e.g. Laibe & Price 2014). We also attempted to generalize the current scheme to multiple dust species. However, the current procedure relies on term identification to build the formulas such that they ensure conservation of momentum, which is much more complex for large systems such as the multiple-size case. Lastly, even if this scheme conserves dust momentum, it is not guaranteed to conserve every Noether's invariants of the problem. A possible solution to ensure their conservation would be to derive the scheme from a variational principle. Additionally, constructing the scheme from a variational principle could also allow the generalization to multiple dust sizes and ensure the correctness of the model by not relying on ansatz.

Using such schemes allows for a finely controlled dynamic of the dust while resolving large Stokes numbers. This can allow, typically, simulations of dusty collapse where the current formalism with terminal velocity approximation is limiting for larger grain sizes (Lebreuilly et al., 2019, 2020). For some applications, such as turbulence dust, the particle is precise enough due to artificial particle clumping for grains with $\mathrm{St} \sim 1$, but the monofluid formalism cannot resolve them either (e.g. Commerçon et al. 2023). This shows the necessity of a full-monofluid formalism without a dust terminal velocity approximation. Additionally, such a scheme can be used to try to reproduce the Lorén-Aguilar & Bate (2015), as it is currently interpreted as being the result of the dust dynamic at large stokes that is not captured by the monofluid formalism in terminal velocity approximation. In such a case, the full monofluid formalism should reproduce the instability. If that is not the case, this instability would be either due to numerical effects or related to the kinetic effects that are not captured by pressure-less fluids. We plan in a close future to implement such a scheme in Shamrock.

References

Commerçon B., Lebreuilly U., Price D. J., Lovascio F., Laibe G., Hennebelle P., 2023, *Dynamics of dust grains in turbulent molecular clouds. Conditions for decoupling and limits of different numerical implementations*, A&A, 671, A128

Hutchison M., 2017, PhD thesis, -

Laibe G., Price D. J., 2014, *Dusty gas with one fluid*, MNRAS, 440, 2136-2146

Lebreuilly U., Commerçon B., Laibe G., 2019, *Small dust grain dynamics on adaptive mesh refinement grids. I. Methods*, A&A, 626, A96

Lebreuilly U., Commerçon B., Laibe G., 2020, *Protostellar collapse: the conditions to form dust-rich protoplanetary disks*, A&A, 641, A112

Lorén-Aguilar P., Bate M. R., 2015, *Toroidal vortices and the conglomeration of dust into rings in protoplanetary discs*, MNRAS, 453, L78-L82

Price D. J., 2012, *Smoothed particle hydrodynamics and magnetohydrodynamics*, Journal of Computational Physics, 231, 759-794

Appendix D

Shearing Box

Contents

Foreword

In this appendix, we present a standard implementation of the local Shearing Box in the SPH solver of Shamrock. We show that such implementation results in noise. At numerical scale, lattice reorganization is destroyed by the local shear, enforcing an unusual level of discretization errors. In order to potentially solve this issue, we present two alternative methods that we derived analytically, but have to be tested in the code. The first method consists of integrating the shearing direction in an axisymmetric shearing box. The other method consists of using a sheared coordinate system, so that the steady dynamics of the shearing box correspond to null velocities, potentially improving SPH precision.

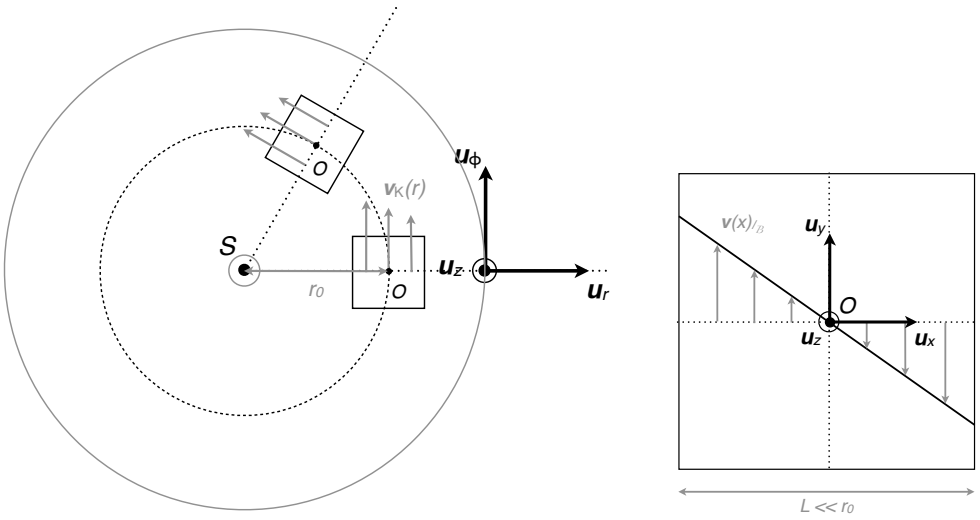
1. Shearing box

We have shown in Chapter 1,Sec. 2.2 that a steady solution of a protoplanetary disc is a differentially rotating flow at sub-Keplerian velocities. In order to simulate a small-scale effect, Goldreich & Lynden-Bell (1965) derived based on the work of Hill (1878) a local box in which the differentially rotating flow results locally in a shear flow (represented in Fig. D.1). From Hawley et al. (1995), the Euler equations in the shearing box can be written in their primitive form as follows:

$$(\partial_t + \mathbf{v}\cdot\boldsymbol{\nabla})\rho = -\boldsymbol{\nabla}\cdot\mathbf{v}, \tag{D.1}$$

$$(\partial_t + \mathbf{v}\cdot\boldsymbol{\nabla})\mathbf{v} = -\frac{1}{\rho}\boldsymbol{\nabla}P - 2\Omega\mathbf{e}_k\times\mathbf{v} + 2\eta x\mathbf{e}_x - \Omega^2 z\mathbf{e}_z, \tag{D.2}$$

$$(\partial_t + \mathbf{v}\cdot\boldsymbol{\nabla})u = -\frac{P}{\rho}\boldsymbol{\nabla}\cdot\mathbf{v}, \tag{D.3}$$



(a) Global. (b) Local

Figure D.1: Representation of the shearing box in a protoplanetary disc. (a) Position of the shearing box in the global disc. (b) Local dynamics in the shearing box resulting from local approximation. (Adapted from Laibe (2020))

where $\eta \equiv q\Omega^2$, and $q \equiv -\mathrm{d}\ln\Omega/\mathrm{d}\ln r$ parametrizes the shear associated to local differential rotation. They correspond to the standard Euler equation with the addition of an external force corresponding to the balance between inertial forces and gravity in the local corotating frame rotating with the disc, as well as the Coriolis force. Close to the midplane, the vertical component of the gravity is $-\Omega^2 z\mathbf{e}_z$. When the vertical gravitational force is omitted, the shearing box is referred to as unstratified. For sake of simplification, in the shearing box, a movement along the x axis corresponds in the global disc to being on an eccentric orbit, which corresponds in the shearing box to an oscillation on the orbit timescale called epicycles. Formally, this effect takes the form of the force in the shearing box.

In order to simulate the shearing box, special care has to be taken for the boundary conditions. Traditionally, the boundary conditions are taken to be so-called shearing-periodic (e.g. Rein & Papaloizou 2010; Hawley & Balbus 1991; Youdin & Johansen 2007). Shearing-periodic boundary conditions refer to a box where the periodic images of the box are moving at the speed of the shear flow relative to the simulation domain. Formally, for a field $\Phi(\mathbf{r}, t)$ with periodic boundary conditions, we have the following statement:

$$\Phi(\mathbf{r}, t) = \Phi\Big(\mathbf{r} + i(L_x\mathbf{e}_x + t\mathbf{v}_\mathrm{s}\mathbf{e}_y) + jL_y\mathbf{e}_y + kL_z\mathbf{e}_z, t\Big), \quad (i,j,k) \in \mathbb{Z}^3. \tag{D.4}$$

This condition is represented graphically in Fig. D.2. Corresponding to the following transformation for the positions in a mirrored domain of index (i, j, k)

$$\mathbf{r} \mapsto \mathbf{r} + i(L_x\mathbf{e}_x + t\mathbf{v}_\mathrm{s}\mathbf{e}_y) + jL_y\mathbf{e}_y + kL_z\mathbf{e}_z,$$

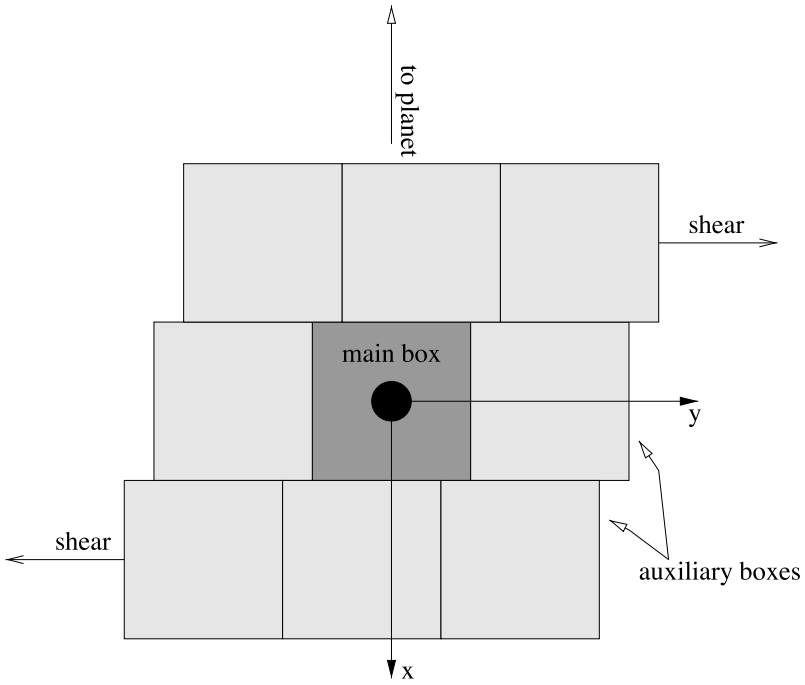


Figure D.2: Shearing-periodic boundary conditions. The main box corresponds to the simulation domain, and the auxiliary box corresponds to the shearing-periodic images of the domain across the boundary condition. (Adapted from Rein & Papaloizou (2010))

thus, the shear speed of the boundary is

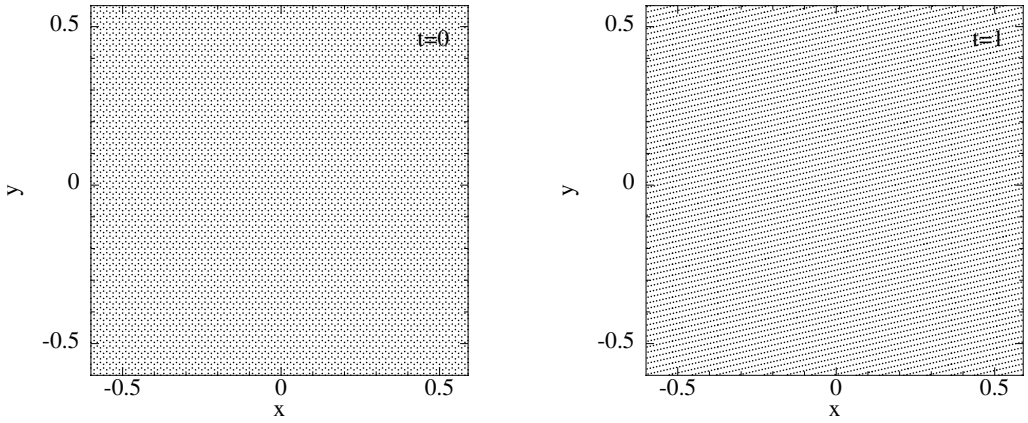
$$\mathbf{v}_\mathrm{s} = q\Omega L_x.$$

Additionally, for the velocity field, the transformation differs as the boundary is time-dependent. Instead, the velocity is transformed as follows across the shearing periodic boundary:

$$\mathbf{v}(\mathbf{r},t) = \mathbf{v}\Big(\mathbf{r} + i(L_x\mathbf{e}_x + t\mathbf{v}_\mathrm{s}\mathbf{e}_y) + jL_y\mathbf{e}_y + kL_z\mathbf{e}_z, t\Big) + i\mathbf{v}_\mathrm{s}\mathbf{e}_y, \quad (i,j,k) \in \mathbb{Z}^3. \quad \text{(D.5)}$$

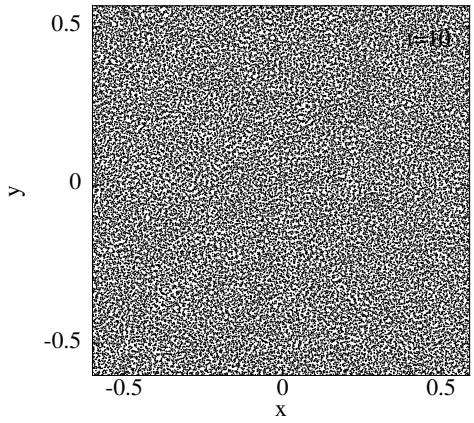
2. SPH implementation

As presented in Chapter 4, Sec. 3.6.2 in Shamrock, a graph of ghost zones is used to compute and exchange the ghost zones. With the implementation of ghost zones in Shamrock based on an interaction criterion, only two modifications are needed to convert the periodic boundary into a shearing periodic boundary First, the interaction criterion is altered such that the domain mirroring is performed according to the shearing periodic boundary conditions instead of the standard periodic ones (as represented on Fig. D.2). Second, when mirrored across the boundary, the corresponding offset in position and velocity is applied to the ghost zone as specified.



