Modélisation des disques astrophysiques à l'ère de l'exascale.

Loïc Strafella

Université Paris Cité
École Doctorale STEP'UP n°560
Laboratoire de Cosmologie et d'Évolution des Galaxies

Modélisation des disques astrophysiques à l'ère de l'Exascale

Par Loïc STRAFELLA

Thèse de doctorat de Physique de l'Univers

Présentée et soutenue publiquement le 12 décembre 2023

Devant un jury composé de :

Dominique AUBERT Professeur, Université de Strasbourg	Rapporteur
Romain TEYSSIER Professeur, Princeton University	Président du jury
Jenny SORCE Chargée de recherche, Université de Lille	Examinatrice
Virginie GRANDGIRARD Directrice de recherche, CEA-IRFM	Examinatrice
Paola DI MATTEO Astronome adjointe, Observatoire de Paris	Examinatrice
Matthias GONZÁLEZ Maître de conférences, Université Paris Cité	Examinateur

Dirigée et encadrée par :

Frédéric BOURNAUD Directeur de recherche, Université Paris-Saclay, CEA-DAP	Co-directeur
Marc PERACHE Directeur de recherche, Université de Paris-Saclay, CEA-DAM	Co-directeur
Damien CHAPON Cadre scientifique des EPIC, CEA-DEDIP	Co-encadrant
Olivier BRESSAND Cadre scientifique des EPIC, CEA-DAM	Co-encadrant

Remerciements

Je tiens d'abord à remercier mes directeurs de thèse et mes encadrants d'avoir accepté de me prendre en tant que doctorant. Un grand merci aussi à tout ceux qui ont rendu possible de cette thèse, ainsi qu'à Pascal et Antoine pour leur écoute pendant les sessions de suivi d'avancement. Je remercie également les doctorants et post-doctorants du Dap, pour les nombreuses discussions fructueuses qui m'ont permis d'y voir plus clair. Et particulièrement, Noé et Tine, les deux experts de `RAMSES`, que j'ai souvent sollicité.

Je pense aussi aux membres du laboratoire qui ont égaillé les journées et les repas du midi. Merci à l'équipe de développement de Dyablo pour les nombreuses discussions en terme de génie logiciel et/ou de `HPC`. Et à Damien, pour ta patience et tes nombreuses explications. En espérant que les portes aux plafonds dans mes codes te serviront un jour, à faire un deuxième étage.

Je tiens particulièrement à remercier, aussi, mon petit *poulettos*, qui depuis le mois de Juillet, s'est attelé à la noble tâche de raccourcir mes nuits et mis un peu de piment dans la rédaction de ce manuscrit. Je n'oublie pas bien sûr, ma lectrice personnelle sans qui, mes trois années de thèse auraient été beaucoup plus difficiles.

Ne devrais-je pas aussi, avoir une pensée pour les réalisateurs des films cultes qui ont bercé les heures difficiles de ces trois années ?

Modélisation des disques astrophysiques à l'ère de l'exascale

Résumé

La fragmentation des disques autogravitants est un sujet d'étude central de l'astrophysique actuelle. Les disques de gaz peuvent se fragmenter en une hiérarchie de structures imbriquées les unes dans les autres, couvrant de très grands facteurs d' échelles sur cinq ordres de grandeur ou plus. Les simulations numériques sont nécessaires pour étudier les processus imbriquées, et identifier les propriétés des processus en fonction des conditions physiques de ces disques : selon la température initiale d'un disque et son état turbulent, une hiérarchie de structures l'effondrement peut se former en partant d'abord des plus grandes échelles (scénario "top-down") ou des plus petites échelles (scénario "bottom-up"). Les codes à raffinement adaptatif de maillage (`AMR` - *adaptive mesh refinement*), dont le code RAMSES, sont un outil puissant pour l' étude de ces processus multi-échelles. L'arrivée prochaine de supercalculateur de classe exascale permet d'envisager la réalisation d'une nouvelle génération de simulations numériques résolvant une grande gamme d' échelles avec une précision inégalée. Un des enjeux techniques à la réalisation de telles simulations reste le volume de données gigantesque qui sera alors mis enjeu, représentant des dizaines de téraoctets à analyser.

Dans le but de rendre de telles simulations possibles sans surcoût prohibitif lié au volume de données, on présentera dans un premier temps, les optimisations effectuées concernant les écritures sur disques du code RAMSES en finalisant son couplage à la librairie d'entrées-sorties `HERCULE` et en intégrant une version finalisée de ce couplage à la distribution open-source du code RAMSES. Ensuite, un nouveau modèle de données spécifique pour le traitement de données, destinés aux codes à maillage `AMR` avec une structure en arbre, a été développé et test 'e sur des jeux données issus de différentes simulations astrophysiques. Ce nouveau modèle, ultra-compact et autoportant, permet de réduire drastiquement le volume de données, jusqu''a un facteur 40x en fonction des simulations, grâce à sa description compacte du maillage mais également à des algorithmes de réduction de redondance et de compression de données avec et sans perte d'information. Le méthode de compression de données utilisée, détaillée dans le manuscrit, surpasse en vitesse et/ou en ratio de compression les bibliothèques à l'état de l'art en compression de données scientifiques tels que : `ZFP`, `FPZIP`, `SZ`. De plus, un nouvel outil d'analyse `C++`, au parallélisme hybride (`MPI`, `OpenMP`), dédiés à ce nouveau format données, a été développé ainsi que plusieurs méthodes d'analyses tels que : la sélection de région d'intérêt (`ROI` - *Region Of Interest*) grâce au pavage cubique minimal, la production d'image par floutage gaussien ou par lancer de rayon.

Tous ces développements, effectués en amont, ont permis de lever les verrous technologiques pour lancer une simulation numérique d'envergure d'un disque galactique isolé auto-gravitant, avec le code `RAMSES`, permettant de résoudre 90% de la masse du disque de gaz à une résolution de 3.4 *pc*, donc 50% à 1 *pc*, résolvant ainsi également les régions peu denses du milieu interstellaire. Les résultats de cette simulation, à la résolution spatiale jusque là inégalée, visant à identifier les signatures observables de l' état thermique et turbulent du disque, ainsi que les effets sur l' évolution ultérieure et la formation stellaire, sont présentés dans ce manuscrit. Enfin, le nouveau format de données, la bibliothèque `HERCULE` et l'outil d'analyses seront interfacés avec l' écosystème de la base de données des simulations numériques astrophysiques, `GALACTICA` http://www.galactica-simulations.eu/db/), permettant ainsi à la communauté scientifique de bénéficier des nouveaux outils.

Mots clés : astrophysique, turbulence, simulation numérique, traitement de données, compression, format de données, HPC

Abstract

The fragmentation of self-gravitating disks is a central topic in astrophysics. Gas disks can fragment into a hierarchy of nested structures spanning over a wide range of scales, with five or more orders of magnitude. Numerical simulations are needed to study those nested processes, and identify the properties of the processes as a function of the physical conditions of these disks : depending on the initial temperature of a disk and its turbulent state, a hierarchy of collapsing structures can form starting from the largest scales ("top-down" scenario) or from the smallest scales ("bottom-up" scenario). Adaptive mesh refinement (`AMR`) codes, such as RAMSES, are a powerful tool for studying these multi-scale processes. The imminent arrival of exascale-class supercomputers opens up to a new generation of numerical simulations resolving a wide range of scales with unrivalled precision. One of the technical challenge in carrying out such simulations remains the gigantic volume of data need to be handled representing tens of terabytes to be analyzed.

With the aim of making such simulations possible without prohibitive overheads linked to data volume, we will first present the optimizations carried out on RAMSES's `I/O` by finalizing its coupling to the HERCULE parallel `I/O` library and integrating a finalized version of this coupling into the RAMSES code open-source distribution. Secondly, a new specific data model dedicated to data processing, designed for tree-based `AMR` mesh codes, was developed and tested on data sets from various astrophysical simulations. This new extremely compac data model makes it possible to drastically reduce data volume, by up to a factor of 40x depending on the simulations, thanks to its compact mesh description, but also to redundancy reduction and data compression algorithms. The data compression method used, detailed in the manuscript, outperforms state-of-the-art scientific data compression libraries such as : `ZFP`, `FPZIP`, `SZ`. In addition, a new `C++` analysis tool, with hybrid parallelism (`MPI`, `OpenMP`), dedicated to this new data format, has been developed, as well as several analysis methods such as, but not limited to : region of interest (`ROI`) selection using minimal cubic paving, image production using Gaussian blurring or raytracing.

All these developments have enabled us to overcome the technological bottlenecks, regarding `I/O` and data analysis, in order to perform a large-scale numerical simulation of an isolated self-gravitating galactic disk, with the `RAMSES` code, resolving 90% of the mass of the gas disk at a resolution of 3.4 *pc* with more than 50% at 1 *pc*, thus also resolving the low density regions of the interstellar medium. The results of this simulation, aimed at identifying observable signatures of the thermal and turbulent state of the disk, as well as the effects on subsequent evolution and star formation, are presented in this manuscript. Finally, the new data format, `HERCULE` library and the new data analysis tool will be interfaced with the ecosystem of the astrophysical numerical simulation database : `GALACTICA`, enabling the scientific community to benefit from the new tools.

Keywords : astrophysics, turbulence, numerical simulation, data processing, compression, data format, HPC

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Dans ce chapitre, on commencera par introduire le sujet d'étude astrophysique, à savoir la cascade de turbulence dans un disque galactique isolé. On soulèvera les limites des simulations numériques de la littérature qui nous permettront de définir les objectifs de la simulation *ExaMilkyWay*, effectuée au cours de ces travaux de thèse. On détaillera les points techniques bloquants en terme de consommation de mémoire, de temps de calcul, d'espace de stockage. De plus, les grands volumes de données induisent également des problèmes de performance d'Entrées/Sorties (`E/S`) et de traitement de données. Afin d'apporter une solution à ces problèmes, on rappellera brièvement les enjeux et les initiatives mises en place à l'échelle nationale et Européenne pour la préparation de l'arrivée des machines de classe *Exascale*, concernant les *Entrées/Sorties* et le traitement de données. Dans un contexte de plan national pour la science ouverte, on introduira les notions de base du partage de données à travers le principe `FAIR`. Ensuite, on présentera le code de simulation `RAMSES`, qui a fait l'objet de nombreuses modifications au cours de ces travaux de thèse. On se focalisera principalement sur sa méthode de maillage, son parallélisme et ses flux de données. Ces aspects sont les plus importants à maîtriser avant de poursuive la lecture de ce manuscrit. Finalement, on mettra en avant les enjeux astrophysiques qui ont motivé les modifications du code `RAMSES` et le développement d'un nouveau modèle de données lors de la première partie de cette thèse. Ceux ci ont permis, dans un second temps, d'étudier la cascade de turbulence au travers d'une simulation de galaxie isolée.

1.1 Sujet d'étude

Ces travaux de thèse ont été motivés par l'étude du développement de la turbulence dans le milieu interstellaire, ainsi que son rôle dans la régulation de la formation d'étoiles. Bien que plusieurs acteurs puissent être à l'origine de cette régulation, le mécanisme principal reste un sujet très débattu. Aussi, trois hypothèses principales ont émergé : le faible rendement de la formation stellaire, la rétro-action des étoiles et la pression turbulente. La première hypothèse est aujourd'hui exclue puisqu'il a été montré que l'efficacité, à l'échelle de la formation d'une étoile individuelle, la conversion des coeurs denses semble être d'au moins 50%, [Elmegreen, 2002], [McKee and Ostriker, 2007]. On évaluera donc les deux autres. La rétro-action des étoiles massives peut détruire les nuages denses et éjecter une grande partie du réservoir de gaz en dehors des galaxies, diminuant ainsi la formation stellaire [Hopkins et al., 2014]. Par ailleurs, la pression turbulente dans le milieu interstellaire permet généralement de stabiliser les nuages contre leur effondrement et/ou leur fragmentation gravitationnelle en coeurs pré-stellaire ; mais elle peut également compresser les nuages et donc favoriser la formation stellaire, [Kraljic et al., 2014], [Renaud et al., 2012], [Colman et al., 2022].

Ces processus sont par nature multi-échelle et multi-physiques. En effet, ils font intervenir des instabilités gravitationnelles depuis les grandes échelles, de la thermo-hydrodynamique du gaz aux moyennes échelles et la formation stellaire avec sa rétro-action aux plus petites échelles. Pour résoudre ces différences d'échelles allant de quelques dizaines de kiloparsec au parsec, avec des densités de gaz pouvant varier sur plus de dix ordres de grandeurs, il est nécessaire d'avoir recours aux simulations numériques sur de grands centres de calculs utilisant des méthodes avancées de maillages adaptatifs. Les besoins d'une grande gamme d'échelles spatiales et de densité sont tels que les études numériques existantes se limitent généralement à des régions sous-galactiques, [Colling et al., 2018], [Brucy et al., 2020], [Colman et al., 2022] et/ou à des modèles très limités de rétro-action des jeunes étoiles, [Bournaud et al., 2010], [Murray et al., 2011], et/ou un gaz isotherme sans formation stellaire, [Fensch, Jérémy

et al., 2023]. D'autres impose une résolution élevée uniquement dans les régions très denses négligeant ainsi les régions peu denses (pourtant cruciales pour la propagation multi-échelles de la turbulence), [Renaud et al., 2013]. Enfin dans la plupart des cas, la durée physique de simulation est très courte, ne permettant pas de couvrir les cycles de formation/destruction des nuages moléculaires denses. Ces études ont été limitées par plusieurs problèmes techniques : la quantité limitée de ressource mémoire, l'espace de stockage limité, la scalabilité du code, les temps importants d'écriture/lecture des données, les nombreuses heures de calculs consommées, et le volume de données à analyser. A titre d'exemple, les temps d'écriture/lecture des données, pour le simulation sur 10000 processus, tel que [Renaud et al., 2013], sont de l'ordre de $\sim 1h$, en croissance fortement non linéaire avec le nombre de processus. De plus, certaines simulations peuvent produire jusqu'à plusieurs centaines de téraoctets de données à stocker et analyser.

On a utilisé le code `RAMSES`, [Teyssier, 2002], pour faire une simulation de galaxie isolée entière avec un modèle thermodynamique de chauffage/refroidissement du gaz, un modèle de formation stellaire et de rétro-action cinétique des étoiles massives, le tout avec une résolution spatiale élevée de l'ordre du parsec y compris dans les régions les moins denses, afin d'obtenir une résolution fine du développement de la turbulence à grande échelle. L'objectif secondaire était de couvrir plus d'un cycle, à haute résolution, de la formation/destruction des nuages denses. Néanmoins, pour rendre envisageable cette simulation ambitieuse, il a fallu dans un premier temps résoudre les points techniques bloquants. Pour cela, on d'abord amélioré la scalabilité du code ainsi que les temps d'écriture/lecture de données, par l'intégration d'une bibliothèque d'entrées/sorties parallèles, chapitre 2. Ensuite, la mise en place d'un nouveau modèle de données ultra-compact et des algorithmes de réduction de redondance et de compression de données ont permis de réduire drastiquement l'espace de stockage consommé par les résultats de simulations, chapitre 3. Afin de traiter de grand volume de données décrites avec ce nouveau modèle, on a développé un logiciel au parallélisme hybride, chapitre 4. De plus, en amont de la simulation `ExaMilkyWay`, des développements spécifiques au sein du code `RAMSES` on a permis de réduire significativement son empreinte mémoire. Le nombre d'heures consommées par la simulation a été réduit par la mise en place d'une stratégie en deux étapes avec un changement de nombre de processus `MPI`, chapitre 6. Finalement, on détaillera la mise en place de la simulation et ses modèles ainsi que les premiers résultats obtenus, qui auront consommé près de 52 millions d'heures de calculs sur le *Très Grand Centre de Calcul du CEA*, `TGCC`, au chapitre 6.

1.2 Préparation à l'Exascale : les Entrées/Sorties

L'impact de *l'Exascale* est attendu dans de nombreuses disciplines allant de la sécurité nationale à la médecine en passant par pratiquement tous les champs scientifiques, des mathématiques à la biologie. En mai 2022, la première machine à devoir franchir la barrière symbolique de l'`exaflops`, à savoir un milliard de milliards d'opérations à virgule flottantes par seconde, était annoncée : `Frontier`, à *Oak Ridge National Laboratory, USA*. En juin 2022, cette machine était désormais disponible et listée en première position au `Top-500` [a] des machines les plus puissantes au monde avec une performance crête sur la suite `LINPACK` à 1.1 exaflops. La construction, la maintenance et l'utilisation efficace d'une telle machine (et ses équivalentes européennes à venir) est un véritable challenge, d'autant plus compte tenu des enjeux climatiques actuels. Aussi, afin d'identifier, de soutenir et de préparer au mieux la communauté des utilisateurs, l'Initiative Européenne pour les Logiciels Exascale [b] (*European Exascale Software Initiative*, `EESI`) a rendu plusieurs rapports mettant en avant les points clés pour le passage à l'échelle des applications. Parmi ces points, on retrouve notamment la nécessité d'améliorer : la scalabilité des applications, le développement de technologies matérielles et logicielles pour les Entrées/Sorties et la gestion de données, mais aussi la capacité d'analyse et de stockage de très grands volumes de données. Un extrait du rapport final sur les questions transverses est très révélateur d'une problématique qui était peu étudiée concernant les `E/S` : *[...] More and more scientists see the scalability of their simulations dropping because of unmatched computation and `I/O` performance [...], HPC scientists predict fundamental changes in the way `I/O` and data management will be handled in the near future.* Avec les travaux qu'on présentera dans ce manuscrit, on montrera qu'une réflexion autour des modèles de données utilisés ainsi que l'utilisation d'une bibliothèque d'`E/S` appropriée permet d'améliorer la scalabilité des `E/S` du code `RAMSES`, de réduire les ressources de stockage nécessaires et d'améliorer les performances de l'analyse de données.

En vue de l'arrivée de la nouvelle machine *Exascale* mentionnée précédemment, une initiative européenne sur le développement de solutions logicielles et matérielles aux problèmes bien identifiés des Entrées/sorties et de

a. https://www.top500.org/
b. http://www.eesi-project.eu/

la gestion de gros volumes de données a été mise en place grâce au financement de plusieurs acteurs : l'*European High-Performance Computing Joint Undertaking* soutenu par le projet de recherche et innovation Horizon 2020 mais également par la France, la République Tchèque, l'Allemagne, l'Irlande, la Suède et la Grande-Bretagne. Le projet IO-SEA [c] permet ainsi de réunir plusieurs acteurs de la recherche dont fait partie le CEA, mais également des groupes industriels tel que BULL, SEAGATE ou encore ParTec. Ce projet a pour objectif de réunir plusieurs domaines applicatifs scientifiques utilisateurs de grands moyens de calcul et producteurs de grands volumes de données, dont fait partie l'astrophysique moderne, afin d'établir une interface commune pour les flux de données. Ainsi, les utilisateurs pourront bénéficier d'un système de gestion hiérarchique du stockage (HSM : Hierarchical Storage Management) de manière transparente et optimale, et ainsi tirer profit de matériaux de stockages mixtes (NVMe, SSD, HDD, bandes magnétiques), afin de mieux répondre aux différents besoins du cycle de vie des données. Cela permet, par exemple, d'avoir un premier niveau de stockage très rapide de données temporaires, pour l'écriture de données de protection/reprise par exemple. On parlera de stockage *"chaud"* sur des supports du type NVMe ou SSD. Ensuite vient un deuxième niveau moins performant, pour du stockage à durée limitée, type SSD ou HDD, visant plutôt les données réduites ou destinées à l'analyse (produite par les simulations d'un projet) : il s'agit du stockage *"tiède"*. Finalemenent, le dernier niveau de stockage est dit *"froid"*, où les données peuvent être stockées de manière pérennes sur du matériel moins performant tels que des HDD ou des bandes magnétiques. Des applications scientifiques ont donc été sélectionnées afin d'évaluer les performances des E/S des différents développements effectués au cours du projet. Pour le cas d'usage en astrophysique, le code sélectionné est : RAMSES avec Hercule.

Dans le cadre de cette thèse et du projet IO-SEA, j'ai donc participé notamment au déploiement de RAMSES comme cas d'usage en astrophysique mais également à la rédaction des rapports [d] *Application use case and traces* et *Application and co-design input*, résumant les flux de données du code, ses besoins en termes de ressources d'E/S mais également les enjeux futurs concernant les volumes de données produits par la communauté computationnelle en astrophysique. Ainsi, RAMSES et son nouveau système d'E/S utilisant la bibliothèque Hercule, résultant de la première partie de cette thèse, voir chapitre 2, a été déployé sur plusieurs centres de calculs en Allemagne et en République Tchèques servant de bancs de tests aux couches matérielles et logicielles du projet IO-SEA.

L'EESI recommande le développement de technologies de traitement *in-situ* des données, c'est-à-dire l'analyse, la visualisation et le traitement des données de simulations **pendant** son exécution. Également, il est important de pouvoir partager le fruit des simulations dans un esprit de science collaborative, de reproductibilité mais aussi de rentabilité des heures de calculs. Pour ce faire, une partie des travaux de cette thèse se concentre sur la mise en place d'un modèle de données compact et auto-portant pour la description des données issus du code RAMSES. Ce nouveau modèle fait parti du Plan de Gestion des Données, PGD, de la simulation effectué dans la deuxième partie de cette thèse, détaillé au chapitre 6. Il est important de noter que la mise en place d'un PGD est devenu un point essentiel de toute demande de financement ou d'attribution de ressources de calcul pour tout projet destiné à produire des données, quel que soit le domaine d'application scientifique.

1.3 Science ouverte et partage de données

Les simulations numériques sont devenues un outil majeur pour l'astrophysique, qui permet non seulement de préparer les campagnes d'observations et d'exploiter les données observationnelles obtenues, mais également de qualifier des modèles théoriques et de les confronter aux données observées. De ce fait, la question du stockage, de l'accessibilité et de la diffusion des données numériques se pose. Réaliser une simulation numérique en astrophysique, comme dans d'autres domaines scientifiques, est une expertise en soi qui requiert la mutualisation des compétences et des expériences. Il est donc nécessaire de rendre facilement accessible les données de simulations mais également les codes qui les ont produites et qui permettent d'en analyser les résultats.

Un autre point important concerne le coût monétaire des simulations numériques qui peut aisément atteindre plusieurs centaines de milliers d'euros pour les plus grosses d'entre-elles et qui invite de fait, dans la mesure du possible, à ne pas dupliquer les simulations, mais bien à les partager dans la perspective de les réutiliser. Dans un contexte de crise climatique, il est crucial de mettre en avant les équivalents émissions carbone de ces expériences numériques. En effet, un million d'heures de calcul sur CPU représente environ quatre tonnes eqCO2, soit environ 16,000 km en ville avec une voiture moyenne, [Berthoud et al., 2020]. Pour une simulation comme celle effectuée

c. https://iosea-project.eu/
d. https://cordis.europa.eu/project/id/955811/results/fr

durant cette thèse, cela revient donc pratiquement à 1 million de kilomètre en ville, soit à peu moins de trois fois la distance Terre-Lune.

Ces préoccupations ont bien entendu une portée plus générale que les simulations numériques en astrophysique et concernent les données scientifiques de manière globale. Finalement, l'objectif est d'essayer de respecter quatre valeurs clés proposées en 2016 par [Wilkinson et al., 2016] : `FAIR`. Il s'agit d'un acronyme pour : **F**indable, **A**ccessible, **I**nteroperable, **R**eusable. Ces principes constituent un objectif affiché au niveau Européen avec la publication d'un rapport et plan d'action [e] d'un comité d'expert sur les données ; mais également au niveau national, avec le plan national pour la science ouverte [f] publié en 2021.

Il existe deux principaux freins à la diffusion des données de simulation de modèles théoriques en astrophysique au sein de la communauté scientifique : le volume des données pouvant atteindre plusieurs centaines de Teraoctets et l'absence de format standardisé pour décrire, stocker, échanger et analyser ces modèles - à l'instar du format `FITS`, [Wells et al., 1981], qui est lui largement utilisé pour les données d'observations -. Contrairement aux observations astronomiques qui ont toutes comme cadre de référence la voûte céleste, les simulations numériques permettent de modéliser n'importe quelle structure ou objet astrophysique en 1, 2 ou 3 dimensions, avec une grande variété de modèles complémentaires avec des résolutions spectrales, spatiales ou temporelles différentes. La grande diversité des codes de calcul, chacun utilisant un modèle de données et/ou un format de données différents, rend difficile la standardisation des formats, et par conséquent, l'appropriation des données par la communauté. Il suffit pour s'en convaincre de regarder, par exemple, le nombre de point d'entrée de lecture de données (*backend* `E/S`) de l'outil `Paraview`, [Ahrens et al., 2005], ou du package d'analyse `Yt`, [Turk et al., 2011] : on y retrouve pratiquement un `backend` par code de calcul.

Il est primordial de fournir, en plus des données, des services de traitement adaptés, pour ainsi permettre au plus grand nombre d'acteurs de les exploiter. Ces services doivent être développés pour pallier la complexité et à la diversité des formats de fichiers et de modèles de données générés par les simulations numériques, et à la difficulté de traiter des volumes importants de données. Ces services indispensables fournissent des produits génériques (images d'observables synthétiques, spectre `1D` ou `2D`, cube `3D` d'une grandeur physique, etc) dans des formats de fichiers standardisées (images `JPEG/PNG`, fichiers `FITS`, `ASCII`, `HDF5`, etc.). La mise à disposition des données de simulations et de ce type de services associés facilite la réutilisation des données par des non spécialistes de la simulation numérique, qui pourront en faire usage dans un large champ d'applications. Ces services prennent souvent la forme de WebServices, directement accessibles en ligne, ne nécessitant de la part de l'utilisateur aucune compétence technique (compilation/déploiement) ni aucune connaissance technique sur la façon dont a été produite la donnée (connaissance du code, du format de fichier ou du modèle de données, etc.). C'est dans cette optique qu'une base de données de simulations numériques en astrophysique a été développée et déployée par le `CEA` appelée `GALACTICA` [g]. On détaillera le fonctionnement global de cette base de données, ainsi que certaines contributions effectuées durant ces travaux de thèse à l'éco-système de `Galactica`, au chapitre 4.

1.4 Code `RAMSES`

`RAMSES` [Teyssier, 2002] est un code `Fortran` 90 massivement parallèle, utilisant l'interface standard `MPI`, pour la simulation hydrodynamique de fluides astrophysiques auto-gravitants et de particules. De nombreux modèles physiques y ont été développés afin d'étudier des processus variés tels que la formation stellaire, le refroidissement du milieu interstellaire, la rétro-action des supernovae, la formation des galaxies, la magnétohydrodynamique, etc. C'est un code utilisé par de nombreuses équipes de recherche dans des champs applicatif différents, on peut citer par exemple la simulation `Extreme-Horizon` [Chabanier et al., 2020] pour la cosmologie, la simulation `Antennae` pour les interactions de galaxies [Teyssier et al., 2010], [Renaud et al., 2014], la simulation `Frigg` pour l'étude du milieu interstellaire et de la turbulence [Hennebelle, 2018] ou encore des simulations liés à la poussière, `DustyCollapse` [Lebreuilly et al., 2020]. Il s'agit d'un code qui a été utilisé sur plusieurs machines du `Top 500` ces vingt dernières années et qui a permis de faire des simulations numériques utilisant jusqu'à plusieurs dizaines de milliers d'unités

e. https://op.europa.eu/en/publication-detail/-/publication/7769a148-f1f6-11e8-9982-01aa75ed71a1/language-en/format-PDF/source-80611283

f. https://www.enseignementsup-recherche.gouv.fr/sites/default/files/2021-09/2e-plan-national-pour-la-science-ouverte-12968.pdf

g. http://www.galactica-simulations.eu/db/

de calcul simultanément.

1.4.1 Maillage et particules

RAMSES utilise une méthode de maillage cartésienne évolutive et qui s'adaptera donc automatiquement au cours de la simulation. On parle alors de raffinement adaptatif du maillage ou AMR (*Adaptive Mesh Refinement*). Cette approche, initialement décrite à la fin des années 80 par [Berger and Oliger, 1984], [Berger and Colella, 1989], permet d'apporter une résolution spatiale plus importante dans les zones d'intérêt pour l'utilisateur. Elle s'oppose à la méthode traditionnelle dite à grille fixe où l'ensemble de l'espace de simulation est composé d'éléments de calculs apportant la même résolution. Pour ce faire, l'utilisateur fournit des critères de raffinement qui seront évalués à chaque pas de temps et qui permettront de diviser (*raffiner*) ou regrouper (*déraffiner*) les éléments de calcul. Les critères sont variés et dépendent du type de simulation effectuée. On retrouve par exemple un critère de masse, de gradient de densité ou pression, sur la résolution de la longueur de *Jeans*, un critère géométrique, etc. Si un des critères est validé alors une cellule, qu'on appellera, par la suite, cellule `parent`, sera divisée avec un facteur de raffinement f_d dans chaque direction de l'espace et générera ainsi $(f_d)^{dim}$ nouvelles cellules, appelée cellule `enfant`. RAMSES utilise un facteur de raffinement unique et identique suivant les directions de l'espace, $f_d = 2$.

A chaque cellule de la simulation, un niveau AMR est attribué allant du niveau 1 jusqu'à un niveau maximum défini par l'utilisateur. Ainsi, une cellule parent sera de niveau l et ses cellules enfants de niveau $l+1$. Afin d'éviter un saut trop brutal en résolution entre deux cellules voisines, ce qui pourrait générer des instabilités numériques en plus de complexifier les schémas numériques aux interfaces, le critère 2 : 1 est respecté. C'est-à-dire qu'il ne peut y avoir au maximum qu'un seul niveau d'écart entre deux cellules adjacentes. Finalement, un niveau minimal est généralement utilisé pour définir une résolution de base dans la boite de simulation, on y fera référence par là suite comme du niveau AMR minimum, voir figure 1.1.

Cette méthode de maillage permet de réduire significativement la quantité de mémoire et le temps de calcul nécessaire pour effectuer une simulation. C'est une approche indispensable pour l'étude de phénomène multi-échelle, comme la cascade de turbulence dans le cas présent. Néanmoins, cette approche nécessite d'utiliser des structures de données complexes pour gérer les différents niveaux des cellules et l'interface entre deux cellules de niveaux différents. RAMSES définit le maillage grâce à une structure en arbre, proposée initialement par [Khokhlov, 1998] en 1998, avec pour élément de base une cellule parent et utilise des listes doublement chaînées. C'est-à-dire que chaque élément d'un niveau l pointe vers sa cellule parent au niveau $l-1$ mais également vers ses cellules enfants au niveau $l+1$. De plus, afin d'accélérer les différents algorithmes, il est également possible de parcourir toutes les cellules d'un même niveau et d'accéder aux cellules voisines. Il faut compter 17 entiers 32 octets de stockage mémoire par cellule parent soit environ 2.125 entiers par cellule enfant en 3D. Cette approche de l'AMR est appelée *Fully Threaded Tree* (FTT) et permet d'avoir accès aux cellules des différents niveaux. Cette méthode est pratique pour mettre en place des solveurs multi-grille pour résoudre l'équation de Poisson, [Guillet and Teyssier, 2011] en propageant rapidement l'information. Cette approche FTT s'oppose aux approches qui ne conservent en mémoire que l'enveloppe des cellules enfants.

Les processus hydrodynamiques sont gouvernés par les équations d'Euler. Au sein du code, les fluides auto-gravitants sont donc décrits par les équations convervatives 1.1, 1.2, 1.3 où ρ est la densité du fluide, $\vec{v}$ la vitesse du fluide, E l'énergie totale et ϕ le potentiel gravitationnel :

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = s_m \tag{1.1}$$

$$\frac{\partial}{\partial t}(\rho \vec{v}) + \nabla \cdot (\rho \vec{v} \otimes \vec{v}) + \nabla P = -\rho \nabla \phi + \vec{s_p} \tag{1.2}$$

$$\frac{\partial}{\partial t}(\rho E) + \nabla \cdot [\rho \vec{v}(E + P/\rho)] = -\rho \vec{v} \cdot \nabla \phi + s_e \tag{1.3}$$

Des termes sources additionnels, noté $s_m, \vec{s_p}, s_e$ interviennent dans les équations d'Euler. Typiquement, le terme s_m est introduit par les processus de formation stellaire qui vont consommer le gaz pour le convertir en particules, on a alors $s_m = -\frac{\partial \rho_*}{\partial t}$. De même, la rétro-action cinétique des supenovae impactera la conservation du moment et nécessite donc de rajouter le terme $\vec{s_m}$ dans l'équation 1.2. Enfin, les processus de chauffage/refroidissement du gaz modifient l'équation de conservation de l'énergie en rajoutant le terme s_e qui peut s'exprimer

par $s_e = n_H^2 (\mathcal{H} - \Lambda)$, avec $\mathcal{H}, \Lambda$ des fonctions de chauffages et refroidissement, [Gnat and Sternberg, 2007]. Enfin, l'évolution du potentiel gravitationnel ϕ est donnée par l'équation de Poisson 1.4 et tient compte à la fois du gaz et des particules :

$$\Delta\phi = 4\pi G \left(\rho_{gas} + \rho_{particles}\right) \tag{1.4}$$

On notera que cette équation 1.4 fait intervenir la masse des particules en plus de la masse du gaz. La contribution au champ de densité des particules se fait via un schéma d'interpolation classique d'une particule sur un maillage, appelé *Cloud-in-Cell*, [Hockney and Eastwood, 1988]. La résolution de cette équation se fait généralement par un *solveur de Poisson* de type Gauss-Seidel, [Teyssier, 2002] ou encore un solveur multi-grille qui tire profit de l'approche `FTT` du code, [Guillet and Teyssier, 2011].

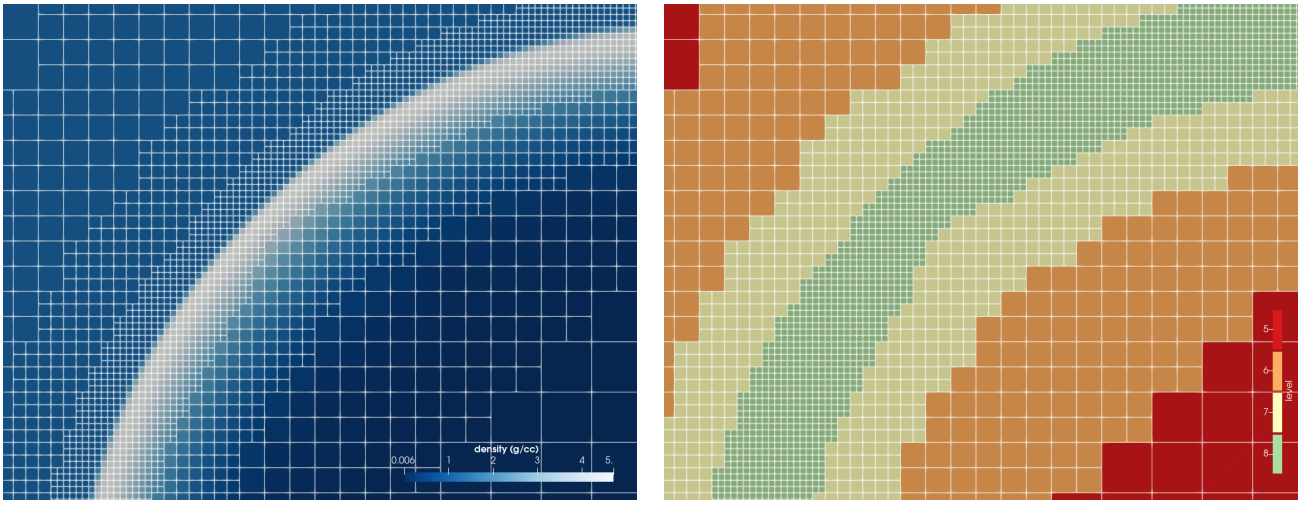

FIGURE 1.1 – Exemple d'une grille à maillage adaptatif avec un raffinement se basant sur le gradient de densité. A gauche, le champ de densité, et à droite, le niveau `AMR` des cellules du maillage. Plus le niveau de la cellule est élevé, plus la cellule est petite et l'erreur lors de la résolution numérique des équations sera faible.

`RAMSES` permet donc de gérer différents types de particules (matière noire, étoiles, débris, etc.), en plus de l'hydrodynamique du gaz, et notamment leurs créations/destructions au court d'une simulation. L'ajout de particules implique de compléter la stratégie de raffinement de la grille par une approche "quasi-Lagrangienne", [Kravtsov et al., 1997], dont l'objectif est de minimiser les effets de relaxation à deux corps, ainsi que les pics locaux de densité liés à une masse ponctuelle. Pour minimiser ces effets sur le solveur hydrodynamique, il est généralement conseillé de maintenir environ 8 à 10 particules par cellule en moyenne, [Kravtsov et al., 1997].

1.4.2 Schéma numérique

`RAMSES` se base sur une méthode de *Godunov* pour la résolution des équations différentielles. Ce type d'approche utilise la méthode des volumes finis ainsi qu'un solveur de *Riemann*. La méthode des volumes finis permet d'établir une relation entre les cellules permettant de mettre à jour le vecteur d'état U_i d'une cellule i par la formulation suivante :

$$U_i^{n+1} = U_i^n - \frac{\Delta t_n}{V_i} \sum_{j \in \nu_i} F_{i,j}^{n+1/2} \tag{1.5}$$

Dans cette formulation, V_i est le volume de cellule, ν_i fait référence aux faces de la cellule avec ses voisines j, voir figure 1.2, pour lesquelles il faut calculer le flux moyen $F_{i,j}$ entre les temps t_n et t_{n+1} :

$$F_{i,j}^{n+1/2} = \frac{1}{\Delta t_n} \int_{t_n}^{t_{n+1}} \int_{S_{i,j}} F(U(\vec{x}, t))\, d\vec{s}\, dt \tag{1.6}$$

L'équation 1.6 fait intervenir la valeur du flux à l'interface qui constitue donc une discontinuité. Ce type de problème est appelé *problème de Riemann* pour lequel il existe de nombreux solveurs permettant de déterminer la valeur du flux à l'interface à partir des états au centre des cellules de part et d'autre de cette interface. Plusieurs types de solveurs sont implémentés dans `RAMSES` : Herten-Lax-von Leer (`HLL`), Harten-Lax-van Leer-Contact (`HLLC`), Harten-Lax-van Leer-discontinuities (`HLLD`), chacun offrant plus ou moins de stabilité et/ou de précision. Il est important de noter, que dans le cadre de maillage `AMR`, il y a quelques subtilités dont il faut tenir compte. En effet, la taille d'une cellule voisine peut être plus grande ou plus petite. En conséquence, certaines interfaces deviennent *non conformes*, comme celle représentée en rouge sur la figure 1.2. Il faut donc corriger le calcul du flux aux interfaces entre deux niveaux `AMR`.

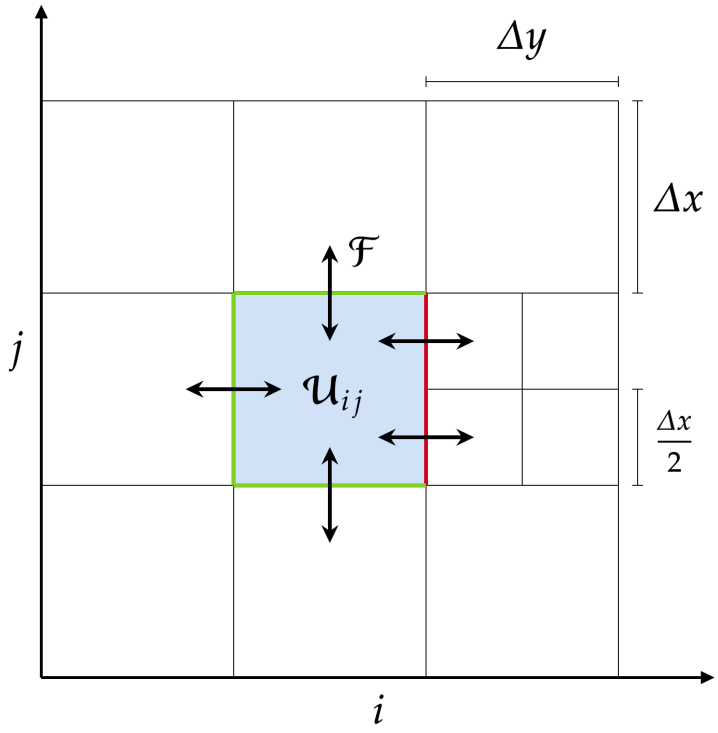

FIGURE 1.2 – Schéma du calcul des flux pour la méthode des volumes finis sur une grille avec une interface non conforme en rouge, et des interfaces conformes en vert.

Pour éviter que les schémas numériques ne deviennent instables et que l'erreur numérique augmente rapidement, il faut respecter une condition de stabilité, appelée condition *Courant-Friedrichs-Lewy*, ou condition `CFL`. Elle permet de limiter le déplacement de "matière" sur un pas de temps, à une distance plus petite que la taille de l'élément local de calcul. Elle s'écrit donc en fonction de la taille d'une cellule Δx à un niveau l, de la vitesse v et du pas de temps Δt :

$$\left| \frac{v \cdot \Delta t}{\Delta x} \right| < 1 \tag{1.7}$$

Cette condition doit bien évidemment être respectée dans toutes les directions de l'espace. Puisque `RAMSES` est un code `AMR`, les cellules ont des tailles variables en fonction des niveaux. Cette condition doit donc être respectée à tous les niveaux, c'est-à-dire pour toutes les tailles de cellules, et pour toutes les cellules. De plus, la structure en arbre induit que le pas de temps au niveau $\Delta t^{l+1} = \Delta t^l/2$. On notera également que le pas de temps est global au sein de la simulation. En conséquence, un évènement quelconque qui fait chuter le pas de temps localement (explosion d'une supernova par exemple), impacte le pas de temps global et toute la simulation est "ralentie". De plus, des contraintes additionnelles sont rajoutées pour le calcul du pas de temps. Par exemple, une condition similaire permet de limiter le déplacement d'une particule à une fraction de la taille de la cellule à laquelle

elle appartient. Ou encore, pour résoudre correctement les effondrements gravitationnels, le pas de temps est limité par le temps de chute libre le plus petit. On notera que d'autres contraintes peuvent être rajoutées en fonction des cas d'usage.

1.4.3 Parallélisme

Malgré les gains apportés par l'`AMR`, il est nécessaire de partager les éléments entre plusieurs unités de calculs. Pour ce faire, `RAMSES` se base sur un parallélisme en mémoire distribuée grâce au standard `MPI` où chaque processus s'occupera d'une quantité variable de cellules. Ce type d'approche est relativement facile à mettre en place car le paradigme de programmation en mémoire distribuée est plus simple a appréhender qu'un parallélisme avec une mémoire partagée par plusieurs `threads`. Les limites du parallélisme de `RAMSES` seront évoqués plus en détail au chapitre 6 sur le dimensionnement de la simulation. L'équilibrage de charge, c'est-à-dire la répartition des cellules entre les différents processus, se fait via le découpage d'une courbe de remplissage de l'espace de simulation. Une courbe de remplissage peut se définir comme un chemin qui passe, une seule fois, par tous les points de l'espace (ici on prendra le centre des cellules comme point de référence). Ce type de courbes sont étudiés depuis de nombreuses années, [Peano, 1890], et elles ont chacune leurs avantages et inconvénients. Sur la figure 1.3, on montre deux célèbres courbes : la courbe de Hilbert à gauche, et la courbe de Morton à droite. Pour définir la trajectoire de la courbe, un indexe unique est calculé pour chaque cellule en fonction de ses coordonnées logiques, c'est-à-dire le duplet/triplet d'entiers qui définit, dans l'espace cartésien d'une grille régulière, la position de la cellule. Cette index permet de faire un passage de n dimensions vers une dimension. Ici, on passe alors d'un système à deux coordonnées d'espace à un système `1D` avec une seule coordonnée. Il suffit alors d'ordonner ces indexes dans l'ordre croissant ou décroissant, et de découper la courbe en n morceaux que l'on réparti entres les n processus.

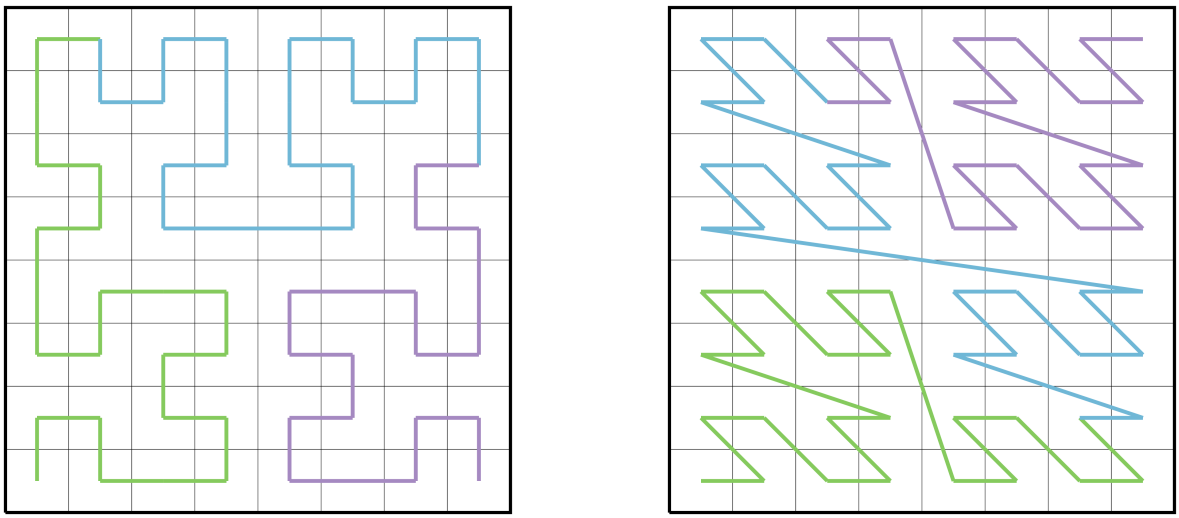

FIGURE 1.3 – Courbes de remplissage de l'espace en 2D sur une grille fixe. A gauche la courbe de Hilbert et à droite la courbe de Morton. Les couleurs définissent les éléments d'un même processus.

`RAMSES` se base sur la courbe de Hilbert pour découper l'espace de simulation en `domaine` dans un contexte de maillage multi-niveaux, voir exemple sur l'image de gauche de la figure 1.4, et ainsi répartir la charge de travail entre les différents processus. Un domaine constitue donc l'ensemble des éléments d'un processus. L'indice de Hilbert possède des propriétés très intéressantes, malgré le fait qu'il soit plus coûteux à calculer que l'indice de Morton. En effet, la localité, c'est-à-dire la capacité d'ordonner un point proche de ces voisins, est une propriété clé. Elle permet d'éviter la création d'îlots, comme on peut le voir clairement pour le domaine en jaune sur l'image de droite, avec une décomposition qui se baserait sur la courbe de Morton, sur la figure 1.5. Il s'agit d'un point important puisque les processus voisins auront besoin de s'échanger des données au cours de la simulation pour assurer la cohérence des champs à l'interface des processus. Les communications entre processus, principalement inter-noeuds, ont un coût qui peut devenir non négligeable dans les algorithmes, même si les avancées matérielles, notamment sur les fibres et les inter-connections, permettent d'avoir un réseaux de communication très performant, voir section 2.1.

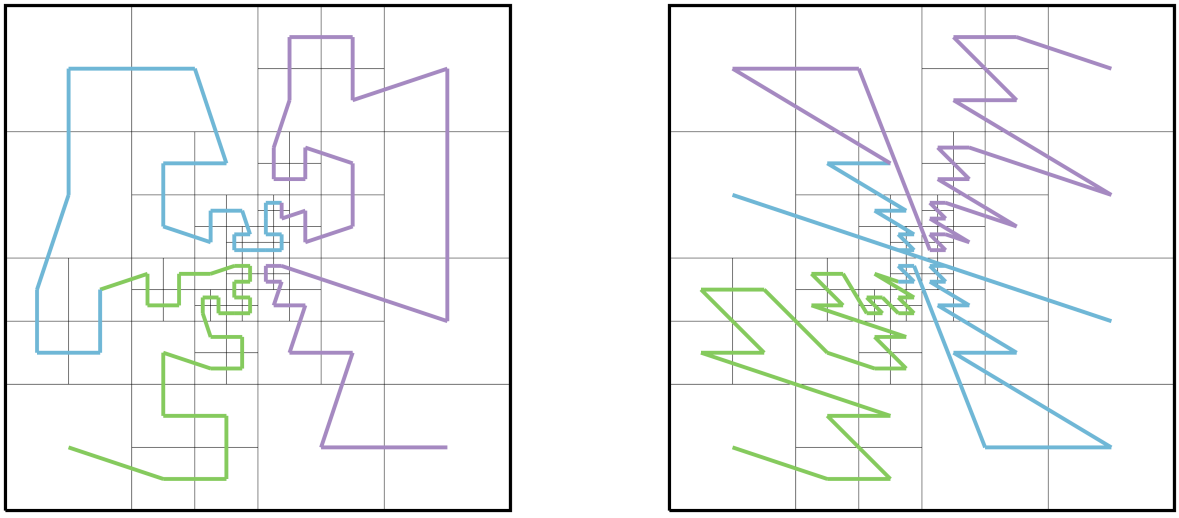

FIGURE 1.4 – Courbes de remplissage de l'espace en 2D sur une multi-niveaux. A gauche, la courbe de Hilbert et à droite la courbe de Morton. Les couleurs définissent les éléments d'un même processus.

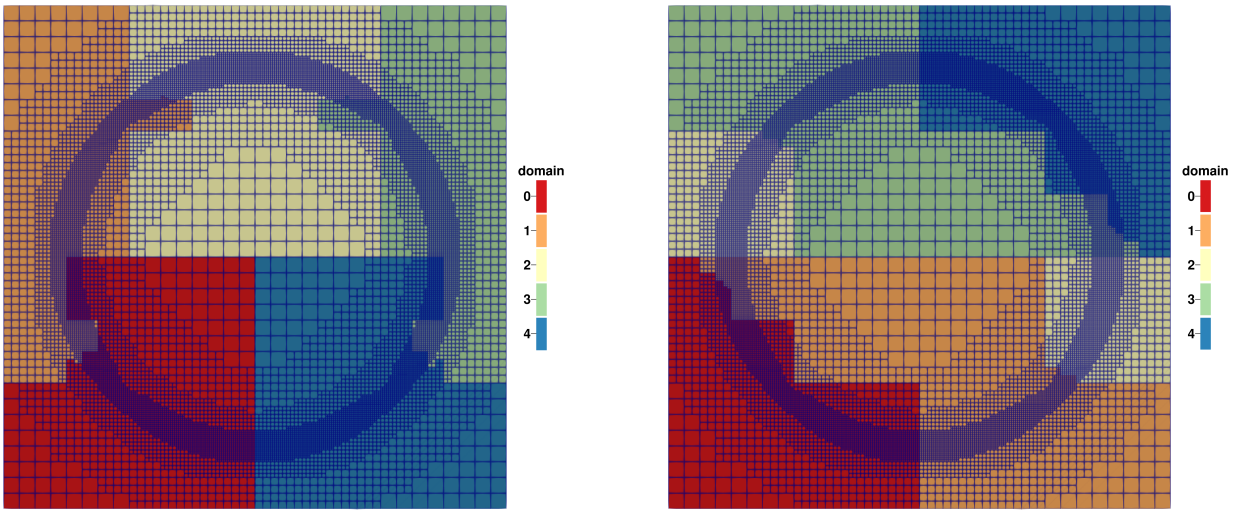

FIGURE 1.5 – Décomposition de domaine suivant la courbe de Hilbert pour un cas test de type blast 2D avec RAMSES sur la figure de gauche, et suivant la courbe de Morton sur la figure de droite. On notera la présence d'un "îlot" pour le domaine n°2 (en jaune) avec la courbe de Morton.

1.4.4 Entrées/Sorties et flux de données

RAMSES a été développé dans les années 2000, avec un paradigme d'E/S "simple" qui permettait de répondre aux besoins des utilisateurs et d'exploiter correctement les machines de cette époque. En effet, l'approche utilisée est celle du *file-per-process*, c'est-à-dire que chaque processus MPI écrira ses données dans un fichier qui lui est propre. Afin de cloisonner les données, le code créera même plusieurs fichiers par processus : un fichier pour le maillage, un fichier pour les grandeurs hydrodynamiques (densité, vitesse, pression, etc.), un fichier pour les particules, un autre pour la gravité, etc. Le nombre de fichier par processus peut ainsi varier en fonction des modules physiques activés par l'utilisateur, de même que la taille des fichiers. Il est important de noter également que tout ces fichiers sont générés dans un dossier propre à chaque déclenchement d'une écriture de données.

Dans le cadre du projet IO-SEA(*Storage I/O and Data Management for Exascale Architectures*), men-

tionné précédemment, il a été demandé d'apporter une vision schématique des flux de données du code, ainsi qu'un plan de gestion générique des données produites en reprenant la nomenclature des différents types de stockage. Sur la figure 1.6, on représente donc un cas d'usage classique. Les bandes de couleurs font référence au type de stockage ciblé par les données écrites ou lues par le code. Les données de protections/reprises sont plutôt destinées à être sauvegardées de manière pérenne c'est-à-dire sur un stockage qui bénéficiera des sauvegardes automatiques par exemple, afin de garder une trace de la simulation, de partager les données avec un futur projet, ou de pouvoir continuer le projet lors d'une autre demande d'allocation de ressources de calcul. Ce type de sortie, programmée à une fréquence choisie par l'utilisateur, a généralement une fréquence faible de l'ordre d'une à deux sorties par *run* ($\simeq 12h$ ou $\simeq 24h$). De plus, le volume de données produit varie en fonction du type de simulation mais aussi au cours de la simulation et peut représenter plusieurs Téraoctets. A priori, la réutilisation de ces données est faible (en dehors d'une relecture pour relancer le code), elles pourraient donc être directement redirigées sur du stockage *froid* (*forever*) en passant par des noeuds dédiés aux écritures, pour éviter de ralentir le code, puisque les performances de ce type de stockage sont faibles et le volume des données important. De même, pour la relecture, les recommandations faites dans le cadre du projet `IO-SEA` suggèrent de pré-charger les données sur du stockage performant avant de relancer la simulation.

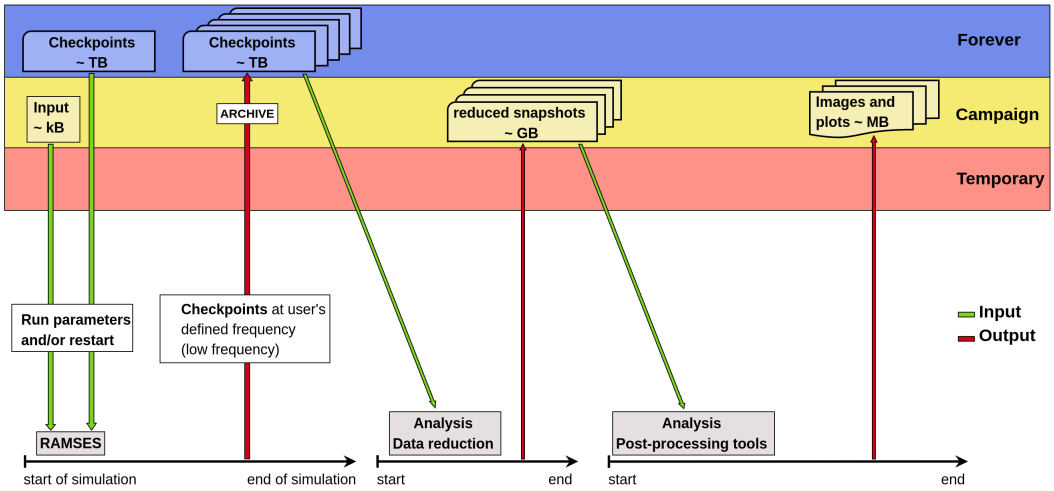

FIGURE 1.6 – Schéma des flux de données d'`E/S` de `RAMSES` dans sa version d'origine, constitué uniquement de point de sauvegarde de l'état du code (*checkpoints*), à double usage (analyse et reprise).

Cependant, dans la version officielle de `RAMSES`, les analyses scientifiques se font depuis ces même données de protection/reprise et après écriture des données, à l'aide d'outils, généralement des scripts personnalisés par les utilisateurs, ou via des packages tel que `PyMSES`, [Guillet et al., 2013], `Yt`, [Turk et al., 2011] ou encore `Pynbody`, [Pontzen et al., 2013]. En conséquence, il serait plus logique de maintenir ces données sur du stockage "tiède" (*campaing*) voir "chaud" (*temporary*), pour bénéficier directement de bonnes performances lors de la relecture des données pour l'analyse et éviter des déplacements inutiles des données. On constate naturellement que cette bivalence de l'utilisation des données du code n'est pas adaptée par rapport aux objectifs du projet `IO-SEA` qui vise à optimiser les flux de données (et le matériel) en fonction de la finalité des données pour les futures architectures. On notera que ce type d'approche, déjà problématique pour les architectures actuelles, deviendra critique pour les futures architectures exascale.

D'autre part, afin de faciliter le partage des données, le transfert sur le réseau ou réduire l'espace de stockage, il est commun, dans les cas d'usage avec `RAMSES`, de passer par une étape de réduction de données afin de réduire à quelques centaines de Gigaoctets le volume des données intéressantes à traiter. Ces données réduites ont une durée de vie de l'ordre du temps du projet ou de l'allocation de ressources, et peuvent donc être écrites directement sur du stockage "tiède" ou éventuellement "chaud" (avec de bonnes, voire très bonnes, performances d'écriture et de lecture). Il est bien évidemment possible de faire un stockage durable de ces données pour garder des historiques des résultats et de pouvoir les partager à plus long terme avec la communauté. Finalement, une dernière étape d'analyse permet de partager les résultats en produisant des bases de données en ligne ou des articles scientifiques, à partir des données à très haute valeur ajoutée. Ce type de données correspond généralement à des

images haute résolution, des courbes, des histogrammes, des données tabulées, des catalogues d'objets, etc. On note également qu'il est commun pour la communauté d'utiliser des méthodes de compression de données sous forme d'archive `Zip` ou `Tar` afin de réduire l'espace de stockage consommé.

Le chapitre suivant abordera les limitations de l'approche *file-per-process* dans le code `RAMSES` ainsi que les améliorations du système de gestion des `E/S` apportées dans le but d'améliorer la scalabilité du code. On verra notamment comment l'intégration d'une bibliothèque d'`E/S` parallèle permet d'améliorer la scalabilité et la gestion des données. De plus, on verra que la séparation du flux de données d'écriture permettra de réduire le volume de données produit. On mettra en avant l'impact de ces changements sur le schéma des flux de données.

Première partie

Entrées/Sorties parallèles et code RAMSES

Chapitre 2

Les entrées/sorties

Dans ce chapitre, avant d'introduire deux bibliothèques (`HDF5`, `Hercule`) et l'interface standard `POSIX` dédiés aux `E/S` ainsi que leurs avantages et inconvénients respectifs, il est important de faire une brève présentation du système de fichier lustre et de son fonctionnement global. En effet, afin de faciliter l'optimisation des flux d'`E/S` il faut avoir une idée du fonctionnement d'un système de fichiers parallèles disponible sur les centres de calculs. Ensuite, on abordera les `E/S` dans un code de calcul et on clarifiera les notions de format de données et format de fichiers, ce qui m'amènera à utiliser plutôt la notion de *modèle de données* et de *format de fichiers* afin de faire une séparation claire de ces deux notions qui sont très différentes et très souvent confondues. Dans un second temps, on met en avant l'importance du cloisonnement dans les codes des routines dédiés aux `E/S` tout en nuançant l'intérêt et la difficulté de rendre une interface agnostique de la bibliothèque d'`E/S` utilisée pour l'écriture ou de lecture de données. Finalement, je présenterai les modifications que j'ai apportées au code `RAMSES` et à son flux de données.

2.1 Le système de fichier LUSTRE

`Lustre` est un système de fichiers parallèle, open-source, déployé sur une très grande majorité de moyens de calcul, du petit *cluster* de laboratoire aux centres de calculs du Top 500. Ce système permet de répondre au besoin de gérer des volumes de données qui sont plus grands que la capacité de stockage d'un seul disque dur mais aussi de supporter les accès simultanés d'un très grand nombre d'utilisateurs à ces données. En effet, la bande passante d'un seul serveur ne permet pas de gérer la quantité de données, ni le nombre d'utilisateurs. `Lustre` permet donc le passage à l'échelle de la capacité de stockage et de la bande passante. Ainsi, sur les plus grands centres de calculs, il est possible de gérer plusieurs milliers d'utilisateurs simultanément et le stockage peut atteindre plusieurs dizaines de Pétaoctets avec une bande passante cumulée de centaines de Gigaoctets/seconde, voir plusieurs Téraoctets/seconde.

Sur la figure 2.1, on représente schématiquement les différents composants du système `Lustre`. On retrouve les points d'entrées de type *client* qui font généralement référence aux noeuds de calculs où tournent les applications ou encore les noeuds de connexion (*les frontales*), à partir desquels les utilisateurs peuvent également consulter et gérer leurs données. Ensuite, il y a deux types de serveurs et de cibles de stockage associées : les serveurs de méta-données, `MDS`, et leur espace de stockage propre, les `MDT`, et les serveurs de stockage objet, `OSS` et leur stockage, les `OST`. Pour relier tous ces composants, un ou plusieurs réseaux sont mis en place avec des performances différentes en fonction des besoins. On peut citer par exemple le célèbre réseau InfiniBand qui équipe la grande majorité des super-calculateurs du top 500 destinés au calcul scientifique haute performance (on met ici de côté les super-calculateurs plutôt destinés à des fins commerciales). Ce réseau permet désormais d'atteindre des bandes passantes allant jusqu'à 400 Gigaoctets par seconde.

Les `MDS` sont des serveurs qui permettent principalement de gérer les noms des fichiers et des dossiers, la localisation du stockage des fichiers (voir `OSS` et `OST`), les verrous sur les fichiers, etc. Les `MDS` ont un système de stockage propre appelé `MDT` où les méta-informations sont stockées. D'après la documentation officielle de `Lustre`[a], le dimensionnement des serveurs de type `MDS`, notamment en terme de mémoire vive, peut être estimé en fonction du nombre de clients et de la taille des dossiers (et donc du nombre de fichiers). Le serveur dispose d'un système de

a. https://doc.lustre.org/lustre_manual.xhtml#idm140261399052864

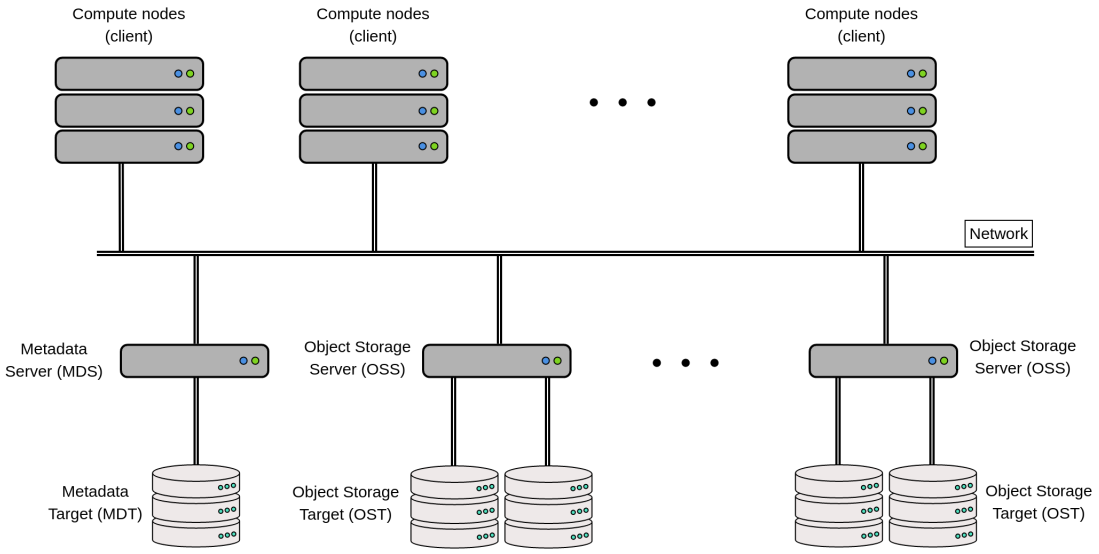

FIGURE 2.1 – Schéma du système de fichier parallèle `Lustre` avec ses composants.

cache permettant de réduire les appels système pour lire les méta-informations des fichiers, ce qui permet d'améliorer les performances d'accès à ces informations pour les utilisateurs. Quand un client ouvre un fichier, cela génère une requête au serveur `MDS` qui va donner en retour les méta-informations associées au fichier au client. C'est pour cette raison que dans les dossiers contenant un très grand nombre de fichiers, la commande `ls` peut parfois montrer des lenteurs excessives. En conséquence, il est de plus en plus commun, pour les centres de calculs, d'imposer aux utilisateurs une restriction sur le nombre de fichiers contenus par dossier et/ou un nombre total de fichiers par utilisateur.

Les `OSS` gèrent les requêtes d'`E/S` pour les accès en lecture/écriture des données des fichiers. Chaque `OSS` dispose généralement de plusieurs cibles de stockage qui lui sont propres (des disques durs), les `OST`. C'est sur ces derniers que sont stockés les fichiers. Les deux propriétés basiques de `Lustre` sont les paramètres `stripe_count` et `stripe_size`, attachés aux dossiers, qui permettent respectivement de définir le nombre d'`OST` sur lesquels sera réparti un fichier, et la taille d'un bloc élémentaire du fichier (généralement de l'ordre de quelques Mégaoctets). En effet, lors de l'écriture d'un fichier, en fonction de la configuration du dossier parent dont le fichier hérite, celui-ci sera "découpé" par le système en bloc élémentaire (appelé `stripe`) de taille `stripe_size`. Chacun de ces blocs seront répartis entre un ou plusieurs `OST` suivant la valeur du `stripe_count`. Enfin, un algorithme de type `round-robin` est généralement utilisé pour répartir les blocs d'un même fichier sur les différents `OST`, voir schéma 2.2. Une mauvaise paramétrisation peut donc engendrer des problèmes d'équilibrage au niveau des `OSTs`. Par exemple, utiliser un seul `OST` pour écrire un très gros fichier peut le saturer ou impacter négativement les performances. A l'opposé, répartir un fichier sur trop d'`OSTs` peut réduire les performances puisqu'il faudra "discuter" avec plusieurs `OSS`. De même, la taille des blocs peut impacter l'équilibrage d'utilisation des disques des `OSTs` et avoir un impact sur les performances d'`E/S`.

Trouver une configuration optimale des paramètres (`stripe_count`, `stripe_size`) est une tâche difficile qui passe par des tests de performance (*benchmarks*). Les benchmarks d'`E/S`, lorsqu'ils sont effectués dans un contexte "utilisateur", c'est-à-dire dans un contexte normal de charge de la machine (ie : avec d'autres applications et d'autres utilisateurs), sont sujets aux éventuels impacts de ces autres applications et autres utilisateurs. Par exemple, si une autre application fait des `E/S` en même temps, les performances pendant le test seront dégradées. D'un autre côté, si les benchmarks sont faits dans un contexte "administrateur" (sans autre application ni utilisateur) les performances seront a priori optimales ou "crête" mais ne refléteront probablement pas la réalité lorsqu'un utilisateur utilisera l'application. D'autre part, il est important de préciser que le schéma des appels d'écriture ou de lecture de l'application elle-même peut impacter significativement les performances, voir parfois même plus que les paramètres (`stripe_count`, `stripe_size`). Par exemple, un code qui fait beaucoup d'appels d'écriture ou de lecture de petits tableaux peut avoir des performances d'`E/S` moins bonne qu'un code qui fait moins d'appels avec des tableaux plus gros, avec une même paramétrisation. En effet, entre l'appel d'écriture ou lecture du code et la disponibilité des données, il y a plusieurs couches de *cache* ou *buffer* qui peuvent être paramétrées en fonction des besoins et de la politique d'utilisation du centre de calcul. Par exemple, il existe un *cache disque* au niveau des

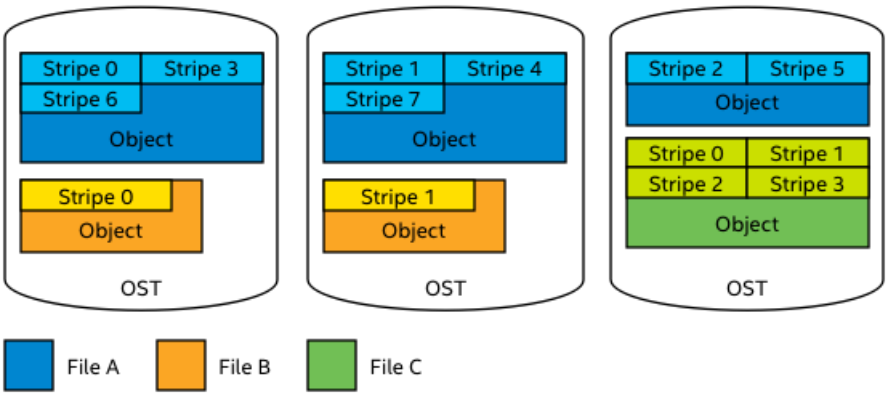

FIGURE 2.2 – Schéma de répartition des blocs (*stripe*) de fichiers en *round-robin* sur plusieurs `OSTs`, source : https://wiki.lustre.org.

`OSS` qui stocke temporairement les données fréquemment demandées. Ainsi, si une application accède à des données qui se situent dans le cache, les performances sont grandement améliorées car cela évite un accès coûteux aux disques.

L'optimisation des `E/S` d'un code nécessite donc une connaissance approfondie du schéma d'écriture/lecture des données du code. Parfois, une simple paramétrisation ne suffit pas ; des changements au sein de l'application sont donc nécessaires afin d'améliorer les performances. C'est ce qui a été fait pour les protections/reprises de `RAMSES` avec l'intégration de la bibliothèque d'`E/S` `Hercule`, [Bressand et al., 2012], voir section 2.3. De plus, on notera que l'utilisation d'une bibliothèque spécialisée pour les `E/S` tel que `Hercule`, `HDF5`, `NetCDF`, `MPI-IO`, etc, peut permettre de tirer profit des optimisations faites en interne, afin d'améliorer les performances tout en diminuant les coûts de maintenance et de modification de l'application par les utilisateurs.

Enfin, pour clôturer cette section, on rappelle quelques bonnes pratiques vis-à-vis des `E/S` avec un système de fichier tel que `Lustre`. Il est généralement conseillé d'éviter autant que possible les petites requêtes d'`E/S` et de préférer l'écriture ou la lecture de grands tableaux (si possible). On mettra ce point en avant lors des modifications de `RAMSES` à la section 2.3. De même, ce système de fichier est optimisé pour offrir une grande bande passante fonctionnant de manière optimale avec des gros fichiers, de l'ordre de quelques Gigaoctets à quelques Téraoctets, plutôt qu'avec une multitude de petit fichiers de quelques dizaines/centaines de Mégaoctets. Ainsi, réduire le nombre de fichier par dossier permet de limiter les contentions au niveau des serveurs `MDS` et `OSS`.

2.2 Les bibliothèques d'Entrées/Sorties et le format de fichier

Avant d'aborder le fonctionnement des bibliothèques d'`E/S`, il est important de mettre en avant les deux approches les plus utilisées pour l'écriture de données : *single-shared-file* et *file-per-process*. On se place bien évidemment dans le contexte d'une application parallèle avec l'interface `MPI`.

L'approche *file-per-process* consiste à faire écrire ou lire un fichier par un processus de manière indépendante vis-à-vis des autres processus. Cette méthode comporte plusieurs avantages. Tout d'abord, elle est très facile à mettre en place puisqu'il n'y pas besoin de gérer l'accès par plusieurs processus au même fichier. Il n'y a donc pas besoin de gérer de verrou (*lock*) sur le fichier, il n'y pas non plus de risque de contention pour les opérations de lecture/écriture. De plus, au sein du code, il n'y pas besoin d'effectuer de synchronisation ou de communication inter-processus pour les opérations `E/S`. On notera également que les données sont donc séparées par processus et il n'y a pas d'agrégation globale pour former un modèle de données (voir section 2.2.2). Enfin, en terme de code, une implémentation *file-per-process* est généralement plus simple et donc il est plus facile de maintenir, étendre et faire évoluer le code. Néanmoins, cette approche a également certains inconvénients. Premièrement, lorsque le nombre de processus augmente, le nombre de fichier augmente également. C'est d'autant plus vrai qu'une application comme `RAMSES` génère plusieurs fichiers par processus, voir section 2.3. Bien qu'en terme de performance, cette approche est, en théorie, capable d'approcher la bande passante maximum, elle est souvent limitée par les performances des `MDS` à cause des opérations sur les méta-données des fichiers, [Harrington, 2018]. En effet, les `MDS` sont mis sous pression

lorsqu'une application essaye d'accéder à un très grand nombre de fichiers simultanément en écriture ou en lecture. Les requêtes sont alors sérialisées et les performances s'effondrent. C'est le cas, par exemple, lorsqu'un utilisateur essaye d'exécuter la commande `ls` dans un dossier contenant de nombreux fichiers. Ces contentions peuvent alors limiter la scalabilité du code. On notera également qu'en fonction de la politique de gestion des ressources du centre de calcul, des quotas sur le nombre de fichiers par dossier et par utilisateur peuvent rendre ce type d'approche difficile à maintenir et même limiter les possibilités en terme de simulation. Deuxièmement, lors de la phase d'analyse, les outils de post-traitement doivent alors gérer un grand nombre de fichiers, ce qui peut augmenter la complexité de l'outil, mais surtout impacter également drastiquement ses performances lors de la lecture des données. Généralement, cette approche atteint ses limites lorsqu'une application essaye d'écrire ou lire simultanément plusieurs dizaines de milliers de fichiers.

L'approche *single-shared-file*, s'oppose à celle décrite précédemment, dans le sens où tous les processus de l'application vont contribuer à l'écriture ou à la lecture d'un unique fichier partagé. Son avantage principal est donc de réduire significativement le nombre de fichiers produits par l'application. Un autre point fort est d'offrir la possibilité d'agréger les données des processus et ainsi optimiser le volume de données lors des appels aux fonctions d'écriture ou de lecture. En conséquence, il est plus difficile avec cette méthode d'approcher la bande passante maximum puisqu'une synchronisation est nécessaire afin de coordonner les processus entre eux. On peut voir un exemple de ce type d'optimisation pour le code `FLASH` dans les travaux de [Latham et al., 2012]. Cette approche permet généralement d'obtenir de meilleurs performances que l'approche *file-per-process*, au-delà d'un certain nombre de processus, variable en fonction des applications et des simulations. Néanmoins, il faut définir comment agréger les données. Il s'agit d'un point essentiel à la fois pour les performances, mais également pour les outils de post-traitement, qui auront à relire et traiter les données. On reviendra sur ces notions d'agrégation des données et de format de données ainsi que sur la corrélation forte qu'il existe entre les deux, à la section 2.2.5.

Il existe une troisième approche mixte entre le *file-per-process* et le *single-shared-file* qu'on appellera *multiple-shared-file*. Elle consiste à faire partager un fichier par un groupe de processus `MPI`. Cette approche est au coeur du fonctionnement de la bibliothèque `Hercule` que l'on détaillera à la section 2.2.3. Elle permet de tirer profit de la réduction du nombre de fichier, tout en limitant la quantité de synchronisation nécessaire entre les processus.