(a) Setup of the simulation, $t = 0$.

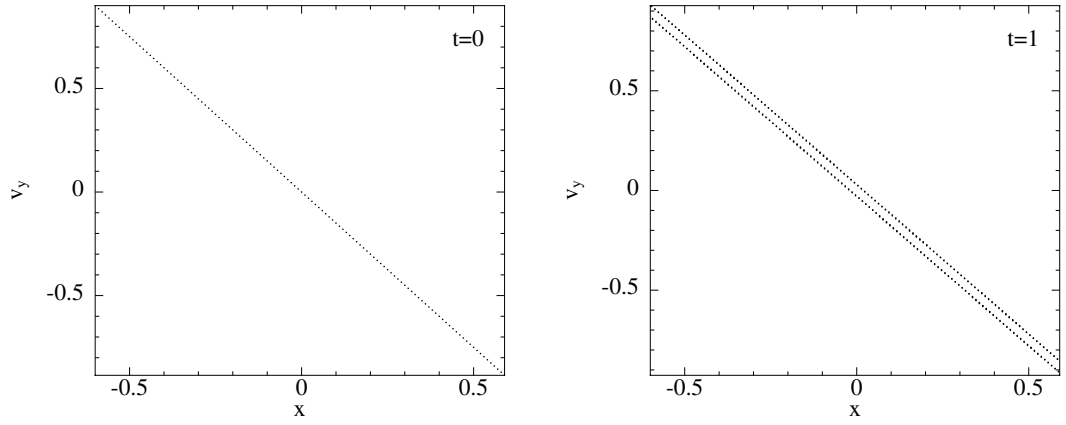
(b) $t = 1$.



(c) $t = 10$.

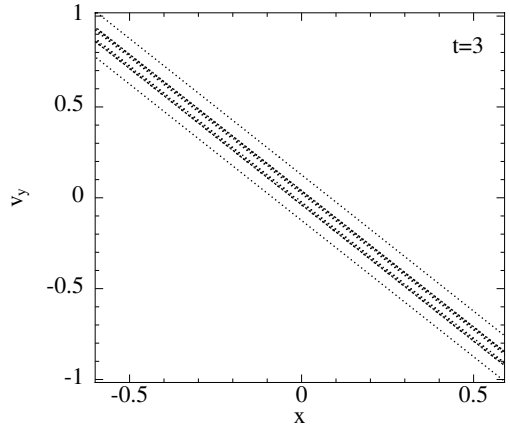
Figure D.3: Result of shearing box simulation in SHAMROCK with the SPH solver using a cubic spline SPH kernel at different values of time, in code units. On plots from a, b, and c, the positions of particles are shown. The corresponding velocity profile is shown in Fig. D.4.

As a basic test of the shearing box, we present here the result of a steady-state simulation in the shearing box. The particles are setup in three dimensions on a hexagonal compact lattice to achieve a constant density in the box. Velocities are set on a linear profile matching the shear velocity of the boundary conditions on both sides (see Fig. D.4). We observe that particles are stable up to 3 shear time, after which the crystal lattice destabilizes, leading to random local particle distribution and therefore noise in the simulated fields. This effect is due to the particle being forced out of a stable lattice arrangement by the shear. Initially, the particles are on a hexagonal compact lattice, which is a stable arrangement of particles. However, after roughly one shear time, the lattice is sheared into a nearly cubic one, which is known in SPH to be unstable and transitioning to a random distribution in a short time (e.g. Price 2012).

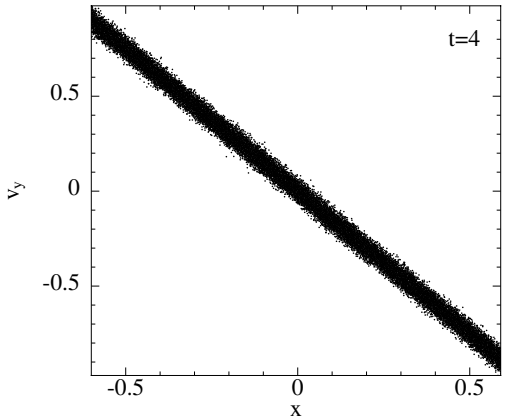


(a) Setup of the simulation, $t = 0$.

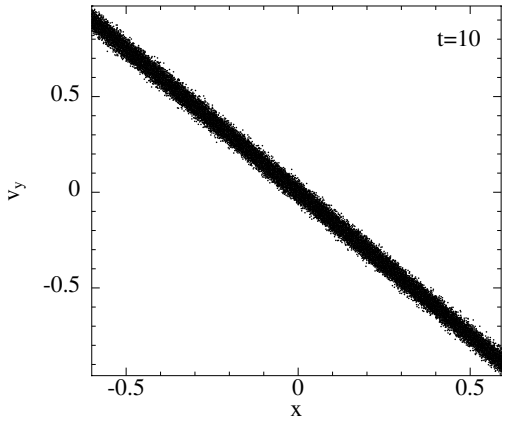
(b) $t = 1$.



(c) $t = 10$.



(d) $t = 10$.



(e) $t = 10$.

Figure D.4: Result of shearing box simulation in Shamrock with the SPH solver using a cubic spline SPH kernel at different values of time, in code units. On plots a, b, c, d, and e, the velocity is shown in code units.

Aside from this result, we have not completed additional physical tests of the implementation yet. We do, however, plan to test the Kida vortex (Kida, 1981), an eccentric vortex solution that should remain steady if the scheme is sufficiently precise. If not, a precession of the vortex should be observed.

3. Axisymmetric shearing box

We present in this section another procedure that we have derived that could be suited for SPH simulation of an axisymmetric shearing box. The goal is to remove the shearing motion of the particle, as this leads to noise in SPH. To do so, we transform the equations of the shearing box as per Stone & Gardiner (2010), and then, using the axisymmetric approximation, we integrate the shearing direction away. The shearing box force can be written as being

$$\begin{aligned}\mathbf{f}_{\text{SB}} &= 2\Omega^2 q x \mathbf{e}_x - 2\Omega \mathbf{e}_z \times \mathbf{v} - \Omega^2 z \mathbf{e}_z \\ &= 2\Omega^2 q x \mathbf{e}_x - 2\Omega \left(-v_y \mathbf{e}_x + v_x \mathbf{e}_y\right) - \Omega^2 z \mathbf{e}_z \\ &= 2\Omega^2 q x \mathbf{e}_x + 2\Omega v_y \mathbf{e}_x - 2\Omega v_x \mathbf{e}_y - \Omega^2 z \mathbf{e}_z \\ &= 2\Omega \left(q\Omega x + v_y\right) \mathbf{e}_x - 2\Omega v_x \mathbf{e}_y - \Omega^2 z \mathbf{e}_z. \end{aligned} \tag{D.6}$$

From this form we see that the shearing box admit a simple steady-state solution:

$$\mathbf{v}_K = -q\Omega x \mathbf{e}_y.$$

We may want to subtract such a steady background state from the actual equations of the shearing box. To start off, we have to compute the convective derivative. Defining

$$\tilde{\mathbf{v}}_x = \mathbf{v}_x, \tag{D.7}$$
$$\tilde{\mathbf{v}}_y = \mathbf{v}_y + q\Omega x. \tag{D.8}$$

The convective derivative can be transformed as such

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \boldsymbol{\nabla} \mathbf{v} = \frac{\partial \tilde{\mathbf{v}}}{\partial t} + \tilde{\mathbf{v}} \cdot \boldsymbol{\nabla} \bar{\mathbf{v}} + \bar{\mathbf{v}} \cdot \boldsymbol{\nabla} \tilde{\mathbf{v}} + \tilde{\mathbf{v}} \cdot \boldsymbol{\nabla} \tilde{\mathbf{v}} \tag{D.9}$$
$$= \frac{\partial \tilde{\mathbf{v}}}{\partial t} + \tilde{\mathbf{v}} \cdot \boldsymbol{\nabla} (-q\Omega x \mathbf{e}_y) + (-q\Omega x \mathbf{e}_y) \cdot \boldsymbol{\nabla} \tilde{\mathbf{v}} + \tilde{\mathbf{v}} \cdot \boldsymbol{\nabla} \tilde{\mathbf{v}} \tag{D.10}$$
$$= \frac{\partial \tilde{\mathbf{v}}}{\partial t} - q\Omega \tilde{v}_x \mathbf{e}_y - q\Omega x \partial_y \tilde{\mathbf{v}} + \tilde{\mathbf{v}} \cdot \boldsymbol{\nabla} \tilde{\mathbf{v}}. \tag{D.11}$$

Using this transformation, we get the Euler momentum conservation equation for the relative velocity

$$\frac{\partial \tilde{\mathbf{v}}}{\partial t} - q\Omega x \partial_y \tilde{\mathbf{v}} + \tilde{\mathbf{v}} \cdot \boldsymbol{\nabla} \tilde{\mathbf{v}} = -\frac{1}{\rho} \boldsymbol{\nabla} P + 2\Omega \tilde{v}_y \mathbf{e}_x + (q-2)\Omega \tilde{v}_x \mathbf{e}_y - \Omega^2 z \mathbf{e}_z, \tag{D.12}$$

If we have a barotropic equation of state ($P = f(\rho)$), the internal energy equation is decoupled from the Euler equation as the pressure does not depend on it. If the system is assumed to be axisymmetric, only the momentum equation is changed in a shearing box based on the relative velocities. Therefore, we can try to rewrite it in the case of axisymmetry ($\partial_y = 0$). We use the Euler equation in the following form:

$$\mathbf{D}_t \mathbf{v} - q\Omega x \partial_y \mathbf{v} = -\frac{1}{\rho}\boldsymbol{\nabla} P + 2\Omega v_y \mathbf{e}_x + (q-2)\Omega v_x \mathbf{e}_y - \Omega^2 z \mathbf{e}_z, \tag{D.13}$$

where $\mathbf{D}_t = \partial_t + \mathbf{v}\cdot\boldsymbol{\nabla}$ is the convective derivative, and we denote the relative velocity $\tilde{\mathbf{v}}$ by simply $\mathbf{v}$ here. In the $\mathbf{e}_y$ axis, the equation is using the axisymmetry ($\partial_y = 0$).

$$\mathbf{D}_t v_y = (q-2)\Omega v_x. \tag{D.14}$$

Integrating over time, we obtain the following solution:

$$v_y = (q-2)\Omega \xi_x, \tag{D.15}$$

where $\xi_x = \int \mathrm{d}t\, v_x$ is the Lagrangian displacement. Replacing the expression of the azimuthal velocity in the Euler momentum conservation equation and accounting for the axisymmetry yield

$$\frac{\partial \tilde{\mathbf{v}}}{\partial t} + \tilde{\mathbf{v}}\cdot\boldsymbol{\nabla}\tilde{\mathbf{v}} = -\frac{1}{\rho}\boldsymbol{\nabla} P + 2(q-2)\Omega^2 \xi_x \mathbf{e}_x - \Omega^2 z \mathbf{e}_z. \tag{D.16}$$

Additionally, from this equation, we observe that for a constant velocity in space, one solution is given on the $\mathbf{e}_x$ axis by

$$\frac{\mathrm{d}^2 \xi_x}{\mathrm{d}t^2} = 2(q-2)\Omega^2 \xi_x \mathbf{e}_x, \tag{D.17}$$

which is a harmonic oscillator of frequency $2(q-2)\Omega^2$, which is the epicyclic frequency. Indeed, this equation describes the oscillatory epicyclic motion mentioned previously.