Dans la section suivante, on détaillera le fonctionnement de l'`API POSIX`, utilisée par le code `RAMSES` dans sa version native. On détaillera ensuite le principe et le fonctionnement de la bibliothèque d'`E/S` spécialisée `HDF5`, une des bibliothèques `E/S` les plus utilisées, et notamment par l'outil d'analyse que j'ai développé, chapitre 4. Enfin, on explicitera le fonctionnement de `Hercule`, une autre bibliothèque d'`E/S` et de gestion de données que j'ai intégré à `RAMSES`. Concernant l'agrégation des données, on verra que `Hercule` se base sur une approche fondamentalement différente de celle de `HDF5`.

2.2.1 Posix

`POSIX` n'est pas une bibliothèque d'`E/S` à proprement parler mais une interface standard pour les systèmes d'exploitation, qui permet notamment la portabilité des applications qui l'utilisent sur différents systèmes d'exploitation. Parmi les différentes fonctions accessibles, on retrouve les fonctions qui permettent d'effectuer les opérations d'ouverture, écriture, lecture et fermeture de fichier. C'est généralement cette interface qui est utilisée par les compilateurs pour implémenter les fonctions intrinsèques du langage de programmation utilisé. En effet, le standard Fortran, par exemple, définit le comportement de ces fonctions intrinsèques et notamment celui des fonctions de manipulations de flux données tels que **open, read, write, close**. Ces comportements sont ensuite implémentés par les différents compilateurs.

Il n'y a pas de problème particulier à l'utilisation des routines intrinsèques **open, read, write, close**. Cependant, en Fortran, le standard définit un certain nombre d'options par défaut pour le comportement de l'opération **open**. En conséquence, et c'est le cas dans le code `RAMSES`, si l'attribut **access** n'est pas renseigné, le comportement par défaut pour l'écriture de données binaires est celui qui consiste à encadrer les données écrites avec la fonction **write**, d'un entier 4 octets, appelé un `record`. Cela permet de renseigner dans le fichier le nombre d'octets écrits par l'appel à la fonction **write**. La taille du **record** est un nombre entier correspondant au nombre de bloc de stockage unitaire du fichier (**file storage unit**) qui, lui, peut dépendre du système ou du processeur. Le

standard Fortran définit par défaut un **file storage unit** de 1 octet. Généralement, le **record** est donc encodé sur 4 octets. Cela signifie également que le comportement devient indéfini si on essaye d'écrire plus de 2^{31} octets soit 2 Go avec un unique appel. Concrètement dans l'exemple 2.1, un **record** de 4 octet encodant la valeur 4 sera donc écrit avant et après, la valeur de l'entier n. Un autre **record** encodant la valeur 44 sera écrit dans le fichier avant et après le tableau. Il est important de noter que cela permet simplement de savoir le nombre d'octets contenus entre deux records mais ne permet pas de retrouver le type de la variable ou du tableau. Le **record** sera le même si la variable n était un flottant 4 octets ou un tableau de 4 entiers de 1 octet.

```
1 integer, parameter :: n = 10
2 integer, dimension(n) :: array1D
3
4 open(unit=42, file="test.bin", access="sequential")
5   write(42) n
6   write(42) array1D(:)
7 close(unit=42)
```

Extrait 2.1 – Ecriture `Fortran` avec l'interface `POSIX` en mode 'sequential'

Cela nous amène au point suivant concernant le format du fichier. En effet, un des gros avantages des bibliothèques d'entrées/sorties, et une des raisons de leur développement et popularité, est de faciliter l'organisation et l'accès aux données. Un fichier binaire écrit avec ou sans `record` est totalement inutile si on ne connaît pas l'organisation des données faite à l'écriture lorsqu'on essaye de relire le fichier. Dans l'exemple 2.1, "un entier sur 4 octets définissant la taille d'un tableau d'entier 4 octets écrit juste après, etc.". De même, si des données sont insérées, les routines de lecture du fichier doivent être modifiées pour tenir compte du changement d'organisation des données, ce qui complexifie la maintenance, l'évolution et la rétro-compatibilité des routines de lectures, qui se situent généralement au niveau des outils de post-traitement. C'est un des problèmes avec `RAMSES` qui peut rajouter des données lors de la phase d'écriture des fichiers binaires en fonction des paramètres définis soit lors de la compilation, ou pire, dans la phase de configuration d'une simulation. Ainsi pour un outil d'analyse tel que *Pymses* (package d'analyse en Python utilisant des routines de lecture en `C`) cela implique d'être très modulable ou de modifier les routines de lecture bas niveau en fonction des évolutions / besoins des utilisateurs ; tout en maintenant un maximum de rétro-compatibilité sur les versions de `RAMSES`.

Finalement, pour un accès *aléatoire* (ou non séquentiel) aux données dans un fichier binaire, il est alors nécessaire de se déplacer "manuellement", de `record` en `record` par exemple, ce qui peut rendre l'opération compliquée et coûteuse. On notera également que pour la relecture en `C/C++` des fichiers binaires écrits avec un accès `sequential` il est important de "lire" les `record` pour s'assurer du bon déplacement du curseur dans le fichier. L'utilisation de l'accès `stream` ou `direct` permet de s'affranchir de l'écriture des `record` et donc faciliter la lecture en `C/C++`. (lien vers le standard Fortran section 12). Afin de s'affranchir de ce type de problème et d'offrir plus de souplesse et de simplicité aux utilisateurs, les bibliothèques d'`E/S` utilisent généralement un système d'indexation avec une clé unique pour accéder à une donnée.

Pour la gestion des fichiers produits dans le cadre d'un code parallèle, l'approche *file-per-process* est la plus simple à mettre en place avec cette `API`, c'est d'ailleurs ce qui est utilisé dans `RAMSES`, voir section 2.3. Pour tirer profit d'une approche de type *single-shared-file* ou *multiple-shared-file*, les opérations de gestion de synchronisation des processus et d'écriture dans un même fichier (choix des *offsets*, des méta-données, etc.) doivent être gérées "à la main". Encore une fois, l'utilisation d'une bibliothèque d'`E/S` spécialisée permet d'abstraire, pour l'application et les utilisateurs / développeurs de l'application (qui ne sont pas forcément des experts en `E/S`), ces opérations délicates.

2.2.2 HDF5

La bibliothèque `HDF5`, [The HDF Group, 2023], pour *Hierarchical Data Format* offre une surcouche aux appels bas niveau pour l'écriture et la lecture des données, qui permet notamment de conserver l'encodage de la donnée. En effet, `HDF5` assure à l'utilisateur que les fichiers produits sont auto-portants sur le type des données qu'ils contiennent (entier, flottants, 1 octet, 4 octets, etc.) et l'encodage des données (*little endian*, *big endian*). Ainsi, lorsqu'on écrit un entier 4 octets par exemple, des méta-informations sont rattachées, de manière transparente pour

l'utilisateur, à cette valeur, typiquement son type "entier" et son encodage "4 octets". De plus, HDF5 maintient une table d'indexation qui nécessite un identifiant pour cet entier. Cela permet de s'affranchir de la connaissance de l'ordre dans lequel les données ont été écrites, ce qui constitue un atout majeur pour la maintenance, l'extensibilité des formats de données et l'évolution des outils d'analyses des codes de simulation. En conséquence, un fichier HDF5 peut facilement être partagé entre plusieurs utilisateurs, sur plusieurs systèmes différents, et garantit une parfaite portabilité. Il s'agit d'une des raisons qui expliquent la popularité de cette bibliothèque.

En plus de cet aspect *"clé-valeur"*, HDF5 permet aux utilisateurs d'organiser de manière logique et hiérarchique leurs données au sein du fichier. En effet, l'utilisateur n'a plus à se soucier du format du fichier puisque c'est HDF5 qui se charge d'organiser les données binaires dans le fichier. En revanche, l'utilisateur peut définir un modèle conceptuel de ses données et/ou les organiser de manière logique, comme sur l'exemple 2.2. Pour ce faire, on retrouve trois notions clés : les **groupes** que l'on peut associer à la notion de **dossier** sur un système de fichier, les **paramètres** qui représente des variables simples (méta-informations) et les **jeux de données** (ou **dataset**).

```
1  |--> group: simulation_info
2      |--> attr: 'code_name', dtype: |S5
3      |--> attr: 'version', dtype: int32
4      |--> group: constant
5          |--> attr: 'length', value : b'cm', dtype : |S2
6          |--> attr: 'time', value : b's', dtype : |S1
7  |--> group: results
8      |--> group: hydrodynamics
9          |--> dset: 'density',  dtype: float32, shape:
               (100,)
10         |--> dset: 'velocity', dtype: float32, shape:
               (100,3)
11     |--> group: particles:
12         |--> attr: 'nparticles', dtype: int32
13         |--> dset: 'position', dtype: float64, shape:
               (200,3)
```

Extrait 2.2 – Exemple de structure pour un fichier HDF5

L'accès à une donnée telle que la variable *time* de l'exemple ci-dessus se fait donc en demandant la lecture de l'attribut *time* qui se situe dans le *chemin* `/simulation_info/constant/`. Cette organisation hiérarchique est la deuxième raison de la popularité de cette bibliothèque. Finalement, un autre aspect de l'intérêt de l'utilisation d'une bibliothèque dédiée pour les E/S est la possibilité de bénéficier d'optimisation et de mise à jour des routines utilisées, mais également de tirer profit d'une interface haut niveau pour faire des E/S et exploiter au mieux le parallélisme de l'architecture des supercalculateurs. Ainsi, les utilisateurs ont pu bénéficier récemment d'une mise à jour de manière quasi-transparente pour les applications, avec le passage de la version 1.10 à 1.12 de HDF5 qui a permis, en autre, de résoudre certains problèmes de consommation mémoire et de performance, qui avaient été soulevés par un grand nombre d'utilisateurs.

Interface pour les E/S parallèles

Cette bibliothèque propose des fonctionnalités additionnelles pour les applications utilisant un parallélisme par processus avec la couche MPI. Pour ce faire, HDF5 se base sur MPI-IO qui est une extension de MPI permettant d'étendre les principes clés aux E/S parallèles. Nativement, il existe deux comportements possibles différents : indépendant ou collectif. Le fonctionnement *indépendant* utilise l'approche *file-per-process* et permet à chaque processus MPI d'être indépendant des autres processus. La version collective, en revanche, est celle qui permet l'utilisation des E/S parallèle à proprement parler et de passer sur un paradigme de type *single-shared-file*, où tous les processus participent aux appels d'écriture/lecture dans un fichier partagé unique.

La sélection du mode E/S passe par l'activation d'une option. Le mode H5FD_MPIO_COLLECTIVE est important pour la paramétrisation du mode de transfert de MPI-IO ; ce paramètre s'oppose à H5FD_MPIO_INDEPENDENT. Dans la version collective, l'appel fait par HDF5 est un appel à la routine MPI_File_write_at_all, qui est un appel collectif (doit être effectué par tous les processus du communicateur MPI donné à HDF5. En conséquence, MPI-IO

connaît les tailles des données qui doivent être écrites par les différents processus et peut donc faire des optimisations en interne. Dans la version indépendante, c'est la routine `MPI_File_write_at` qui est utilisé. Il s'agit d'une routine non collective ainsi `MPI-IO` ne "sait" pas qu'un autre processus doit écrire des données et ne peut donc pas effectuer d'optimisation ce qui peut donc engendrer des problèmes de performance.

Dans l'approche *collective* se pose alors la question de l'agglomération des données. Il s'agit d'un point très essentiel et qui est lié à la granularité des données qu'on évoquera dans le chapitre 3 sur le modèle de données `lightAMR`. L'approche choisie par `HDF5` est de centraliser, dans un même fichier, l'ensemble des données des processus. Pour ce faire, la notion d'**hyperslabs** a été introduite. Imaginons qu'une application utilise 10 processus `MPI` et que chaque processus souhaite écrire un tableau de 3 valeurs. L'application va définir grâce, aux interfaces, une zone dans le fichier de $10 * 3$ valeurs, c'est-à-dire l'espace total nécessaire, tous processus confondus, pour écrire les données. Chaque processus va ensuite envoyer ses données à écrire en renseignant en plus un offset d'écriture au sein de cette zone. Par exemple, le processus 0, va demander à écrire ses données entre les indices 0 et 2, le processus 1, entre 3 et 5, etc. Au final, tous les processus auront écrit leurs données dans un seul tableau (mémoire contiguë dans le fichier), mais chacun dans des "cases" du tableau disjointes les uns des autres. Afin de pallier une grande variété de besoins des applications, `HDF5` propose des mécanismes riches avec : un système de *padding*, des conversions à la volée, des *layout* différents pour l'orientation des tableaux multidimensionnels, de la compression de données, etc. Il est important de noter que certaines options, telle que la conversion de type, peut avoir un surcoût mémoire qui peut devenir prohibitif pour certaines applications.

Cette centralisation des données grâce aux *hyperslabs* et la "construction" mémoire sont utilisées au sein des applications pour construire un modèle de données approprié, en se basant par exemple sur une représentation spatiales des données de la simulation. Cette méthode est intéressante pour les codes à grille régulières. En effet, pour le post-traitement il est possible d'exploiter les `hyperslabs` pour limiter la relecture à des régions d'intérêt dans les données, [Kumar et al., 2014]. Néanmoins, cette approche de centralisation dans un même fichier des données peut rapidement devenir problématique lorsque la simulation s'agrandit et que les données deviennent volumineuses. En effet, bien que les systèmes de fichiers actuels préfèrent gérer de gros fichiers de plusieurs centaines de Gigaoctets, des simulations, comme celle effectuée durant cette thèse, pourraient générer un fichier unique de plusieurs dizaines de Téraoctets en fonction du modèle de données utilisé. Cela peut engendrer différents problèmes, par exemple en cas de corruption du fichier, c'est l'ensemble des données qui sont perdues. Il peut également y avoir un problème de contention lorsque trop de processus contribuent au même fichier. Finalement, très récemment (février 2023), avec une nouvelle version de `HDF5`, un nouveau *pilote* (driver) bas niveau est disponible et permet de découper automatiquement un fichier *logique* en plusieurs sous-fichiers physiques. Un concentrateur d'`E/S` permet de rediriger les flux vers le bon sous fichier correspondant.

D'autre part, la structure hiérarchique de `HDF5` déplace le problème de maintenance du format de fichier évoqué à la section précédente, vers un problème de structuration hiérarchique des données. Il est alors important pour les applications "productrices" de données, de définir une structuration stricte et/ou standardisée de l'organisation des données au sein des fichiers `HDF5`, afin que limiter les modifications nécessaires au sein des outils d'analyses. Si on reprend la structure illustrée sur la figure 2.2, il est important de conserver, par exemple, le chemin `/results/hydrodynamics/` pour stocker les champs physiques. Ainsi, il est aisé pour un outil d'analyse d'aller chercher et lister les champs disponibles en récupérant les `datasets` stocké à cet endroit, sans pour autant empêcher les utilisateurs de rajouter des champs en fonction de leurs besoins. On présentera à la section suivante la bibliothèque `Hercule` qui propose une approche différente, plus proche des données et qui simplifie la gestion de l'organisation de l'agglomération des données par les applications.

2.2.3 Hercule

La bibliothèque `Hercule` a pour objectif de mettre en place des services d'écriture, de lecture mais également de mise en forme des données complexes et volumineuses issues des codes de simulation numérique sur les machines massivement parallèles. Elle permet une gestion de l'information sous la forme de bases de données destinées au code lui-même pour les protections/reprises, en passant par des codes de post-traitement pour les sorties destinées au dépouillement, jusqu'au échanges inter-codes. Il s'agit donc d'une bibliothèque qui permet de mettre l'accent sur la donnée pour être au coeur de la chaîne de simulation. La communication avec une base de données

se fait via une API haut niveau. Chaque processus communique à Hercule le ou les domaines de la simulation dont il est responsable. Avec RAMSES par exemple, chaque processus gère un seul domaine, constitué de l'ensemble des cellules dans l'intervalle de la courbe de Hilbert dont il est responsable. Avec ces informations, Hercule va construire une base de données contenant D domaines pour une application utilisant N processus (avec RAMSES, on a $N = D$). Lors de la relecture, l'application est libre de choisir sa répartition des domaines à lire entre les processus. Pour les protections/reprises par exemple, la répartition des domaines est généralement la même que lors de l'écriture de la base sauf pour un éventuel changement d'équilibrage ou pour relancer une simulation sur un nombre de processus différents. Pour la relecture des données, l'application a la liberté de pouvoir accéder à l'ensemble des domaines, à une sous-partie ou même à certains domaines spécifiques. Cette approche sera exploitée dans le chapitre 4 sur le post-traitement de grands volumes de données.

Lorsqu'une application souhaite ajouter (ou rajouter) des informations dans une base de données, elle doit y associer un identifiant chronologique (le temps de simulation par exemple, ou le facteur d'expansion dans les simulations cosmologiques). Ainsi, à la fin d'une simulation, une base de données contiendra donc une collection de pas de temps qu'on appellera contexte et chaque contexte contiendra un nombre D de domaines. Le stockage de cette base se fait sous la forme de fichiers binaires. Pour tirer profit de l'architecture parallèle, Hercule va regrouper les données de P processus dans le même fichier jusqu'à atteindre une taille maximum paramétrable, au-delà de laquelle un nouveau fichier sera créé pour contenir la suite des données. Le paramètre P représente donc le *nombre de contributeurs par fichier* (NCF) et fait partie des paramètres qui peuvent être ajustés. Avec $P = 1$ on retrouve le système *file-per-process* et avec $P = N$ le système *single-shared-file*. Un compromis intéressant est $P = 16$, qui permet généralement de conserver des performances correctes tout en réduisant le nombre de fichiers, par rapport à $P = 1$, voir section 2.3.4. On notera également que l'accumulation des données dans les mêmes fichiers permet également de réduire le nombre de fichiers produits. Il est important de noter que cette mécanique reste totalement transparente pour les utilisateurs et permet à Hercule de pouvoir faire évoluer ses mécanismes en fonction des architectures et des systèmes de fichiers.

Enfin, Hercule propose deux approches pour l'écriture de données : asynchrone et synchrone. La différence entre ces deux approches est cruciale lors de l'écriture de tableaux. Dans la première méthode, les données sont envoyées via un pointeur et une taille (ou plusieurs en fonction du dimensionnement) et Hercule *conserve* ces informations sans faire de copie des données. Les écritures étant, en principe, déclenchées lors de la fermeture du contexte, il est important que l'application ne modifie pas les données entre l'appel de demande d'écriture et la fermeture du contexte. Dans la seconde approche, les données sont copiés en mémoire par Hercule. Cette méthode est nécessaire lorsque l'application réutilise un même tableau temporaire pour sauvegarder différentes informations. L'inconvénient de cette méthode étant bien sûr un surcoût mémoire, qui peut éventuellement être réduit en forçant l'écriture avant la fermeture du contexte (c'est d'ailleurs ce qui est fait lors de l'écriture du modèle lightAMR). Il y a également un coût à payer au niveau des performances d'E/S qui sont généralement diminuées.

Les données issues des applications ont des objectifs très différents : protection/reprise, dépouillement ou inter-codes. Hercule a donc un comportement différent en fonction du type de sortie ou base de données crée. On s'intéressera principalement au fonctionnement des bases de données de protections/reprises et aux bases de dépouillement. Comme on le détaillera à la section suivante, dans le cas des protections/reprises, le consommateur des données est le producteur des données, en conséquence une base de ce type n'est pas destinée à être partagée entre des applications. L'écriture et la lecture de données se fait donc de manière très classique pour une bibliothèque clé-valeur. A l'opposé, les données de dépouillement sont destinés à être consommées par un code d'analyse, de post-traitement et/ou de visualisation. L'organisation et la sémantique des données deviennent alors des aspects cruciaux. Pour ce faire, Hercule se base sur un dictionnaire qui permet de définir des objets de types simples (entier, réel, tableau) et de types complexes (ensemble d'objet simple ou complexe). Ce dictionnaire est utilisé par les applications pour décrire les données dans la base. Par exemple, on retrouve des objets complexes pour gérer les maillages structuré, non structuré, AMR, etc. De plus, un support peut-être rajouté à un objet ce qui permet d'y associer des grandeurs physiques. Par exemple, imaginons un code qui permet de gérer un nuage de point 3D avec 4 processus MPI et qui souhaite faire une sortie destinée à la visualisation et au post-traitement. On peut rajouter dans le dictionnaire la configuration de la figure 2.3.

```
1 <ComplexType name="Point">
2   <Attr name="identifiant" type="int_8"/>
3   <Attr name="coordinates" type="float_8[3]"/>
```

```
4  </ComplexType>
5
6  <ComplexType name="PointCloud">
7    <Attr name="nbElements" type="int_4"/>
8    <Attr name="elements"   type="Point[nbElements]"/>
9  </ComplexType>
```

Extrait 2.3 – Exemple de description au format XML de `Hercule`.

On définit un `Point`, comme un objet complexe possédant deux attributs : un identifiant et un tableau de coordonnées flottantes. On notera que les types `int_8` et `float_8` font référence à des types simples, correspondant respectivement à un entier sur 8 octets et un flottant double précision. On définit ensuite un autre objet complexe `PointCloud` comme étant une collection de `Point` dont le nombre sera porté par l'attribut `nbElements`, qui permet de dimensionner l'attribut `elements`. Automatiquement, par héritage, l'attribut `identifiant` associé à `elements` sera dimensionné suivant la taille $nbElements * 1$ de même, `elements.coordinates` prendra la taille $nbElements * 3$. Le nom des attributs servira de point d'entrée lors de l'utilisation de l'`API` pour créer des objets, les attributs, etc. Par exemple, pour créer un nuage de points, dans le code parallèle (ou non) chaque processus, peut contribuer à la base de données en définissant un `PointCloud` dont il est en charge. On montre un exemple partiel de l'utilisation de l'`API` sur la figure 2.4. Dans un premier temps, on crée l'objet global au niveau de la racine de la base (`her_root`), que l'on récupère dans la variable `her_cloud`. Ensuite, on crée l'attribut de dimensionnement `nbElements` de `elements` ce qui nous permet de renseigner ensuite les identifiants des points.

```
1  type(HIc_Obj) :: her_root, her_cloud, her_elems, her_masses
2
3  ! Open Hercule database of type "HDep" and create + open context
4  [...]
5  ! ready to write data
6
7  np = 2  ! number of points
8  points_ids(1) = 1; points_ids(2) = 2;
9  points_coords(1, 1) = 1.0; points_coords(1, 2) = 1.0; points_coords(1, 3) = 1.0;
10 points_coords(2, 1) = 2.0; points_coords(2, 2) = 2.0; points_coords(2, 3) = 2.0;
11 points_masses(1) = 1.0; points_masses(2) = 3.0;
12
13 call hicf_Obj_create(her_root, "NuageDePoints", "PointCloud", her_cloud, err)
14 call hicf_Obj_setAttrVal(her_cloud, "nbElements", np, err)
15 call hicf_Obj_getAttr(her_cloud, "elements", her_elems, err)
16 call hicf_Obj_setAttrVal(her_elems, "identifiant", points_ids, np, err)
17 call hicf_Obj_setAttrVal(her_elems, "coordinates", points_coords, np*3, err)
18
19 call hicf_Obj_create(her_elems, "masses", "GrandeurScalaireFlottant", her_masses, err)
20 call hicf_Obj_setAttrVal(her_masses, "val", points_masses, np, err)
```

Extrait 2.4 – Exemple d'utilisation de l'`API` objet de Hercule avec l'interface Fortran, pour une base de dépouillement (`HDep`).

Il est possible de rajouter des `supports`, mentionnés précédemment, pour rajouter une masse par exemple, aux objets de type *Point*, voir ligne 19-20. Cela se fait en créant un objet de type *GrandeurScalaire-Flottant* que l'on rattache à l'objet `her_elems` (de type `elements`). Il est essentiel de noter que le tableau décrivant les masses doit être dimensionné au nombre d'éléments (attribut `nbElements`), sinon une erreur de l' `API` d'`Hercule` est levée. En suivant le même principe, on peut aussi rajouter une unité pour cette nouvelle grandeur, etc. Toute cette sémantique est indépendante de la mécanique interne d'`Hercule` pour l'agrégation des données au sein des fichiers constituant la base. Néanmoins, puisque chaque processus décrit un objet appelé `NuageDePoints` de type `PointCloud`, la base contiendra, dans chaque domaine, un objet de ce type (éventuellement vide, si un processus n'a pas de point à sauvegarder).

Afin de faciliter la gestion et l'exploration de la base de données, un ensemble d'outils similaires à *h5dump* (outils associés à la bibliothèque `HDF5`) et à `Panoply`, [NASA, 2023] (visualiseur de données `HDF5` de la `NASA`), sont disponibles et permettent aux utilisateurs d'interagir avec leurs données. Par exemple, l'outil `hercule_view` sur la figure 2.3 permet de facilement naviguer dans le contenu d'une base. Ici, on a une base de type `HDep` (dépouillement)

contenant 7 pas de temps, chacun contenant les données des 4096 domaines (*sous-domaine* dans la terminologie d'`Hercule`). On y retrouve des méta-informations sous l'objet *RamsesSimulation*, mais également différents maillages de type *Particules* et *Ramses3D*. Des méta-informations sur les attributs sont également disponibles. En effet, `Hercule` calcule des grandeurs telles que le minimum ou le maximum des tableaux mais aussi une `checksum`, qui fait partie des mécanismes internes de vérification de corruption de données. Il s'agit uniquement d'un outil d'exploration et non pas de visualisation ou d'analyse.

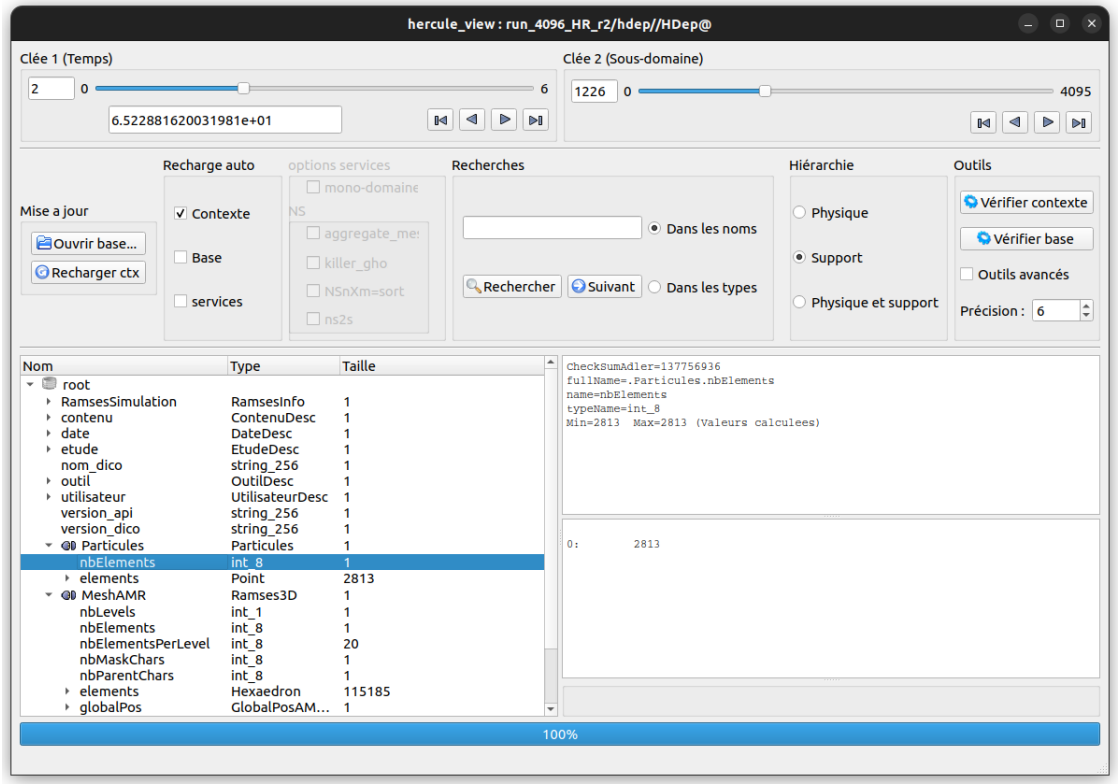

FIGURE 2.3 – Visualisation du contenu d'une base de donnée `Hercule`, de type `HDep`, avec l'outil `hercule_view`.

2.2.4 Le choix d'utilisation de `Hercule`

Initialement, les raisons qui ont poussé à l'utilisation de la bibliothèque `Hercule`, plutôt qu'une autre (telles que `HDF5`, `MPI-IO`, `POSIX` ou `NetCDF`), sont nombreuses. Dans un premier temps, `Hercule` est une bibliothèque développée en interne au `CEA`, ce qui permet de faciliter les contacts directs avec les développeurs et de bénéficier rapidement de nouvelles fonctionnalités ainsi que de leurs expertises. De plus, il s'agit d'une bibliothèque qui a déjà fait ses preuves dans plusieurs codes en interne, tous dans un contexte de calcul haute performance.

Deuxièmement, l'approche *clé-valeur* est indispensable pour faciliter les accès aux données. De plus, l'`API` est simple d'utilisation et permet de cacher les subtilités de la gestion de l'agrégation de données, contrairement à `HDF5` par exemple, où l'utilisateur doit fournir les `offsets` pour chaque processus `MPI` au sein des `hyperslabs`. Avec `Hercule`, quel que soit la paramétrisation des `E/S` (nombre de contributeur par fichier, taille des fichiers et répartition des pas de temps, etc), l'interface reste inchangée. C'est un point primordial qui permet de faciliter grandement le développement d'outils de post-traitement. En effet, avec `HDF5`, répartir les données dans plusieurs fichiers est possible en créant un sous-communicateur `MPI` par exemple ; mais cela nécessite de développer une surcouche à la bibliothèque, qu'il faut ensuite maintenir, faire évoluer, partager entre les outils, etc. D'autre part, la réduction du nombre de fichiers produits, grâce au partage des fichiers entre plusieurs contributeurs, mais aussi grâce à l'accumulation de plusieurs pas de temps, sont des points importants. En effet, la communauté des utilisateurs de `RAMSES` souffre du grand nombre de fichiers produits par le code notamment à cause des restrictions imposées sur le nombre d'inodes maximum par utilisateur imposés sur les systèmes Lustre des centres de calculs.

Troisièmement, l'aspect *centré sur les données* (*data-centric*). Ainsi, l'accent est mis sur ce que représente les données grâce au mécanisme du dictionnaire sémantique pour décrire des modèles. Il apporte une solution qu'on ne retrouve pas (ou pas de manière aussi riche) dans les bibliothèques concurrentes. De cette façon, un outil capable de comprendre le dictionnaire natif de `Hercule`, ou un dictionnaire compatible partagé par une communauté, est capable de lire, écrire et manipuler les données. En conséquence, `Hercule` permet de mettre l'accent sur le développement de modèles de données, mais également de standardiser les modèles de manière à abstraire le code qui produit les données, du code qui va les relire et les traiter.

Enfin, son utilisation grâce à son interface haut niveau homogène dans différents langage, `Fortran`, `C`, `C++`, est un atout important par rapport aux bibliothèques concurrents.

2.2.5 Les Entrées/Sorties dans les codes de calcul

Une fois qu'une bibliothèque d'`E/S` a été choisie pour écrire et lire des données, vient alors un problème plus épineux : *Quoi écrire comme données et sous quel "format" ?*. Dans un premier temps, quelques précisions sont nécessaires pour éclaircir la notion de `format de fichier` et `format de données`. En effet, le format de fichier fait référence à la structuration du format de données dans le fichier. Par exemple, pour le format de fichier `PNG`, on s'attend à trouver, dans l'ordre, les 8 octets de la signature `PNG`, suivi de 25 octets de l'en-tête ou `méta-données` (largeur de l'image en pixel, hauteur en pixel, etc.), puis des blocs de données de l'image à proprement parler (qui sont de taille variable en fonction du modèle utilisé) et enfin, un bloc de 12 octets stipulant la fin du fichier, [Adler and Al., 2022]. Le format de données `PNG` fait plutôt référence au modèle utilisé pour représenter l'image (matrice 2D). Par exemple, on peut citer le modèle 8 bits par canal (rouge, vert, bleu) soit un peu plus de 16 millions de couleurs, ou encore le modèle 32 bits qui permet de rajouter 8 bits pour le canal de transparence. Il est important de noter que le format `PNG` est auto-portant. C'est-à-dire que le fichier contient toutes les informations nécessaires et ne nécessite pas de faire des "a priori" ou des suppositions sur le contenu. De plus, il est standardisé : n'importe quel outil ou développeur qui implémente le standard peut relire le contenu et manipuler les données. Bien évidemment, un fichier `PNG`, est un fichier binaire comme on pourrait l'écrire avec l'interface standard `POSIX`. On pourrait utiliser la bibliothèque `HDF5` pour stocker des images `PNG` pour faciliter la gestion des données en les regroupant en sous-groupe en fonction d'une catégorie par exemple mais il faudrait alors définir une structure hiérarchique et probablement développer un lecteur pour interfacer les outils d'analyses d'images avec les données du fichier `HDF5`. En astronomie par exemple, on retrouve les fichiers `FITS`, [Wells et al., 1981], qui ont été standardisés et conçus pour les données astronomiques et permettent de stocker des tableaux multi-dimensionnels, des images (sous forme de tableau bidimensionnel), des tables, des cube de données, des spectres, etc. L'objectif étant bien sûr de faciliter les collaborations et les échanges de données entre les scientifiques.

Pour les codes de simulation destinés à tourner sur des centres de calcul haute performance, tel que `RAMSES`, on va généralement distinguer plusieurs types de sorties avec des modèles de données différents en fonction des besoins : *protection/reprise*, *post-traitement*, *monitoring*, etc. Malgré l'utilisation de plusieurs milliers d'unités de calculs en simultané, une simulation s'étale généralement sur plusieurs jours ou plusieurs semaines, voire même plusieurs mois pour les plus importantes. De plus, elles nécessitent souvent d'être monitorées scientifiquement pour vérifier la bonne paramétrisation des processus physiques et sont donc découpées en petits `runs` de quelques heures ou quelques jours. Il est donc nécessaire de pouvoir reprendre une simulation lorsque celle-ci s'est arrêtée. On appellera donc sortie de *protection/reprise* les écritures/lectures qui visent à relancer une simulation pour en poursuivre l'évolution. Le consommateur de ce type de données est donc le code lui même. A l'opposé, on retrouve les sorties de *post-traitement* qui sont destinées à être analysées pour étudier les processus physiques mis en jeu au cours de la simulation, dans le but de produire du matériel scientifique qui pourra ensuite être communiqué (publication scientifique, vulgarisation, etc). Le consommateur de ce type de données est un code et/ou des scripts d'analyse qui sont modulables et variés en fonction des besoins. Finalement, il peut aussi y avoir des sorties beaucoup plus succinctes et légères, destinées au monitoring de la simulation. La plupart du temps, il s'agit tout simplement des `logs` qui permettent d'afficher des informations relatives au bon déroulement de la simulation (description du maillage, énergie totale, performances de calcul, occupation mémoire, pas de temps, etc). Pour les deux premiers types de sorties, il est important de mettre un place un modèle de données approprié.

Le point important des sorties de *protection/reprise* (et son modèle de données associé), est qu'elles

doivent permettre de poursuive une simulation comme si elle ne s'était pas arrêtée, le code doit donc être réinitialisé dans un *état identique*. Cela implique certaines contraintes. En effet, elles doivent donc naturellement contenir toutes les informations nécessaires à la reprise. Elles sont donc parfois plus volumineuses que les sorties d'analyses. De plus, il est préférable d'utiliser les données brutes telles qu'elles sont stockées en mémoire et ainsi éviter des calculs et/ou le ré-ordonnancement, qui en plus d'être inutiles et consommateurs de mémoire temporaire, sont potentiellement sources d'erreurs (e.g. conversion de grandeurs conservatives/non conservatives pour éviter des problèmes d'arrondis). A la section suivante, ces principes seront appliqués au code `RAMSES` pour améliorer les performances d'`E/S` du flux de données de protection/reprise. Néanmoins, puisque les données de ce type de sortie sont destinées uniquement (en principe) au code qui les a produit, il n'y a pas de contrainte forte sur la structuration des données ou de l'utilisation d'un modèle standard. En effet, elles ne sont pas destinées à être partagées et/ou analysées scientifiquement (au sens métier) et/ou utilisées par d'autres codes. On y retrouve généralement, comme dans l'exemple du modèle `PNG`, des `méta-données` (il s'agit généralement des paramètres de la simulation) et des `données brutes`.

Les sorties de post-traitement sont donc destinées à l'analyse scientifique des résultats de la simulation. Ces données sont traitées par des outils d'analyses dédiés ou des scripts personnalisés par les utilisateurs. En conséquence, elles doivent être partagées entre plusieurs applications (voir plusieurs utilisateurs) et elles requièrent donc un modèle de données plus riche, auto-portant et si possible, standardisé. Les notions de standardisation et d'"auto-portance" du modèle sont capitales. En effet, on entend par "auto-portant" le fait que les données contiennent toutes les informations nécessaires pour décrire le contenu et pouvoir être analysées. Deux exemples simples sont : la nomination des champs hydrodynamiques et la description des unités utilisées. Il peut également être important de rajouter des variables qui permettent de définir le pas de temps analysé tel que le temps physique de la simulation ou un nombre de pas temps d'intégration, etc. Finalement, il s'agit principalement de méta-données descriptives. Ces méta-données descriptives sont, au même titre que la standardisation du modèle (voir ci-après), un point essentiel pour le partage de données, [Gray et al., 2005].

On entend par "standard" un modèle à minima documenté et détaillé dans la littérature. Par exemple, le modèle `lightAMR`, a fait l'objet d'une publication qui détaille sa construction et son fonctionnement. Ainsi, n'importe quel utilisateur ou application, qui implémente ce "standard" pourra lire, interpréter et analyser les données, indépendamment du code qui les a produites. C'est avec la standardisation que vient ensuite la notion de portabilité et facilité d'échange des données entre utilisateurs. On notera également, que cela permet de développer et mutualiser les efforts de développement au niveau des outils d'analyses. En effet, si plusieurs codes implémentent un même modèle de données standardisé pour les sorties de post-traitement, alors il est possible de mettre en place un outil d'analyse qui pourra être partagé par l'ensemble des utilisateurs et bénéficier des retours d'une communauté plus large, voir chapitre 4. Par exemple, le modèle `non structuré` est très répandu et standardisé, et peut donc être interprété par la plupart des outils d'analyses et de visualisation. Des outils tels que `Paraview`, [Ahrens et al., 2005], [Kitware, 2023], implémente même directement des lecteurs spécifiques pour relire des données au modèle *non structuré* écrite au format `HDF5` avec un fichier descriptif au format `XDMF`. Ainsi, en quelques cliques les utilisateurs peuvent analyser et visualiser leurs données.

Bien évidemment, des données ou méta-données "optionnelles" peuvent être rajoutées à un modèle, si elles sont pertinentes ou si elles peuvent faciliter le travail des outils d'analyses. Par exemple, dans le modèle `lightAMR`, voir chapitre 3, on rajoute un tableau donnant le nombre de cellules `AMR` par niveau. Cette information pourrait être recalculée à partir des données décrivant le maillage mais cela implique de lire des données et d'effectuer des calculs alors que le surcoût de rajouter un "petit" tableau `1D`, même sur tous les domaines de la simulation, est finalement très faible et permet de faciliter les post-traitements. A l'opposé, rajouter, par exemple, la norme d'un champ vectoriel en plus de ses composantes doit être réfléchi parce que le surcoût en terme de stockage, peut être non négligeable par rapport au coût de recalculer la grandeur. Il y a donc un bon équilibre à trouver. On notera que dans certaines applications, l'utilisation de techniques dites *In-Situ* permet d'effectuer l'analyse directement dans l'application, et de produire directement des données réduites tels que des images, des courbes, etc. On reviendra sur ces méthodes d'analyses qui essayent de répondre au problème du très grand volume de données produit par les simulations numériques, voir chapitre 4.

Sur la figure 2.4, on représente schématiquement le flux de sortie de données d'analyse. Le schéma serait quasiment identique pour une sortie de protection/reprise, à la différence des modèles de données qui serait propre au code. On met en avant les trois aspects différents : les structures de données computationnelles, les modèles de données utilisées pour les sorties d'analyses et enfin le format du fichier associé à la bibliothèque d'`E/S` utilisé.

Les structures de données "computationnelles" sont utilisées par le code pour faire les calculs de manière efficace et performant. On notera qu'elles peuvent être influencés par la cible matérielle. Ensuite il vient les modèles de données dédiés aux analyses. Il est important de noter qu'il doit y avoir une forme de cohérence et de compatibilité entre le modèle et le type de maillage utilisé par le code. Par exemple, il n'est pas idéal d'utiliser le modèle *non structuré* pour décrire une grille `AMR` comme on le montrera au chapitre 4. Néanmoins, pour les phases de développement, et bien que le modèle *non structuré* est loin d'être optimal, la facilité d'analyse des données, notamment avec `Paraview`, est très utile.

Ensuite, on s'intéresse au module d'`E/S`. D'un point de vue plutôt orienté génie logiciel, une fois le modèle décrit en mémoire, il n'est constitué que des structures informatiques de base qui doivent pouvoir être écrites (ou lues) par des bibliothèques d'`E/S` (tableau ou variable). Pour faciliter le changement de bibliothèque, il est généralement préférable de mettre en place une interface agnostique qui se chargera d'écrire ou lire les données avec un `back-end` (bibliothèque `E/S`) choisi. Certaines bibliothèques ont même été développées spécifiquement pour servir d'interface à toute une collection de bibliothèques `E/S` : c'est le cas de `PDI` (Parallel Data Interface), [Roussel et al., 2017]. Pour les `protection/reprise`, il peut être relativement facile de mettre en place une telle interface commune, parce que le modèle de données ne souffre pas de contrainte stricte, comme on le verra à la section suivante. Toutefois, pour un modèle de données de post-traitement, l'opération est plus délicate parce que le format est plus strict. Par exemple, une interface commune entre `HDF5` et `Hercule` n'est pas évidente à cause de la description stricte du modèle de données et de la notion d'objet et de support, évoquée lors du fonctionnement de `Hercule`.

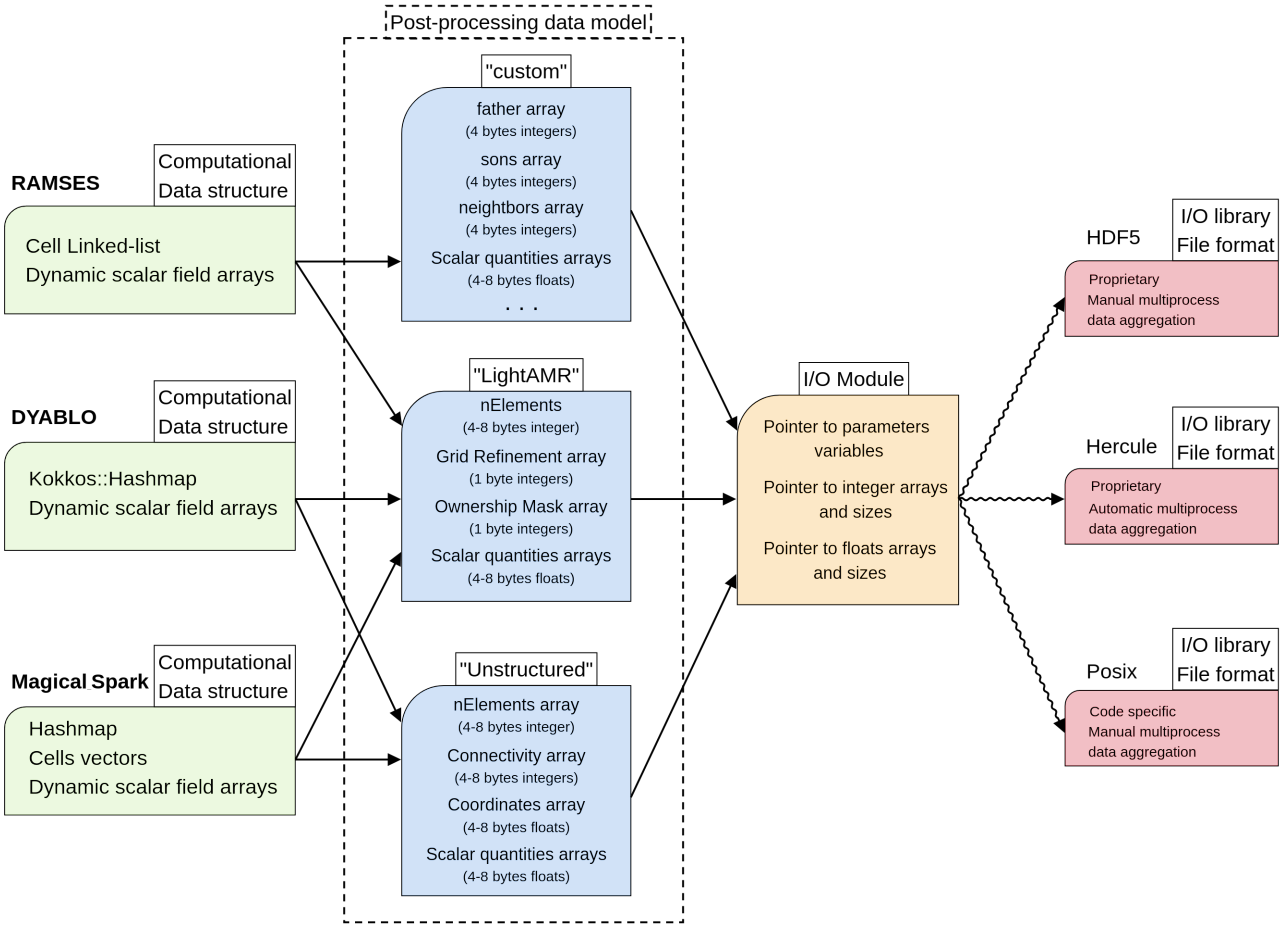

FIGURE 2.4 – Différences entre les structures de données pour le calcul, les modèles de données pour les sorties de post-traitement et la structure de données dans les fichiers de sortie.

Au bout de la chaîne, on retrouve le format de fichier. Il y a une différence majeure entre le modèle de données utilisé et le format de fichier. Par exemple, en utilisant l'interface `POSIX` avec le modèle *non structuré*, dans le cadre d'un code qui utilise le paradigme d'`E/S` *file-per-process* : chaque processus va décrire, dans son fichier,

le maillage et les grandeurs suivant le modèle. Le format de fichier correspondra à l'ordonnancement des données binaires au sein du fichier. De fait, même si le format *non structuré* est standardisé, ce n'est pas pour autant que le format de fichier est standard. En conséquence, dans ce cas précis, il est indispensable de fournir un fichier descriptif du format pour qu'une application puisse relire et traiter les données. Cette remarque reste vraie avec la bibliothèque `HDF5`. Néanmoins, ce n'est plus l'ordonnancement des données binaires, et donc le format de fichier à proprement parlé, qui doit être standardisé, puisque qu'il est géré directement par `HDF5` ; mais plutôt la structure hiérarchique du fichier. `Paraview` contourne ce problème en proposant aux utilisateurs de fournir un fichier `XDMF` (*eXtensible Data Model and Format*) qui permet de décrire, de manière standardisée, les maillages et les grandeurs qui sont stockés dans le fichier `HDF5`. Ce problème ne se pose pas avec `Hercule`, qui dispose nativement d'un fichier descriptif : le dictionnaire `Hercule`.

Lorsqu'on se place dans le contexte d'`E/S` parallèles et de la création d'un fichier partagé par plusieurs processus avec l'approche *multiple-shared-file* ou *single-shared-file*, une contrainte supplémentaire s'ajoute vis-à-vis de la gestion de l'écriture du modèle de données. En effet, avec le fonctionnement de `HDF5`, il faut utiliser les `hyperslabs` (voir section 2.2.2) pour agglomérer les données. Certains modèles de données peuvent être facilement agglomérés en fonction de leur granularité (voir chapitre 3). Par exemple, il est facile d'agglomérer les données *non structurées* de plusieurs processus. Typiquement, cela signifie que l'on peut concaténer deux tableaux du modèle issus de deux processus différents, sans dénaturer le modèle. La visualisation des deux maillages séparés ou du maillage décrit par la réunion des deux tableaux (comme si c'était un unique maillage *non structuré*) sera identique. Cette opération sera donc transparente pour les outils d'analyses et de visualisation. Pour d'autres modèles, par exemple le modèle `lightAMR`, cette opération n'est pas possible, et l'analyse d'un maillage par "concaténation" ne fonctionnera pas. On revient plus en détail sur cette problématique au chapitre 3. Avec `Hercule`, la gestion de l'agrégation des données des processus au sein des fichiers est gérée directement par la bibliothèque de manière transparente pour l'utilisateur.

Pour la mise en place d'une interface commune, on se retrouve donc rapidement avec la nécessité de se calquer sur la bibliothèque `E/S` la plus contraignante, tout en essayant de rester suffisamment générique. Pour `HDF5` et `Hercule`, cela implique notamment de se conformer au modèle de fonctionnement de `Hercule`, plus contraignant que `HDF5`. Une autre option consiste à développer une surcouche de gestion à `HDF5` pour gérer l'écriture et la lecture de données avec une gestion transparente de l'agrégation des données, comme le fait `Hercule`. J'ai développé ce type de surcouche à `HDF5` qui est utilisée dans l'outil d'analyse présenté au chapitre 4.

2.3 RAMSES : intégration de Hercule

2.3.1 Les problèmes des `E/S`

Comme on l'a mentionné précédemment, `RAMSES` est un code qui utilise l'approche *file-per-process* pour écrire les résultats de simulation, et va même créer plusieurs fichiers par processus. Tant que les utilisateurs lancent des simulations qui ne requièrent pas beaucoup de processus `MPI` (< 8000 processus), le nombre de fichiers produits reste relativement raisonnable et administrable. Cependant, avec l'amélioration rapide de l'équipement des centres de calculs et la diminution de la quantité de mémoire par coeur de calcul, les simulations `RAMSES` (code uniquement `MPI`) sont devenues de plus en plus gourmandes en ressources et en nombre de processus. Comme on l'a vu précédemment, l'approche *file-per-process* n'est plus soutenable en terme de performance des `E/S`, et même délétère pour les `MDS` (meta-data server) des centres de calculs, surtout pour un code qui génère plusieurs fichiers par processus. La scalabilité des `E/S` de `RAMSES` était ainsi limitée à environ 8000 processus `MPI`. Afin de résoudre partiellement ce problème, un système de groupe d'écriture à été mis en place. Ce système permet de limiter le nombre de fichier par dossier, en répartissant les fichiers des processus `MPI` dans des dossiers différents (chaque groupe de N processus disposant de son propre dossier, N étant paramétrable). Ainsi, la charge d'`E/S` à gérer pour le système de fichier est mieux répartie. D'autant plus qu'au sein d'un groupe, les processus écrivent les uns à la suite des autres grâce à un échange de *jetons*. Néanmoins, cette approche ne permet pas de résoudre le problème fondamental des `E/S` dans le code et implique des communications inter-processus pour s'échanger des jetons d'autorisation d'écriture. On notera également que la politique des centres de calculs a été durcie et des limites sur le nombre total de fichiers autorisés par utilisateur rendent les simulations `RAMSES` très compliquées lorsque l'utilisation de plusieurs milliers de processus sont nécessaires.

De plus, la taille des fichiers produits est variable, et dépend du raffinement du maillage (et donc de la simulation ou du type de simulation), mais aussi des modèles physiques résolus. Par exemple, une simulation utilisant de la magnéto-hydrodynamique, va nécessiter de stocker 6 champs scalaires supplémentaires (le champ magnétique) dans les fichiers `HYDRO`. D'autre part, d'un point de vue technique, les données dans les routines d'écriture sont regroupées suivant le maillage `AMR`, du niveau le plus petit (niveau 0), vers le niveau `AMR` le plus élevé (niveau `AMR` maximum) et sont écrites niveau par niveau. Ainsi, avec une approche d'écriture par niveau, les appels sont donc fréquents et les volumes de données relativement petits. On rappelle que `RAMSES` utilise uniquement la parallélisation en mémoire partagée et qu'en conséquence pour maximiser l'utilisation des noeuds de calcul, il y a autant de processus que de coeurs par noeuds (voir section sur le dimensionnement du chapitre 6). La mémoire disponible par processus est relativement limitée (de 4 *Go* à 2 *Go* en fonction des configurations), la taille des données par processus est donc assez petite.

Une tentative d'intégration de `MPI-IO` et `HDF5` a été fait par [Wautelet and Kestener, 2011] qui ont conclu que le schéma des `E/S` induit par l'utilisation du maillage adaptatif est complexe et non régulier dans `RAMSES`. En effet, le nombre de cellules géré par chaque processus `MPI` peut-être très variable, jusqu'à un facteur 10 suivant les simulations, lié principalement au coût prohibitif d'une opération d'équilibrage de charge trop régulière. Il a donc été montré que l'utilisation de `HDF5` ou de `MPI-IO`, avec ou sans fichier partagé, ne permettait pas d'obtenir un gain ou une meilleure scalabilité du code comparée à la version standard *file-per-process* avec les appels d'`E/S POSIX` de `RAMSES`.

Pour résoudre ce problème, il est essentiel de noter qu'un des premiers points bloquants est lié à l'unicité du flux de données. En effet, dans sa version principale, `RAMSES` ne possède qu'un seul type de sortie dont l'objectif est double : faire une protection/reprise et/ou analyser les données. Le problème de ce type d'approche est que les données doivent donc être suffisamment complètes pour pouvoir reprendre l'exécution du code dans un état identique, mais également suffisamment facile à manipuler pour les outils d'analyses. Cela implique, par exemple, d'écrire tous les champs physiques calculés avec le maximum de précision (double précision) afin de s'assurer de limiter les erreurs et de pouvoir faire repartir la simulation comme si le code ne s'était pas arrêté. Or pour le post-traitement, comme nous le verrons dans le chapitre sur l'analyse de données, la double précision n'est pas forcément toujours nécessaire, de même que tous les champs ne sont pas toujours nécessaires. De la même façon, le choix dans `RAMSES` est de sortir des grandeurs primitives, plus facile à conceptualiser, malgré l'utilisation des grandeurs conservatives dans le code. En conséquence, des tableaux temporaires sont alloués en mémoire et des calculs inutiles (en écriture lors des protections et à la lecture lors des reprises) sont effectués lors de la lecture ou de l'écriture d'une sortie de protection/reprise. Ce qui est donc nécessaire pour une sortie destinée aux post-traitement et à l'analyse scientifique ne l'est pas forcément pour une sortie destinée simplement à être un point de sauvegarde de l'avancement de la simulation, et inversement. Ainsi, la première action pour améliorer le système d'`E/S` du code est de faire une séparation claire entre le flux des données de protection/reprise et celui destiné à l'analyse.

De plus, avec l'utilisation des appels `POSIX`, les fichiers binaires produits sont le résultat de l'entrelacement des champs à différents niveaux, ce qui complexifie la tâche des outils d'analyses pour la phase de relecture. On notera par exemple que rajouter un champ ou même une simple variable, décale toutes les données. Et ces changements doivent alors être répercutés dans tous les outils d'analyses, dont la maintenance et l'évolution sont donc complexifiés. On adressera également ce problème grâce à l'intégration de `Hercule` et le fonctionnement *clé-valeur*.

2.3.2 Séparation des flux de données

On a vu que `Hercule` permet d'avoir une approche *data-centric* et de proposer des bases de données adaptées à l'objectif principal des données : les bases `HProt` (protection/reprise) et les bases `HDep` (post-traitement). J'ai donc tiré profit de cette possibilité pour rajouter un flux de données spécialisé pour le post-traitement et adapter le modèle de données des protections/reprises. On représente schématiquement sur la figure 2.5, le nouveau système d'`E/S`. Le nouveau mode de fonctionnement est celui du *multiple-shared-file* utilisé nativement par `Hercule`. Désormais, on utilise le paramètre `NCF` pour définir le nombre P de processus qui contribueront à un même fichier partagé, conformément au mode de fonctionnement de la bibliothèque. On notera que cette approche est valable pour les deux bases de données produites. De plus, comme on le voit sur la figure, on regroupe plusieurs sorties aux temps t_i dans la même base de données (dans le même ensemble de fichiers). Avec cette méthodologie, le nombre de fichiers

produits pour une simulation (plusieurs sorties d'analyses et/ou de protection/reprise) est réduit significativement et varie en fonction du paramètre `NCF`. De plus, les fichiers produits sont plus volumineux, ce qui correspond mieux au mode fonctionnement du système de fichiers `Lustre`.

Cette séparation des flux permet également de faciliter la gestion des données produites. En effet, puisque les flux sont physiquement séparés, l'utilisateur à la possibilité de définir des fréquences de sortie différentes. Typiquement, on s'attend naturellement à ce que la fréquence des sorties des protections/reprises soit beaucoup plus faible que celle des sorties destinées au post-traitement. L'utilisateur a également beaucoup plus de souplesse sur les données de la base `HDep`. Il a la possibilité de choisir les champs scalaires ou vectoriels à écrire, d'activer ou non la compression des données, d'activer ou non la réduction de précision (*down-casting*),etc, voir chapitre 3.

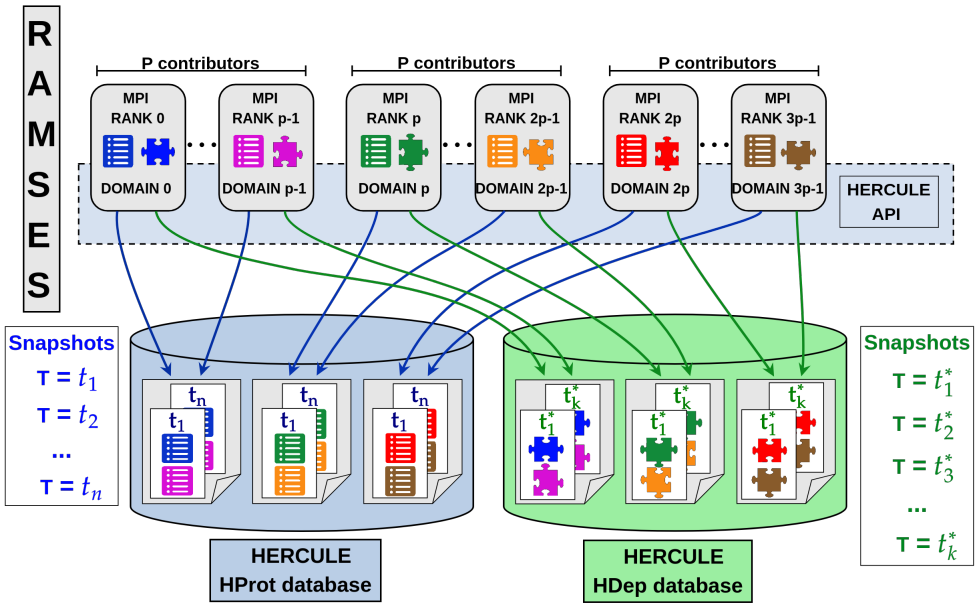

FIGURE 2.5 – Bases de données produites par une simulation `RAMSES` sur $3P$ processus `MPI` où chaque processus écrit un domaine d'une base de données `HProt` ou `HDep`. Chaque groupe de P processus contribue à un même fichier, qui contient plusieurs sorties à différents temps.

Se pose ensuite la question du modèle de données à utiliser dans les deux types de sorties. On s'intéressera ici au format de données pour les protections/reprises et on introduira le format `lightAMR`, spécifique aux sorties d'analyses, au prochain chapitre. Comme on l'a mentionné précédemment, afin d'optimiser les `E/S` d'un code, il faut parfois changer le modèle / format des données utilisé. Bien que cette approche puisse être coûteuse en terme de développement (perte de compatibilité avec les outils d'analyses ou de monitoring) elle s'est révélée efficace pour l'optimisation des `E/S` du code *FLASH*, [Latham et al., 2012]. On verra à la section sur les *benchmarks*, voir section 2.3.4, que c'était également la bonne approche avec le code `RAMSES`.

2.3.3 Nouveau format de données pour les protections/reprises

Avant de changer le format des données, plusieurs approches ont été essayées avec des granularités de données différentes. La première tentative consistait à remplacer strictement au sein des codes les appels d'écriture des données en `Fortran` par des appels aux fonctions `Hercule` équivalentes. Bien que cette méthode a l'avantage de respecter strictement le format d'origine de `RAMSES`, elle implique l'utilisation de tableaux temporaires où les données sont regroupées par niveau et/ou des conversions de grandeurs primitives/conservatives sont faites. En conséquence, il n'est pas possible de tirer profit de l'asynchronisme de `Hercule`. Cette approche n'est donc pas viable, puisqu'elle induit un surcoût mémoire prohibitif. De plus, les appels d'écriture sont constitués de petits volumes de données et répétés de nombreuses fois dans des boucles. Il y a donc un surcoût non négligeable des appels aux fonctions avec les différentes interfaces.

Comme évoqué aux sections précédentes, l'optimisation des `E/S` peut parfois nécessiter de changer une partie de l'application afin d'obtenir de meilleures performances. La deuxième approche consistait donc à écrire directement les tableaux nécessaires à la reprise du code. On supprime ainsi la nécessité d'utiliser des tableaux temporaires et cela permet d'utiliser le fonctionnement asynchrone (sans copie) de `Hercule`. D'un point de vue théorique, cette méthode correspond beaucoup plus aux attentes du système `Lustre` puisque qu'il y peu d'appels d'écriture avec des tableaux de données de grande taille, pour chaque processus `MPI`. Néanmoins, cette approche induit un surcoût, en terme de stockage, variable qui peut devenir non négligeable en fonction des simulations. En effet, `RAMSES` utilise de l'allocation *dynamique* mais également statique au cours du *run*. C'est-à-dire que pour éviter les allocations et déallocations successives durant l'exécution, les tableaux principaux sont alloués dynamiquement uniquement au début de la simulation, en fonction de paramètres données par l'utilisateur : `ngridmax` et `npartmax`. Ils permettent respectivement de définir le nombre maximum de cellules `AMR` et de particules d'un processus `MPI`. On notera que le code défragmente régulièrement ses tampons mémoire afin de conserver une bonne localité des données et tirer profit du cache des `CPU`. Afin de conserver de la marge pour les équilibrages de charges, il est généralement conseillé de conserver $\sim 10 - 20\%$ d'espace libre. Avec cette méthode, la sortie à donc un surcoût de 10% à 20%, ce qui peut devenir conséquent pour les plus grosses simulations (un surcoût de ~ 750 *Go* si cette méthode avait été adoptée pour la simulation `ExaMilkyWay`).

Finalement, la troisième méthode qui a été adoptée, découle naturellement des tentatives précédentes. Il s'agit d'écrire directement la partie des tableaux ne contenant que les cellules/particules effectivement utilisées, à un pas de temps donné (de taille `ngridcurrent` ou `npartcurrent`) en tirant profit de la défragmentation faite par `RAMSES`. Ainsi, on réduit le nombre d'appels d'écriture aux nombres de tableaux à écrire, il n'y a pas de duplication en mémoire et on restreint le volume de données à écrire au strict nécessaire, sans surcoût. En effet, cette approche permet de s'affranchir de l'utilisation de tableau temporaire, d'augmenter la taille des données à écrire (ou lire) et de bénéficier de l'asynchronisme au sens de `Hercule`. Enfin, on s'affranchit également des conversions inutiles des grandeurs scalaires : conservatives vers non conservatives qui ne sont désormais plus pertinentes.

Pour conclure, on donne le nouveau schéma des flux de données avec l'intégration de `Hercule` et la séparation des flux : protection/reprise et post-traitement sur la figure 2.6. Avec ce nouveau système d'`E/S`, le flux de données de protections/reprises (`HProt`) peuvent être, dans le cadre d'`IO-SEA`, redirigé directement sur du stockage *froid* (en bleu) ; et les données des bases `HDep` sur du stockage *chaud* (rouge) pour être réutilisées rapidement par un code d'analyse pour produire des données à haute valeur ajoutée, stockées sur du stockage *tiède* (jaune), destinées à l'analyse et la communication scientifique.

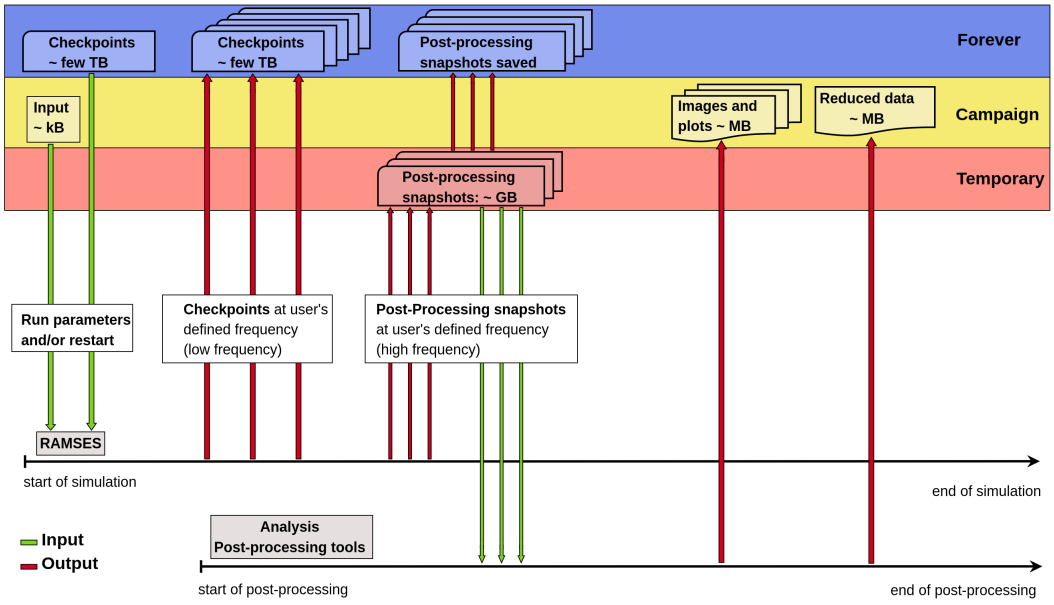

FIGURE 2.6 – Nouveau schéma des flux de données d'Entrées/Sorties de `RAMSES` avec `Hercule`.

2.3.4 Benchmarks des protections/reprises

Des *benchmarks* de scalabilité ont été fait sur la machine `Joliot-Curie` du `TGCC` sur la partition *scratch* (haute performance d'`E/S`) dont la capacité de débit a été évaluée à 300 Go/s. Ces évaluations ont été faites dans un environnement utilisateur sans privilège spécifique et dans une phase de production normale de la machine (hors contexte *Grand Challenge*, etc). En conséquence, d'autres applications peuvent avoir effectué des opérations d'`E/S` au même moment, ce qui peut expliquer l'écart-type élevée dans les résultats. Néanmoins, ce type de *benchmarks* permet de donner une idée précise à la communauté des utilisateurs des performances d'`E/S` (pour les protections/-reprises) de ce qu'ils peuvent attendre de l'intégration d'`Hercule`, dans un contexte réaliste de production.

On a utilisé un cas test *Sedov3D* (explosion d'une bulle dans une boite cubique) avec une configuration du type 2048^3 (2048 cellules dans chaque direction de l'espace). Le raffinement `AMR` et l'intégration temporelle ont été désactivés afin de ne tester que les routines d'écriture. En conséquence, la charge est répartie parfaitement entre les processus `MPI`. On a mesuré la capacité d'écriture de `RAMSES`, c'est-à-dire la quantité d'octets écrits par le code par unité de temps. Le nombre de processus `MPI` a été augmenté de 2048 à 8192 tout en conservant la taille du problème constante. Tous les *benchmarks* ont été réalisés 5 fois et on donne les résultats moyens ainsi que l'écart-type.

Compte tenu de la physique résolue, dans sa version d'origine `RAMSES` a généré deux types de fichiers par processus `MPI` : un fichier `AMR` et un fichier `HYDRO` contenant 5 champs scalaires. Dans la version modifiée avec `Hercule`, la taille maximale des fichiers produits est celle définie par défaut et permet de contenir dans un seul fichier (partagé par P processus `MPI`) toutes les données des processus. De plus, on a fait varier le nombre de contributeurs par fichier `NCF`, entre 4 et 16. Le `NCF` correspond donc au nombre P de processus mentionnés précédemment. Deux paramètres `Lustre` ont été également modifiés : `stripe_size` et `stripe_count`. La taille des *stripe* pouvant impacter l'équilibrage des disques au sein des `OSTs`, elle est en fait forcée à 1 Mo dans la configuration du système de fichier. Après une étude préliminaire sur le nombre d'`OST` à attribuer au dossier de sortie (et aux fichiers qu'il contient), on a conclu que l'ajuster au `NCF` permettait d'obtenir des performances optimales.

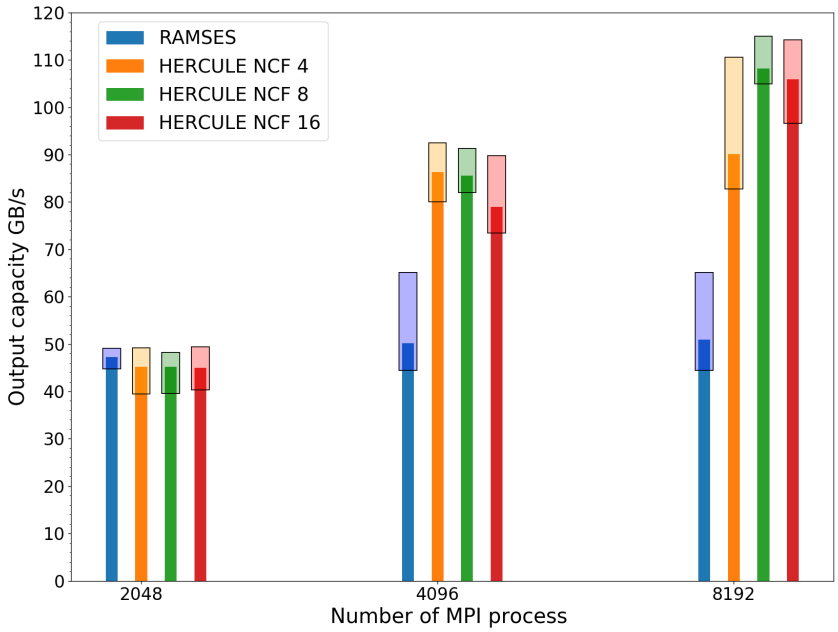

FIGURE 2.7 – Benchmarks des écritures de protection/reprise avec *Ramses* et *Ramses* + *Hercule*. Cas test d'un blast `3D` sphérique dans une grille 2048^3 sans raffinement `AMR`, moyenné sur 5 tests. La barre d'erreur correspond à l'écart-type. `RAMSES` produit respectivement 4096, 8192 et 16384 fichiers. Avec `Hercule`, le nombre de fichiers produits est de $(512, 256, 128)$, $(1024, 512, 256)$ et $(2048, 1024, 512)$.

Sur la figure 2.7, on rapporte les résultats des benchmarks d'écriture pour un cas test de *sedov3d* (disponible dans `RAMSES`), [Strafella and Chapon, 2020]. Les courbes en bleu font référence à la version d'origine de `RAMSES`. Le nombre de fichiers produit est de 2 fois le nombre de processus `MPI`. Les courbes orange, verte et rouge, font référence à la version avec `Hercule` pour trois paramétrisations différentes du nombre de contributeurs par fichier, respectivement, 4, 8 et 16. On voit clairement la saturation de la capacité d'écriture de la version d'origine qui sature à $\sim 50\ Go/s$ lorsque le nombre de processus passe de 4096 à 8192. La version avec `Hercule` en revanche voit sa capacité d'écriture augmenter avec le nombre de processus. Avec 8192 processus, la différence entre les deux versions du code est d'un facteur ~ 2. Il est important de noter également qu'à 8192 processus, la version avec $NCF = 16$ produit seulement 512 fichiers alors que la version d'origine produit 16384 fichiers. Le nombre de fichiers produits par la version avec `Hercule` correspond à $N_o/(2 * NCF)$, avec N_o le nombre de fichiers dans la version d'origine. On notera que le facteur 2 au dénominateur correspond au nombre de fichier différent produit par `RAMSES`, ici 2 (`AMR` et `HYDRO`). On constate également, que le `NCF` a peu d'influence sur les performances, on utilisera donc un $NCF = 16$ pour la simulation `ExaMilkyway` effectuée au chapitre 6.

On notera également que ces benchmarks, bien qu'effectués avec des privilèges d'utilisateur classique et une charge d'exploitation normale de la machine, peuvent ne pas refléter les performances obtenues lors de simulation réelles. En effet, les performances d'`E/S` peuvent être affectées par la présence d'autres applications, d'autres utilisateurs sur la machine et également les différents évolutions matérielles et/ou logicielles. De plus, on se place ici, avec le test *Sedov3d*, et donc dans un cas idéal et symétrique. En effet, le *blast* se présente sous la forme d'une région sphérique. En conséquence, le découpage de Hilbert et l'équilibrage de charge produisent des domaines de simulation très bien équilibrés. Il serait intéressant de poursuivre ces benchmarks avec une simulation réelle et avec un déséquilibrage du volume de données écrit par chaque processus `MPI`, ce qui correspond à pratiquement tous les cas d'usage de `RAMSES`. On notera toutefois que les simulations `Extreme-Horizon`, [Chabanier et al., 2020] ou encore d'une galaxie isolée [Renaud et al., 2013], effectuées sur plus de 10000 processus `MPI`, nécessitaient jusqu'à plus d'une heure pour les phases d'`E/S`. La simulation `ExaMilkyWay` (voir chapitre 6) utilisant les développements détaillés précédemment avec la bibliothèque `Hercule`, sur 16384 processus, effectue ses écritures/lectures de protections/reprises en moins de 2 minutes, pour un volume de données de $\sim 4.5\ To$ à haute résolution. On notera également que les sorties d'analyses avec le modèle `lightAMR` (voir chapitre suivant), beaucoup plus légères et compressées, sont écrites en quelques dizaines de secondes pour un volume (compressé) $< 200\ Go$ à haute résolution.

Enfin, j'ai développé durant ma thèse, sous la forme d'une librairie externe, une surcouche à `HDF5` avec une interface `Fortran`, qui permet de mettre en place différentes stratégies d'écritures parallèles : `single-shared-file`, `file-per-process` et `multiple-shared-file`, dans le but de confronter `Hercule` à `HDF5`. J'ai également intégré ses développements dans un prototype du code `RAMSES`. Il serait intéressant d'effectuer de nouveaux benchmarks d'écriture afin de comparer les deux bibliothèques parallèles entre en fonction des différentes stratégies. Néanmoins, l'intégration de `Hercule` est donc un atout qui permet d'améliorer la scalabilité des `E/S` de `RAMSES`, dont on va tirer profit lors de la simulation `ExaMilkyWay` effectuée durant mes travaux de thèse, voir chapitre 6, mais aussi pour l'analyse de données, chapitre 4.