In SPH, the particles are Lagrangian. Therefore, ξ_x simply corresponds to the difference between their starting position at $t = 0$ and their position at time t. This could result in precise integration of the epicyclic motion in such an approach. Additionally, the shear being along the axis $\mathbf{e}_y$ is therefore removed in this approach as the y direction has been integrated away. Lastly, for mechanisms such as the SI or the MRI, the axisymmetric approximation is taken, allowing potential simulation of such processes with this method.

4. Sheared coordinates

We want to remove the shearing motion of particles. The previous method does so by integrating the motion along the azimuthal axis. However, this only allows 2.5D

simulations. In order to perform three-dimensional simulations without having the shearing motion in the particles, we need to transform the equations in such a way that the steady solution of the shearing box is of null velocity. This can be done by subtracting the background state, as in Eq. D.12. However, this approach results in a modified equation for the conservation of mass. SPH, by design, cannot simulate an Euler equation where the equation of conservation of mass is modified. In the following section, we present how the steady-state shear flow can be moved to the coordinate system such that the steady-state shear flow is of null velocity in the new coordinates. This idea was motivated by trying to extend the derivation of the FARGO fast advection algorithm to SPH, which also modifies the conservation of mass. This work was performed in collaboration with Q. Vigneron (postdoctoral position at Nicolaus Copernicus University).

Note that in this section we use notations of differential geometry with Einstein's summation convention, where repeated indices in superscripts and subscripts denote implicit summation over all values. Superscripts denote contravariant components, and subscripts denote covariant ones.

4.1. General coordinate transform

We want to subtract a background velocity profile $\bar{v}^i$. We first define a shift vector S^i corresponding to the advection due to the background velocity field

$$S(x^\mu, t)^i = \int_0^t \bar{v}^i(x^\mu, t')\mathrm{d}t'. \tag{D.18}$$

We can then subtract this shift vector from the coordinates to cancel the background shear. We define the following change of coordinates:

$$\begin{cases} x^i = \tilde{x}^i + S(x^\mu, t)^i \Rightarrow v^i = \tilde{v}^i + \bar{v}^i \\ t = \tilde{t} \end{cases}, \tag{D.19}$$

where $\tilde{v}^i$ is the velocity relative to the background shear flow. Under such transform the derivative becomes:

$$\begin{cases} \partial_t = \tilde{\partial}_t - \bar{v}^i\tilde{\partial}_i \\ \partial_i = \dfrac{\partial\tilde{x}^j}{\partial x^i}\tilde{\partial}_j = \tilde{\partial}_i - \dfrac{\partial S^j}{\partial x^i}\tilde{\partial}_j = \tilde{\partial}_i - \sigma_i^j\tilde{\partial}_j \end{cases}. \tag{D.20}$$

Even if the time is unchanged by the transformation, the time derivatives are modified as the transformation is time-dependent (see Fig. D.5). In differential geometry, vectors are generally defined in the so-called tangent space as $\mathbf{V} = V^\mu\partial_\mu$. Such a definition is independent of any coordinate change, but the coordinates V^μ are subject to changes. In our case, the vectors will have to be changed as such

$$\begin{cases} V^t = \dfrac{\partial t}{\partial\tilde{x}^i}\tilde{V}^i = \tilde{V}^t \\ V^i = \dfrac{\partial x^i}{\partial\tilde{x}^j}\tilde{V}^j = \tilde{V}^i + \dfrac{\partial S^i}{\partial\tilde{x}^j}\tilde{V}^j = \tilde{V}^i + \tilde{\sigma}_j^i\tilde{V}^j \end{cases}. \tag{D.21}$$

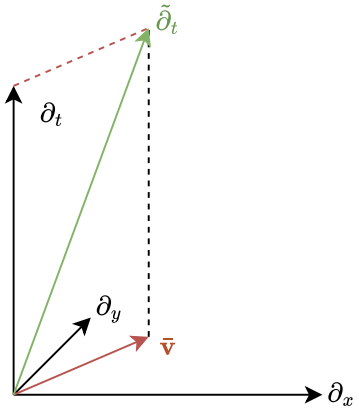


Figure D.5: Illustration of the definition of the time derivative in the modified coordinates.

4.2. The continuity equation

The previously defined coordinate transform can be applied to the continuity equation. The continuity equation in general coordinates can be written as follows:

$$\left[\partial_t + v^\mu \partial_\mu\right] \rho = -\nabla_\mu v^\mu. \tag{D.22}$$

By decomposing the velocity in the background and residual components

$$\left[\partial_t + \bar{v}^\mu \partial_\mu + v'^\mu \partial_\mu\right] \rho = -\partial_\mu \bar{v}^\mu - \partial_\mu v'^\mu. \tag{D.23}$$

We then apply the transform described by eq.D.20, D.21

$$\left[\tilde{\partial}_t + \tilde{v}^\mu \tilde{\partial}_\mu\right] \tilde{\rho} = -\tilde{\partial}_\mu \bar{v}^\mu - \tilde{\partial}_\mu \tilde{v}^\mu. \tag{D.24}$$

The resulting continuity equation here differs by a term $-\tilde{\partial}_\mu \bar{v}^\mu$. By stating that the background velocity must be divergence-free (which is the case in the shearing box), we get

$$\left[\tilde{\partial}_t + \tilde{v}^\mu \tilde{\partial}_\mu\right] \tilde{\rho} = -\tilde{\partial}_\mu \tilde{v}^\mu, \tag{D.25}$$

which is the continuity equation, but in the new coordinate system. To summarize, if we want to absorb a shearing flow as background in a transformation, by changing the velocity in a coordinate change, and keep the same continuity equation, we need the background velocity to be divergence-free.

4.3. Momentum equation

For the momentum equation, instead, we have to deal with the term $\mathbf{v} \cdot \nabla \mathbf{v}$, which requires special care. In order to proceed, we use the Lie derivative, which for a scalar f derived relative to a vector $\mathbf{A}$ is

$$(\mathcal{L}_{\mathbf{A}} f) = A^\mu \nabla_\mu f = A^\mu \partial_\mu f, \tag{D.26}$$

and the lie derivative of a vector $\mathbf{v}$ relative to the same vector $\mathbf{A}$ is

$$(\mathcal{L}_{\mathbf{A}}\mathbf{v})^{\mu} = A^{\nu}\nabla_{\nu}v^{\mu} - v^{\nu}\nabla_{\nu}A^{\mu}. \tag{D.27}$$

This operator is antisymmetric, therefor $\mathcal{L}_{\mathbf{v}}\mathbf{v} = 0$. We note that here the symbol ∇ denotes covariant derivatives. The self-advection of the velocity can be transformed as follows:

$$\begin{aligned}
\mathbf{f} &= (\partial_t + \mathbf{v}\cdot\nabla)\,\mathbf{v} && \text{(D.28)}\\
&= (\mathcal{L}_{\partial_t} + \mathbf{v}\cdot\nabla)\,\mathbf{v} && \text{(D.29)}\\
&= (\mathcal{L}_{\partial_t+\mathbf{v}} + \mathbf{v}\cdot\nabla)\,\mathbf{v} && \text{(D.30)}\\
&= \left(\mathcal{L}_{\tilde{\partial}_t+\tilde{\mathbf{v}}} + \mathbf{v}\cdot\nabla\right)\mathbf{v} && \text{(D.31)}\\
&= \left(\mathcal{L}_{\tilde{\partial}_t} + \mathcal{L}_{\tilde{\mathbf{v}}} + \mathbf{v}\cdot\nabla\right)\mathbf{v} && \text{(D.32)}\\
&= \left(\tilde{\partial}_t + \mathcal{L}_{\tilde{\mathbf{v}}} + \mathbf{v}\cdot\nabla\right)\mathbf{v} && \text{(D.33)}\\
&= \tilde{\partial}_t\mathbf{v} + \mathcal{L}_{\tilde{\mathbf{v}}}\mathbf{v} + \mathbf{v}\cdot\nabla\mathbf{v} && \text{(D.34)}\\
&= \tilde{\partial}_t\mathbf{v} + \tilde{\mathbf{v}}\cdot\nabla\mathbf{v} - \mathbf{v}\cdot\nabla\tilde{\mathbf{v}} + \mathbf{v}\cdot\nabla\mathbf{v} && \text{(D.35)}\\
&= \tilde{\partial}_t\bar{\mathbf{v}} + \tilde{\partial}_t\tilde{\mathbf{v}} + \tilde{\mathbf{v}}\cdot\nabla\bar{\mathbf{v}} - \bar{\mathbf{v}}\cdot\nabla\tilde{\mathbf{v}} + \mathbf{v}\cdot\nabla\mathbf{v} && \text{(D.36)}\\
&= \tilde{\partial}_t\bar{\mathbf{v}} + \tilde{\partial}_t\tilde{\mathbf{v}} + \tilde{\mathbf{v}}\cdot\nabla\bar{\mathbf{v}} - \bar{\mathbf{v}}\cdot\nabla\tilde{\mathbf{v}} + \bar{\mathbf{v}}\cdot\nabla\bar{\mathbf{v}} + \bar{\mathbf{v}}\cdot\nabla\tilde{\mathbf{v}} + \tilde{\mathbf{v}}\cdot\nabla\bar{\mathbf{v}} + \tilde{\mathbf{v}}\cdot\nabla\tilde{\mathbf{v}} && \text{(D.37)}\\
&= \tilde{\partial}_t\bar{\mathbf{v}} + \tilde{\partial}_t\tilde{\mathbf{v}} + 2\tilde{\mathbf{v}}\cdot\nabla\bar{\mathbf{v}} + \bar{\mathbf{v}}\cdot\nabla\bar{\mathbf{v}} + \tilde{\mathbf{v}}\cdot\nabla\tilde{\mathbf{v}} && \text{(D.38)}\\
&= \left(\tilde{\partial}_t + \tilde{\mathbf{v}}\cdot\nabla\right)\tilde{\mathbf{v}} + \left(\tilde{\partial}_t + \bar{\mathbf{v}}\cdot\nabla\right)\bar{\mathbf{v}} + 2\tilde{\mathbf{v}}\cdot\nabla\bar{\mathbf{v}}. && \text{(D.39)}
\end{aligned}$$

Hence, the self-advection can be rewritten as such, where $\mathbf{f}$ contains the pressure and external forces,

$$\left(\tilde{\partial}_t + \tilde{\mathbf{v}}\cdot\nabla\right)\tilde{\mathbf{v}} = \mathbf{f} - \left(\tilde{\partial}_t + \bar{\mathbf{v}}\cdot\nabla\right)\bar{\mathbf{v}} - 2\tilde{\mathbf{v}}\cdot\nabla\bar{\mathbf{v}}. \tag{D.40}$$

4.4. Shearing metric tensor

As used in previous sections, the divergence $\partial_\mu v^\mu$ is invariant by coordinate change. For other differential operators, such as the gradient and curl, we need to compute the metric tensor. To define the metric, we can rely on the Cartesian distance and then apply the coordinate transform. In such a case, the metric $\eta_{\mu\nu}$ is defined by the following relation:

$$\mathrm{d}s^2 = \mathrm{d}x^\mu\mathrm{d}x^\nu\delta_{\mu\nu} = \mathrm{d}\tilde{x}^\mu\mathrm{d}\tilde{x}^\nu\eta_{\mu\nu}. \tag{D.41}$$

The following computation ensues:

$$\begin{aligned}
\mathrm{d}s^2 &= \mathrm{d}x^\mu\mathrm{d}x^\nu\delta_{\mu\nu} && \text{(D.42)}\\
&= \mathrm{d}\tilde{x}^\mu\mathrm{d}\tilde{x}^\nu\delta_{\mu\nu} + \tilde{\sigma}^\mu_j\mathrm{d}\tilde{x}^j\mathrm{d}\tilde{x}^\nu\delta_{\mu\nu} + \mathrm{d}x^\mu\tilde{\sigma}^\nu_k\mathrm{d}x^k\delta_{\mu\nu} + \tilde{\sigma}^\mu_j\mathrm{d}x^j\tilde{\sigma}^\nu_k\mathrm{d}x^k\delta_{\mu\nu} && \text{(D.43)}\\
&= \mathrm{d}\tilde{x}^\mu\mathrm{d}\tilde{x}^\nu\delta_{\mu\nu} + \tilde{\sigma}^j_\mu\mathrm{d}\tilde{x}^\mu\mathrm{d}\tilde{x}^\nu\delta_{j\nu} + \mathrm{d}x^\mu\tilde{\sigma}^k_\nu\mathrm{d}x^\nu\delta_{\mu k} + \tilde{\sigma}^j_\mu\mathrm{d}x^\mu\tilde{\sigma}^k_\nu\mathrm{d}x^\nu\delta_{jk} && \text{(D.44)}\\
&= \underbrace{\left[\delta_{\mu\nu} + (\tilde{\sigma}^i_\mu\delta_{i\nu} + \tilde{\sigma}^i_\nu\delta_{\mu i}) + \tilde{\sigma}^i_\mu\tilde{\sigma}^j_\nu\delta_{ij}\right]}_{\eta_{\mu\nu}}\mathrm{d}x^\mu\mathrm{d}x^\nu. && \text{(D.45)}
\end{aligned}$$

Therefore, the shearing metric is

$$\eta_{\mu\nu} = \delta_{\mu\nu} + (\tilde{\sigma}^i_\mu \delta_{i\nu} + \tilde{\sigma}^i_\nu \delta_{\mu i}) + \tilde{\sigma}^i_\mu \tilde{\sigma}^j_\nu \delta_{ij} \tag{D.46}$$

The contravariant metric tensor $\eta^{\mu\nu}$ is defined as being the matrix inverse of the matrix defined by $\eta_{\mu\nu}$.

4.5. Cartesian shear metric

In a Cartesian shear, the shift vector is $\mathbf{S} = \mathbf{e}_y \alpha x t$, where α is the constant shear gradient $\alpha = q\Omega$. The shift vector as a function of the sheared coordinate system is unchanged from its definition, since the direction is constant along the direction of the shear flow. Therefore,

$$S^i(\tilde{x}^\mu, t) = \delta i_y \alpha \tilde{x}^x t, \tag{D.47}$$

$$S^i(x^\mu, t) = \delta i_y \alpha x^x t, \tag{D.48}$$

which results in the following Jacobian matrix for the coordinate transformation:

$$\tilde{\sigma}^i_j = \begin{pmatrix} 0 & 0 & 0 \\ \alpha t & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{D.49}$$

$$\sigma^i_j = \begin{pmatrix} 0 & 0 & 0 \\ \alpha t & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{D.50}$$

When replaced in the shearing metric tensor Eq. D.46, it yields the Cartesian shear metric tensor:

$$\eta_{\mu\nu} = \begin{pmatrix} 1 + (\alpha t)^2 & \alpha t & 0 \\ \alpha t & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{D.51}$$

This metric is explicitly dependent on time, which expresses the shearing motion of the coordinate system. Additionally, an argument can be made that the shearing box being a first-order approximation, the term $(\alpha t)^2$ can be neglected because it is of second-order (E.Lynch, private communication). Lastly, the inverse of the metric tensor is

$$\eta^{\mu\nu} = \begin{pmatrix} 1 & -\alpha t & 0 \\ -\alpha t & (\alpha t)^2 + 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{D.52}$$

4.6. Differential operators

As used in previous sections, the divergence $\partial_\mu v^\mu$ is invariant by coordinate change. For other differential operators, such as the gradient and curl, we need to use the metric tensor. To define the metric, we can rely on the Cartesian distance and then apply the coordinate transform. Using the metric tensor, we can introduce the generalized gradient

$$(\nabla A)^i = \eta^{ij}\partial_j A.$$

4.7. Euler's equation in the sheared coordinate system

In summary, the Euler equations in the sheared coordinates are:

$$\begin{aligned}
\left(\tilde{\partial}_t + \tilde{\mathbf{v}}\cdot\nabla\right)\rho &= -\nabla\cdot\tilde{\mathbf{v}},\\
\left(\tilde{\partial}_t + \tilde{\mathbf{v}}\cdot\nabla\right)\mathbf{v} &= -\frac{1}{\rho}\nabla P - \left(\tilde{\partial}_t + \bar{\mathbf{v}}\cdot\nabla\right)\bar{\mathbf{v}} - 2\tilde{\mathbf{v}}\cdot\nabla\bar{\mathbf{v}},
\end{aligned}$$

with the metric

$$\eta_{\mu\nu} = \begin{pmatrix} 1+(\alpha t)^2 & \alpha t & 0\\ \alpha t & 1 & 0\\ 0 & 0 & 1 \end{pmatrix}.$$

As the conservation of mass equation is unchanged, this allows the use of an SPH implementation in a general metric to simulate such flow without modifying density summation. SPH was indeed generalized to non-Cartesian metric tensors (see Liptai & Price (2019)). In such a framework, the particles in the test presented in Sec. 2 would be static, implying that the steady solution of the shearing box would be stable and without noise as they would not move from their stable lattice arrangement.

Two drawbacks could come from such an approach. First, when using SPH with non-Cartesian metric tensors, the Cullen & Dehnen (2010) switch is not usable as no extension of it was performed to generalized coordinates. Second, the metric being time-dependent means that the off-diagonal terms will increase over the course of the simulation to the point of reducing the method's precision over a large period of time. This last issue can be overcome by remapping the coordinates every fixed number of shearing times to avoid the time explicit terms becoming too large.

5. Summary

We have presented the shearing box formalism as well as a first implementation of it in the code Shamrock. We have shown that the shearing motion of particles results in an unstable arrangement that transitions to a random particle arrangement and noise. To solve such an issue, a first approach is to consider an axisymmetric shearing box, where the shear motion is removed by integrating the shearing direction. Using

such an approach, we obtain a system of equations that can be simulated using two-dimensional SPH. Lastly, for the 3-dimensional shearing box, we have shown that it is possible to use sheared coordinate systems to keep the conservation of mass equation unchanged while removing the shearing motion in that coordinate system. This yields modified equations of moitions in a modified metric, which can be simulated by GRSPH methods (e.g. Liptai & Price 2019; Rosswog 2010a,b). We plan on implementing both of those approaches in Shamrock, as the first one requires minimal change to the code, and the second would also provide GRSPH simulations as well as the shearing box with sheared coordinates.

References

Cullen L., Dehnen W., 2010, *Inviscid smoothed particle hydrodynamics*, MNRAS, 408, 669-683

Goldreich P., Lynden-Bell D., 1965, *II. Spiral arms as sheared gravitational instabilities*, MNRAS, 130, 125

Hawley J. F., Balbus S. A., 1991, *A Powerful Local Shear Instability in Weakly Magnetized Disks. II. Nonlinear Evolution*, ApJ, 376, 223

Hawley J. F., Gammie C. F., Balbus S. A., 1995, *Local Three-dimensional Magnetohydrodynamic Simulations of Accretion Disks*, ApJ, 440, 742

Hill G. W., 1878, *Researches in the Lunar Theory*, American Journal of Mathematics, 129

Kida S., 1981, *Motion of an elliptic vortex in a uniform shear flow*, Journal of the Physical Society of Japan, 50, 3517-3520

Laibe G., 2020, habilitation a diriger des recherches, -

Liptai D., Price D. J., 2019, *General relativistic smoothed particle hydrodynamics*, MNRAS, 485, 819-842

Price D. J., 2012, *Smoothed particle hydrodynamics and magnetohydrodynamics*, Journal of Computational Physics, 231, 759-794

Rein H., Papaloizou J. C. B., 2010, *Stochastic orbital migration of small bodies in Saturn's rings*, A&A, 524, A22

Rosswog S., 2010a, *Relativistic smooth particle hydrodynamics on a given background spacetime*, Classical and Quantum Gravity, 27, 114108

Rosswog S., 2010b, *Conservative, special-relativistic smoothed particle hydrodynamics*, Journal of Computational Physics, 229, 8591-8612

Stone J. M., Gardiner T. A., 2010, *Implementation of the Shearing Box Approximation in Athena*, ApJS, 189, 142-155

Youdin A., Johansen A., 2007, *Protoplanetary Disk Turbulence Driven by the Streaming Instability: Linear Evolution and Numerical Methods*, ApJ, 662, 613-626

Appendix E

Fast Multipole Method

Contents

Foreword

In this appendix, we detail the prototype of the FMM (Fast Multipole Method) implementation to resolve self-gravity in Shamrock. Currently, the FMM is implemented in an N-body solver using a leapfrog integrator as a prototype for the SPH implementation of self-gravity. We first present the principle of the FMM, followed by its implementation in Shamrock and corresponding tests. In the end, the current implementation seems to offer significant performance, and its extension to multiple GPUs needs to be finalized.

1. Fast Multipole Method

1.1. Solved equations

In a self-gravitating system, the gravitational force f_{SG} is related to the local density of the medium via the Poisson equation

$$\mathbf{f}_{\text{SG}} = -\boldsymbol{\nabla}\Phi, \tag{E.1}$$

where the gravitational potential Φ is given as the solution of the Poisson equation

$$\triangle\Phi = 4\pi\mathcal{G}\rho, \tag{E.2}$$

where $\mathcal{G}$ is the gravitational constant and ρ the density of the fluid. Assuming free boundary conditions, the green function G of the linear equation Eq. E.2, which is

a solution of the Poisson equation for a Dirac function as density is

$$G(\mathbf{x}) = -\frac{\mathcal{G}m}{||\mathbf{x}||}. \tag{E.3}$$

The gravitational potential can subsequently be expressed as the following convolution

$$\phi(\mathbf{x}) = \iiint_V \rho(\mathbf{x}_j) G(\mathbf{x} - \mathbf{x}_j).$$

A difficulty of the Poisson equation in numerical code is that the equation is elliptic, implying instantaneous long-range interactions between all fluid elements.

1.2. Basic multipole expansion

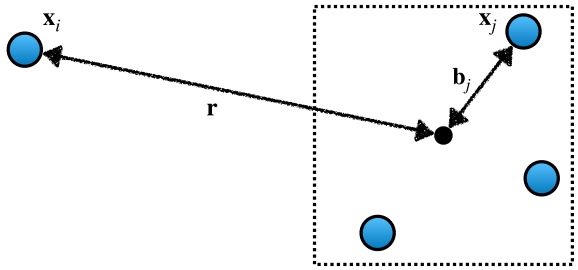


Figure E.1: Illustration of the variables in a basic multipole expansion. We have two particles at positions $\mathbf{x}_i$ and $\mathbf{x}_j$, particle j is in a box and its relative position to the box center is $\mathbf{b}_j$. The relative position of the particle i to the box is $\mathbf{r}$.