Deuxième partie

Modèle de données `lightAMR` et post-traitement

Chapitre 3

Le modèle de données LightAMR

L'objectif de ce chapitre est la description d'un modèle de données compact et auto-portant pour les codes `AMR` en arbre tel que `RAMSES`. Avant de détailler le modèle `lightAMR`, on expliquera les raisons qui rendent ce nouveau modèle de données indispensable. Ensuite, des algorithmes complémentaires seront présentés. On verra que ce modèle permet de facilement supprimer la redondance d'information mais également de compresser les données, aussi bien de description du maillage, que les données scalaires associées. Finalement, on testera ce modèle sur des jeux de données réelles issus de simulations `RAMSES` et on quantifiera les gains en terme de stockage. De plus, les algorithmes de compressions de données seront comparés à ceux de la littérature. On présentera également une extension *avec perte* de la compression de données flottantes dont la gestion de l'erreur est simple et garantie, par construction, point à point (par opposition à une erreur garantie sur l'ensemble d'un jeu de données). A la fin du chapitre, on discutera de la compatibilité du modèle avec une structure de données mise en place par le package `VTK` : les *HyperTreeGrid*, pour la visualisation. Le modèle `lightAMR` présenté à fait l'objet d'une publication durant la thèse : [Strafella and Chapon, 2022] (à l'exception de l'extension *avec perte* de l'algorithme).

3.1 Raisons justifiant la mise en place d'un nouveau modèle

Bien que la communauté scientifique s'accorde sur l'intérêt majeur de l'utilisation du raffinement adaptatif de maillage dans les simulations hydrodynamiques pour réduire l'empreinte mémoire des codes et accélérer les calculs, il n'existe pas de modèle de données `AMR` standardisé pour le stockage, l'analyse et la visualisation. Une des principales raisons étant le large éventail d'approches utilisées pour gérer et mettre en place ce type de maillage. Parmi les plus utilisées, on retrouve le type `patch-based` proposé initialement par [Berger and Oliger, 1984], [Berger and Colella, 1989] qui est utilisée dans plusieurs applications tels que `SAMRAI`, [Hornung and Kohn, 2002] ou `CHOMBO`, [Adams et al., 2000]. Le modèle de type `cell-based` proposé et décrit par [DeZeeuw and Powell, 1993], [Khokhlov, 1998] est celui utilisé dans `RAMSES`, [Teyssier, 2002]. Il y a également l'approche `block-based` utilisée par les codes `Flash`, [Fryxell et al., 2000], le code `AMRVAC`,[van der Holst and Keppens, 2007] ou encore `Athena++`, [Stone et al., 2020]. On notera également que l'utilisation d'un même modèle `AMR`, ne signifie pas que les structures de données *computationnelles* utilisées soient identiques puisque celles-ci auront un impact significatif sur les performances du code, sa scalabilité, le type de schéma numérique utilisé pour la résolution des équations et peut même constituer un point bloquant pour l'utilisation de certains matériels tel que le `GPU`.

Pour décrire les données en vue du post-traitement, il y a donc deux solutions possibles : utiliser un modèle de données propre au code (*customisé*, non standardisé), soit utiliser un modèle standardisé mais non adapté. Par exemple, il existe un modèle standard et très générique pour définir un maillage d'éléments quelconques (triangle, quadrilatère, cube, tétraèdre, etc) appelé modèle `non structuré`, qui permet de renseigner des coordonnées de points dans l'espace et leur connectivité. Cette approche peut être utilisée afin de décrire une collection de cellules et peut donc s'appliquer facilement à un maillage `AMR`. Bien que ce modèle, très utilisé, permette de rendre les données compatibles avec de nombreux outils d'analyses, elle pose plusieurs problèmes.

Premièrement, elle génère rapidement des fichiers très volumineux. Pour une grille simple comme celle de la figure 3.1, ce modèle nécessite l'utilisation de 296 valeurs flottantes de coordonnées cartésiennes (en `2D`) plus

148 entiers pour en décrire la connectivité pour seulement 37 cellules, soit 48 octets par cellule. Avec le modèle que l'on détaillera plus loin, pour la même information, 98 valeurs booléennes sont nécessaires, soit seulement 2.7 octets par cellule. De plus, il est important de noter qu'il peut y avoir beaucoup de redondance lors de la description des coordonnées des points des cellules. Mettre en place un algorithme qui permette de réduire cette redondance tout en maintenant la cohérence de la connectivité n'est pas simple, surtout au sein d'un code en mémoire partagée et/ou distribuée. Il est commun pour les simulations actuelles d'atteindre rapidement plusieurs centaines de millions voir plusieurs dizaines de milliards de cellules. En conséquence, l'utilisation du modèle `non structuré` ne permettrait pas d'établir des plans de gestion de données cohérents compte tenu du volume de données qui serait produit. Deuxièmement, les algorithmes d'analyse compatibles avec ce modèle n'ont pas été développés pour les maillages contenant des éléments avec une interface *non conforme*, c'est-à-dire dont l'une des faces d'un élément est partagée entre plusieurs éléments voisins, comme c'est le cas entre deux cellules `AMR` de niveaux différents. Il suffit pour s'en convaincre d'essayer de faire un *iso-contour* avec `Paraview`. Troisièmement, avec ce modèle, la structure hiérarchique du maillage n'est pas conservée parce que les cellules sont décrites de manière individuelle. D'autant plus, pour éviter plus de redondance et d'incohérence dans les données, seule l'enveloppe feuille de l'arbre `AMR` est exportée lors de la création des donnés du modèle `non structuré`. Or la structure hiérarchique du maillage constitue un avantage significatif pour les outils de post-traitement comme on le verra au chapitre 4.

Le modèle de données et le format `IDX`, [Pascucci and Frank, 2001], propose d'exploiter les propriétés de la courbe de Morton pour organiser de très grands volumes de données de grille régulière. L'intérêt de cette approche est qu'elle met en avant l'importance du modèle et de l'organisation des données en amont (au niveau de l'application qui produit les données) pour améliorer les performances d'`E/S` des codes de post-traitement, bien que ce soit un point souvent négligé. De plus, elle permet d'utiliser des méthodes de type *Level-of-Detail* mais également des *external memory algorithm* (ou *out-of-core algorithm*). Ces méthodes sont vitales lorsqu'il y a un très grand volume de données à traiter et que la mémoire (au sens `RAM`) disponible n'est pas suffisante. Les travaux de [Kumar et al., 2012] ont étendu ce modèle et format, avec une `API` parallèle, pour l'écriture de données au format `IDX`, nommée `PIDX`, pour les codes parallèles multi-résolution. Finalement en 2014, une approche étendue spécifiquement pour les codes `AMR` est proposée par [Kumar et al., 2014]. On y retrouve l'utilisation d'*aggrégateurs* et des *concentrateurs* d'`E/S` pour améliorer les performances des codes multi-résolutions. Le problème de cette approche, avec `RAMSES`, est le surcoût mémoire impliqué par l'agrégation des données sur quelques processus. De plus, l'agrégation des données se fait de manière globale et la gestion des données binaires produites n'est pas claire. Néanmoins, on discutera, plus tard dans ce chapitre, de la notion d'agrégateurs et de concentrateurs d'`E/S` qui pourraient être une extension des travaux présentés ici.

Jusque là, à l'exception de quelques simulations de type *Grand Challenge*, les volumes produits par les utilisateurs de `RAMSES` restaient relativement faciles à manipuler, à transférer via le réseau et à analyser. Avec l'arrivée des machines *Exascale* et les nombreuses initiatives de passage à l'échelle des codes, ainsi que l'enrichissement des modèles physiques, on s'attend naturellement à une augmentation du volume moyen produit même pour les plus "petites" simulations. En conséquence, les différents codes `AMR` cités précédemment ont fait le choix d'utiliser un modèle de données qui leur est propre, plus compact que le modèle `non structuré`, quitte à laisser le soin à la communauté des utilisateurs de gérer la relecture et l'analyse des données. Des initiatives de collaboration des outils de post-traitement ont vu le jour, pour proposer un outil dédié à un code spécifique tel que `PyMSES` spécifique à `RAMSES`, ou pour proposer un outil compatible avec plusieurs codes tel que `Yt` qui doit donc implémenter plusieurs lecteurs pour s'adapter aux nombreux modèles différents et aux différentes bibliothèques `E/S` utilisées pour relire les données. L'objectif du modèle `lightAMR` est d'essayer de converger vers un modèle standard et compact, pour les codes `AMR` en arbre, pensé pour le post-traitement.

Finalement, le modèle spécifique de `RAMSES` n'est pas idéal pour l'analyse. Comme on l'a vu précédemment, la bivalence de l'utilisation des données (protection/reprise et analyse) implique trop de contraintes sur le contenu des données. Si on prend l'exemple des simulations `Extreme-Horizon`, [Chabanier et al., 2020] ou `Cosmic Dawn III`, [Lewis et al., 2022], on constate que l'étude de la physique résolue par une simulation `RAMSES` peut rapidement être freinée simplement par le volume de données à traiter. En effet, une protection (ou *snapshot*) d'une simulation comme `Extreme-Horizon` pèse environ 3.6 To, $\sim$ 500 Go de données de maillages, $\sim$ 2.3 To de champs scalaires et $\sim$ 800 Go de particules. En dehors des problèmes de performance et de la scalabilité des `E/S` (lié au nombre de fichiers, ou à l'accès des données qui sont plutôt une problématique lié à la bibliothèque `E/S` utilisé), un tel volume ne permet pas de réaliser un grand nombre de sorties d'analyse (limitation de l'espace disque disponible ou du nombre de fichier par utilisateur sur les centres de calculs) et ne permet donc pas d'avoir un

suivi temporel fin des processus physiques que l'on souhaite étudier. Il est donc important de repenser le modèle de données utilisé pour le post-traitement afin de mettre en place un modèle beaucoup plus compact. Pour cela, il est important de définir le cadre et les besoins des utilisateurs en matière de post-traitement pour concevoir un modèle pertinent et pas seulement très compact. Les besoins répertoriés sont les suivants :

- conserver la structure hiérarchique du maillage,
- conserver l'intégralité de l'information et/ou avoir la possibilité de choisir la perte d'information (*downcasting*, champs scalaires à conserver, etc),
- limiter les calculs nécessaires en post-traitement,
- accéder facilement et rapidement à des régions d'intérêts,
- avoir une description simple, auto-portante et standardisée/documentée du modèle utilisé.

3.2 L'importance de la granularité

Il est important de noter qu'on se situe dans un contexte de code parallèle a minima multiprocessus avec `RAMSES`. Ainsi, afin de limiter les coûts de construction, et notamment de communication inter-processus, mais également afin d'offrir plus de souplesse et de faciliter le post-traitement des données, il est important de considérer la *granularité* de l'information. On entend par là, le bloc élémentaire. Cette granularité à un impact sur l'agrégation des données et permet de définir si le modèle de données "global" est le même que le modèle de données "local" ce qui impactera les outils d'analyse. Cette notion est importante si on cherche à regrouper les données de plusieurs processus pour ensuite les analyser comme étant un seul bloc cohérent. Dans la littérature, on retrouve systématiquement cette approche lors de la mise en place d'une modèle de données, e.g. `PIDX`. Prenons un exemple simple pour illustrer cette problématique et la notion de granularité. En utilisant un modèle qui permet de décrire les coordonnées d'une cellule par un entier, par exemple sa position sur une courbe de remplissage de l'espace. Alors la granularité des données sera minimale, puisque le bloc élémentaire sera un entier. Chaque processus `MPI` décrira donc une collection de cellules sous la forme d'un tableau d'entiers. Ce tableau d'entiers constitue donc le modèle de données local (au processus). On notera qu'on suppose ici que cet entier seul, pour une cellule, permet d'accéder à l'information nécessaire au traitement des données. On notera notamment qu'il n'y a pas d'inter-dépendance entre les entiers pour chaque cellule. Avec cette approche, si les données sont agrégées, par exemple pour l'écriture en parallèle avec `HDF5` ou par choix de l'utilisateur, la concaténation des tableaux des processus ne posera pas de problème vis-à-vis du modèle de données. Ainsi, le modèle de données "global", issu de la concaténation des tableaux des processus, sera le même que le modèle "local". On notera également que la répartition des cellules sur un nombre de processus `MPI` différent ne pose aucun problème puisqu'il suffit de relire qu'une sous-partie du tableau concaténé. Enfin, il est également important de noter qu'aucune communication n'est nécessaire entre les processus, ni pour construire le modèle "local", ni pour conserver la cohérence du modèle lors de l'agrégation (il n'est pas nécessaire de s'échanger des cellules par exemple). Bien évidemment, des communications sont nécessaires pour la synchronisation des processus lors de l'agrégation (l'échange des tailles de tableaux par exemple).

Ce type de modèle avec une faible granularité est très pratique. Par exemple, le modèle `non-structuré` permet de décrire une cellule comme un polygone ou un polyèdre : coordonnées cartésiennes des sommets ainsi que leurs connectivités. Au sein d'un code parallèle, chaque processus décrira donc une collection de polygones ou polyèdres. En conséquence, les données des processus peuvent donc facilement être concaténées, et par la suite, facilement "découpées" pour relire ou analyser uniquement une sous-partie de cette collection, puisqu'il n'y a pas de lien entre les différents blocs élémentaires. On notera qu'il faut malgré tout apporter une correction au tableau de connectivité pour conserver la cohérence du modèle global, ce qui implique une phase de communication entre les processus. Néanmoins, la description du modèle est simple, son agrégation également et donc l'utilisation de bibliothèque d'`E/S` parallèle, tel que `HDF5`, est plus facile à mettre en place sans impacter les outils d'analyse. Malheureusement, comme on l'a mentionné précédemment, ce n'est pas un bon candidat pour les codes `AMR` à cause du volume de données qu'il produirait (il reste néanmoins très utile pour le déboguage).

Dans le cas de `RAMSES`, puisque le paradigme utilisé est celui du *file-per-process*, il n'y a donc pas d'agrégation et donc pas de modèle global mais seulement un modèle de données local à chaque processus, identique entre tous les processus. En conséquence, les données produites ne donnent pas une vue d'ensemble du maillage et des champs scalaires/vectoriels de toute la simulation mais une vision par morceau qu'on appellera des *domaines* pour faire le lien avec le découpage du maillage. Les outils d'analyse, tel que *PyMSES*, vont donc traiter les données

domaine par domaine et feront même des optimisations pour réduire la liste des domaines à traiter, voir chapitre 4. Cette approche est très intéressante parce qu'elle met en avant le fait qu'il n'est pas forcément nécessaire de mettre un place un modèle global cohérent (ou dont la concaténation des modèles locaux soit cohérente). Et offre différentes possibilités pour faciliter le post-traitement de très grand volume de données, voir chapitre 4.

Notre approche avec le modèle `lightAMR` requiert une restructuration des données, et donc l'utilisation de tableaux temporaires, mais reste local au processus `MPI`. Ainsi, aucune communication additionnelle inter-processus n'est requise pour établir les tableaux descriptifs du modèle au sein de chaque processus. Le modèle `lightAMR` a une granularité très grande comparé au modèle *non structuré*. En effet, le bloc élémentaire est la partie de l'arbre `AMR` (et ses champs scalaires associés) gérée par le processus `MPI` courant. En conséquence, la concaténation des tableaux du modèle (ou la relecture d'une sous-partie d'un tableau) ne conserve pas la cohérence du modèle de données, voir section 3.7. Ainsi, il y aura autant de bloc de données "*lightAMR*" que de processus `MPI` dans la simulation, on conserve donc l'approche domaine par domaine. Il n'y a donc pas d'agrégation inter-processus des données faites du point de vue du modèle de données ou au sein du format de données comme c'est le cas pour `PIDX`. On notera ici que l'utilisation d'`Hercule` joue un rôle essentiel puisqu'elle permet de s'affranchir de l'agrégation des données pour l'écriture, et de pouvoir, en post-traitement, accéder facilement aux données domaine par domaine. On reviendra sur ce point à la section 3.7.

3.3 Description de la grille `AMR` et des données

Afin de décrire le modèle `lightAMR`, on prendra pour exemple une grille `AMR` de type `cell-based` en 2 dimensions, découpée en 3 domaines, voir figure 3.1. Ce type de maillage et sa structure hiérarchique peuvent se représenter sous la forme d'un arbre acyclique, voir figure 3.2. On appellera *noeud* un point de l'arbre qui représente une cellule *parent* du maillage, et *feuille* une cellule *enfant*. Il existe en fait plusieurs représentations possibles, on choisit d'utiliser celle qui se base sur un unique noeud que l'on appellera sommet de l'arbre ou *noeud racine*, et qui permet de faire référence à la boite de simulation, que l'on peut voir comme étant la cellule de niveau 0 du maillage. On discutera à la section 3.6 d'une autre représentation possible sous la forme d'une collection d'arbres.

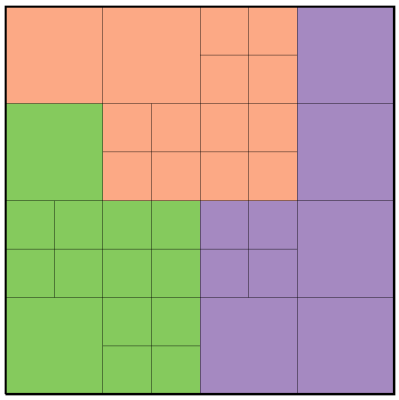

FIGURE 3.1 – Exemple d'une grille `AMR 2D`, (niveau minimum 2, niveau maximum 3) répartie sur 3 domaines en fonction du découpage avec la courbe de Hilbert : domaine 0 en vert, domaine 1 en orange et domaine 2 en mauve.

Pour décrire cet arbre à des fins de stockage, on adopte une notation booléenne où on associe à chaque point de l'arbre (noeud ou feuille), deux valeurs booléennes. Une pour décrire l'état de la cellule : 1 pour un noeud et 0 pour une feuille, et une valeur pour définir son appartenance au domaine courant : 0 si le point appartient au domaine, 1 si le point ne lui appartient pas. En conséquence, et comme mentionné précédemment, un arbre est construit pour un domaine donné, même si sa description commence depuis la racine (la boite de simulation). Si la description est correcte, alors la somme des volumes, sur tous les domaines, des cellules feuilles dont la valeur d'appartenance vaut 0, doit être égale au volume de la boite de simulation.

Ces valeurs booléennes sont stockées dans deux tableaux clés du modèle : le tableau de raffinement et

le tableau d'appartenance. Afin d'avoir une bijection unique entre la représentation en arbre et le maillage de la simulation, il faut définir un ordre de parcours imposé. Puisqu'il est naturel lors des analyses de commencer par une résolution dégradée, et au besoin d'aller chercher plus de résolution, on choisit de stocker cette description en arbre de haut en bas, c'est-à-dire du niveau AMR le plus petit, niveau 0, jusqu'au niveau AMR le plus grand. Les noeuds et les feuilles sont ordonnées suivant la courbe de Morton. Une fois ces deux tableaux de booléens décrivant le maillage AMR établi, les données scalaires correspondant aux grandeurs physiques portées par la grille AMR sont données sous forme de tableaux 1D qui suivent rigoureusement le même ordre que les tableaux de raffinement et d'appartenance, voir figure 3.3. On notera que les champs vectoriels comme la vitesse, par exemple, sont donc décrits composante par composante, sous la forme de tableaux 1D.

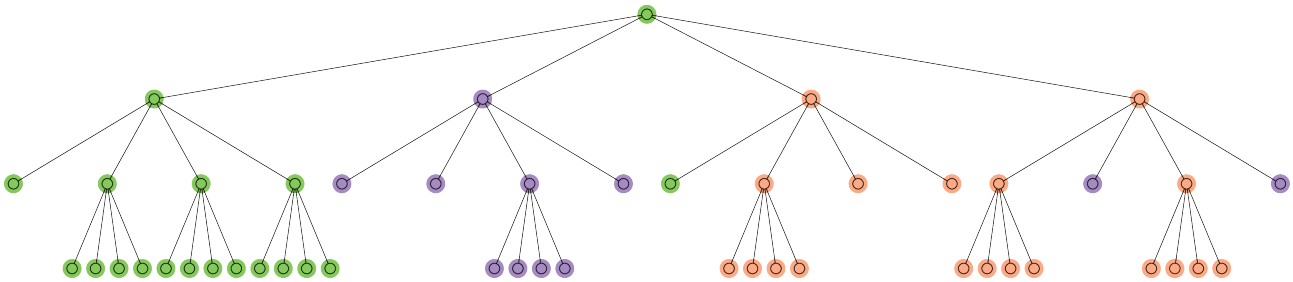

FIGURE 3.2 – Représentation sous la forme d'un arbre de la grille AMR de la figure 3.1. Les couleurs correspondent aux cellules des différents domaines, respectivement : (vert, domaine 1), (bleu, domaine 2), (mauve, domaine 3).

Avec ce modèle, l'information nécessaire pour reconstruire le maillage exact de la simulation est conservée tout en n'explicitant pas les quantités qui peuvent être facilement recalculées lors de du post-traitement. En effet, contrairement au modèle de données de RAMSES, de nombreuses quantités tels que les coordonnées du centre des cellules, leurs voisinages, leurs parents et descendants ne sont plus explicitement stockées et devront donc être recalculées. Afin de faciliter ces calculs et de maintenir la cohérence par domaine, des méta-données légères sont rajoutées tels que : le nombre de niveau AMR minimum et maximum, le nombre de cellules par niveau, la taille de la boite de simulation, etc. On notera également que l'information précise du domaine d'appartenance d'une cellule n'est pas conservée. En effet, dans le format natif, le tableau cpu_map, stocké dans les fichiers AMR permet d'associer à chaque cellule le numéro de domaine (rang MPI) auquel elle appartient. Dans le cas présent, seule une information booléenne est conservée, on ne peut donc pas directement savoir à quel domaine appartient une cellule masquée. Il faut nécessairement passer par le calcul de l'indice de Hilbert. On détaille cette méthode au chapitre 4.

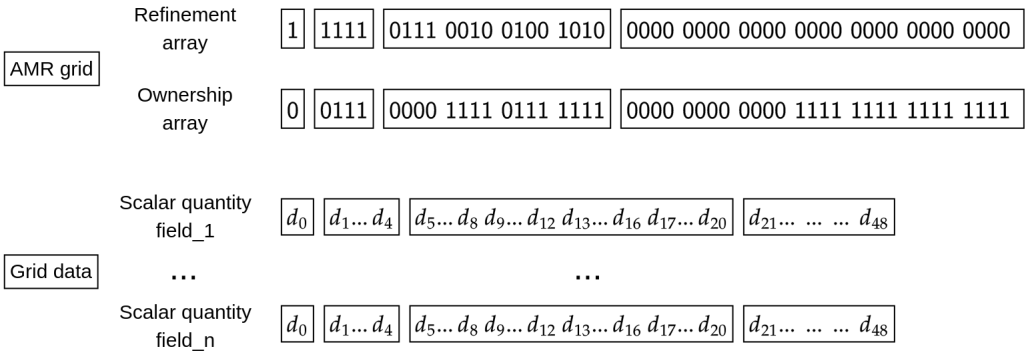

FIGURE 3.3 – Encodage de l'arbre de la figure 3.2 sous la forme de deux tableaux booléens : raffinement et appartenance, ainsi que les grandeurs scalaires associées.

Bien que l'exemple utilise une facteur de raffinement $f_r = 2$, ce type de description est compatible avec d'autres valeurs que 2 pour autant que ce facteur soit appliqué à toutes les cellules de la simulation dans toutes les dimensions de l'espace. Pour un code, tel que *Dyablo*, [Durocher and Delorme, 2024], dont les structures de données internes ne conservent que l'enveloppe feuille de l'arbre avec un facteur de raffinement $f_r = 2$ au niveau des noeuds et où chaque cellule feuille est une grille régulière sous la forme d'un bloc de taille $b_x \cdot b_y \cdot b_z$, cette description peut également être utilisée. Soit la taille des blocs est la même dans toutes les dimensions **et** est une puissance n de 2, dans ce cas, une cellule feuille peut être représentée sous la forme de n niveaux AMR supplémentaires totalement

raffinés. Il s'agit du cas le plus simple pour lequel aucune modification n'est nécessaire, ni pour les algorithmes de compression qui seront décrits par la suite, ni pour l'outil d'analyse au chapitre 4. Si les blocs sont de tailles différentes, à chaque cellule feuille du tableau de raffinement, il faudra associer $b_x \cdot b_y \cdot b_z$ valeurs pour les champs scalaires au sein du bloc. De plus, un ordre de parcours devra être défini et une extension devra être implémentée aux niveaux de l'outil d'analyse et des algorithmes de compression. On notera également que les méta-données devront être enrichies avec la taille des blocs.

Finalement, cette description est parfaitement adaptée à un code comme `RAMSES` qui utilise une approche *fully threaded* et où les noeuds de l'arbre sont accessibles et contiennent une valeur pour chaque champ scalaire. La construction de ce modèle est donc relativement aisée dans ce cas. Pour un code, qui utiliserait une approche différente sans conserver de valeur aux noeuds, une simple valeur moyenne est suffisante et permettra de tirer profit de l'algorithme de compression de données flottantes décrit dans ce chapitre, ainsi que des approches *Level-Of-Details* en post-traitement, voir chapitre 4.

3.4 Suppression de la redondance

Une fois que la grille `AMR` est décrite suivant les règles du modèle de données, il peut y avoir de la redondance d'information. En effet, certaines cellules, et leur descendance, sont décrites dans plusieurs domaines, par exemple les cellules issues de la 4 ème grille du niveau 1 seront décrites au minimum par le domaine 2 et le domaine 3 sur l'exemple à la figure 3.2. Ces cellules redondantes sont nécessaires au code car elles font parties de l'enveloppe des cellules fantômes qui est utilisée dans les schémas numériques pour accéder au voisinage. Dans le cas de `RAMSES`, où chaque domaine stocke les cellules feuilles ainsi que les noeuds de l'arbre jusqu'au niveau 0 du maillage, les besoins de ce voisinage à différents niveau de la grille et la redondance qu'il induit peut se révéler coûteux en terme de stockage. En effet, comme on l'a mentionné précédemment, à chaque cellule décrite dans le modèle `lightAMR`, il faut associer une valeur du champ scalaire ou vectoriel.

Il est commun dans certains domaines d'étude utilisant des arbres de décision, d'utiliser des algorithmes qui permettent de simplifier un arbre profond et complexe, au profit d'un arbre plus simple quitte à sacrifier un peu de précision, [Bohanec and Bratko, 1994]. Ces méthodes sont appelés *algorithme de tree pruning*. On propose ici d'appliquer le même principe à l'arbre décrit par le tableau de raffinement. L'algorithme qu'on propose conserve toute l'information du maillage, il est donc *sans perte*. L'objectif est de retirer les cellules fantômes décrites dans le tableau de raffinement et le tableau d'appartenance, et de propager ces modifications aux tableaux des données scalaires associées. Il est important de noter que l'on peut se passer du stockage des cellules fantômes pour plusieurs raisons. Premièrement, ces cellules sont rarement exploitées par les outils d'analyses. Deuxièmement, cette information pourrait être reconstruite à un coût acceptable en fonction des besoins des analyses. On notera que le post-traitement des données n'implique pas forcément une analyse complète de tous les domaines d'une simulation, voir chapitre 4. Troisièmement, cette information est pertinente pour `RAMSES` compte tenu de ses algorithmes et sa gestion du maillage, mais les outils de post-traitement n'utilisent pas forcément la même approche pour analyser les données. Par exemple, l'outil que j'ai développé, décrit au chapitre 4, extrait l'enveloppe feuille du maillage d'un domaine. De plus, une collection de domaines est gérée par un seul processus `MPI`. Finalement, on notera que le gain en terme de stockage est généralement non négligeable en fonction des simulations, comme on le verra dans la section sur les résultats 3.8. En conséquence, l'application de cet algorithme est la deuxième étape importante, après la description compacte du maillage, pour réduire significativement le volume de données produit à chaque sortie de post-traitement.

D'un point de vue technique, pour que l'algorithme d'élagage ne retire pas de feuille ou de noeud de l'arbre dont la descendance pourrait appartenir au domaine courant il est nécessaire dans un premier temps de corriger *temporairement* la valeur du masque d'appartenance au niveau des noeuds afin de valider la règle suivante : un noeud dont au moins une des feuilles ou noeuds de sa première descendance appartient au domaine courant, doit également appartenir au domaine courant. Par exemple, sur la figure 3.2, lors de la description du domaine 3 en mauve, cela revient à changer la couleur du noeud racine en mauve, du 4 ème noeud au niveau 1 de orange à mauve, et de son 3 ème noeud de orange à mauve également. Une fois cette correction effectuée, il faut parcourir l'arbre de bas en haut, du niveau maximum au niveau 0, et appliquer la règle d'élagage suivante : un noeud dont toutes les feuilles n'appartiennent pas au domaine courant peut être supprimé et remplacé par une feuille. On notera que l'appartenance du noeud est inchangée. Une fois l'élagage fini, le tableau d'appartenance d'origine est réappliqué.

Compte tenu du modèle de donnés et du découpage de la courbe de Hilbert au niveau AMR maximum, la totalité de la redondance ne peut pas être supprimée. En effet, par exemple, les 4 noeuds du niveau 1 sur la figure 3.2 seront décrits par les 3 domaines.

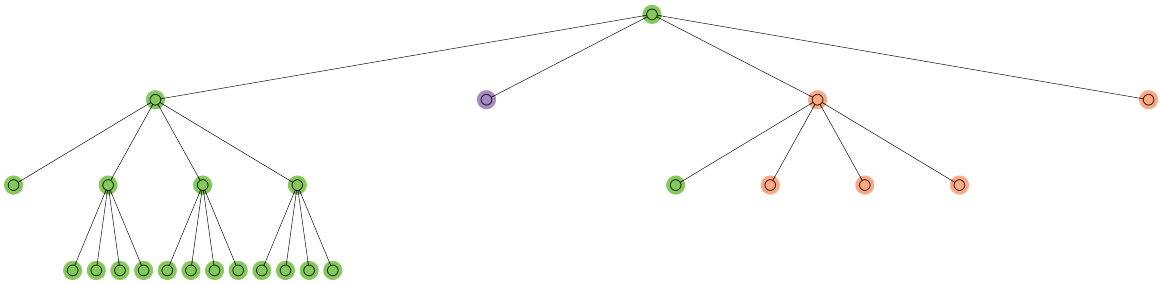

FIGURE 3.4 – Effet de l'élagage de la représentation AMR pour le domaine 1 (vert)

Sur la figure 3.5, on représente le maillage associé à l'arbre élagué vu par le domaine 1. On constate que la balance 2 :1 du maillage, c'est-à-dire la conservation d'un niveau d'écart maximum entre deux cellules adjacentes, n'est plus conservée aux frontières des domaines. On le met en évidence, en rouge, à la frontière entre le domaine en vert et le domaine en mauve sur la figure. Néanmoins, on notera que cette balance 2 :1 est conservée au sein d'un même domaine. C'est une heureuse conséquence des règles de description du maillage et des conditions d'élagage. En effet, on tire profit de cette propriété pour accélérer l'algorithme de lancer de rayon, voir chapitre 4. On rappelle cependant que les couleurs sont données ici pour faciliter la compréhension, d'un point de vue technique le domaine en vert, n'a pas l'information du numéro de domaine auquel appartient une cellule masquée (marquée avec un "1" dans la tableau d'appartenance).

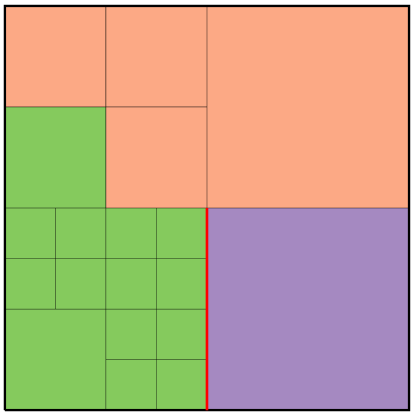

FIGURE 3.5 – Grille AMR vue par le domaine 1 (vert) après application de l'algorithme de *tree pruning*, la balance 2 : 1 n'est plus conservée au niveau des cellules fantômes, à la frontière en rouge entre le domaine 1 (vert) et 3 (mauve).

3.5 Compression des données

La compression de données a été initialement développée dans l'objectif de réduire le temps de transfert d'une information d'un point $\mathcal{A}$ à un point $\mathcal{B}$. Avec l'augmentation des volumes de données issus des codes de simulation numérique et les problèmes d'E/S associés, la compression de données est devenue un champ de recherche très actif permettant d'apporter une partie de la solution pour optimiser le transfert et le stockage de grands volumes de données. Par abus de langage, on appelle un algorithme de compression un processus qui comporte en fait deux étapes. La première est la transformation de données brutes vers des données compressées ou encodées. La deuxième étape est la fonction réciproque permettant de retrouver les données brutes à partir des données compressées. Il y a deux types de compression : *sans perte* et *avec perte*. Si un algorithme est qualifié de *sans perte*, alors la

donnée décompressée sera *strictement identique*, au bit près, à la donnée brute. A l'opposé, la qualification *avec perte* signifie qu'une partie de l'information initiale est *définitivement* perdue. Afin de quantifier l'impact de la perte d'information, une erreur de compression est généralement associée à ce type d'algorithme. D'après [Salomon and Motta, 2009], les algorithmes de compression peuvent être rangés en fonction de différentes caractéristiques :

1. *adaptatif* ou *non-adaptif* : si l'algorithme modifie ses opérations ou ses paramètres en fonction des données d'entrées (à compresser) alors il peut être qualifié d'adaptatif,
2. *single-pass* ou *multiple-pass* : un algorithme *single-pass* ne fera qu'un seul passage sur les données, alors qu'un *multi-pass* fera plusieurs passages. Généralement, un algorithme *multiple-pass* permet d'obtenir un meilleur ratio au prix d'une vitesse réduite,
3. *symmetric* ou *asymmetric* : un algorithme symétrique appliquera les mêmes traitements à la compression et à la décompression, alors qu'un algorithme asymétrique appliquera des traitements différents. En conséquence, la charge de travail n'est pas forcément la même et les temps de compression et de décompression peuvent être significativement différents (c'est le cas par exemple de l'archivage avec la commande `unix` *tar*),
4. *universal* ou *custom* : un algorithme universel n'utilise pas d'a priori sur les données contrairement à un algorithme *custom* qui exploite la connaissance du contenu ou du contexte des données pour améliorer les performances de compression : ratio et/ou vitesse.

L'objectif principal d'un algorithme de compression est assez simple : supprimer la redondance ou toute forme de corrélation dans les données. On peut ainsi introduire une notion d'entropie. En effet, en théorie de l'information, la formule 3.1, définit l'*entropie de Shannon*, [Shannon, 1948], d'une variable aléatoire discrète x comportant n symboles, chaque symbole x_i ayant une probabilité $\mathcal{P}_i$ d'apparaître :

$$\mathcal{H}_s = -\sum_{i=1}^{n} P_i \log_b(P_i) \tag{3.1}$$

En utilisant la base 2 pour le logarithme, on peut associer un nombre de bits par symbole. Prenons l'exemple de deux distributions différentes d'une variable entière quelconque, χ, comprise entre $[0, 14]$, représentées sur la figure 3.6. A gauche, la distribution est uniforme, son entropie de Shannon est de 3.98, ce qui signifie qu'il faut en moyenne 4 bits pour encoder un entier. Ce résultat est attendu puisque $2^4 = 16$, on peut donc, avec 4 bits, encoder tous les "*symboles*" (ici les nombres entiers entre 0 et 14). En effet, puisque les probabilités d'apparition des symboles sont toutes quasiment identiques, l'information portée par un symbole est importante. C'est-à-dire que si on retire un entier d'une séquence, l'impact peut être important sur l'information portée par la séquence. A l'opposé, sur la figure de droite, on a une distribution de Poisson centrée sur la valeur 5 (on a retiré artificiellement la valeur 4 pour accentuer l'impact sur l'entropie), son entropie de Shannon est de 2.69, on peut donc encoder en moyenne les symboles sur 3 bits. Tous les entiers ne portent donc pas la même quantité d'information, retirer le 9 ou le 10 d'une séquence impactera peu l'information globale. Il est donc possible de réduire, en moyenne, le nombre de bits pour encoder l'information. Enfin, il est important de noter que l'entropie de Shannon est une mesure de l'information pour une séquence donnée, mais ne représente pas une limite pour la compression. Par exemple, si on prend une séquence qui répète de manière consécutive, n fois les chiffres de 1 à 14, du type $[1, .., 1, 2, .., 2, 3, ..3, ..]$, la distribution sera homogène et l'entropie de l'ordre de 4. Des méthodes de type *Run-Length-Encoding* (`RLE`) arriveront parfaitement à compresser ce type de séquence. On utilisera d'ailleurs ce type d'approche pour compresser les tableaux de description du maillage à la section suivante.