In tree-based N-body codes (as in SHAMROCK), the particles are organized together in the cells of a tree. In order to avoid computing the gravitational force for any pair of particles (complexity of order $\mathcal{O}(N^2)$), one can instead approximate the solution by considering the gravitational interaction with a group of particles that corresponds to a tree cell. This corresponds to the situation shown in Fig. E.1. Assuming that the box size is small compared to the distance to the particle (e.g. $||\mathbf{r}|| \gg ||\mathbf{b}_j||$), the Green function can be expanded into multipoles, to the order p in powers of $||\mathbf{b}_j||/||\mathbf{r}||$:

$$\begin{aligned} G(\mathbf{x}_i - \mathbf{x}_j) &= G(\mathbf{r} + \mathbf{b}_j) \\ &\simeq \sum_{n=0}^{p} \frac{1}{n!} \nabla_r^{(n)} G(\mathbf{r}) \cdot \mathbf{b}_j^{(n)}. \end{aligned}$$

Using that expansion in the solution of the Poisson equation Eq. E.4, the integral

can be moved inside the sum as such:

$$\begin{aligned}\phi(\mathbf{x}_i) &= \iiint_V \rho(\mathbf{x}_j) G(\mathbf{x}_i - \mathbf{x}_j)\ \mathrm{d}^3\mathbf{x}_j \\ &\simeq \iiint_V \rho(\mathbf{x}_j) \sum_{n=0}^{p} \frac{1}{n!} \nabla_r^{(n)} G(\mathbf{r}) \cdot \mathbf{b}_j^{(n)}\ \mathrm{d}^3\mathbf{x}_j \\ &= \sum_{n=0}^{p} \frac{1}{n!} \underbrace{\nabla_r^{(n)} G(\mathbf{r})}_{D_n} \cdot \underbrace{\left(\iiint_V \rho(\mathbf{x}_j) \mathbf{b}_j^{(n)}\ \mathrm{d}^3\mathbf{x}_j \right)}_{Q_n^B},\end{aligned}$$

where D_n are the gradients of the Green function and Q_n^B are the moments of the mass distribution. Hence, the force can then be written as follows:

$$f_{\mathrm{g}}(\mathbf{x}_i) = \nabla\phi(\mathbf{x}_i) = \sum_{n=0}^{p} \frac{1}{n!} D_{n+1} \cdot Q_n^B.$$

For point-mass particles, the moments reduce to

$$Q_n^B = \sum_j m_j \mathbf{b}_j^{(n)}.$$

It is important to note here that the gravitational field of an SPH particle is not the gravitational field of a point mass particle, since gravity is regularized close to the particle to avoid numerical divergences associated to the singularity at the origin on the gravitational potential. This is traditionally accounted for by modifying the Green function using so-called softened gravity, where the Green function is taken to be the solution of the Poisson equation where the source term is an SPH kernel instead of the Dirac function instead of modifying the multipole (e.g. Springel et al. 2021; Price et al. 2018).

In the basic multipole expansion method, rather than evaluating the Green function for each pair of particles, we compute it once for each pair of cell-particles, assuming the multipole moments are known.

1.3. Fast Multiple Method

The Fast Multiple Method (FMM) is an alternative method to reduce the number of evaluations of the Green function to once per pair of cells instead of once per pair of particle-cells. Consider two cells as represented on Fig. E.2, we now have two cells. The distance between the two particles i and j is $\mathbf{x}_i - \mathbf{x}_j$, which can be rewritten as $\mathbf{r} + (\mathbf{b}_j - \mathbf{a}_i)$ here. Under the assumption $||(\mathbf{b}_j - \mathbf{a}_i)|| \ll ||\mathbf{r}||$, one can perform an expansion with respect to $\mathbf{a}_i$ and $\mathbf{b}_j$ instead of only $\mathbf{b}_j$ as done for the basic multipole expansion.

Firstly, for two vectors $\mathbf{a}_i$ and $\mathbf{b}_j$ and T a symmetric tensor of order larger than n, we have the following equality:

$$T \cdot (\mathbf{b}_j + \mathbf{a}_i)^{(n)} = T \cdot \sum_{k=0}^{n} \binom{n}{k} \mathbf{b}_j^{(n-k)} \mathbf{a}_i^{(k)},$$

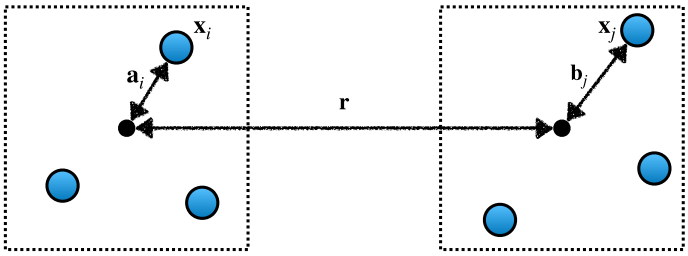


Figure E.2: Illustration of the variables involved in a fast multipole method approach.

where the superscript (n) denotes the operation performing n times the tensorial product of a vector with itself.

Using that expression, the Green function can be developed according to

$$\begin{aligned}
G(\mathbf{x}_i - \mathbf{x}_j) &= G(\mathbf{r} + \mathbf{b}_j - \mathbf{a}_i) \\
&\simeq \sum_{n=0}^{p} \frac{1}{n!} \nabla_r^{(n)} G(\mathbf{r}) \cdot (\mathbf{b}_j - \mathbf{a}_i)^{(n)} \\
&= \sum_{n=0}^{p} \frac{1}{n!} \nabla_r^{(n)} G(\mathbf{r}) \cdot \sum_{k=0}^{n} \binom{n}{k} \mathbf{b}_j^{(n-k)} \, (-\mathbf{a}_i)^{(k)} \\
&= \sum_{k=0}^{p} \frac{(-1)^k}{k!} \mathbf{a}_i^{(k)} \cdot \sum_{n=0}^{p-k} \frac{1}{n!} \nabla_r^{(n+k)} G(\mathbf{r}) \cdot \mathbf{b}_j^{(n)} \\
&= \sum_{k=0}^{p} \frac{(-1)^k}{k!} \mathbf{a}_i^{(k)} \cdot \sum_{n=0}^{p-k} \frac{1}{n!} D_{n+k} \cdot \mathbf{b}_j^{(n)}.
\end{aligned}$$

From Eq. E.4, and applying the same procedure as above, one obtains

$$\begin{aligned}
\phi(\mathbf{x}_i) &= \iiint_V \rho(\mathbf{x}_j) G(\mathbf{x}_i - \mathbf{x}_j) \, \mathrm{d}^3\mathbf{x}_j \\
&\simeq \iiint_V \rho(\mathbf{x}_j) \sum_{k=0}^{p} \frac{(-1)^k}{k!} \mathbf{a}_i^{(k)} \cdot \sum_{n=0}^{p-k} \frac{1}{n!} D_{n+k} \cdot \mathbf{b}_j^{(n)} \, \mathrm{d}^3\mathbf{x}_j \\
&= \sum_{k=0}^{p} \frac{(-1)^k}{k!} \mathbf{a}_i^{(k)} \cdot \sum_{n=0}^{p-k} \frac{1}{n!} D_{n+k} \cdot \iiint_V \rho(\mathbf{x}_j) \mathbf{b}_j^{(n)} \, \mathrm{d}^3\mathbf{x}_j \\
&= \sum_{k=0}^{p} \frac{(-1)^k}{k!} \mathbf{a}_i^{(k)} \cdot \underbrace{\sum_{n=0}^{p-k} \frac{1}{n!} D_{n+k} \cdot Q_n^B}_{\text{symetrical tensor}} \\
&= \sum_{k=0}^{p} \mathbf{a}_i^{(k)} \cdot \underbrace{\frac{(-1)^k}{k!} \sum_{n=0}^{p-k} \frac{1}{n!} D_{n+k} \cdot Q_n^B}_{M_k},
\end{aligned}$$

and the corresponding gravitational force is:

$$f_{\mathrm{g}}(\mathbf{x}_i) = \iiint_V \rho(\mathbf{x}_j) \nabla G(\mathbf{x}_i - \mathbf{x}_j)\ \mathrm{d}^3\mathbf{x}_j \tag{E.4}$$

$$\simeq \iiint_V \rho(\mathbf{x}_j) \sum_{k=0}^{p} \frac{(-1)^k}{k!} \mathbf{a}_i^{(k)} \cdot \sum_{n=0}^{p-k} \frac{1}{n!} D_{n+k+1} \cdot \mathbf{b}_j^{(n)}\ \mathrm{d}^3\mathbf{x}_j \tag{E.5}$$

$$= \sum_{k=0}^{p} \frac{(-1)^k}{k!} \mathbf{a}_i^{(k)} \cdot \sum_{n=0}^{p-k} \frac{1}{n!} D_{n+k+1} \cdot \iiint_V \rho(\mathbf{x}_j) \mathbf{b}_j^{(n)}\ \mathrm{d}^3\mathbf{x}_j \tag{E.6}$$

$$= \sum_{k=0}^{p} \frac{(-1)^k}{k!} \mathbf{a}_i^{(k)} \cdot \underbrace{\sum_{n=0}^{p-k} \frac{1}{n!} D_{n+k+1} \cdot Q_n^B}_{\text{symetrical tensor}} \tag{E.7}$$

$$= \sum_{k=0}^{p} \mathbf{a}_i^{(k)} \cdot \underbrace{\frac{(-1)^k}{k!} \sum_{n=0}^{p-k} \frac{1}{n!} D_{n+k+1} \cdot Q_n^B}_{dM_k}. \tag{E.8}$$

Eq. E.8 implies that the Green function is required to be evaluated only once per pair of cells. Lastly, the expansion relies on the development in order of $||(\mathbf{b}_j - \mathbf{a}_i)||/||\mathbf{r}||$. This implies that, provided an upper bound of $||(\mathbf{b}_j - \mathbf{a}_i)||$, it is possible to quantify the error of the FMM compared to the analytical solution. Springel et al. (2021) gives the following angle to use as a criterion:

$$\theta = \frac{L_a + L_b}{r}, \tag{E.9}$$

where L_a and L_b are the side lengths of the boxes a and b, respectively, and r is the distance between the boxes. Using a critical angle θ_c, the FMM expansion is allowed only if $\theta < \theta_c$; otherwise, direct pairwise summation is used. This results in an upper bound in the error of the expansion (see Sec. 4).

2. Moment translation & recombination

We now have detailed the procedure behind the FMM. However, as in N-body codes based on tree algorithms, for large cells, one may want to combine moments of chlidren cells instead of computing them directly from the contained particles (see Fig. E.3). Springel et al. (2021) state using such a technique, but the details of the procedure are not mentioned. In Shamrock we have derived an analytical procedure to combine moments, who is not, to our knowledge explicitely described in the literature.

First, we define the following vectors: for a particle at position $\mathbf{x}$, its relative position to the center of the child cell is $\mathbf{b}$, its relative position to the center of the parent cell is $\mathbf{b}'$, and lastly, the offset from the center of the child cell to the center of the parent cell is $\mathbf{g}$. Here we have $\mathbf{b}' = \mathbf{b} + \mathbf{g}$.

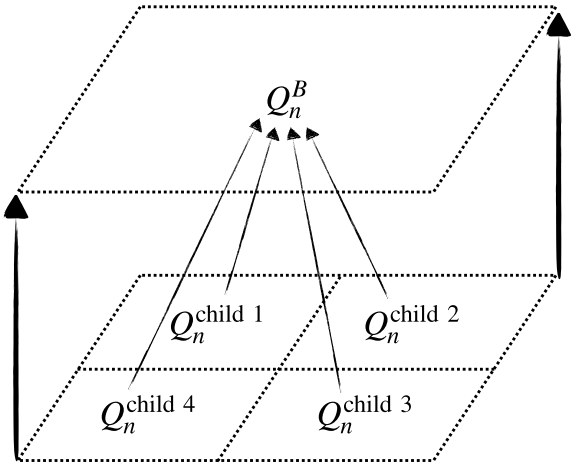


Figure E.3: Illustration of the principle of moment recombination.