L'entropie de Shannon est à la base des méthodes de compression entropique, qui se basent sur la fréquence d'apparition de symboles. L'exemple le plus connu est la méthode de compression de Huffman, utilisée notamment pour compresser du texte en fonction de la fréquence d'apparition des lettres qui varie en fonction de la langue. Typiquement, la méthode consiste à encoder sur moins de bits les symboles les plus fréquents. Néanmoins, ce type d'approche est difficilement applicable à des données flottantes, parce que le nombre de "symboles" est très grand, la distribution complexe voire continue. De très nombreuses méthodes plus adaptées ont donc été développées : on peut citer par exemple les méthodes de compression par *transformation* ou encore les méthodes par *prédiction*. Les méthodes par *transformation* transforment les données d'une représentation à une autre, où la redondance et/ou la corrélation entre les données est plus facile à déterminer. On peut citer par exemple les méthodes à base d'ondelettes qui permettent de décomposer les données sous la forme de contributions d'ondes à différentes échelles et fréquences, comme on le ferait avec un développement en série de Fourier d'un signal. Ces méthodes sont très utilisées pour la compression de contenu multimédia, mais sont par nature des méthodes de compression *avec*

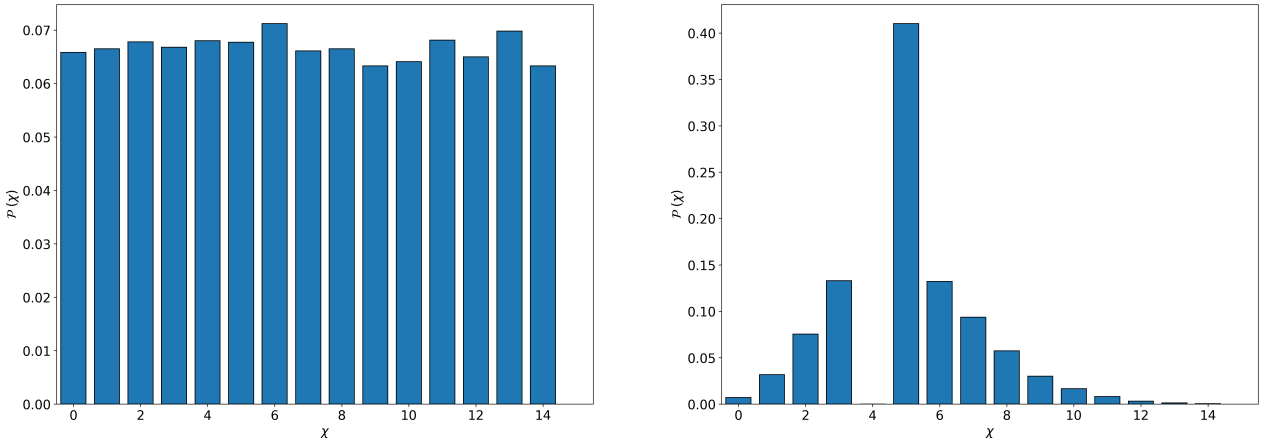

FIGURE 3.6 – Exemple de distribution pour le calcul de l'entropie de Shannon d'une quantité χ quelconque. A gauche, une distribution homogène et à droite une loi de Poisson.

perte. Les méthodes par *prédiction*, et notamment la *Delta* compression, sont très populaires pour la compression de données flottantes. En effet, ce type d'approche utilise une fonction mathématique pour prédire une valeur. En conséquence, plus la fonction sera précise pour prédire les données, plus le taux de compression sera élevé. Ce type de méthode est populaire parce qu'il est facile, en théorie, d'optimiser l'algorithme de compression en fonction des cas d'usages en adaptant la fonction de prédiction aux données. Il est commun d'essayer plusieurs fonctions de prédiction et d'utiliser celle qui donne le meilleur résultat. Néanmoins, le problème est alors de trouver une bonne fonction de prédiction, [Fout and Ma, 2012]. On notera également que l'utilisation d'une fonction mathématique (qui engendre des calculs) ne permet pas de garantir l'aspect *sans perte* de l'algorithme et peut induire un surcoût en temps de calcul.

3.5.1 Run-length Encoding : CPS52

Une fois que la structure de l'arbre du maillage (après élagage) a été décrite à l'aide de tableaux à valeurs booléenes, un algorithme de compression sans perte a été développé dans le cadre de cette thèse pour compresser ces informations : raffinement et appartenance. Conformément aux caractéristiques énoncées précédemment, l'algorithme proposé est donc : *non-adaptatif*, *single-pass*, *symétrique* et *universel*. Il s'agit d'une méthode de type *run-length-encoding* (`RLE`), [Robinson and Cherry, 1967]. J'ai fait le choix de développer cet algorithme plutôt que d'utiliser une bibliothèque extérieure pour plusieurs raisons. Tout d'abord, l'algorithme est rapide à implémenter et cela permet d'avoir un contrôle sur l'encodage, ce qui m'a permis de rajouter des octets pour marquer la fin des niveaux (voir ci-après). Ensuite, cela permet de limiter la dépendance aux bibliothèques extérieures qui peut parfois devenir problématique lors du déploiement du code sur différentes architectures, avec des versions parfois différentes des bibliothèques disponibles. Enfin, comme on le verra dans la section sur les résultats, les performances des bibliothèques testées n'étaient pas convaincantes, voir section 3.8.

Les algorithmes `RLE` se basent donc sur une technique simple et efficace pour compresser des données qui contiennent des répétions consécutives de symboles. Au lieu de stocker individuellement chaque symbole, on stocke le nombre de répétitions consécutives d'un symbole suivi du symbole en lui-même. Dans notre cas, on utilise cette approche pour compresser les deux tableaux de description du maillage `AMR` que sont les tableaux de raffinement et de d'appartenance. En effet, il s'agit de tableaux de booléens qui seront interprétés comme une liste ordonnée de paquets de valeurs binaires identiques, soit des 0 successifs, soit des 1 successifs (voir étape ①, fig. 3.7). La deuxième étape consiste à compter les longueurs de ces paquets continus (étape ②, fig. 3.7). L'étape ③ consiste à numériser les tailles de ces paquets ordonnés à l'aide d'un encodage spécial et à stocker le résultat dans un tableau d'entiers 1-octet, qui constituera le champ compressé. Pour coder les tailles des paquets, on utilise une décomposition en base 52. Lorsque la décomposition nécessite plus d'un chiffre, on rajoute le nombre de chiffres dans le tableau. Ainsi, les octets dont la valeur est comprise entre 1 et 8 sont utilisés pour stocker le nombre d'octets qui encodent la décomposition de la taille du paquet. Les 52 valeurs comprises entre 11 et 62 permette de stocker (avec un décalage de 11) les chiffres de la décomposition. Par exemple, pour coder un paquet de taille 2904 qui se décompose en

$1 \cdot 52^2 + 3 \cdot 52^1 + 44 \cdot 52^0$, le plus grand exposant nécessaire plus 1, est d'abord stocké suivi des chiffres $1, 3, 44$, codés sous la forme $[12, 14, 55]$ ($[1 + 11, 3 + 11, 44 + 11]$), ce qui donne donc 4 octets : [3.12.14.55] soit 32 bits au lieu des 2094 bits ou 262 octets nécessaires pour stocker 2904 zéros ou uns. Ainsi, le champ compressé est un tableau `1D` d'entier 1 octet dont les valeurs seront comprises dans l'intervalle $[0, 64]$.

Afin de rendre possible la décompression par niveau ou jusqu'à un niveau donné, des octets "clés" sont insérés pour marquer la fin des niveaux `AMR` avec les valeurs 9 ou 10. Une valeur 9 signifie que le niveau se termine à la fin d'un paquet de bits (0 ou 1) identiques (e.g. fin du niveau 2 dans la fig. 3.7) tandis que 10 signifie que la fin du niveau coupe en deux un paquet de bits identiques (e.g fin des niveaux 0 et 1 dans la fig. 3.7). Afin de commencer correctement l'alternance des valeurs, le tout premier octet du champ compressé contient la valeur du premier paquet.

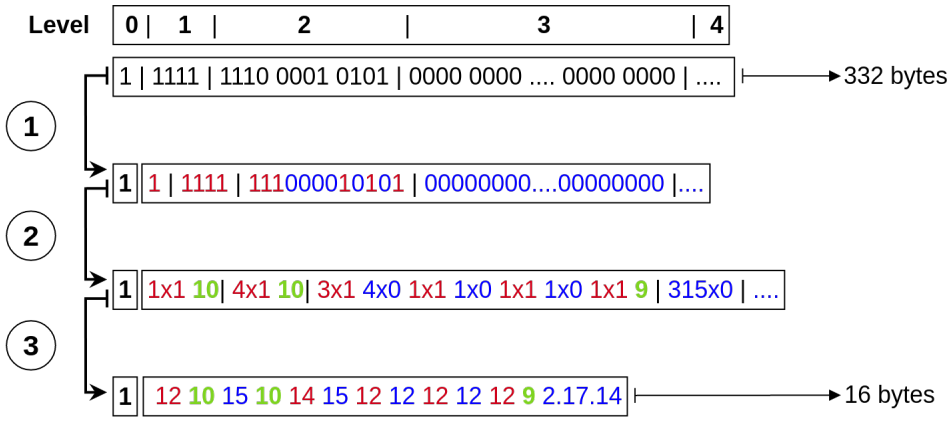

FIGURE 3.7 – Étape de la compression `CPS52` pour les tableaux à valeur booléenne de la description de la grille `AMR`.

On notera que ces tableaux ne sont pas les plus gourmands en ressource de stockage, puisqu'un domaine contenant $3 \cdot 10^6$ cellules (que l'on peut considérer comme un très gros domaine pour une simulation `RAMSES` de manière générale), ne nécessite finalement que 5.6 *ko* (raffinement + appartenance). Néanmoins, comme on le montrera dans la section sur les résultats, sur l'ensemble d'une simulation comme `ExaMilkyWay`, ces tableaux non compressés peuvent représenter plusieurs dizaines de gigaoctets, environ ~ 70 *Go* à haute résolution dans notre cas. L'algorithme de compression `RLE`, bien qu'assez simple, permet un gain non négligeable en terme de stockage et de réduire ce volume à moins de 2 *Go*, voir section 3.8.

3.5.2 Compression sans perte : `PCP`

Dans cette section et la suivante, on détaillera un algorithme de compression de type *prediction-based*. Comme on l'a mentionné, ce type de méthode utilise une fonction pour prédire la prochaine valeur d'un champ. Par exemple, si on souhaite compresser un champ donné sous la forme d'un tableau `1D` du type $[x_0, x_1, x_2, ..., x_n]$. Il est généralement courant d'utiliser les p dernières valeurs pour estimer la prochaine valeur du champ. Par exemple, on pourrait utiliser une combinaison quelconque des valeurs x_0, x_1, x_2 pour prédire x_3, puis x_1, x_2 et x_3 pour prédire x_4, etc, de manière itérative. On peut citer par exemple, le prédicteur de *Lorenzo*, qui est très utilisé, notamment pour les tableaux à plusieurs dimensions issus de données sur grille régulière, [Ibarria et al., 2003].

On associe à ce type de méthode, un encodage *delta* (*Delta encoding*) qui consiste à encoder dans le champ compressé, non pas les données de prédiction, mais la différence (le "delta"), entre la prédiction et la valeur réelle (initiale). Ainsi, plus la fonction de prédiction est bonne, plus le delta sera petit, et plus la compression sera élevée. Lors de la décompression, il suffit de connaître la fonction de prédiction utilisée et le delta pour retrouver la valeur d'origine. Généralement, pour la compression de données scientifiques flottantes, un meilleur ratio de compression est obtenu en spécialisant les algorithmes en fonction d'a priori sur les données, c'est-à-dire en ajustant la fonction de prédiction, [Cappello et al., 2019].

Le problème principal de ce type d'algorithme est donc la fonction de prédiction. De plus, l'organisation des données dans le tableau à compresser peut également impacter le ratio de compression, [Lindstrom and Isenburg, 2006]. Par exemple, dans le cas des tableaux scalaires issu du modèle `lightAMR`, peut-être que si les données étaient rangées du niveau le plus fin vers le niveau le plus grossier, le taux de compression serait meilleur. Ou alors, peut-être qu'il faudrait compresser les données par niveau dans des tableaux séparés, etc. En effet, compte tenu de l'ordre d'agencement des données imposé par le modèle `lightAMR`, il est naturel de penser que les prédicteurs se basant sur le "voisinage" des données du tableau à compresser, ne puissent pas donner de bons résultats. De fait, des cellules adjacentes à un niveau l, rangées côte à côte dans le tableau, peuvent avoir des valeurs très différentes de par la nature hiérarchie du maillage. Pour pallier ce type de problème, certains algorithmes se basent sur une table de hachage pour extraire des relations non linéaires, sans se baser sur le voisinage (typiquement les p dernières valeurs dans le tableau) dans les données, [Burtscher and Ratanaworabhan, 2007], [Ratanaworabhan et al., 2006]. Néanmoins, cette table a un coût mémoire et elle doit être stockée avec les données compressées pour l'étape de décompression. L'algorithme que l'on propose ici, en plus d'être très rapide, ne consomme pas de mémoire additionnelle (en dehors de celle utilisée pour le champ compressé). Il peut être qualifié de : *custom*, *non-adaptive*, *single-pass* et *symmetric* (et *prediction-based*). Il se découpe en trois grandes étapes : prédiction, décorrélation et encodage, représenté avec un exemple sur la figure 3.8. On verra à la section suivante, qu'il est possible de rajouter une quatrième étape, optionnelle, d'"approximation" qui permet d'améliorer le ratio de compression en introduisant de la *perte* d'information.

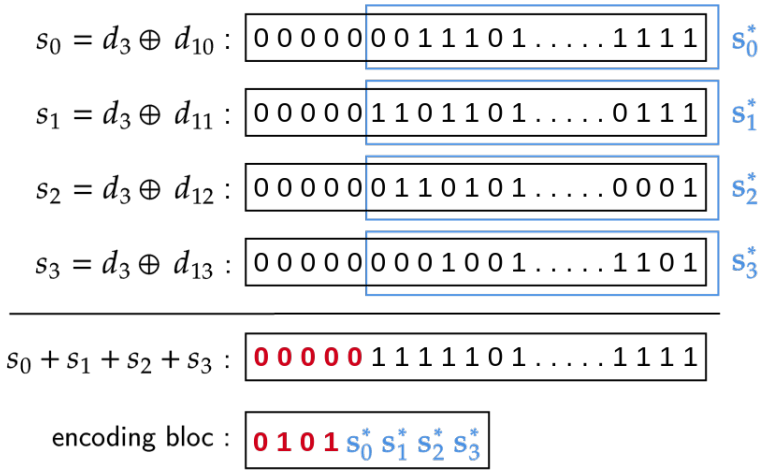

FIGURE 3.8 – Étape de la compression sans perte `PCP4` pour les champs scalaires de la grille `AMR`.

L'algorithme qu'on propose est *custom* parce qu'on utilise une fonction de prédiction propre aux données de type `AMR` *tree-based* décrite suivant le modèle `lightAMR`. En effet, la fonction première de ce type de maillage est de raffiner une cellule afin de mieux suivre les processus physiques et donc afin d'éviter de trop grands gradients des quantités hydrodynamiques entre deux cellules adjacentes. En conséquence, la valeur d'une cellule parent (noeud de l'arbre) peut être utilisée comme prédicteur de la valeur de ses cellules enfants et cela sans opération supplémentaire dans le cas de grandeurs physiques non-conservatives, où la valeur de la cellule parent est égale à la moyenne arithmétique des valeurs des cellules enfants. Sur la figure 3.8, la variable d_3 est considérée comme étant la valeur de la cellule parent et les variables $d_{10}, d_{11}, d_{12}, d_{13}$ la valeur de chacun des cellules enfants. Cette fonction de prédiction possède plusieurs avantages. Premièrement, la valeur de la cellule parent est stockée dans le modèle `lightAMR`, il n'y a donc pas besoin de faire des calculs, ce qui permet un gain significatif en temps de calculs et donc en vitesse de compression. Deuxièmement, il n'est pas nécessaire d'allouer des tableaux temporaires ou des tables de hachage, ni de stocker des informations complémentaires dans le champ de compression. Troisièmement, cette fonction suit naturellement les variations spatiales des champs scalaires à compresser, on a donc la garantie de toujours avoir une prédiction en accord avec ces variations. Il est important de noter que cette fonction de prédiction implique de fournir à la fonction de compression et de décompression le tableau de raffinement de la grille `AMR`.

La seconde étape est la décorrélation des données. Pour ce faire, on utilise une méthode classique qui consiste à calquer les données flottantes (double précision ou simple précision) sur des entiers (long ou simple). Ainsi, on peut utiliser des opérations entières dont les résultats sont reproductibles (et portables sur différentes

architectures), contrairement aux opérations à virgule flottante qui sont sensibles aux erreurs d'arrondis. De plus, ces opérations sont généralement très rapides. Pour calculer les *deltas*, on utilise l'opération `XOR` entre la représentation entière de la valeur de la cellule parent et la représentation entière de la valeur des cellules enfants. Ainsi, si deux bits sont identiques, le résultat sera 0, sinon il vaudra 1. Sur la figure 3.8, l'opération `XOR` est représenté par le symbole $\oplus$ et les *delta* par la variable s_i. On notera que $d_3 \oplus d_{10}$ permet d'obtenir le *delta* s_0 et que l'opération inverse permettant de retrouver la valeur d'origine d_{10} est $s_0 \oplus d_3$. Cette relation est à la base de la symétrie de l'algorithme. Enfin, il est important de souligner que cette méthode de décorrélation est utilisée par exemple par l'algorithme de compression `FPC`, [Burtscher and Ratanaworabhan, 2007].

La troisième étape consiste à extraire pour chaque *delta*, s_i, un résidu, noté s_i^* sur la figure 3.8. Pour extraire les résidus, on détermine le nombre de bits de poids fort à valeur 0, de chaque *delta* des cellules enfants d'une même cellule parent, et on prend la valeur minimale de ces nombres que l'on notera `LZ` (pour *leading zero*). Dans la figure 3.8, `LZ`=5 et correspond aux 5 zéro exclus des rectangles bleus. Ce choix permet de regrouper, dans le champ compressé, l'information du nombre de `LZ` retirés de chaque *delta* de toutes les cellules enfants d'une cellule parent du maillage. Cette factorisation du `LZ` permet d'améliorer le ratio de compression. On reviendra sur l'intérêt de cette approche à la section 3.8.5. L'opération permettant de calculer le nombre de `LZ` se trouve être cruciale pour les performances de l'algorithme. On notera donc qu'il est préférable d'utiliser l'opération `OR`, noté +, entre tous les résidus et déterminer le nombre de `LZ`, une seule fois, du résultat, plutôt que de le calculer pour chaque résidu et de prendre la valeur minimale. Enfin, on choisit d'encoder le nombre de `LZ` sur 4 bits, suivi des 4 (en `2D`) ou 8 (en `3D`) résidus s_i^*. Avec un encodage sur 4 bits du `LZ`, cela signifie que l'on autorise à retirer au maximum 15 bits de poids forts des résidus. On reviendra également sur ce choix à la section 3.8.5.

En se basant sur le nombre maximum de bit de poids fort retiré P_{lz} (qui est fonction de l'encodage utilisé), la dimension de la simulation d (`2D` ou `3D`), le type de données E_{enc} (32 ou 64 bits), on peut calculer le ratio de compression théorique maximal, voir formule 3.2. Cette formule ne tient pas compte d'une éventuelle sur-couche d'encodage que l'on retrouve dans la plupart des bibliothèques de compression.

$$C_{max}^{d} = \frac{E_{enc} \cdot 2^d}{(E_{enc} - P_{lz}) \cdot 2^d + p} \tag{3.2}$$

En conséquence, pour un encodage du nombre de `LZ` sur 4 bits soit 15 zéros maximum, une simulation en `3D`, on obtient un ratio théorique maximal de 1.293 pour un champ scalaire double précision et 1.829 pour un champ scalaire en simple précision. On tient compte dans cette formulation du lissage par paquet de 2^d cellules du nombre de `LZ` que l'on peut retirer et de la factorisation de cet encodage. Pour avoir une estimation du ratio de compression par cellule, il suffit de prendre $d = 0$, on obtient ainsi $C_{max}^{0} = 1.208$ pour des données double précision et $C_{max}^{0} = 1.524$ pour des données simple précision. On verra à la section 3.8.5 lors de tests sur des données réelles, si le gain de l'encodage avec factorisation est systématique. Enfin, on notera que compte tenu de la représentation binaire standardisée par la norme `IEEE` des valeurs flottantes 32 bits et 64 bits, un encodage sur 4 bits permet donc de retirer, en théorie, l'intégralité de l'exposant de ces deux représentations. De plus, on notera que si le champ scalaire à compresser fluctue autour de 0.0, c'est-à-dire que la valeur des cellules enfants ont des signes différents, alors aucun bit ne pourra être retiré de la représentation, le premier bit de poids fort encodant le signe. Finalement, on choisit d'appeler cet algorithme par la fonction de prédiction, c'est-à-dire : *Parent-Child-Predictor*, et sera par la suite référencé par l'acronyme `PCP` suivi du nombre de bit encodant le nombre de `LZ` (ici 4).

3.5.3 Compression avec perte des données flottantes

Bien que les algorithmes de compression *sans perte* permettent de s'assurer de n'avoir aucun impact de la compression sur les données, et donc sur les analyses en post-traitement, il est généralement difficile d'obtenir des ratios de compression élevés pour des données flottantes. Néanmoins, afin d'aller plus loin et d'obtenir de meilleur ratio de compression, pour améliorer le stockage ou le transfert sur le réseau, les algorithmes *avec perte*, sont devenus populaires pour les données scientifiques. En effet, la compression *avec perte* est issue de méthodes héritées de la compression d'image tel que les ondelettes (*Wavelet compression*), qui sont très efficaces pour compresser des données tout en conservant les caractéristiques fines de l'image, [Boix and Cantó, 2010]. Néanmoins, l'évaluation de l'impact de la compression de ces méthodes se base sur ce qui est perceptible par l'oeil. En effet, en termes simples, si le taux de compression est élevé mais que l'oeil ne détecte pas d'artefact dans l'image ou la vidéo, on considéra la

compression comme très bonne. Le problème lorsqu'on applique ces algorithmes à des données scientifiques, issus de codes de simulations et destinées au post-traitement, est d'évaluer l'impact de cette perte d'information sur les résultats scientifiques. En effet, il est commun lors des phases d'analyses, de calculer des champs dérivés (norme d'un champ vectoriel, énergie, température, etc..) ou de produire des images, des courbes ou des spectres afin d'extraire des caractéristiques particulières. En conséquence, il est important de pouvoir définir une erreur maximale, relative ou absolue, autorisée lors de la compression. C'est ce qui a été apporté par les méthodes de type *error-bound* tels que `SZ`, [Di and Cappello, 2016], [Tao et al., 2017], `ZFP`, [Lindstrom, 2014]. On notera cependant, que même si une erreur peut être paramétrée, il est important de faire une étude quantitative pour estimer l'impact réel sur les analyses. Ainsi, on retrouve des études telles que [Baker et al., 2016] qui s'intéressent non seulement aux bénéfices de la compression mais également à l'impact sur les résultats scientifiques, pour éventuellement contraindre les erreurs ou les méthodes de compression à utiliser. Enfin, on notera qu'en fonction du type de méthode utilisée pour la perte d'information, il peut y avoir un effet de lissage sur les données. C'est le cas par exemple des approches par transformation, [Baker et al., 2016].

Dans les algorithmes *avec perte*, une étape additionnelle est rajoutée. En effet, après la *décorrélation*, une étape "d'approximation" est insérée et consiste à retirer de l'information non importante, éventuellement sous la contrainte d'une erreur imposée par l'utilisateur. Les bibliothèques `SZ` et `ZFP` proposent toutes les deux de définir une erreur relative ou absolue, sur le jeu de données, lors de la configuration de la compression. On notera que l'algorithme peut devenir asymétrique et la compression plus lente que la décompression, c'est le cas de `SZ` par exemple. On observera également, que pour certains jeux de données, la paramétrisation de l'erreur ne garantit pas que la tolérance demandée sera respectée, ce qui peut freiner certains utilisateurs à utiliser la compression avec perte. Néanmoins, dans notre cas, la méthode que l'on utilise pour supprimer de l'information permet de garantir une erreur relative bornée "point à point" : c'est-à-dire que l'erreur relative entre la valeur d'une cellule non compressée et la version décompressée (après compression avec perte) est bornée, et ce pour chaque cellule. Ce type de méthode est appelé *point-wise relative error bound*. De plus, l'erreur ne se propage pas lors de la décompression malgré ce qu'on pourrait penser à première vue, compte tenu du schéma de compression *Parent-Child* hiérarchique détaillé précédemment. Enfin, un dernier avantage considérable est que l'erreur peut être facilement centrée autour de 0 en rajoutant une simple opération lors de la décompression. Cette propriété est extrêmement importante parce que cela signifie que si on intègre une quantité le long d'une ligne de visée avec un algorithme de lancer de rayon par exemple, statistiquement, l'erreur devrait s'annuler au lieu de s'accumuler.

Contrairement à [Fu et al., 2017] qui propose d'ajuster le nombre de bits utilisés pour décrire l'exposant et la mantisse, notre approche est plus simple et consiste à conserver l'approche *sans perte* décrite précédemment sur les bits de poids fort et de retirer au moment de l'encodage un nombre de bits de poids faible, défini en paramètre d'entrée, que l'on appellera N_{loss}, voir figure 3.9. Sur cette figure, on notera que les $N_{loss} = 3$ bits de poids faible sont retirés des résidus s_i^*. Techniquement, cette suppression est faite lors de l'encodage dans le champ compressé, des résidus par superposition. En conséquence, la compression avec perte est donc une extension de l'algorithme sans perte avec un coût quasiment nul. On notera, de plus, que l'algorithme reste symétrique, ainsi comme on le verra à la section 3.8.5 la vitesse de compression est inchangée.

Lorsque les données sont en simple précision, il est important de faire attention à la somme potentielle des bits retirés. En effet, en simple précision, en `PCP4`, il ne faut pas dépasser la valeur $N_{loss} = 16$ parce que la partie *sans perte* de l'algorithme peut retirer jusqu'à 15 bits. L'encodage actuel nécessite de décrire une valeur avec au minimum 1 bit. En double précision, la marge de manoeuvre est beaucoup plus grande. Néanmoins, au-delà d'une certaine valeur de N_{loss} la précision sera équivalente, ou moins bonne, que si le champ était réduit en simple précision et le gain sur le ratio de compression nettement moins important. Enfin, l'erreur relative maximale peut se calculer de manière théorique par la formule 3.3 à partir du nombre de bits constituant la mantisse, noté N_{man} (23 bits en simple précision, 52 bits en double).

$$err_{rel,max} = \frac{2^{N_{loss}}}{2^{N_{man}}} \tag{3.3}$$

Pour l'instant, l'erreur est bornée entre $[0, 2^{N_{loss}} - 1]$. Lors de la décompression, il est très facile de centrer cette erreur autour de 0 et de la réduire d'un facteur 2. En effet, il suffit pour cela de rajouter $2^{N_{loss}-1}$ (ou mettre à 1 le $2^{N_{loss}-1}$ ème bit) à la représentation entière de la valeur décompressée. Ainsi, l'erreur sur une valeur donnée n'est plus bornée entre $\left[0, 2^{N_{loss}}\right]$ mais devient centrée autour de 0 : $\left[-2^{N_{loss}-1}, 2^{N_{loss}-1} - 1\right]$. Si on reprend l'exemple sur la figure 3.9, $N_{loss} = 3$ donc la différence maximale sur la représentation entière entre la valeur théorique et la

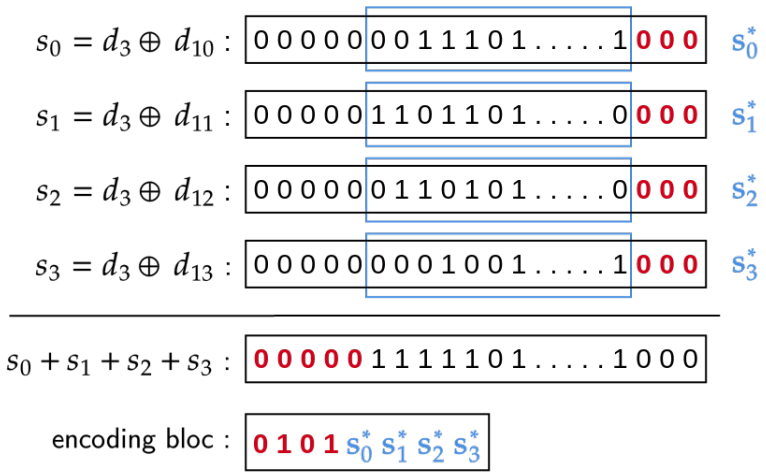

FIGURE 3.9 – Étape de la compression avec perte, avec un encodage sur 4 bits du nombre de zéros de poids fort retirés et avec $N_{loss} = 3$ bits de poids faibles retirés des résidus.

valeur décompressée sera de $2^N_{loss} - 1$ soit 7. L'erreur sera donc strictement positive et comprise entre $[0, 7]$. Si on rajoute $2^{N_{loss}-1} = 4$, alors l'erreur est ramenée entre $[-4, 3]$. Comme on l'a mentionné précédemment, cette propriété est très intéressante pour le post-traitement. Lorsqu'une grandeur est accumulée, on peut espérer une erreur moyenne proche de 0, plutôt qu'une erreur qui serait accumulée avec une erreur strictement positive, voir chapitre 4.

Finalement, la formule de calcul du ratio de compression théorique maximale 3.2 peut être étendue pour intégrer l'aspect *avec perte* :

$$C^d_{max} = \frac{E_{enc} \cdot 2^d}{(E_{enc} - N_{loss} - P_{lzrm}) \cdot 2^d + p} \tag{3.4}$$

Il est important de noter que contrairement à la plupart des bibliothèques de compression, on n'utilise pas d'étape complémentaire sous forme d'"encodeur" qui permet généralement d'obtenir de meilleurs ratios de compression. On reviendra sur ce point dans la discussion.

Compte tenu de la nature hiérarchique de la méthode compression, il est important de noter que la première valeur ne peut pas être compressée et il faut de toute façon l'écrire en clair afin de pouvoir amorcer la décompression. Avec les notations de la figure 3.3, cela signifie d'encoder les bits de d_0 en clair. On notera de plus, qu'on fait le choix d'encoder le champ compressé sur des entiers non signés avec le même nombre de bits que le champ à compresser i.e. : double précision sur des entiers 64 bits, simple précision sur des entiers 32 bits. Enfin, afin de faciliter les développements futurs et de pouvoir gérer différentes versions de la bibliothèque, tout en maintenant une rétro-compatibilité, la toute première valeur du tableau compressé contient les informations de version, le nombre de bits pour l'encodage du `LZ` et le nombre de N_{loss} (0 dans le cas *sans perte*). Ainsi, il est très facile à partir du champ compressé de connaître la précision du champ, la version de la bibliothèque, le nombre de `LZ` maximum retirables, le nombre de bits de poids faible retirés et donc l'erreur maximale sur les valeurs.

3.6 Compatibilité avec la structure des vtkHyperTreeGrid

Depuis 2017, une nouvelle structure de données est apparue au sein du package de visualisation `VTK` pour pallier le manque de structure de données pour la visualisation de grille `AMR` en arbre : les `HyperTreeGrid` et les `HyperTree`, [Harel et al., 2017a], [Harel et al., 2017b]. Un `vtkHyperTreeGrid` (`HTG`) est défini comme une grille uniforme à un niveau l dont chaque cellule, identifiée par un triplet de coordonnées logiques (i, j, k) (en `3D`), peut être la racine d'un `vtkHyperTree` (`HT`). Un `HT` est défini comme un jeu de données qui peut être représenté sous la forme d'un arbre et où chaque noeud contient exactement f^d enfants, avec f est le facteur de raffinement et d la dimension $\{2, 3\}$. Il est essentiel de noter qu'au sein d'un `HTG`, tous les `HT` doivent avoir les même paramètres (f, d). De plus, pour le moment, seuls des facteurs de raffinement de 2 ou 3 sont disponibles.

Compte tenu de ces caractéristiques, il est relativement simple de construire un `HTG` à partir de données issues du modèle `lightAMR`. Néanmoins, afin d'accélérer les traitements des différents filtres et d'alléger le nombre d'objets pour le rendu `2D`/`3D`, un masque est généralement associé à chaque noeud/feuille d'un `HTG` et ses `HT` correspondant. Ce masque, comparable au tableau d'appartenance du modèle `lightAMR`, permet de définir si une cellule doit être traitée (pour la visualisation ou durant l'application d'un filtre). Il est important que noter que pour se déplacer dans un `HT`, différents *curseurs* sont disponibles avec plus ou moins de fonctionnalités, voir [Harel et al., 2017b] pour plus d'informations. Ces curseurs ont été développés pour des parcours récursifs des `HT` à partir des cellules de la grille du `HTG`. En conséquence, si une cellule raffinée est masquée alors sa descendance ne sera pas explorée, et il est même interdit (levé d'une exception arrêtant le code) de se déplacer en profondeur depuis une cellule masquée et raffinée. Afin d'être compatible avec cette description, il est donc nécessaire d'appliquer la correction mentionnée précédemment concernant l'appartenance d'un noeud au domaine courant.

De plus, la grille est décrite depuis le niveau 0 par tous les domaines, ce qui constitue déjà un cas particulier dans la définition des *HyperTreeGrid*, puisqu'il n'y a donc qu'une seule racine dans une grille régulière de niveau 0. Cela pose un problème pour le parallélisme au sein de `VTK`, ou pour la visualisation de grands volumes de données qui nécessitent l'utilisation de plusieurs processus (tâche `MPI`). En effet, les noeuds de l'arbre sont décrits par plusieurs processus en même temps, ce qui engendre des effets de bord, par exemple, pour un rendu à un niveau `AMR` plus petit que le niveau maximum. Pour résoudre ces problèmes, et rendre le parallélisme natif au niveau des *HyperTree*, une approche consiste à adapter le modèle `lightAMR` afin de commencer la description de l'arbre `AMR` à partir d'un niveau `AMR` "plancher" (`levelmin` selon la terminologie `RAMSES`) qui est atteint partout dans l'espace de simulation. Il est alors nécessaire de rajouter une information de position des noeuds racines sous la forme d'un duplet/triplet de coordonnées logiques de la position du noeud, dans une grille uniforme à ce niveau plancher. Par exemple, pour la grille `2D` de la figure 3.1, cela revient à adapter la représentation en arbre comme représenté sur la figure 3.10.

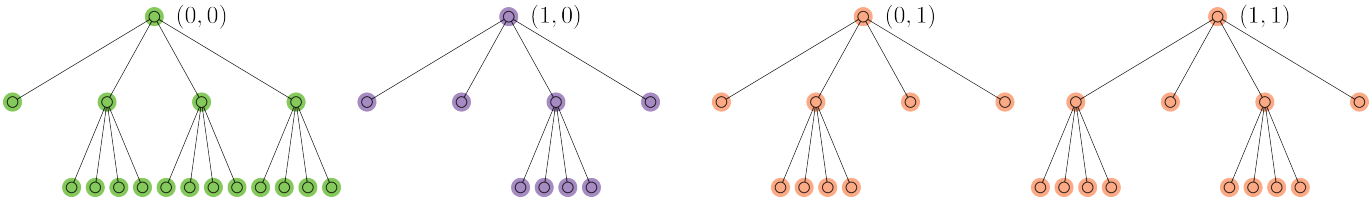

FIGURE 3.10 – Description en arbre compatible avec le modèle des *vtkHyperTreeGrid*.

Le problème de cette approche est double. Premièrement, afin de construire cet arbre correctement dans le code `RAMSES` au moment d'une sortie de post-traitement, cela implique de nombreuses communications entres les différents processus pour s'échanger des morceaux d'arbres. En effet, avec cette structure `VTK`, on suppose un découpage au niveau des cellules grossières de la grille régulière décrite. De plus, les simulations astrophysiques sont par essence largement multi-échelles et impliquent le plus souvent des arbres `AMR` profonds avec un écart de 6 à 12 niveaux `AMR` entre le niveau minimum et le niveau maximum. Avec un tel écart, il ne serait pas possible pour certaines simulations de construire une telle description par manque de ressource mémoire.

Deuxièmement, le coût en terme de stockage n'est pas négligeable. Par exemple, pour une simulation en 3 dimensions dont le niveau minimum serait de 8, cela impliquerait de rajouter $3.\ 2^{10*3} = 3.221.225.472$ entiers 4 octets soit $\sim 12\,GB$ de données sur l'ensemble du *snapshot*, pour décrire la position logique des noeuds. Bien évidemment, les niveaux précédents n'étant pas décrits, le gain en stockage peut s'exprimer par grandeur double précision : $\sum_{i=0}^{l-1} 2^{i*3}.2^3$ avec $l = 10$. On a donc une économie de $\sim 1.2\,Gb$ pour une grandeur double précision (on néglige ici le gain au niveau des tableaux de raffinement et d'appartenance). L'opération deviendrait donc potentiellement intéressante si plus de 11 champs scalaires double précision étaient stockés. De plus, l'algorithme de compression de données flottantes ne peut pas compresser, par construction, la valeur racine de l'arbre, puisqu'elle est utilisée comme valeur de départ pour la décompression, le gain serait donc encore moins intéressant si les champs étaient compressés.

Afin de s'affranchir de ces problématiques, on choisira d'utiliser la description de la grille `AMR` avec un arbre commençant au niveau 0. Néanmoins, les manipulations nécessaires pour produire des données compatibles avec le modèle de `VTK`/`Paraview` seront faites en post-traitement, voir chapitre 4.

3.7 Détails d'implémentation dans RAMSES

Avant de rentrer dans les détails techniques, il est important de préciser que la compression de données a été implémentée dans une bibliothèque à part, écrite en C et C++ pour laquelle j'ai développé une interface Fortran pour l'intégration dans RAMSES. Cela signifie notamment que toute amélioration des algorithmes de compression se fera de manière transparente vis-à-vis des codes qui l'utilisent (RAMSES et les outils d'analyses). Afin de s'assurer de la bonne implémentation, et pour faciliter les évolutions futures, une centaine de tests unitaires ont été développés avec googletest [a].

Comme on l'a mentionné au chapitre précédent, le flux de données du code a été séparé en deux. Concrètement cela signifie qu'il est désormais possible d'avoir deux types de sorties différentes avec des fréquences différentes. Afin d'apporter plus de flexibilité et en vue de pouvoir réduire davantage le volume de données, la possibilité de ne sortir qu'une sous-partie des champs scalaires a été rajoutée. On notera également qu'il est possible d'activer le *downcasting* des champs scalaires. Bien évidemment, des valeurs booléennes permettent d'activer l'élagage de l'arbre, la compression de la description AMR (CPS52), la compression des données flottantes (PCP). Si la compression *avec perte* est activée, il est alors possible, également, depuis le fichier d'entrée, de renseigner le paramètre N_{loss}. L'encodage du nombre de bits retirés P_{lzrm} est pour l'instant limité par défaut à 4 bits et donc 15 bits retirés au maximum. Les options sont récapitulées dans le tableau 3.1. On notera qu'elles sont contenues dans la *namelist* Fortran, appelée &HERCULE_PARAMS.

Options	Valeur	Signification
hdep_format	bool	Activation des sorties en lightAMR.
hdep_prune	bool	Activation de l'élagage de l'arbre AMR.
hdep_downcast	bool	Activation du *downcasting* des champs scalaires.
hdep_compression	bool	Activation du CPS52.
hdep_data_compression	bool	Activation du PCP-4.
hdep_lossless	bool	Mode *sans perte* pour PCP-4.
hdep_lossy	bool	Mode *avec perte* pour PCP-4.
hdep_lossy_nloss	int	Nombre de bits de poids faible à retirer.
houtput	int	Fréquence des sorties HDep.
hdep_nvars	int	Nombre de champs scalaires à écrire.
hdep_savevars	list, int	Index des champs scalaires à écrire.
hdep_dirout	str	Dossier d'écriture de la base HDep.

TABLE 3.1 – Options de configuration des sorties d'analyses au format HDep avec le modèle lightAMR dans RAMSES. On retrouve les options permettant de piloter les algorithmes d'élagages et de compression présentés, ainsi que la gestion des champs scalaires à écrire.

Au sein du code, trois tableaux sont utilisés pour construire le tableau de raffinement et le tableau d'appartenance : father, son, cpu_map. Ils permettent respectivement, à partir d'un indice de grille, d'obtenir l'indice de la cellule parent, de la cellule enfant et le numéro du rang MPI d'appartenance de la cellule. Afin de conserver l'ordre des cellules pour la description des champs scalaires, un tableau d'indirection sous la forme des indices des cellules est conservé. Par défaut, la construction de l'arbre se fait jusqu'au niveau maximum de la simulation, mais il est possible de définir un niveau maximum de description en modifiant le paramètre de niveau AMR maximum de la fonction build_tree_classic qui construit l'arbre. Bien que cette option ait été validée, elle n'est pas utilisée dans le cadre de la simulation ExaMilkyWay parce qu'une grande partie de l'information, même à des fins de *monitoring*, est portée sur les deniers niveaux.

a. https://google.github.io/googletest/

Finalement, le schéma global pour établir une sortie de post-traitement au modèle `lightAMR` peut se résumer en 6 grandes étapes :

1. construction des tableaux de raffinement, d'appartenance, des indices des cellules,
2. application de l'algorithme d'élagage de l'arbre, `[optionnel]`,
3. compression `CPS-52` du tableau de raffinement et d'appartenance, `[optionnel]`,
4. agrégation des grandeurs scalaires demandées dans l'ordre des cellules du tableau de raffinement,
5. compression `PCP` *avec perte* ou *sans perte*, `[optionnel]`,
6. envoie des donnés à une bibliothèque d'`E/S` pour écriture (ici, `Hercule`).

D'un point de vue technique, la création du *contexte* `Hercule` pour le domaine courant et le temps courant est fait entre les étapes 3 et 4. Ensuite, les données sont agrégées dans un tableau temporaire qui est envoyé de manière synchrone à la bibliothèque d'`E/S`, champ par champ. En conséquence, avec `Hercule`, une copie mémoire des données est faite. Pour limiter l'empreinte mémoire, on force l'écriture des données champ par champ. Cette méthode avec forçage de l'écriture a un impact fort sur les performances d'écriture. On pourrait éviter cette recopie mémoire et donc le forçage d'écriture. En effet, le tableau d'indirection utilisé pour ordonner les grandeurs scalaires est conservé jusqu'à ce que toutes les grandeurs soient envoyées pour écriture. On pourrait donc enrichir l'interface de `Hercule` afin de pouvoir fournir un pointeur vers un tableau d'indirection et un pointeur vers un tableau de données brutes, et laisser la bibliothèque faire l'indirection au moment des écritures, sans forcer l'écriture manuellement. Néanmoins, pour l'écriture des grandeurs différentes de celles stockées dans les tableaux de données brutes, il sera nécessaire d'utiliser des tableaux temporaires. De plus, pour activer la compression de données, il faudrait que celles-ci soient implémentées, intégrées ou reliées à `Hercule`. On notera cependant, que le surcoût de cette approche est compensé par un volume de donnée réduit. Par exemple, dans le cas de la simulation `ExaMilkyway`, une protection/reprise nécessite l'écriture de ~ 4.5 To de données, alors qu'une sortie d'analyse ne pèse que ~ 200 Go.

Concernant les particules et leurs champs associés, elles sont bien évidemment rajoutées dans la base de données, par domaine, entre les étapes 3 et 4, sans compression. Il est néanmoins possible d'activer également le *downcasting* pour ces données. En fonction des simulations, le volume de données représenté par les particules peut devenir important, notamment pour les simulations cosmologiques telle que `Extreme-Horizon` où elles représentent ~ 800 *Go*. On notera qu'il est communément admis que les analyses associées aux particules sont sensibles à la précision. En conséquence, il pourrait être intéressant de voir l'apport, sur ce type de jeux de données, de méthode de compression *sans perte* telle que [Isenburg et al., 2005]. On pourrait envisager d'exploiter la structure hiérarchique de telle sorte que le centre de la cellule parent contenant les particules, soit utilisé pour compresser leurs positions. Enfin, on notera également que le lien particule $\leftrightarrow$ maillage n'est pas conservé car il s'agit d'une information qui peut être très facilement récupérée en post-traitement avec un simple algorithme d'échantillonnage (*point sampling*).

Finalement, il est important de souligner que la construction du modèle et des champs scalaires n'est pas liée à la bibliothèque d'`E/S` choisie. En effet, les étapes 1 à 5 sont indépendantes des écritures de données à l'étape 6. En conséquence, les tableaux produits par les différentes étapes peuvent donc être écrits avec n'importe quelle bibliothèque d'`E/S` (parallèle ou non). On pourrait, par exemple, choisir d'utiliser l'interface `POSIX` et le paradigme *file-per-process* d'origine, pour écrire un fichier par processus `MPI` avec un format binaire, accompagné d'un fichier descriptif spécifiant un identifiant et un type pour un champ. Il est ensuite facile de pouvoir analyser le fichier descriptif et de construire une table d'indexation par le nom des champs et l'offset dans le fichier binaire pour pouvoir accéder à un champ directement sans devoir lire l'ensemble du fichier. Avec cette approche, le maillage et toutes les grandeurs associées aux cellules seraient regroupées au sein du même fichier, ce qui réduirait déjà le nombre de fichiers produits par rapport à la version d'origine du code.

Bien que cette approche permette de réduire les dépendances rajoutées au code `RAMSES`, on la déconseille vivement pour toutes les raisons mentionnées à la section `POSIX` dans le chapitre 2. Une autre méthode consisterait à utiliser `HDF5` dans sa version parallèle avec l'approche *single-shared-file*. Le problème mentionné dans le chapitre précédent, avec cette approche, est la gestion d'un fichier unique. J'ai développé une bibliothèque, `Groot`, qui permet de mimer la méthodologie de `Hercule`, et de produire N/P fichiers depuis une simulation sur N processus `MPI` en regroupant les écritures par paquets de P processus dans un même fichier `HDF5`, grâce aux *hyperslabs* et à la mise en place de sous-communicateur `MPI`. Néanmoins, comme on l'a mentionné à la section sur la granularité, la

```
refinement_array_size:uint64_t
refinement_array:uint8_t
[...]
density_array_size:uint64_t
density_array:double
```

```
open(unit=42, file=out_file, [...])
    write(42) lightAMR_tree_size
    write(42) lightAMR_tree(:)
    [...]
    write(42) lightAMR_ncells
    write(42) lightAMR_scalar_rho(:)
close(unit=42)
```

FIGURE 3.11 – Sur la gauche un exemple de fichier descriptif du format `ASCII` (unique pour tous les processus) et à droite, le code `Fortran` pour générer le fichier binaire avec les donnés au modèle `lightAMR`.

concaténation de deux `lightAMR`, n'est pas un troisième `lightAMR`. De plus, en post-traitement, il est très intéressant de pouvoir gérer les données par domaines, voir chapitre 4. On donne un exemple de structurations possibles d'un fichier `HDF5` sur la figure 3.12.

```
group: simulation_info
  dset: 'mpi_rank', dtype: int32, range: [0, 3], shape : (4,)
  attr: 'time', type: float32, value: 1.1012,
  attr: 'code_name', type : |S5, value: b'ramses'
group: results
  group: lightAMR
    group: mesh
      group: refinement
        dset: 'mpi_offsets', type: int32, range: [0, 4811], shape: (4,)
        dset: 'mpi_sizes', type: int32, range: [1420, 2145], shape: (4,)
        dset: 'values', type: int8, range: [0, 1], shape: (7221,)
        [...]
    group: data
      group: density
        dset: 'mpi_offsets', type: int32, range: [0, 4811], shape: (4,)
        dset: 'mpi_sizes', type: int32, range: [1420, 2145], shape: (4,)
        dset: 'values', type: float64, range: [0.001, 1.125], shape: (7221,)
```

FIGURE 3.12 – Exemple d'une structure hiérarchique possible pour le contenu d'un fichier `HDF5` produit avec 4 processus `MPI` pour le modèle `lightAMR`.

Il convient d'écrire les données de tableaux concaténés au sein d'un *groupe* agissant comme une *méta-structure* contenant les informations pour un traitement par domaine. En effet, sur la figure 3.12, si on prend l'exemple du groupe *tree* (tableau de raffinement), il contient trois *datasets* : `mpi_offsets`, `mpi_sizes`, `values`. Le tableau `1D values` contient les K (ici $K = 4$) tableaux de raffinement concaténés au sein d'une même *hyperslabs*. Les *datasets* `mpi_sizes` et `mpi_offsets` permettent de renseigner respectivement les tailles des tableaux individuels par domaine et les *offsets* d'écriture dans le tableau concaténé. On notera que dans le groupe `simulation_info`, on stocke les numéros des processus `MPI` (numéro de domaine) des processus ayant contribué au fichier. Ainsi, en post-traitement, il est possible de conserver tous les bénéfices de l'approche par domaines du traitement des données, à condition d'implémenter l'interface d'accès aux données par domaine. Il serait cependant intéressant de d'évaluer les performances d'écriture (et de lecture par l'outil d'analyse) de cette approche afin de la comparer à celle utilisée avec `Hercule`.

3.8 Résultats

Afin de pouvoir évaluer les gains apportés par ce modèle et les algorithmes associés, j'ai développé un convertisseur de données permettant de lire les fichiers d'un dossier de protection `RAMSES` et de convertir les données, à la volée, en une base de dépouillement `HDep` avec le modèle `lightAMR`. Il est également possible d'activer les différents algorithmes présentés dans ce chapitre et de sélectionner les champs scalaires à rajouter dans la base. Afin de tirer profit des `E/S` parallèles, ce convertisseur peut être utilisé en séquentiel comme en parallèle (`MPI`), avec un nombre quelconque de processus. Il est donc possible de convertir un *snapshot* `RAMSES` d'une simulation

qui a tourné sur N processus avec P processus de conversion, tout en produisant une base `Hercule` de N domaines (l'association processus/domaine est conservée). On notera également qu'il est possible de ne convertir qu'une sous-partie des fichiers (domaines) pour générer des bases `HDep` partielles. Du fait de la difficulté de déplacer et/ou de stocker de grands volumes de données `RAMSES`, les performances de vitesse de compression n'ont pas pu être mesurées avec le même type de matériel (`CPU`). On précisera donc le type de matériel utilisé ainsi que ses caractéristiques principales.

3.8.1 Données de tests utilisées

On sélectionne des jeux de données issus de différents types de simulations, toutes effectuées avec `RAMSES`. Ces simulations appartiennent à différents domaines d'études allant de la cosmologie à la formation stellaire. Ainsi, le maillage `AMR` et les ordres de grandeurs des champs scalaires varient énormément d'une simulation à l'autre. Par exemple, dans le cas d'une simulation cosmologique, la grille `AMR` est généralement plus uniformément raffinée que dans une simulation de type zoom. De même, le nombre de niveaux actifs peut varier en fonction des sujets d'étude. L'utilisation de ces différents jeux de données permet donc d'estimer, au plus près des données et des cas d'usages réels, les performances du modèle et de ses algorithmes associés. Il est important de noter que les outils d'analyse gèrent les données `RAMSES` par fichier (et donc par domaine), mais également que le modèle `lightAMR` est auto-portant par domaine (et sera traité par domaine également, voir chapitre 4). On considère donc qu'un jeu de données (*dataset*) est représenté par les données d'un domaine.

On résume dans le tableau 3.2, le nombre de processus `MPI`, le nombre de cellule (sans élagage), le numéro de sorties ainsi que les volumes cumulés des fichiers `AMR` et `HYDRO` pour chaque simulation. Les projets `Orion`, [Ntormousi and Hennebelle, 2019], et `Frig`, [Hennebelle, 2018], sont des simulations d'étude de nuages moléculaires et sont donc des simulations avec un grand écart de niveau `AMR` (> 10) et un maillage `AMR` très localisé dans la boîte. Le projet `Extreme-Horizon`, [Chabanier et al., 2020], est une simulation cosmologique avec un raffinement du maillage "relativement" homogène dans toute la boite. Le projet `Cosmic Dawn III`, [Lewis et al., 2022], est également une simulation cosmologique qu'on utilisera comme cas particulier puisqu'elle a été faite à grille fixe avec `RAMSES` et à très haute volumétrie (niveau `AMR` minimum égal au niveau `AMR` maximum :13). Enfin, la simulation `ExaMilkyWay` est la simulation de galaxie isolée, avec un raffinement très localisé, faite dans le cadre de ces travaux de thèse, voir chapitre 6. Les chiffres donnés pour cette simulation ne sont pas issus du convertisseur puisque les données ont directement été produites au modèle `lightAMR` avec `RAMSES` et `Hercule`. On utilisera la sortie à l'étape d'intégration 2874, à $t \simeq 20\ Myr$ après l'activation de la haute résolution.

Simulation	Domaines	Snapshot	Cellules (10^6)	AMR (Go)	HYDRO (Go)
ORION	512	87	445.28	8.5	46.9
FRIG	4096	249	1288.15	29.1	117
ExH	12 800	280	23 660.29	490	2360
ExaMilkyway	16 384	35	37 672.45	71.85*	1488*
Cosmic Dawn III	131 200	106	549 755.81	15 800	52 500

TABLE 3.2 – Caractéristiques principales des simulations dont les jeux de données, dans le format natif de `RAMSES`, ont été utilisés pour analyser les performances du modèle `lightAMR` et des algorithmes de compression. Les données de la simulation `ExaMilkyWay` (avec une astérisque) sont au format `lightAMR` non compressé (après élagage).

3.8.2 Métriques et bibliothèques

Avant de commenter les résultats, il est essentiel de détailler les métriques qui seront utilisées pour quantifier les performances du modèle et des algorithmes associés. Le volume de données de description du maillage du modèle `lightAMR`, et donc des tableaux *raffinement* et d'*appartenance*, sera comparé au volume de la description du maillage d'origine et donc des fichiers `AMR` produit par `RAMSES`. Il est important de noter que le modèle `lightAMR` contient quelques méta-données pour être auto-portant, de l'ordre de quelques dizaines d'entiers 32 bits par domaine. Ces méta-données seront négligées parce que les volumes qu'elles représentent sont négligeables comparées

au poids des tableaux de description du maillage `AMR` (environ $\simeq 9\ Mo$ pour le cas extrême de `Cosmic Dawn III` et inférieur à $1\ Mo$ pour les autres simulations).

Comme on l'a vu précédemment, les données sont cohérentes par domaine et comme on le verra au chapitre 4, elles sont destinées à être analysées par domaine. En conséquence, puisqu'un domaine peut être analysé seul en fonction des besoins des analyses, on considéra donc un jeux de données comme étant les données d'un domaine. On donnera donc des statistiques des différentes métriques pour l'ensemble des domaines d'une simulation avec une valeur minimale, maximale, moyenne et une déviation standard.

Pour l'algorithme d'élagage de l'arbre, on donnera le taux de cellules retirées en pourcent, P_r, calculé par le rapport du nombre total de cellules après application de l'algorithme, n_p, sur le nombre total de cellules dans les données d'origine, n_o :

$$P_r = \left(1.0 - \frac{n_p}{n_o}\right) * 100.0 \tag{3.5}$$

Pour les algorithmes de compression, on utilise un *ratio de compression* que l'on notera C_r, défini comme le rapport de la taille d'origine du jeux de données s_{orig} par la taille compressée s_{comp} :

$$C_r = \frac{s_{orig}}{s_{comp}} \tag{3.6}$$

La vitesse de compression C_s sera calculée comme étant la capacité de consommation, par l'algorithme, de données, exprimé en Mégaoctet par seconde :

$$C_s = \frac{s_{orig}}{t} \tag{3.7}$$

Enfin, pour la compression *avec perte*, il est généralement fréquent d'exprimer la qualité de la compression en fonction de métriques associés à la visualisation tel que le `PSNR` (*Peak Signal to Noise Ratio*, [Liang et al., 2018], [Lindstrom, 2014], [Tao et al., 2017], [Ballester-Ripoll et al., 2018], [Cappello et al., 2019]. Cette approche est très intéressante parce qu'elle permet d'établir un lien entre la compression de données et les analyses qui seront faites à partir des données compressées. On définira donc le `PSNR` tel que proposé par [Tao et al., 2017] et [Liang et al., 2018], en fonction du `RMSE` (*Root-Mean-Square Error*), equation 3.8. On précise que x fait référence à une donnée d'origine et $\tilde{x}$ à une donnée décompressée (après compression avec perte). En conséquence, $x_{max} - x_{min}$ fait donc référence à l'intervalle des valeurs d'un champ. Dans le cas des simulations astrophysiques et les processus multi-échelles, cet intervalle peut être très grand (10 ordre de grandeur en densité par exemple). Cette métrique est donc discutable, mais elle permet de confronter les chiffres obtenus aux valeurs de la littérature.

$$\begin{aligned} RMSE &= \left(\frac{1}{N}\Sigma_{i=1}^{N}\left(x_i - \tilde{x_i}\right)^2\right)^{1/2} \\ PSNR &= 20\ \log_{10}\left(\frac{x_{max} - x_{min}}{RMSE}\right) \end{aligned} \tag{3.8}$$

Pour la méthode de compression `CPS52`, on confrontera les résultats obtenus par l'algorithme proposé avec ceux de bibliothèques *open-source* : `LZ4` [b], Zlib [c] et Snappy [d]. Pour la compression de données flottantes *sans perte* et *avec perte*, on comparera les résultats obtenus avec plusieurs bibliothèques majeures à l'état de l'art, spécialisées dans la compression de données scientifiques : `FPZIP`, [Lindstrom and Isenburg, 2006], `FPC`, [Burtscher and Ratanaworabhan, 2007], `ZFP`, [Lindstrom, 2014] et `SZ3`, [Liang et al., 2023].

3.8.3 Algorithme d'élagage

Dans le tableau 3.3, on donne les taux d'élagage de la représentation en arbre. On constate, pour les simulations `Orion` et `Frig`, que l'on obtient des taux assez élevés. L'algorithme est 2 à 3 fois plus performant que pour la simulation `Extreme-Horizon` ou `ExaMilkyWay`. Les valeurs maximales montrent que pour certains domaines, plus de la moitié de la description de l'arbre peut être retirée. Pour la simulation `Frig`, le domaine correspondant

b. https://zlib.net
c. https://lz4.github.io/lz4/
d. https://google.github.io/snappy/

au taux de 65.4 % est le domaine 2829. Il a un nombre de cellules moyen par rapport aux autres domaines, soit environ $\simeq$ 190000 cellules dans sa description en arbre (cellules feuilles et noeuds), dont près de 125000 sont masquées (cellules fantômes en dehors du domaine). L'enveloppe feuille contient des cellules du niveau 14 au niveau 18. Après l'application de l'algorithme d'élagage, le nombre de cellules masquées est réduit à 131 (13 noeuds et 118 feuilles). Visuellement, la différence est flagrante comme le montre la figure 3.13. Sur cette figure, on constate que l'information réellement portée par le domaine (les cellules feuilles appartenant au domaine) est localisée dans une petite région. Néanmoins, une grand partie de la boite de simulation est décrite. La raison d'un si grand taux d'élagage est donc lié à cette "sur-description" nécessaire aux schémas numériques, qui ont besoin de connaître la valeurs des cellules, y compris des cellules fantômes, aux différents niveaux `AMR`.

Simulation	Minimum (%)	Maximum (%)	Moyenne (%)	Déviation (%)	Global (%)
`Orion`	17.23	47.32	31.38	6.27	26.35
`Frig`	20.05	65.42	47.94	9.50	38.65
`Extreme-Horizon`	6.99	23.24	11.72	2.07	11.42
`ExaMilkyWay`	6.12	27.15	11.24	2.16	11.21
`Cosmic Dawn III`	8.66	8.67	8.66	0.002	8.66

TABLE 3.3 – Taux d'élagage de l'arbre pour les jeux de données testées. La colonne *global* donne le taux d'élagage cumulé sur l'ensemble de la simulation.

Les performances d'élagage sur `Extreme-Horizon` sont moins bonnes en moyenne. Ce résultat est attendu, car le raffinement de la grille est homogène dans la boite de simulation. Ces résultats sont en accord avec le fait que `RAMSES` est plus efficace en mémoire sur des simulations cosmologiques que sur des simulations de type *zoom*. Néanmoins, bien que le taux d'élagage global soit de 11.42% dans le cas d'`Extreme-Horizon`, cela représente un gain en terme de volume de données sur disque d'environ 240 *Go*. Dans le cas de la simulation `ExaMilkyWay`, on aurait pu s'attendre à des performances similaires aux autres simulations zoom, voir même meilleures. Cependant, cette simulation est une simulation zoom poussée à l'extrême, où la totalité du raffinement se situe dans un volume d'environ 0.2% de la boite, ce qui n'est pas le cas des simulations `Orion`[e] et `Frig`[f].

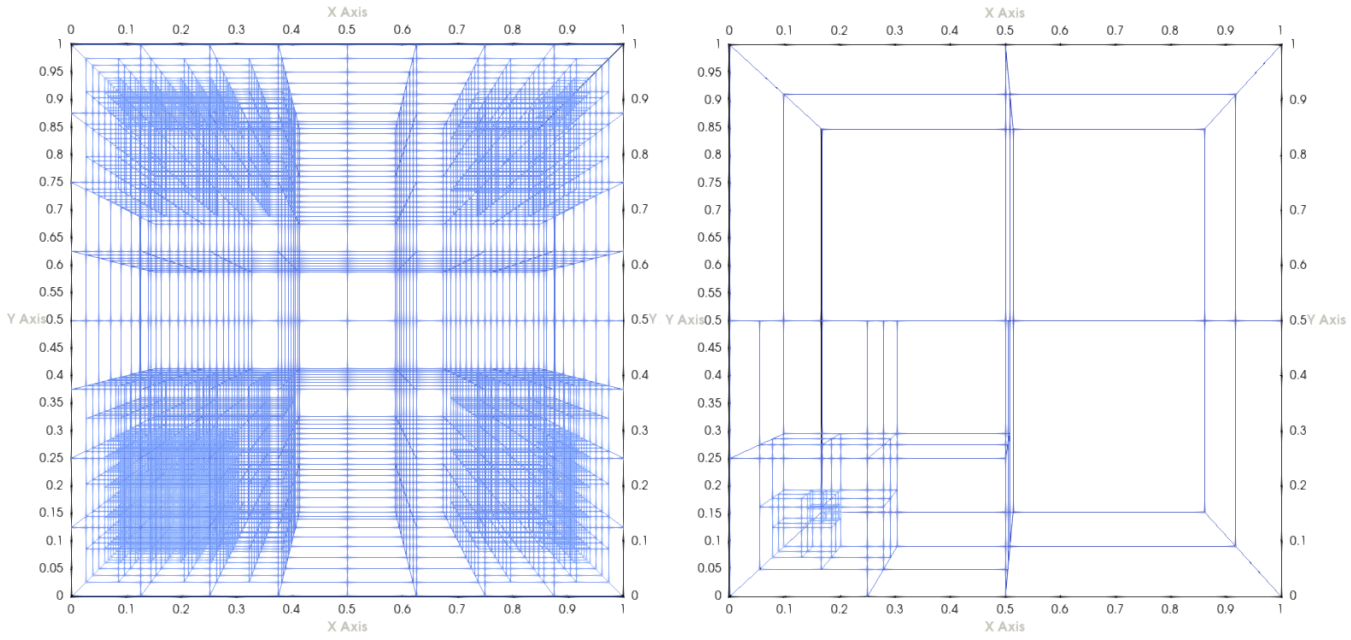

FIGURE 3.13 – Représentation des cellules feuilles en non structuré pour le domaine 2829 de la simulation `Frig`. A gauche, avant élagage et à droite, après application de l'algorithme.

Concernant la simulation en grille fixe, `Cosmic Dawn III`, on obtient des valeurs constantes entre les

e. http://www.galactica-simulations.eu/db/STAR_FORM/ORION/ORION_FIL_HYD_RES/
f. http://www.galactica-simulations.eu/db/STAR_FORM/FRIG/FRIG6_ZOOM7/

domaines, de l'ordre de 8%, ce qui est un résultat attendu puisqu'il s'agit d'une simulation en grille fixe. En conséquence, le découpage des domaines suivant la courbe de Hilbert est très régulier et on a donc un rapport volume sur surface des domaines maximisé. Ainsi le voisinnage des cellules aux différents niveaux `AMR` (entre le niveau 0 et le niveau de grille fixe) est très simple et régulier. Néanmoins, compte tenu des volumes conséquents de cette simulation, 8% représente un gain de plus de 4 To sur disque. On rappelle qu'enlever une cellule de la description `AMR` signifie également enlever sa valeur des champs scalaires décrivant les grandeurs physiques associées au maillage. En conséquence, l'algorithme d'élagage est la première étape pour réduire le volume de données produit par les simulations `RAMSES`.

On donne dans le tableau 3.4, les informations récapitulatives concernant le nombre de cellules total dans la simulation après application de l'algorithme d'élagage, ainsi que les pourcentages des noeuds et des feuilles de l'arbre qui sont masqués et non masqués (cumulé sur tous les domaines). Ces informations permettent de mettre en évidence le surcoût lié au modèle avec une description en arbre. Puisque toutes les simulations testées sont en `3D`, on converge vers 12.5% de surcoût du stockage des noeuds de l'arbre, c'est-à-dire $1/f_d^{DIM=3} = 1/8$. Ces 12.5% représentent le prix à payer pour conserver la structure hiérarchique du maillage, bénéficier des approches de type *level-of-details* en post-traitement mais également de la compression de données qui utilise l'information des noeuds de l'arbre. On notera qu'on pourrait choisir un autre modèle de données, qui ne stocke que les cellules feuilles et leurs grandeurs, mais cela impliquerait, pour chaque cellule, de stocker en plus ses coordonnées logiques et son niveau. Si on prend l'exemple de la simulation `Extreme-Horizon`, le volume de données après élagage des grandeurs hydrodynamiques est de l'ordre de $2300 * 0.8858 = 2037$ Go, et donc ~ 255 Go de "surplus" représenté par le stockage de l'information des noeuds de l'arbre. Si, on stocke les coordonnées logiques des cellules grâce à l'indice de Hilbert (disponible dans `RAMSES`) ou Morton, ce qui permet d'utiliser qu'un entier 64 bits au lieux de 3 coordonnées logiques sur des entiers 32 bits, alors cela implique un surcoût de stockage d'environ 137 $Go + 17$ Go. Il y a un gain de ~ 100 Go mais on perd les avantages importants apportées par la conservation de la structure hiérarchique mentionnés précédemment. Il est important de noter également que l'indice de Hilbert en double précision, ne permet pas de dépasser le niveau 16, au-delà de ce niveau, il faut utiliser la quadruple précision, et dans ce cas là, il n'y a plus de gain. Enfin, on constate également sur ce tableau que les noeuds et les feuilles masqués ne représentent qu'une fraction négligeable du nombre total de cellules et donc un coût en terme de stockage qui est négligeable.

Simulation	Cellules	Feuilles $\in$ (%)	Feuilles $\notin$ (%)	Noeuds $\in$ (%)	Noeuds $\notin$ (%)
ORION	327 926 704	87.4831	0.0169	12.4977	0.0022
FRIG	790 282 960	87.4389	0.0611	12.4918	0.0081
ExH	20 975 129 336	87.4944	0.0056	12.4993	0.0007
ExaMilkyway	37 672 445 744	87.4955	0.0045	12.4994	0.0006
Cosmic Dawn III	628 298 481 664	87.4389	0.0009	12.4999	0.0001

TABLE 3.4 – Bilan du nombre total de cellules (noeuds + feuilles) après élagage, accumulés sur tous les domaines en modèle `lightAMR` pour les différents jeux de données, ainsi que les pourcentages de feuilles et de noeuds appartenant aux domaines (symbole $\in$) et n'appartenant pas aux domaines (symbole $\notin$).

3.8.4 Compression CPS52

On compare les performances en terme de ratio, C_r, et de vitesse de compression de notre algorithme `CPS52`, avec plusieurs bibliothèques communément utilisées, *open source* et spécialisées dans la compression de données "universelle" : `LZ4` (version 1.9.3), `Zlib` (version 1.2) et `Snappy` (version 1.1.7). Il est important de noter que ces bibliothèques font l'objet d'évolutions régulières qui peuvent affecter leurs performances, les chiffres présentés sont donc valables pour les versions utilisées. On précise également que les tests de compression sont faits sur les tableaux après élagage de l'arbre. La bibliothèque `LZ4` est un algorithme asymétrique et universel, du type `LZ77`, [Ziv and Lempel, 1977]. L'idée principale de ce type d'algorithme est de rechercher les occurrences de motif dans les données à compresser et de les encoder de manière compacte. Il existe de nombreuses approches différentes pour choisir la taille des motifs, rechercher les occurrences, encoder les données, etc. Généralement, des tables de hachage sont utilisées parce qu'elles permettent d'accélérer significativement la recherche d'occurrence et donc la

vitesse de compression. La bibliothèque `Snappy`, bien qu'utilisant une méthode de recherche et un encodage différent de la `LZ4`, se base sur la même idée de base pour compresser les données. La `Zlib` se base également sur la méthode `LZ77` (recherche et encodage des occurrences de motifs) mais utilise un encodage plus riche en se basant sur l'encodage de Huffman, [Huffman, 1952], afin d'améliorer le taux de compression. Avec ce type d'encodage, un motif est encodé avec une longueur variable, inversement proportionnelle à sa probabilité d'apparition. On notera que la nécessité de connaître la probabilité d'apparition d'un motif impliqué généralement que l'algorithme devient *multi-pass*. En conséquence, on s'attend donc à avoir des vitesses de compression "élevées" avec `LZ4` et `Snappy`, avec des ratios moins élevés que ceux de la `Zlib` qui offre une vitesse de compression généralement plus faible à cause de son encodage plus complexe. On notera également que la `Zlib` propose différent niveau de compression, de 1 à 9 favorisant soit la vitesse, soit le ratio. On teste donc le niveau 1 (meilleur vitesse) et le niveau 9 (meilleur ratio)

Les ensembles des données des domaines sont traitées de manière séquentielle, les tailles non compressées et compressées sont cumulées ainsi que les temps de compression. Les vitesses C_s et les ratios de compression C_r sont ensuite calculés avec les formules présentées précédemment. On donne donc dans le tableau 3.5 les ratios globaux, ce qui permet de refléter le gain réel sur disque. On constate tout d'abord qu'il y a une différence significative entre le tableau de raffinement et le tableau d'appartenance. En effet, le taille des paquets, i.e. : le nombre de $0/1$ consécutif, est plus grande dans le tableau d'appartenance et plus chaotique dans le tableau de raffinement. En conséquence, l'algorithme `CPS52` permet d'atteindre des ratios très élevés comparés aux autres algorithmes lorsque la taille des paquets est très grande. Curieusement, `Snappy` ne permet pas d'atteindre des ratios très élevés même pour le tableau d'appartenance. On atteint probablement une limite lié à l'implémentation de l'algorithme qui fait saturer le ratio de compression à ~ 21. Le tableau de raffinement est plus difficile à compresser pour tous les algorithmes (à l'exception de la simulation `Cosmic Dawn III`). La `Zlib` au niveau 9 permet d'obtenir le meilleur ratio au prix d'une vitesse de compression significativement réduite, voir tableau 3.6. L'algorithme `CPS52` permet d'obtenir un ratio similaire, pour ce tableau, avec le niveau 1 de la `Zlib`. La simulation `Cosmic Dawn III` fait exception à cause de son maillage `AMR` totalement raffiné (grille "fixe"), qui s'exprime dans le tableau de raffinement par de très grands paquets de 1 consécutifs, favorisant ainsi très nettement le `CPS52`. De même, le découpage du maillage génère un tableau d'appartenance très régulier avec des grands paquets de 0.

Simulation	CPS52		LZ4		snappy		Zlib (1)		Zlib (9)	
	Raff	App	Raff	App	Raff	App	Raff	App	Raff	App
`Orion`	48.1	4397.5	29.9	240.9	15.2	21.3	43.9	219.0	83.3	887.1
`Frig`	39.4	1627.2	25.0	220.9	14.4	21.1	37.6	202.2	69.3	705.7
`Extreme-Horizon`	24.0	11 546.1	17.2	249.5	12.0	21.3	26.0	225.5	48.1	973.3
`ExaMilkyWay`	20.0	17 608.9	19.3	251.4	12.1	21.3	28.6	226.8	75.4	992.1
`Cosmic Dawn III`	83 275.2	97 213.1	254.5	254.7	21.3	21.3	228.7	228.8	1020.9	1022.7

TABLE 3.5 – Ratio de compression des différents algorithmes pour le tableau de raffinement (noté *Raff*), colonne de gauche, et le tableau d'appartenance (noté *App*), colonne de droite, pour les jeux de données testés.

Dans le tableau 3.6, on donne la vitesse de compression pour les deux tableaux pour les 5 jeux de données utilisés. On notera que les tests pour les simulations `Extreme-Horizon` et `Cosmic Dawn III` ont été effectués sur du matériel différent (les données de ces simulations étant particulièrement volumineuses et donc difficilement déplaçables), ce qui peut impacter les vitesses de compression. La taille des paquets consécutifs impacte également la vitesse de compression pour toutes les bibliothèques. On notera que `LZ4` permet d'obtenir une vitesse nettement supérieure à toutes les autres, dépassant 10 Go/s pour le tableau d'appartenance. L'algorithme `CPS52` permet d'avoir une vitesse de l'ordre de 1 Go/s à 2 Go/s en fonction du tableau. Comme attendu, la `Zlib` à une vitesse moins bonne que `CPS52`, `LZ4` et `Snappy`. Finalement, le niveau 9, bien que permettant d'obtenir le meilleur ratio de compression sur le tableau de raffinement, ne dépasse pas 30 Mo/s en dehors du cas particulier de `Cosmic Dawn III`. Il est important de rappeler qu'avec `RAMSES`, les tableaux, par processus, sont relativement petits. En conséquence, le surcoût en temps de calcul de l'ajout de la compression est négligeable. En effet, les tableaux de raffinement/appartenance des plus grands domaines possédant 3 à 4 millions de cellules sont compressés en quelques millisecondes.

Comme on l'a mentionné précédemment, les données vont être post-traitées par domaine, il est donc intéressant de regarder les performances par domaine, voir chapitre 4. On donne sur la figure 3.14, les courbes pour

Simulation	CPS52		LZ4		snappy		Zlib (1)		Zlib (9)	
	Raff	App	Raff	App	Raff	App	Raff	App	Raff	App
`Orion`	1468.7	1952.6	5835.1	14 335.1	4029.8	3660.8	455.6	640.5	21.3	277.4
`Frig`	1321.2	1861.0	4894.7	12 521.0	3660.8	9119.2	490.0	726.8	18.6	264.7
`Extreme-Horizon`	765.7	1947.4	2102.7	10 301.6	1579.7	3437.2	230.9	351.4	7.3	177.3
`ExaMilkyWay`	1240.7	2103.3	5302.1	11 456.8	4210.2	5112.5	465.4	728.8	22.1	277.0
`Cosmic Dawn III`	2014.2	2054.6	10 260.4	11 212.1	3285.4	3359.4	279.7	291.0	167.0	172.0

TABLE 3.6 – Vitesse de compression en Mo/s des différents algorithmes pour le tableau de raffinement (noté *Raff*), colonne de gauche, et le tableau d'appartenance (noté *App*), colonne de droite, pour les jeux de données testés. Matériel pour `Orion`, `Frig`, `ExaMilkyWay` : `CPU AMD` Ryzen 7 @ 3.60 GHz, `Extreme-Horizon`, `Cosmic Dawn III` : `CPU Intel` Xeon Gold 5118 @ 2.30 GHz.

la simulation `Orion`, du ratio de compression du tableau de raffinement et de la vitesse de compression en Mo/s. On obtient les mêmes résultats que ceux énoncés précédemment. L'algorithme `CPS52` permet d'obtenir un ratio similaire à celui de la `Zlib` au niveau 1, avec une vitesse en moyenne 2 fois plus élevée. De plus, `CPS52` est meilleur en terme de ratio sur tous les domaines, comparé à `LZ4` et `Snappy`, tout en maintenant une vitesse de compression de l'ordre de $\sim 1\ Go/s$. Enfin, comme mentionné précédemment, la `Zlib` niveau 9, permet d'obtenir le meilleur ratio sur tous les domaines à l'exception de quelques domaines ponctuels au prix d'une vitesse de compression réduites de deux ordres de grandeurs par rapport au `CPS52`. Finalement, l'algorithme que l'on propose permet de faire un bon compromis entre vitesse de compression et ratio, aussi bien domaine par domaine, que tout domaine confondu.