The moment of the parent cell B' is

$$\begin{aligned} Q_n^{B'} &= \iiint_V \rho(\mathbf{x}_j)\mathbf{b}'^{(n)}\ \mathrm{d}^3\mathbf{x} \\ &= \iiint_V \rho(\mathbf{x}_j)\,(\mathbf{g}+\mathbf{b})^{(n)}\ \mathrm{d}^3\mathbf{x} \\ &= \iiint_V \rho(\mathbf{x}_j)\,\underbrace{(\mathbf{g}+\mathbf{b})\otimes\cdots\otimes(\mathbf{g}+\mathbf{b})}_{n\text{ times}}\ \mathrm{d}^3\mathbf{x}_j. \end{aligned}$$

We now aim to rewrite each order of the moments as a function of the moments Q_n^B of the child cell B by factorizing in powers of $\mathbf{g}$. The moment of the parent will simply be the sum of the contributions of the moments of the children translated to the parent cell center. We now provide the moment-translation formulas up to the fifth order.

Order 0

$$Q_0^{B'} = \iiint_V \rho(\mathbf{x}_j)\ \mathrm{d}^3\mathbf{x} = Q_0^B.$$

Order 1

$$\begin{aligned} Q_1^{B'} &= \iiint_V \rho(\mathbf{x}_j)\mathbf{b}'\ \mathrm{d}^3\mathbf{x} \\ &= \iiint_V \rho(\mathbf{x}_j)\,(\mathbf{g}+\mathbf{b})\ \mathrm{d}^3\mathbf{x} \\ &= \mathbf{g}\iiint_V \rho(\mathbf{x}_j)\ \mathrm{d}^3\mathbf{x} + \iiint_V \rho(\mathbf{x}_j)\mathbf{b}\ \mathrm{d}^3\mathbf{x} \\ &= \mathbf{g}\otimes Q_0^B + Q_1^B. \end{aligned}$$

Order 2

$$\begin{aligned}
Q_2^{B'} &= \iiint_V \rho(\mathbf{x}_j)\mathbf{b}'^{(2)}\ \mathrm{d}^3\mathbf{x} \\
&= \iiint_V \rho(\mathbf{x}_j)\,(\mathbf{g}+\mathbf{b})\otimes(\mathbf{g}+\mathbf{b})\ \mathrm{d}^3\mathbf{x}_j \\
&= \iiint_V \rho(\mathbf{x}_j)\left(\underbrace{\mathbf{gg}+\mathbf{gb}}_{\mathbf{g}Q_1^{B'}}+\underbrace{\mathbf{bg}}_{Q_1^B\mathbf{g}}+\underbrace{\mathbf{bb}}_{Q_2^B}\right)\ \mathrm{d}^3\mathbf{x}_j \\
&= \mathbf{g}\otimes Q_1^{B'}+Q_1^B\otimes\mathbf{g}+Q_2^B.
\end{aligned}$$

Order 3

$$\begin{aligned}
Q_3^{B'} &= \iiint_V \rho(\mathbf{x}_j)\,(\mathbf{g}+\mathbf{b})\otimes(\mathbf{g}+\mathbf{b})\otimes(\mathbf{g}+\mathbf{b})\ \mathrm{d}^3\mathbf{x} \\
&= \iiint_V \rho(\mathbf{x}_j)\left(\underbrace{\mathbf{ggg}+\mathbf{ggb}+\mathbf{gbg}+\mathbf{gbb}}_{\mathbf{g}Q_2^{B'}}+\mathbf{bgg}+\mathbf{bgb}+\mathbf{bbg}+\mathbf{bbb}\right)\ \mathrm{d}^3\mathbf{x}_j \\
&= \mathbf{g}\otimes Q_2^{B'}+\iiint_V \rho(\mathbf{x}_j)\,(\mathbf{bgg}+\mathbf{bgb}+\mathbf{bbg}+\mathbf{bbb})\ \mathrm{d}^3\mathbf{x}_j.
\end{aligned}$$

Using tensorial notations, we can further simplify the expression:

$$\begin{aligned}
Q_3^{B'}{}_{\mu\nu\delta} &= \mathbf{g}_\mu Q_2^{B'}{}_{\nu\delta}+\iiint_V \rho(\mathbf{x}_j)\left(\mathbf{b}_\mu\mathbf{g}_\nu\mathbf{g}_\delta+\mathbf{b}_\mu\mathbf{g}_\nu\mathbf{b}_\delta+\underbrace{\mathbf{b}_\mu\mathbf{b}_\nu\mathbf{g}_\delta}_{Q_2^B{}_{\mu\nu}\mathbf{g}_\delta}+\underbrace{\mathbf{b}_\mu\mathbf{b}_\nu\mathbf{b}_\delta}_{Q_3^B{}_{\mu\nu\delta}}\right)\ \mathrm{d}^3\mathbf{x}_j \\
&= \mathbf{g}_\mu Q_2^{B'}{}_{\nu\delta}+\iiint_V \rho(\mathbf{x}_j)\,(\mathbf{b}_\mu\mathbf{g}_\delta\mathbf{g}_\nu+\mathbf{b}_\mu\mathbf{b}_\delta\mathbf{g}_\nu)\ \mathrm{d}^3\mathbf{x}_j+Q_2^B{}_{\mu\nu}\mathbf{g}_\delta+Q_3^B{}_{\mu\nu\delta} \\
&= \mathbf{g}_\mu Q_2^{B'}{}_{\nu\delta}+\mathbf{g}_\nu\iiint_V \rho(\mathbf{x}_j)\,(\mathbf{b}_\mu\mathbf{g}_\delta+\mathbf{b}_\mu\mathbf{b}_\delta)\ \mathrm{d}^3\mathbf{x}_j+Q_2^B{}_{\mu\nu}\mathbf{g}_\delta+Q_3^B{}_{\mu\nu\delta} \\
&= \mathbf{g}_\mu Q_2^{B'}{}_{\nu\delta}+\mathbf{g}_\nu\left(Q_2^{B'}{}_{\mu\delta}-\mathbf{g}_\mu Q_1^{B'}{}_{\delta}\right)+Q_2^B{}_{\mu\nu}\mathbf{g}_\delta+Q_3^B{}_{\mu\nu\delta}.
\end{aligned}$$

Order 4

$$\begin{aligned}
Q_4^{B'} &= \iiint_V \rho(\mathbf{x}_j)\,(\mathbf{g}+\mathbf{b})\otimes(\mathbf{g}+\mathbf{b})\otimes(\mathbf{g}+\mathbf{b})\otimes(\mathbf{g}+\mathbf{b})\ \mathrm{d}^3\mathbf{x} \\
&= \iiint_V \rho(\mathbf{x}_j)\Big(\underbrace{\mathbf{gggg}+\mathbf{gggb}+\mathbf{ggbg}+\mathbf{ggbb}+\mathbf{gbgg}+\mathbf{gbgb}+\mathbf{gbbg}+\mathbf{gbbb}}_{\mathbf{g}\otimes Q_3^{B'}}+ \\
&\qquad \mathbf{bggg}+\mathbf{bggb}+\mathbf{bgbg}+\mathbf{bgbb}+\mathbf{bbgg}+\mathbf{bbgb}+\mathbf{bbbg}+\mathbf{bbbb}\Big)\ \mathrm{d}^3\mathbf{x}
\end{aligned}$$

$$Q_4^{B'}{}_{\mu\nu\delta\epsilon} = \mathbf{g}_\mu Q_3^{B'}{}_{\nu\delta\epsilon} + \iiint_V \rho(\mathbf{x}_j)\,(\,\mathbf{bggg}+\mathbf{bggb}+\mathbf{bgbg}+\mathbf{bgbb}+ \mathbf{bbgg}+\mathbf{bbgb}+\mathbf{bbbg}+\underbrace{\mathbf{bbbb}}_{Q_4^B{}_{\mu\nu\delta\epsilon}}\,)_{\mu\nu\delta\epsilon}\ \mathrm{d}^3\mathbf{x}$$

$$\begin{aligned}
=&\mathbf{g}_\mu Q_3^{B'}{}_{\nu\delta\epsilon} + \iiint_V \rho(\mathbf{x}_j) \\
&(\ \mathbf{bggg}+\mathbf{bggb}+\mathbf{bgbg}+\mathbf{bgbb}+\mathbf{bbgg}+\mathbf{bbgb}+\mathbf{bbbg}\ )_{\mu\nu\delta\epsilon} \\
&\mathrm{d}^3\mathbf{x} + Q_4^B{}_{\mu\nu\delta\epsilon} \\
=&\mathbf{g}_\mu Q_3^{B'}{}_{\nu\delta\epsilon} + \iiint_V \rho(\mathbf{x}_j) \\
&(\ \mathbf{gggb}+\mathbf{ggbb}+\mathbf{bggb}+\mathbf{bgbb}+\mathbf{gbgb}+\mathbf{gbbb}+\mathbf{bbgb}\ )_{\nu\delta\epsilon\mu} \\
&\mathrm{d}^3\mathbf{x} + Q_4^B{}_{\mu\nu\delta\epsilon} \\
=&\mathbf{g}_\mu Q_3^{B'}{}_{\nu\delta\epsilon} + \iiint_V \rho(\mathbf{x}_j)\,(\\
&\mathbf{g}_\delta\,(\mathbf{ggb}+\mathbf{gbb}+\mathbf{bgb}+\mathbf{bbb})_{\nu\epsilon\mu} + (\mathbf{gbgb}+\mathbf{gbbb}+\mathbf{bbgb})_{\nu\delta\epsilon\mu} \\
&)\ \mathrm{d}^3\mathbf{x} + Q_4^B{}_{\mu\nu\delta\epsilon} \\
=&\mathbf{g}_\mu Q_3^{B'}{}_{\nu\delta\epsilon} + \iiint_V \rho(\mathbf{x}_j)\,(\\
&\mathbf{g}_\delta \underbrace{(\mathbf{ggb}+\mathbf{gbb}+\mathbf{bgb}+\mathbf{bbb})_{\nu\epsilon\mu}}_{\left(Q_3^{B'}-\mathbf{g}\otimes Q_2^{B'}\right)_{\mu\epsilon\nu}} + (\mathbf{ggbb}+\mathbf{gbbb}+\mathbf{bgbb})_{\nu\epsilon\delta\mu} \\
&)\ \mathrm{d}^3\mathbf{x} + Q_4^B{}_{\mu\nu\delta\epsilon} \\
=&\mathbf{g}_\mu Q_3^{B'}{}_{\nu\delta\epsilon} + \mathbf{g}_\delta\left(Q_3^{B'} - \mathbf{g}\otimes Q_2^{B'}\right)_{\mu\epsilon\nu} + \iiint_V \rho(\mathbf{x}_j)\,(\\
&\underbrace{\mathbf{g}_\epsilon\mathbf{g}_\nu\mathbf{b}_\delta\mathbf{b}_\mu + \mathbf{g}_\epsilon\mathbf{b}_\nu\mathbf{b}_\delta\mathbf{b}_\mu}_{\mathbf{g}^{(2)}_{\epsilon\nu}Q_2^B{}_{\delta\mu}+\mathbf{g}_\epsilon Q_3^B{}_{\nu\delta\mu}} + \underbrace{\mathbf{g}_\nu\mathbf{b}_\delta\mathbf{b}_\epsilon\mathbf{b}_\mu}_{\mathbf{g}_\nu Q_3^B{}_{\delta\epsilon\mu}} \\
&)\ \mathrm{d}^3\mathbf{x} + Q_4^B{}_{\mu\nu\delta\epsilon}.
\end{aligned}$$

This computation yields the following result for the 4th order moment:

$$\begin{aligned}
Q_4^{B'}{}_{\mu\nu\delta\epsilon} = &\mathbf{g}_\mu Q_3^{B'}{}_{\nu\delta\epsilon} + \mathbf{g}_\delta\left(Q_3^{B'}{}_{\mu\epsilon\nu} - \mathbf{g}_\mu Q_2^{B'}{}_{\mu\epsilon\nu}\right) + \mathbf{g}^{(2)}_{\epsilon\nu} Q_2^B{}_{\delta\mu} \\
&+ \mathbf{g}_\epsilon Q_3^B{}_{\nu\delta\mu} + \mathbf{g}_\nu Q_3^B{}_{\delta\epsilon\mu} + Q_4^B{}_{\mu\nu\delta\epsilon}.
\end{aligned}$$

Order 5

$$\begin{aligned}
Q_5^{B'}{}_{\mu\nu\delta\epsilon\sigma} =&\mathbf{g}_\mu Q_4^{B'}{}_{\nu\delta\epsilon\sigma} + \iiint_V \rho(\mathbf{x}_j)\,(\\
&\mathbf{bgggg}+\mathbf{bgggb}+\mathbf{bggbg}+\mathbf{bggbb}+\mathbf{bgbgg}+\mathbf{bgbgb}+\mathbf{bgbbg}+\mathbf{bgbbb}+ \\
&\mathbf{bbggg}+\mathbf{bbggb}+\mathbf{bbgbg}+\mathbf{bbgbb}+\mathbf{bbbgg}+\mathbf{bbbgb}+\mathbf{bbbbg}+\mathbf{bbbbb}\ )_{\mu\nu\delta\epsilon\sigma}\ \mathrm{d}^3\mathbf{x}
\end{aligned}$$

$$
\begin{aligned}
Q_5^{B'}{}_{\mu\nu\delta\epsilon\sigma} =& \mathbf{g}_\mu Q_4^{B'}{}_{\nu\delta\epsilon\sigma} + \iiint_V \rho(\mathbf{x}_j) \\
& (\mathbf{bgggg} + \mathbf{bgggb} + \mathbf{bggbg} + \mathbf{bggbb} + \mathbf{bgbgg} + \mathbf{bgbgb} + \mathbf{bgbbg} + \mathbf{bgbbb})_{\mu\nu\delta\epsilon\sigma} + \\
& (\mathbf{bbggg} + \mathbf{bbggb} + \mathbf{bbgbg} + \mathbf{bbgbb} + \mathbf{bbbgg} + \mathbf{bbbgb} + \mathbf{bbbbg} + \mathbf{bbbbb})_{\mu\nu\delta\epsilon\sigma} \ \mathrm{d}^3\mathbf{x}
\end{aligned}
$$

$$
\begin{aligned}
Q_5^{B'}{}_{\mu\nu\delta\epsilon\sigma} =& \mathbf{g}_\mu Q_4^{B'}{}_{\nu\delta\epsilon\sigma} + \iiint_V \rho(\mathbf{x}_j) \\
& (\mathbf{gbggg} + \mathbf{gbggb} + \mathbf{gbgbg} + \mathbf{gbgbb} + \mathbf{gbbgg} + \mathbf{gbbgb} + \mathbf{gbbbg} + \mathbf{gbbbb})_{\nu\mu\delta\epsilon\sigma} + \\
& (\mathbf{bbggg} + \mathbf{bbggb} + \mathbf{bbgbg} + \mathbf{bbgbb} + \mathbf{bbbgg} + \mathbf{bbbgb} + \mathbf{bbbbg} + \mathbf{bbbbb})_{\mu\nu\delta\epsilon\sigma} \ \mathrm{d}^3\mathbf{x}
\end{aligned}
$$

$$
\begin{aligned}
Q_5^{B'}{}_{\mu\nu\delta\epsilon\sigma} =& \mathbf{g}_\mu Q_4^{B'}{}_{\nu\delta\epsilon\sigma} + \mathbf{g}_\nu \left(Q_4^{B'} - \mathbf{g} \otimes Q_3^{B'}\right)_{\mu\delta\epsilon\sigma} + \iiint_V \rho(\mathbf{x}_j) \\
& (\mathbf{bbggg} + \mathbf{bbggb} + \mathbf{bbgbg} + \mathbf{bbgbb} + \mathbf{bbbgg} + \mathbf{bbbgb} + \mathbf{bbbbg} + \mathbf{bbbbb})_{\mu\nu\delta\epsilon\sigma} \ \mathrm{d}^3\mathbf{x}
\end{aligned}
$$

$$
\begin{aligned}
Q_5^{B'}{}_{\mu\nu\delta\epsilon\sigma} =& \mathbf{g}_\mu Q_4^{B'}{}_{\nu\delta\epsilon\sigma} + \mathbf{g}_\nu \left(Q_4^{B'} - \mathbf{g} \otimes Q_3^{B'}\right)_{\mu\delta\epsilon\sigma} + \iiint_V \rho(\mathbf{x}_j) \\
& \Bigg(\underbrace{\mathbf{bbggg}}_{Q_2^B{}_{\mu\nu}\mathbf{g}^3_{\delta\epsilon\sigma}} + \underbrace{\mathbf{bbggb}}_{Q_3^B{}_{\mu\nu\sigma}\mathbf{g}^2_{\delta\epsilon}} + \underbrace{\mathbf{bbbgg}}_{Q_3^B{}_{\mu\nu\delta}\mathbf{g}^2_{\epsilon\sigma}} + \underbrace{\mathbf{bbbgb}}_{Q_4^B{}_{\mu\nu\delta\sigma}\mathbf{g}_\epsilon} + \underbrace{\mathbf{bbgbg}}_{Q_3^B{}_{\mu\nu\epsilon}\mathbf{g}^2_{\delta\sigma}} + \underbrace{\mathbf{bbgbb}}_{Q_4^B{}_{\mu\nu\epsilon\sigma}\mathbf{g}_\delta} + \underbrace{\mathbf{bbbbg}}_{Q_4^B{}_{\mu\nu\delta\epsilon}\mathbf{g}_\sigma} + \underbrace{\mathbf{bbbbb}}_{Q_5^B{}_{\mu\nu\delta\epsilon\sigma}} \Bigg)_{\mu\nu\delta\epsilon\sigma} \mathrm{d}^3\mathbf{x}.
\end{aligned}
$$

Therefore, we have the following result for the fifth order moment translation:

$$
\begin{aligned}
Q_5^{B'}{}_{\mu\nu\delta\epsilon\sigma} =& \mathbf{g}_\mu Q_4^{B'}{}_{\nu\delta\epsilon\sigma} + \mathbf{g}_\nu \left(Q_4^{B'} - \mathbf{g} \otimes Q_3^{B'}\right)_{\mu\delta\epsilon\sigma} \\
& + Q_2^B{}_{\mu\nu}\mathbf{g}^3_{\delta\epsilon\sigma} + Q_3^B{}_{\mu\nu\sigma}\mathbf{g}^2_{\delta\epsilon} + Q_3^B{}_{\mu\nu\delta}\mathbf{g}^2_{\epsilon\sigma} + Q_4^B{}_{\mu\nu\delta\sigma}\mathbf{g}_\epsilon \\
& + Q_3^B{}_{\mu\nu\epsilon}\mathbf{g}^2_{\delta\sigma} + Q_4^B{}_{\mu\nu\epsilon\sigma}\mathbf{g}_\delta + Q_4^B{}_{\mu\nu\delta\epsilon}\mathbf{g}_\sigma + Q_5^B{}_{\mu\nu\delta\epsilon\sigma}
\end{aligned}
$$

All the translation formulas presented above were implemented and tested in Shamrock. Those translation formulas are exact down to floating-point precision. In the Shamrock test suite, we test for the difference between directly computing the moment of a cell and computing for another cell, then translating the result to the wanted cell. In relative errors, they are tested to be lower than 10^{-20}.

3. Implementation

We now detail the implementation performed in Shamrock of the FMM method to solve for self-gravity on a single GPU.

3.1. Symmetric tensors

In the FMM algorithm, we only manipulate symmetric tensors. To program the corresponding formulas, we have implemented a class for symmetric tensors as well as the associated contraction operators. During the implementation, we regularly

checked the generated assembly to ensure that the compiler was properly vectorizing the tensorial operators. Additionally, we also provide a so-called tensor collection class, which is a wrapper for a list of tensors of different ranks. We show here a snippet from the implementation in SHAMROCK of the function computing the dM_k tensor defined Eq. E.8.

Exemple of the use of the symetric tensor class

```
template<class T>
inline SymTensorCollection<T,1,5> get_dM_mat(
    SymTensorCollection<T,1,5> & D, SymTensorCollection<T,0,4> & Q
    ){

    SymTensor3d_1<T> & TD1 = D.t1;
    SymTensor3d_2<T> & TD2 = D.t2;
    SymTensor3d_3<T> & TD3 = D.t3;
    SymTensor3d_4<T> & TD4 = D.t4;
    SymTensor3d_5<T> & TD5 = D.t5;

    T & TQ0 = Q.t0;
    SymTensor3d_1<T> & TQ1 = Q.t1;
    SymTensor3d_2<T> & TQ2 = Q.t2;
    SymTensor3d_3<T> & TQ3 = Q.t3;
    SymTensor3d_4<T> & TQ4 = Q.t4;

    constexpr T _1i2 = (1./2.);
    constexpr T _1i6 = (1./6.);
    constexpr T _1i24 = (1./24.);

    auto M_1 = TD1 * TQ0 + TD2 * TQ1 + (TD3 * TQ2)*_1i2
                    + (TD4 * TQ3)*_1i6 + (TD5 * TQ4)*_1i24;
    auto M_2 = (T(-1.)*TD2 * TQ0) - TD3 * TQ1
                    - (TD4 * TQ2)*_1i2 - (TD5 * TQ3)*_1i6;
    auto M_3 = _1i2 * (TD3 *TQ0 + TD4 *TQ1 + (TD5 * TQ2)*_1i2);
    auto M_4 = _1i6 * ((T(-1.)*(TD4 *TQ0)) - TD5 *TQ1);
    auto M_5 = (TD5 * TQ0)*_1i24;

    return SymTensorCollection<T, 1, 5>{
        M_1,M_2,M_3,M_4,M_5
    };

}
```

The use of the tensor class ensures the absence of errors in their use, as well as proper vectorization and convenience in their usage.

3.2. Computing moments (Upward step)

As presented in Alg. 10, we can compute the moments by first computing the ones of the leaves directly from the particles and then translating the children moments

Algorithm 10: Simplified pseudocode of the moment compute step

Data: Cells i of the tree, $\mathbf{c}_i$ the associated cell center, and Q_n^i the associated moment of order n (initialized to zeros on all components). The moments are computed up to order p. $\mathbf{x}_a$ is the position of particle a, and m_a is its mass.

Result: The computed moments Q_n^i.

```
// Accumulate moments in leafs
foreach leafs cells i, in parallel do
    foreach particle a in cell i do
        for n ∈ [0, p] do
            Q_n^i ← Q_n^i + m_a(x_a − c_i)^(n);
// Propagate moments up the tree
foreach tree level l, decreasing from the lowest one do
    foreach cells i in level l, in parallel do
        foreach cells j child of cell i do
            for n ∈ [0, p] do
                T ← translate moments Q_n^j from center c_j to c_i;
                Q_n^i ← Q_n^i + T;
```

to combine them in the parent cell up the tree recursively.

3.3. Computing force (Downward step)

As mentioned, the FMM is valid only up to a critical angle (θ_c) defined by the user. As performed in Springel et al. (2021), the procedure goes as follows. To accumulate the contribution of self-gravity in a leaf cell of a tree, we start at the root node of the tree. If the angle is not satisfied, we go in with the children of that cell and check the angle of the children, repeating the same process. We recursively go down the tree until either the critical angle is satisfied, in which case we use the multipole expansion, or the cell is a leaf, in which case we perform a pairwise summation. Note that the target leaf cell will also accumulate its own contribution in such a procedure. The procedure is illustrated in Fig. E.4. and can be summarized as presented in Alg. 11. This procedure is a modification of the traversal algorithm normally used in Shamrock, where instead of discarding a leaf if the interaction criterion is not satisfied, we instead perform a long-range multipole summation of the force. In the pseudo-code shown in Alg. 11, we have in practice optimized the indexing logic and performed some optimizations to minimize the memory movements. In general, the FMM can be summarized as first propagating the moments upward the tree, then performing the downward step to compute the forces for each leaf cells.

Algorithm 11: Simplified pseudocode of the downward step of the FMM

Data: Cells i of the tree, $\mathbf{c}_i$ the associated cell center, and Q_n^i the associated moment of order n. The order of the FMM multipoles p to use. $\mathbf{x}_a$ the position of particle a.

Result: The gravitational force $\mathbf{f}_i$ applied on the particle i.

```
1  foreach leafs cells a, in parallel do
       // Setup cell stack
2      s ← {};
       // Enqueue the root node
3      push 0 on stack s;
4      do
           // Pop top of the stack
5          j ← s_top;
           // Check if interaction angle is larger
6          α ← (L_a + L_j/||c_a − c_j|| > θ_c);
7          if α then
8              if if node j is a leaf then
                   // pairwise summation
9                  foreach particles μ in cells a do
10                     foreach particles ν in cells j do
11                         f_μ ← f_μ + ∇G(x_ν − x_μ)
12             else
                   // Push node childs on the stack s
13         else
               // Multipole summation
14             for k ∈ [0, p − 1] do
15                 dM_k ← ((−1)^k / k!) Σ_{n=0}^{p−k} (1/n!) D_{n+k+1} · Q_n^B;
16                 foreach particles μ in cells a do
17                     f_μ ← f_μ + Σ_{k=0}^{p} (x_μ − c_i)^(k) · dM_k;
18     while stack not empty;
```

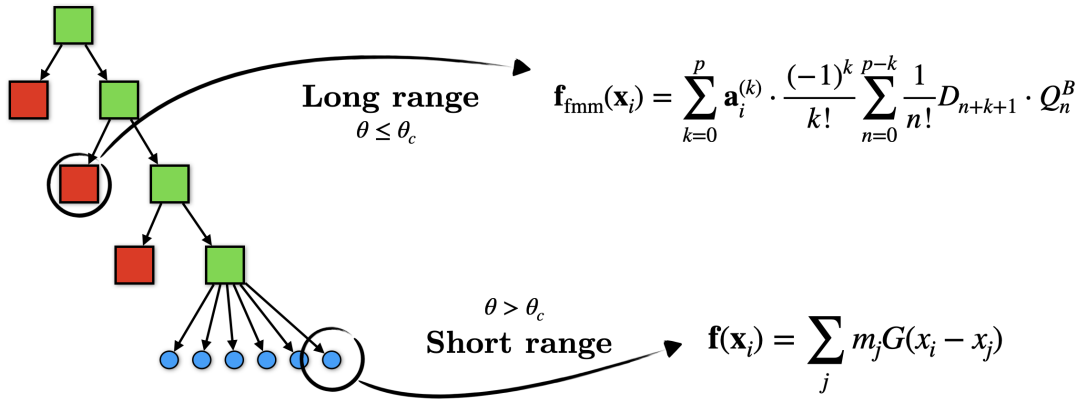


Figure E.4: Illustration of the recursive opening of the tree: cells in green correspond to ones where the angle is larger than the critical angle, and red cells correspond to ones where the angle is smaller than the critical one.

4. Results

4.1. Precision

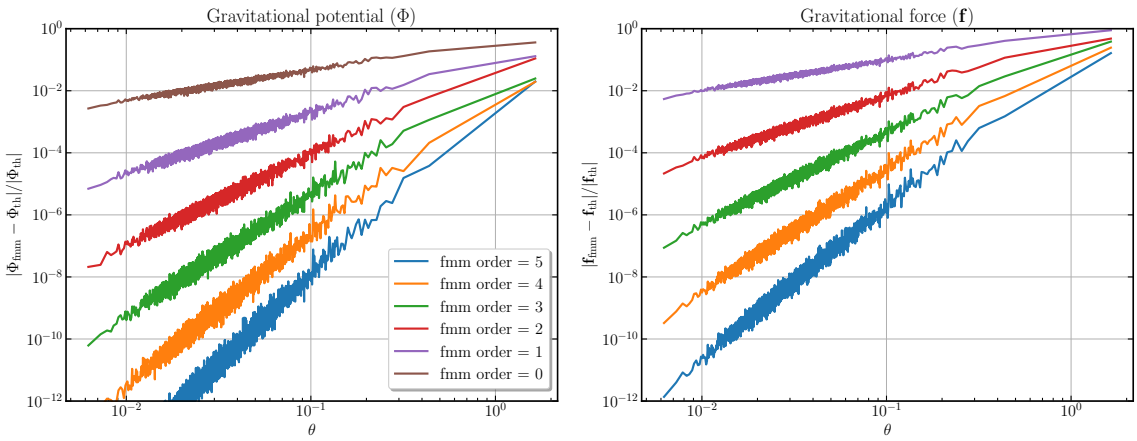


Figure E.5: Results of the FMM precision test.

We start by presenting the tests used to test the precision of the method. For the first test, we generated two random particles and two random cell centers. We compute moments associated with both cells and compute the corresponding long-range force between the two particles. We report the normalized norm difference between the direct pairwise force and the long-range one. We repeat this test for millions of particles at multiple FMM orders. This result is presented for both the potential and the force in Fig. E.5. We observe that the error converges to zero when lowering the angle and that the orders of convergence match the expected ones, validating this test.

To test the complete procedure, we generate a random set of particles. Build the radix tree as for the SPH solver in SHAMROCK, build the moments on that tree, and use the tree traversal with FMM to compute the forces on each particle. Lastly, we compare the resulting forces against the ones obtained by direct pairwise summation on the complete set of particles. The test passes if both values agree at a relative precision of 10^{-5} with an opening angle of $\theta_c = 3$ for fifth-order expansions. Note that the tolerance is lower than the value reported on Fig. E.5. This is excepted as the test in Fig. E.5 corresponds to the worst-case scenario; in practice, inaccuracies will partially cancel out, lowering the error by two orders of magnitude in most of our tests.

4.2. Performance

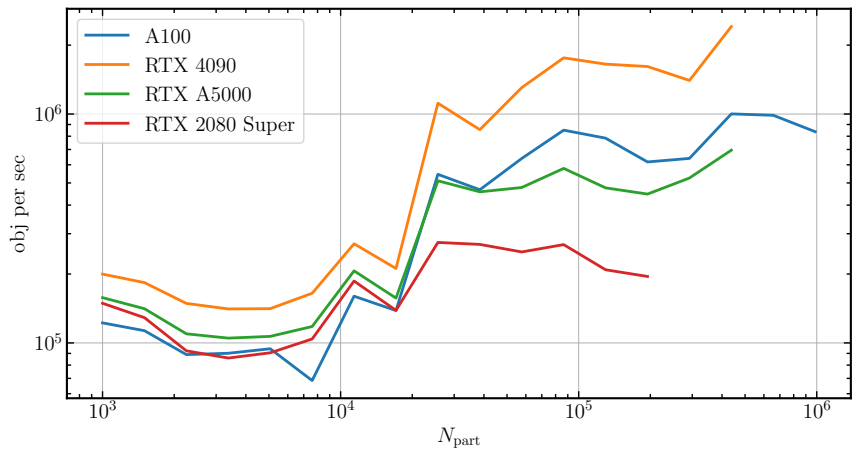


Figure E.6: Performance of the FMM implementation with fifth-order multipoles and an opening angle of $\theta = 0.3$.

Using the same test as previously described on a set of randomly distributed particles, we measure the number of particles processed per second compared to the number of particles in the simulation on a single GPU to be of the order of a million particle per seconds. The results are reported on Fig. E.6. Even if the implementation in its current state is working and yields correct results, we expect the possibility of performing significant optimizations. Specifically, reduce the register usage in the downward-step GPU kernel, group some memory load operations to reduce latency, and change the algorithm layout to have more similar threads, resulting in increased performance due to SIMT. We note, however, that the use of the reduction algorithm the tree of SHAMROCK improves the performance significantly, as shown in Fig. E.7, where the same test is performed with multiple tree reduction levels and shows speedup by an order of magnitude due to reduction.

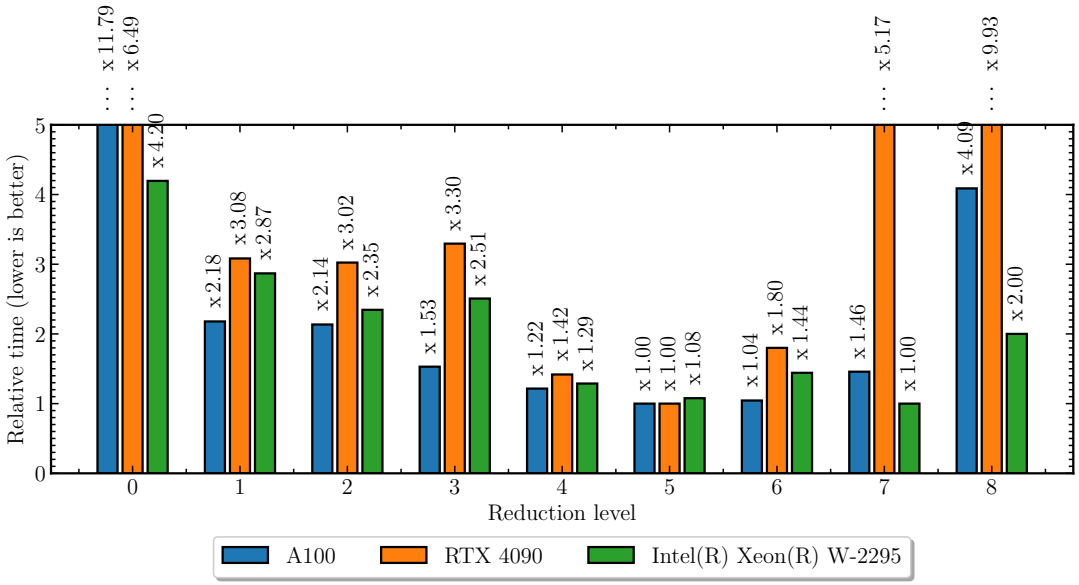


Figure E.7: Relative performance of the FMM test for different reduction levels.

5. Extension to multiple GPUs

As is, the implementation is not capable of running on multiple GPUs. However, it is possible to extend it by transferring a partial tree with the required cells for the downward step along the particles in the ghost zones. We do have a prototype of such an implementation that requires further validation and performance testing as the FMM is communication-intensive, which may result in bottlenecks on the MPI side.

6. Summary

We have presented in this appendix the FMM to solve for the self-gravitating system as it is presented in Springel et al. (2021). In addition to the FMM, we have derived the moment-translation formulas to have exact moment recombination. Very crudely, one estimates a acceleration of a factor $\simeq 100$ compared to the procedure of Gadget-4 on a average server class CPU, also this number should be confirmed by a rigorous quantitative benchmark. The current implementation works on a single GPU, and a prototype is yet to be tested for the multiple GPU case. We expect significant optimizations to be possible in the current implementation. Lastly, the current implementation is standalone in a N-body solver and needs to be connected to the SPH solver to enable self-gravitating simulations in SPH.

References

Price D. J., et al., 2018, *Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics*, PASA, 35, e031

Springel V., Pakmor R., Zier O., Reinecke M., 2021, *Simulating cosmic structure formation with the GADGET-4 code*, MNRAS, 506, 2871-2949

Résumé

Les phénomènes astrophysiques se caractérisent par une interaction complexe entre des processus multi-physiques, multi-échelles, hors d'équilibre et non linéaires. Récemment, la puissance de calcul des supercalculateurs a augmenté jusqu'à atteindre la performance Exascale, à savoir un quintillion d'opérations par seconde. En principe, cette puissance de calcul permet de résoudre des questions cruciales sur la formation des planètes, grâce à des simulations d'une précision sans précédent. Pour y parvenir, il est nécessaire de développer un code basé sur des algorithmes capables de tirer parti de cette nouvelle puissance de calcul.

L'objectif de cette thèse est de développer Shamrock, le premier code Exascale astrophysique qui utilise des méthodes multiples (particules ou grilles adaptatives). Le cœur de ce travail est l'adaptation et l'optimisation d'une nouvelle procédure de construction et de traversée d'arbre pour trouver des voisins distribués aléatoirement, qui est entièrement parallélisée sur des architectures utilisant des cartes graphiques, ainsi qu'une nouvelle approche de la décomposition de domaine pour abstraire l'équilibrage de la charge et la communication. La version actuelle de l'integrateur SPH de Shamrock atteint une efficacité parallèle supérieure à 90% sur les supercalculateurs et fournit des facteurs d'accélération de plusieurs milliers par rapport aux simulations standard de pointe, ouvrant la voie à la résolution de nouveaux problèmes astrophysiques.