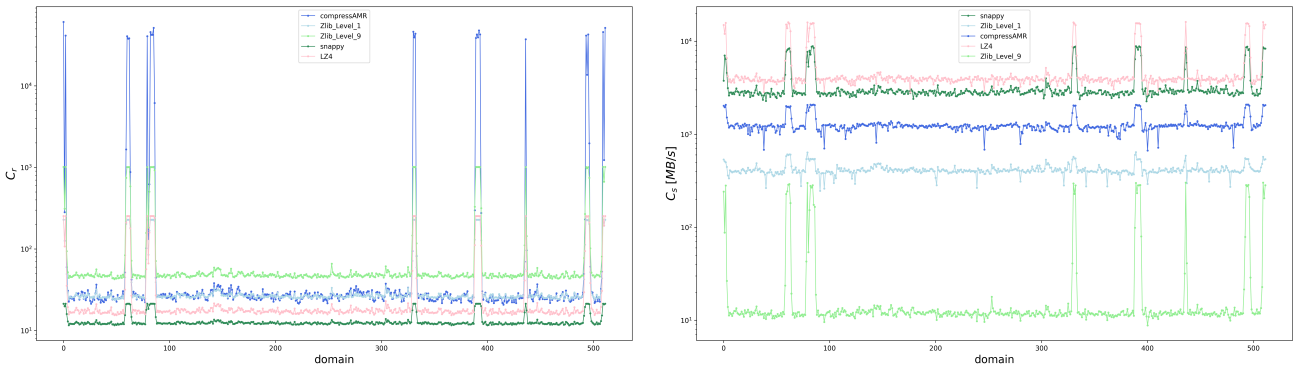

FIGURE 3.14 – Ratio de compression C_r, par domaine, pour les données de la structure `AMR` de la simulation `Orion` à gauche, et la vitesse de compression C_s à droite.

Il est intéressant de regarder l'entropie de Shannon par domaine, sur la figure 3.15. Pour le calcul de l'entropie, on utilise une longueur de symbole égale à un octet puisque le champ non compressé est stocké sur 1 octet. Naturellement, l'entropie du champ non compressé est à de l'ordre de 1 bit, ce qui est normal puisque les données sont binaires : 0 ou 1. Néanmoins, sur le champ compressé, elle est variable en fonction des bibliothèques. On constate qu'elle est assez élevée pour les deux niveaux de la `Zlib`. En dehors des domaines où le nombre de 0/1 consécutifs est très grand, l'entropie est proche de 8, on peut donc conclure qu'a priori ils exploitent l'ensemble des valeurs disponibles sur un octet. Une explication possible pour les creux d'entropie sur l'encodage de certains domaines est que les différentes bibliothèques découpent les grands paquets de 0/1 successifs en plus petits paquets avec le même encodage et donc un large pic dans les probabilités d'apparition de certains symboles. Pour le `CPS52`, l'entropie est plus constante avec une moyenne proche de 3.5, soit en moyenne 4 bits nécessaires par symbole (octet). C'est un résultat attendu puis que l'encodage utilisé exploite uniquement les nombres entre 0 et 64. On pourrait donc exploiter cette propriété pour utiliser un encodage sur 6 bits et donc un gain de 25% (au prix d'un encodage plus coûteux et donc d'une vitesse de compression réduite). Néanmoins, puisque l'entropie donne en moyenne 4 bits, cela signifie que la distribution des symboles (des nombres de 0 à 64) n'est pas homogène. On pourrait donc utiliser un encodage entropique, tel que l'encodage de Huffman, afin d'obtenir encore un gain moyen de l'ordre de 20% (au prix encore une fois d'une réduction de la vitesse de compression).

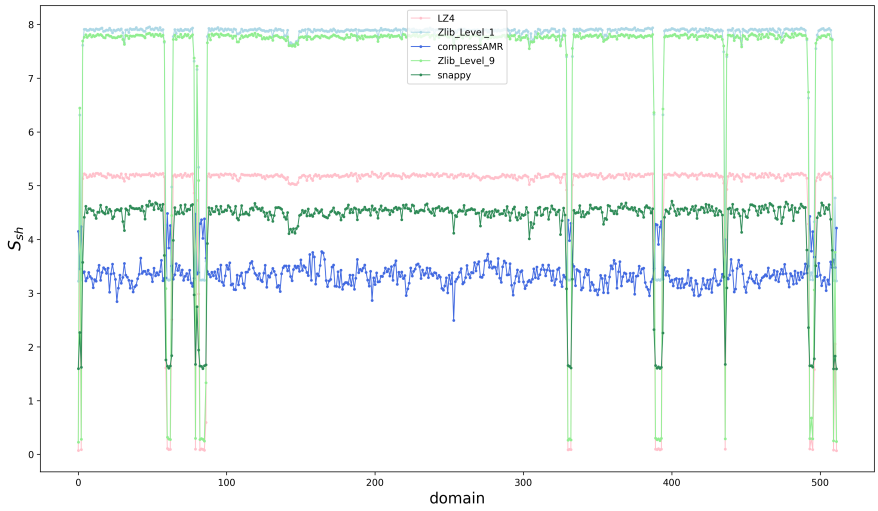

FIGURE 3.15 – Entropie de Shannon par domaine pour le tableau de raffinement compressé en fonction des différentes bibliothèques pour la simulation `Orion`.

Pour conclure, on résume dans le tableau 3.7 le volume des données de description du maillage `AMR` non compressé et compressé avec le `CPS52` pour chaque jeu de données. Même si le ratio de compression peut être amélioré d'environ ~ 40 %, on constate que le volume occupé par la description du maillage a été significativement réduit. Comme on le mettra en avant à la section suivante, ce volume, une fois compressé, ne représente plus qu'une partie négligeable du volume total d'un *snapshot* `RAMSES` avec le modèle `lightAMR`.

Simulation	Non compressé (Mo)	CPS-52 (Mo)	C_r
ORION	625.47	6.58	95.51
FRIG	1507.35	19.61	76.86
ExH	40 006.60	836.92	47.80
ExaMilkyway	71 854.50	1799.47	39.93
Cosmic Dawn III	1 198 380	13.36	89 699.10

TABLE 3.7 – Volume de données que représente la description du maillage (hors méta-données) pour les différents jeux de données avec et sans compression CPS-52.

3.8.5 Compression `PCP` *sans perte*

Il existe de nombreuses bibliothèques pour la compression de données scientifiques, la plupart proposant d'être soit *sans perte*, soit implémentant une méthode *avec perte*. Parmi les plus utilisées on retrouve donc `TTresh` (*avec perte*), [Ballester-Ripoll et al., 2018] plutôt spécialisée dans les données sur grille régulière, `FPZIP`, (principalement *sans perte*) [Lindstrom and Isenburg, 2006], pour les données `2D` / `3D` avec une corrélation spatiale forte et de faibles variations, `Isabela` (*avec perte*), [Lakshminarasimhan et al., 2011], [Lakshminarasimhan et al., 2013], utilisant des `B-spline` pour l'interpolation, `ZFP`, [Lindstrom, 2014], (*avec ou sans perte*) utilisant une compression par transformation de petit blocs, `FPC`, [Burtscher and Ratanaworabhan, 2007], prédiction, `SZ3`, [Cappello et al., 2019], se base sur une méthode par prédiction par interpolation. Enfin, il est important de noter que l'engouement des méthodes d'apprentissage, depuis ces dernières années, a fait émerger des approches prometteuses de compression à base d'auto-encoder, qui permettent d'atteindre des rations jusqu'à 50 fois meilleurs que ceux obtenus avec `ZFP` par exemple, sur des données scientifiques réelles (hors image), [Liu et al., 2023].

Pour le choix des bibliothèques, pour la compression *sans perte*, on se focalise sur les bibliothèques les plus utilisées à savoir `ZFP`, `FPC`, et `FPZIP`. On notera que l'efficacité de `ZFP`, dans la compression de données scientifiques a été éprouvée dans plusieurs études dans des domaines scientifiques différents : [Lindstrom, 2014],

[Baker et al., 2016], [Tao et al., 2019], [Jin et al., 2020]. Il est important de souligner également que l'approche de compression par bloc indépendant de `ZFP` permet de l'utiliser sur `GPU`, [Jin et al., 2020]. On inclura également des bibliothèques se basant sur le principe `LZ77` mentionné précédemment, à savoir `Zlib` et `LZ4`, massivement utilisées pour la production d'archive. Récemment, la bibliothèque `Zstandard` s'est montré efficace dans le cadre de la compression de données pour le *Big Data* capable d'atteindre de bon taux de compression (comparé à la `Zlib` et `LZ4` notamment) tout en maintenant une vitesses de compression élevée ($\sim Go/s$), on l'intégrera également dans les tests. On rapportera donc les performances en terme de ratio et de vitesse de compression pour ces différentes bibliothèques. Afin de limiter le nombre de figures et de tableaux dans ce manuscrit, on ne montrera pas tous les résultats avec toute la combinatoire possible de paramètres sur tous les champs de données. On fait une sélection des résultats pertinents à montrer et à discuter. Il est capital de rappeler que dans le cas du `lightAMR`, les tableaux scalaires sont `1D`, ce qui les rend plus difficile à compresser pour `ZFP` et `FPZIP`, toutes les deux optimisées pour les données multidimensionnelles. On notera, dans le cas de `FPZIP`, que la prédiction se fait via un prédicteur de Lorenzo, [Ibarria et al., 2003], qui se base sur les valeurs adjacentes. En `1D`, ce prédicteur se dégrade en simple *last-value* (`LV`) c'est-à-dire que la valeur x_{i-1} est utilisée pour prédire la valeur x_i. Les trois autres bibliothèques, également très largement répandues, sont basées sur le principe `LZ77` mentionné précédemment.

On commence par donner sur la figure 3.16, un aperçu de la qualité du prédicteur `LV` sur une partie des données d'un domaine de la simulation `Orion` pour le champ de densité. Sur la courbe, on retrouve en orange le lissage par paquet de cellules du prédicteur (valeur de la cellule parent) utilisé par le `PCP`, et en vert le prédicteur de Lorenzo (`LP`) `1D`. Par construction, le `PCP` suit les variations du champ lorsqu'on change de paquet (bloc de 8 cellules enfants), contrairement au `LP`, comme on peut le voir notamment pour la prédiction lors du changement de paquet. Sur l'ensemble du domaine, l'erreur relative moyenne du `PCP` est de -2.22 avec une déviation de 155.74 et de -22.49 avec une déviation de 3363.46 pour le `LP`. Bien qu'il s'agisse d'un résultat attendu, on verra par la suite que la bibliothèque `FPZIP` permet d'obtenir un ratio de compression équivalent, voir très légèrement supérieur au `PCP`.

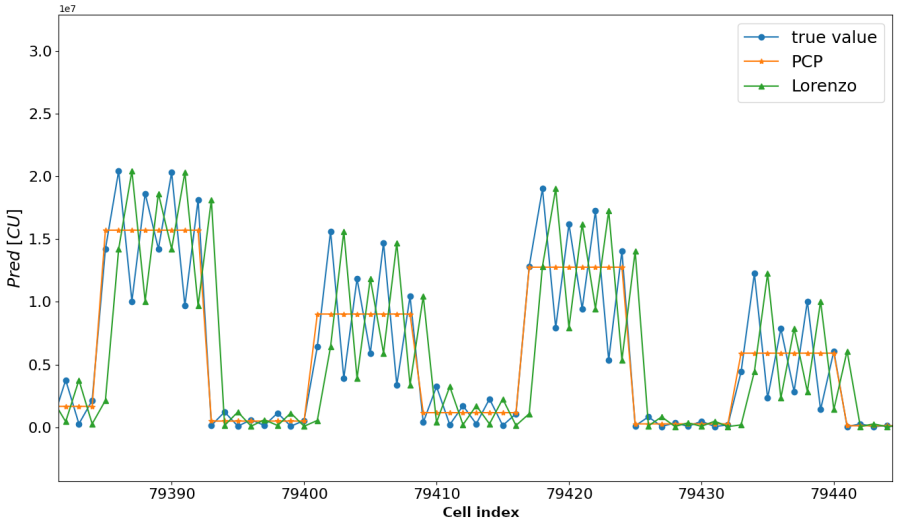

FIGURE 3.16 – Différence entre la fonction de prédiction de Lorenzo `1D` et la fonction du `PCP`. Les points bleus représentent la valeur réelle des cellules (79382 à 79444) du champ de densité d'un domaine de la simulation `Orion` en unité code (`CU`). Les points verts et oranges correspondent respectivement à la prédiction de Lorenzo `1D` et à la prédiction `PCP`.

Quelle que soit la version de l'algorithme `PCP` utilisé, *sans perte* ou *avec perte*, la première partie étape est la même : la suppression du nombre de bits de poids fort. On commence donc par montrer et discuter les résultats autour de la distribution par cellule et par paquet du nombre de bits de poids fort retirable, ce qui permettra de justifier des paramètres par défaut utilisés pour l'encodage. Les distributions normalisées du nombre de bits de poids fort retirables par parquet de 2^{dim} cellules et par cellule individuelle sont données sur la figure 3.17, pour le champ de densité en double précision sur la simulation `Orion`, tout domaine confondu. Le champ de densité

dans les tableaux décrits en `lightAMR` a un intervalle de valeur de $[4.0 \cdot 10^{-1}, 8.0 \cdot 10^{10}]$ en unité code. Sur cette figure, la barre en rouge indique le seuil imposé par l'encodage sur 4 bits (15 zéros maximum). En conséquence, on sature à 15 zéros toute valeur supérieure, et la compression sera donc sous-optimale. On constate que la moitié de la distribution du nombre de `LZ` par cellule individuelle se trouve au-delà de la valeur seuil. Lorsqu'on lisse par paquet le nombre de `LZ` la distribution est décalée vers la gauche et la valeur seuil englobe alors une bonne partie des données. Il est important de noter que dans la norme `IEEE` de l'encodage d'une grandeur double précision, le signe et l'exposant sont encodés sur les 12 premiers bits (9 en simple précision). On constate donc qu'il est possible de retirer jusqu'à l'intégralité de l'exposant dans la majorité des cas (pour ce champ, dans cette simulation) et donc que la valeur du prédicteur est une bonne approximation de l'ordre de grandeur. C'est un résultat attendu puisque cela signifie que la valeur d'une cellule parent est du même ordre de grandeur que celles de ses cellules enfants. C'est une conséquence directe de l'utilisation de l'`AMR` qui constitue donc une base essentielle qui permet de justifier de la pertinence de l'algorithme `PCP` et du choix de la fonction de prédiction.

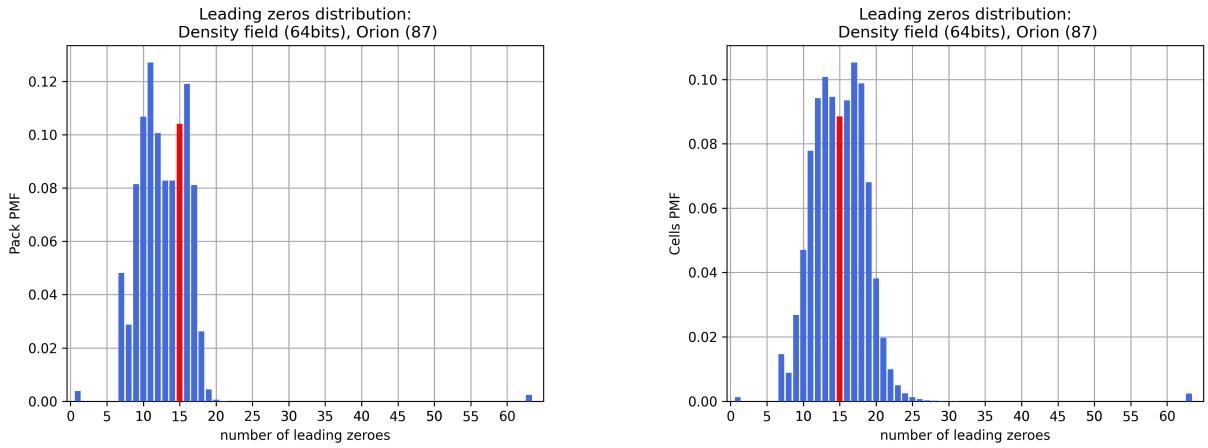

FIGURE 3.17 – Distribution du nombre de bits de poids fort retirables par paquet, à gauche, et par cellule individuelle, à droite, pour le champ de densité en double précision de la simulation `Orion`. En rouge, la valeur seuil par défaut (15 bits).

Néanmoins, on pourrait se demander si un encodage par cellule, avec un seuil plus élevé, par exemple sur 5 bits pour pouvoir retirer jusqu'à 31 zéros, ne permettrait pas d'obtenir un meilleur ratio de compression, puisqu'on intégrerait quasiment toute la distribution. Sur le tableau 3.8, on affiche les informations T_{lzrm}, C_r^* et C_r représentant respectivement, la fraction totale du nombre de bits retirables, le ratio de compression *sans* encodage du nombre de bits retirés et le ratio *avec* encodage du nombre de bits. Le ratio C_r^* est donné à titre indicatif, en pratique le nombre de bits retirés est une donnée *nécessaire* pour la décompression et doit donc être encodée. On le donne ici afin de mettre en avant l'impact de l'encodage du `LZ` sur le ratio. On donne ces informations par paquet de (2^3) cellules (simulation `3D`) et par cellule individuelle. On constate naturellement que T_{lzrm} est plus élevée dans le cas individuel et augmente avec le seuil, de même que le ratio de compression C_r^* augmente. Finalement, le ratio de compression réel C_r n'augmente pas systématiquement avec la valeur seuil. Dans le cas présent, on constate que le meilleur compromis (en gras) serait d'utiliser un encodage sur 5 bits. Cependant, le ratio $C_r^*(C)$ par cellule est nettement plus élevé, 1.303 contre 1.248 avec cet encodage. Mais lorsqu'on rajoute l'encodage du `LZ`, à savoir 4, 5 ou 6 bits par paquet de 8 cellules ou par cellule, le ratio de compression final est plus élevé avec un encodage par paquet. Malgré cela, il est important de noter que l'encodage utilisé est simple et ne tire pas profit de la distribution non homogène du nombre de `LZ`. Une piste d'amélioration de l'algorithme serait donc de regarder l'apport d'un encodage entropique qui pourrait permettre d'améliorer le ratio de compression et favoriser un encodage par cellule individuelle (il faudra également estimer le coût sur la vitesse de compression).

Si on réduit le champ en simple précision, en plus du gain d'un facteur 2 sur le stockage, il y a un impact sur ces distributions. En effet, sur la figure 3.18, on constate qu'elles sont déplacées vers la gauche (vers les plus petits nombres de bits retirables). C'est un résultat attendu parce que la norme `IEEE` définit le signe et l'exposant d'une grandeur simple précision sur les 9 premier bits. Et comme on l'a vu précédemment, on retire principalement l'exposant puisque l'ordre de grandeur est à peu près conservé entre une cellule enfant et sa cellule parent. On arrive donc plus rapidement sur les bits de la mantisse où il n'y pas de raison particulière d'avoir beaucoup de similitudes.

Encodage	T_{lzrm} (P) %	C_r^* (P)	C_r (P)	T_{lzrm} (C) %	C_r^* (C)	C_r (C)
4 bits (15)	19.19	1.237	1.223	21.12	1.268	1.175
5 bits (31)	19.84	1.248	**1.233**	23.27	1.303	**1.183**
6 bits (63)	19.95	1.249	1.231	23.39	1.305	1.163

TABLE 3.8 – Données cumulées sur tous les domaines, pour le champ de densité (64 bits) d' `Orion` pour différents encodages, du nombre de bits retirables en pourcentage T_{lzrm}, du ratio de compression *avec* (C_r) et *sans* (C_r^*) encodage du nombre de bits retirés, en regroupant par paquets (P) ou par cellules (C).

En effet, plus on se déplace vers les bits de poids faibles de la mantisse, plus la valeur se précise. Naturellement, on peut donc envisager de pouvoir encore retirer quelques bits de poids fort de la mantisse, mais on s'attend à ce que la distribution s'atténue rapidement. C'est ce qu'on constate ici, ainsi que sur les graphiques précédents. De même, les ratios de compression réels C_r sont plus élevés sur un champ simple précision que sur un champ double précision. En effet, retirer le signe et l'exposant d'un double signifie retirer $12/64 \simeq 18\%$ alors que sur une variable en simple précision, la fraction retirée est plus grande $9/32 \simeq 28\%$. En conséquence, réduire en simple précision les champs scalaires permet d'apporter un gain significatif en terme de stockage mais également d'améliorer dans notre cas le ratio de compression. On recommandera donc aux utilisateurs de définir les besoins en terme de précision pour leurs analyses scientifiques et, si cela est acceptable, de favoriser la réduction de précision. La même étude sur différents champs des simulations montre qu'en simple précision, un encodage sur 4 bits est suffisant.

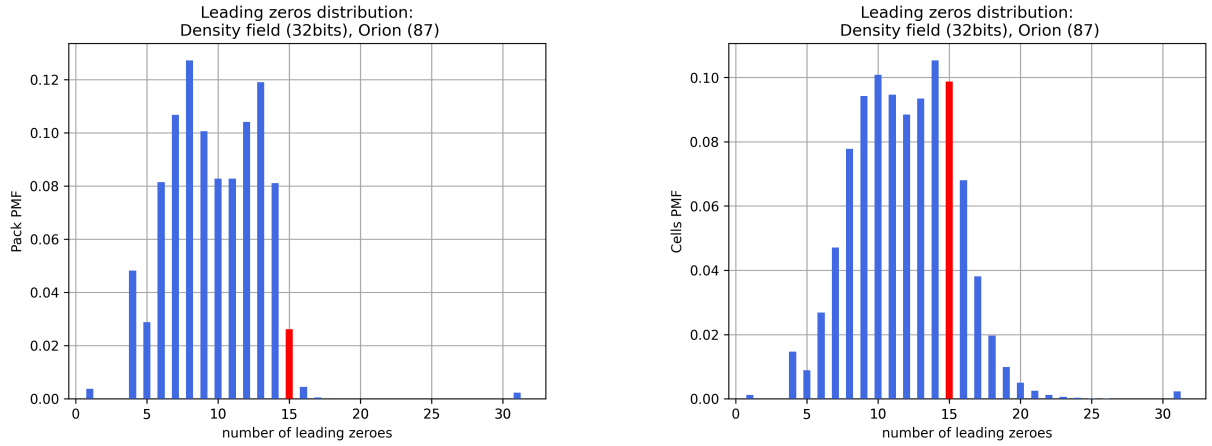

FIGURE 3.18 – Distribution du nombre de bits de poids fort retirables par paquet, à gauche, et par cellule individuelle, à droite, pour le champ de densité en simple précision de la simulation `Orion`. En rouge, la valeur seuil par défaut (15 bits).

Encodage	T_{lzrm} (P) %	C_r^* (P)	C_r (P)	T_{lzrm} (C) %	C_r^* (C)	C_r (C)
4 bits (15)	30.23	1.433	1.402	36.14	1.566	1.309
5 bits (31)	30.36	1.436	1.397	37.20	1.592	1.275

TABLE 3.9 – Données cumulées sur tous les domaines, pour le champ de densité (32 bits) d' `Orion` pour différents encodages, du nombre de bits retirables en pourcentage T_{lzrm}, du ratio de compression *avec* (C_r) et *sans* (C_r^*) encodage du nombre de bits retirés, en regroupant par paquets (P) ou par cellules (C).

Avec les résultats sur les champs en simple et en double précision précédents, on choisit un encodage par défaut sur 4 bits. On notera bien que l'encodage sur 5 bits en double précision apporte le meilleur ratio, mais ce résultat peut varier en fonction des champs et des simulations. Après des tests sur différents champs sur les jeux de données, on choisit donc de définir un encodage sur 4 bits, le gain du passage à 5 bits étant souvent marginal. De plus, limiter à 15 bits retirés par défaut permet d'éviter les problèmes de conflit avec l'approche *avec perte*. On rappelle qu'il faut faire attention à toujours pouvoir décrire une valeur avec au minimum 1 bit. Cependant, j'ai développé une version *multi-pass* de l'algorithme qui permet d'adapter automatiquement l'encodage en fonction du meilleur ratio de compression. On notera que cette approche permet également de pouvoir calculer des statistiques

et donc (dans une prochaine version de la bibliothèque) de pouvoir tester un encodage entropique. Le surcoût sur la vitesse de compression (en *mono-thread*) de cette approche *multi-pass* (hors encodage entropique) est d'environ 10% à 15%. (Il s'agit d'un travail en cours et ne sera donc pas inclus dans les tests).

On donne dans le tableau 3.10 les ratios de compressions pour le champ de densité, une composante du champ de vitesse et la pression thermique, en double précision, pour les différents algorithmes testés sur les jeux de données. Dans le tableau 3.11, on donne les vitesses de compression associées et le tableau 3.12, les vitesses de décompression. Tout d'abord, on constate que le meilleur ratio est obtenu par la bibliothèque `FPZIP` avec le prédicteur de Lorenzo `1D` légèrement meilleur de ~ 2 % à ~ 3 % que celui du `PCP4` et `PCP5`. Il est très intéressant de constater qu'un prédicteur aussi "universel" que le `LV` soit aussi performant que le prédicteur spécifique du `PCP` compte tenu de l'agencement des données au modèle `lightAMR`. Néanmoins, il est important de noter que `FPZIP` bénéficie d'un encodage entropique. On peut donc supposer que la différence de ratio soit principalement dûe à cet encodage plus qu'au prédicteur et donc qu'il serait possible d'améliorer le ratio de compression du `PCP` avec ce type d'approche. Néanmoins, on constate que sur la vitesse de compression, cet encodage plus riche a un coût significatif puisque le `PCP4` ou `PCP5` permet d'obtenir une vitesse de compression supérieure à 2 Go/s, alors que `FPZIP` atteint seulement ~ 35 Mo/s. On notera d'ailleurs qu'il n'y a pas de surcoût, en terme de vitesse, à utiliser un encodage sur 5 bits (`PCP5`) plutôt qu'un encodage sur 4 bits. L'algorithme de `FPC` est en moyenne 10 % moins bon que `FPZIP` ou les `PCP`, mais reste meilleur que la `ZFP` en étant 10 fois plus rapide.

Les algorithmes basés sur `LZ77` sont légèrement moins bon que la `ZFP` qui est pourtant spécialisée dans les données scientifiques. Le faible ratio de compression de la `ZFP` s'explique par le schéma de transformation utilisé, qui se base sur la décomposition en coefficient des données découpées en blocs indépendant de taille 4^d, où d est la dimension du tableau. Les variations entre les paquets de cellules, comme on l'a montré sur la figure 3.16, peuvent expliquer le faible ratio obtenu par cette bibliothèque. Il serait pertinent de modifier la taille des blocs et de les aligner avec les cellules pour évaluer les performances de l'approche de `ZFP`. Il est intéressant de noter que la `Zlib` et la `Zstandard` obtiennent des ratios similaires, mais offrent une vitesse de compression significativement différente, puisque `Zstandard` permet de compresser les données à une vitesse de l'ordre du ~ 1 Go/s, soit ~ 40 fois plus rapide que la `Zlib`. Pour les utilisateurs qui souhaitent utiliser des archives avec des données `RAMSES` (sans passer par une étape de compression au sein du code), on recommande donc de favoriser cet algorithme, disponible avec l'option `--zstd` pour la commande `Unix tar`. Un simple test sur 15 Go de fichiers `HDF5` (données `lightAMR` issues de `RAMSES` converties en `HDF5`) permet de confirmer l'intérêt en terme de vitesse de compression de la `Zstd`. En effet, il faut 2min20s à `tar` pour créer une archive de 14 Go avec l'option `--zstd` (commande complète : `tar --zstd -c -f hdf5.tar.zst hdf5/*`) et 11min33 avec la `Zlib` (avec l'option `cvzf`), pour une archive de 14 Go également. On notera que l'utilisation de `tar`, implique des `E/S`, d'où la différence par rapport aux benchmarks effectuées où la compression est faite depuis des données en mémoire vers des données en mémoire.

Algorithme	Orion			Frig			ExaMilkyWay			Extreme-Horizon		
	ρ	V_x	P_t	ρ	V_x	P_t	ρ	V_x	P_t	ρ	V_x	P_t
PCP4	1.226	1.219	1.224	1.191	1.223	1.212	1.280	1.279	1.284	1.225	1.258	1.169
PCP5	1.235	1.224	1.229	1.192	1.229	1.214	1.391	1.344	1.397	1.231	1.291	1.171
FPZIP	1.250	1.237	1.244	1.210	1.245	1.229	1.426	1.363	1.430	1.246	1.300	1.184
ZFP	1.095	1.095	1.093	1.077	1.099	1.089	1.159	1.140	1.161	1.098	1.120	1.051
FPC	1.133	1.126	1.128	1.091	1.132	1.107	1.278	1.233	1.286	1.124	1.181	1.067
Zlib 9	1.085	1.077	1.074	1.058	1.070	1.057	1.175	1.133	1.177	1.066	1.095	1.042
ZStd 20	1.085	1.079	1.081	1.059	1.072	1.064	1.271	1.174	1.274	1.068	1.110	1.041

TABLE 3.10 – Ratio de compression obtenu pour les différentes bibliothèques concernant le champ de densité, ρ, une composante du champ de vitesse, V_x et la pression thermique, P_t en **double** précision des différentes simulations.

En terme de vitesse de décompression, on constate comme attendu que l'algorithme `PCP4` (et `PCP5`) ont une vitesse similaire à la vitesse de compression, puisque l'algorithme est symétrique : c'est un résultat prévisible. On remarque particulièrement l'asymétrie de la `Zlib` qui permet de décompresser les données ~ 9 fois plus rapidement que lors de la compression. En effet, les algorithmes se basant sur `LZ77` doivent, durant la phase de compression, faire une recherche des occurrences, qui une opération coûteuse. `ZFP` et `FPC` sont toutes les deux environ deux fois plus lentes lors de la phase de décompression. D'autre part, contrairement à `ZFP` qui permet de compresser des

Algorithme	Orion			Frig			ExaMilkyWay			Extreme-Horizon		
	ρ	V_x	P_t	ρ	V_x	P_t	ρ	V_x	P_t	ρ	V_x	P_t
PCP4	2348	2256	2059	2236	2307	2049	2811	2638	2804	1105	1145	1149
PCP5	2213	2240	2076	2309	2277	2083	2413	2075	2119	1213	1176	1154
FPZIP	36	36	71	35	36	35	39	38	39	66	69	62
ZFP	141	140	138	140	141	138	143	144	143	72	72	70
FPC	1459	1461	1409	1434	1566	1432	1655	1612	1663	1061	1090	893
Zlib 9	27	28	34	30	28	32	39	37	38	28	31	27
ZStd 20	1007	1004	980	1084	1089	1075	1030	1029	1024	623	631	620

TABLE 3.11 – Vitesse de compression en Mo/s obtenue pour les différentes bibliothèques (en mono-thread) pour le champ de densité **double precision**, ρ, une composante du champ de vitesse, V_x et la pression thermique, P_t des différentes simulations. Les *benchmarks* sont effectués avec un AMD Ryzen 7 5700 X @ 3.6 GHz pour les simulations Orion, Frig, ExaMilkyWay, et un Intel Xeon Gold 5118 CPU @ 2.30GHz pour Extreme-Horizon

Algorithme	Orion			Frig			ExaMilkyWay			Extreme-Horizon		
	ρ	V_x	P_t	ρ	V_x	P_t	ρ	V_x	P_t	ρ	V_x	P_t
PCP4	2460	2390	2335	2366	2441	2158	2161	2043	2171	1219	1210	1193
PCP5	2310	2254	2301	2210	2345	2041	2033	2005	2014	1113	1074	1084
FPZIP	52	50	50	51	52	50	53	52	49	64	62	61
ZFP	81	78	77	79	78	77	81	80	79	49	46	47
FPC	601	610	573	563	614	569	581	603	610	426	453	417
Zlib 9	270	268	259	278	268	253	274	265	273	181	192	197
ZStd 20	950	900	938	1116	1024	1076	910	987	924	793	841	901

TABLE 3.12 – Vitesse de décompression en Mo/s obtenue pour les différentes bibliothèques (en mono-thread) pour les champs **double precision**, ρ, V_x et P_t des différentes simulations. Les *benchmarks* sont effectués avec un AMD Ryzen 7 5700 X @ 3.6 GHz pour les simulations Orion, Frig, ExaMilkyWay, et un Intel Xeon Gold 5118 CPU @ 2.30GHz pour Extreme-Horizon

blocs indépendamment et donc peut être facilement accéléré avec du multithreading (une version CUDA est même disponible), l'algorithme PCP est plus difficile à paralléliser. On notera également que l'on n'évalue pas la consommation mémoire des différents algorithmes. Néanmoins, l'algorithme PCP n'a besoin d'allouer que le tableau du champ compressé, contrairement par exemple à FPC qui utilise une table de hachage qui peut consommer énormément de mémoire, [Almeida et al., 2014].

Sur le tableau 3.13, on rapporte les ratios de compressions pour les champs en simple précision. Les ratios ne tiennent pas compte du facteur 2 de réduction. Pour toutes les bibliothèques testées, les ratios sont plus élevés, ce qui renforce l'intérêt de réduire les données en simple précision lors que cela est possible. On remarque également qu'en simple précision, c'est FPZIP qui permet d'obtenir le meilleur ratio. Il est important de préciser que compte tenu du système d'encodage de la prédiction utilisé par FPZIP, un changement de signe entre deux valeurs ne pénalise pas la compression, contrairement au PCP où un changement de signe entre la valeur du prédicteur et d'une seule des valeurs des cellules enfants impactera la compression de tout le bloc des cellules enfants. On notera que la Zlib et la Zstd permettent d'obtenir des ratios similaires, voire plus élevés que ZFP, particulièrement pour la simulation ExaMilkyWay. Dans cette simulation, comme on le verra au chapitre 6, la matière est concentrée dans une petite région de l'espace de simulation ($< 1\%$ en volume). Les méthodes avancées de recherche de la Zstd permettent donc visiblement d'obtenir de bons ratios dans les domaines "vides", comme on peut le voir sur la figure 3.19, augmentant ainsi le gain global donné dans cette table.

Il est intéressant de regarder les ratios de compression domaine par domaine, sur la simulation ExaMilkyWay par exemple. On donne sur la figure 3.19 le ratio de compression du champ de densité double précision pour les 850 premiers domaines. On constate que sur les ~ 350 premiers domaines, le PCP4 est quasiment constant avec $C_r \simeq 1.3$ (limite théorique de compression avec un encodage sur 4 bits), alors que le PCP5 à un ratio supérieur à 1.4. La Zstd (niveau 20) a un ratio meilleur que le PCP4 et même que le FPC. Au-delà du 350 ème domaine, on retrouve la tendance de le tableau 3.5. Dans cette simulation, l'objet astrophysique étudié est centré dans la boite de simulation et donc cette tendance du ratio de compression (très bruité à partir du domaine 350, puis "constant" et de nouveau

Algorithme	Orion			Frig			ExaMilkyWay			Extreme-Horizon		
	ρ	V_x	P_t	ρ	V_x	P_t	ρ	V_x	P_t	ρ	V_x	P_t
PCP4	1.402	1.380	1.384	1.301	1.393	1.349	1.717	1.661	1.723	1.394	1.537	1.345
PCP5	1.404	1.383	1.387	1.302	1.394	1.350	1.909	1.731	1.902	1.398	1.559	1.524
FPZIP	1.435	1.404	1.419	1.338	1.431	1.386	2.018	1.842	2.029	1.423	1.576	1.532
ZFP	1.124	1.125	1.121	1.088	1.133	1.112	1.269	1.224	1.274	1.128	1.178	1.269
FPC	1.139	1.126	1.126	1.081	1.140	1.119	1.428	1.412	1.512	1.134	1.246	1.201
Zlib 9	1.158	1.146	1.142	1.110	1.145	1.130	1.393	1.279	1.395	1.126	1.186	1.275
ZStd 20	1.156	1.144	1.147	1.108	1.144	1.137	1.456	1.311	1.449	1.130	1.197	1.279

TABLE 3.13 – Ratio de compression obtenu pour les différentes bibliothèques pour le champ de densité, ρ, une composante du champ de vitesse, V_x et la pression thermique, P_t en **simple** précision des différentes simulations.

bruité) est liée au découpage de Hilbert des domaines. Dans les zones "vides", le comportement des bibliothèques basé sur une méthode LZ77 est très bon et chute rapidement dès que les données deviennent moins homogènes, au profit de bibliothèques plus spécialisées. FPZIP et le PCP5 restent les deux meilleures méthodes indépendamment de la région.

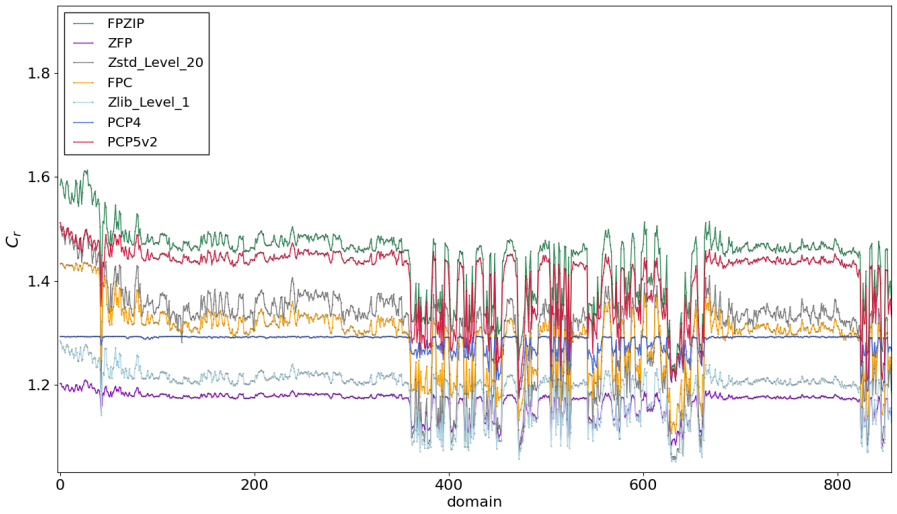

FIGURE 3.19 – Évolution du ratio de compression pour différentes bibliothèques pour les 850 premiers domaines de la simulation ExaMilkyWay.

La déviation standard du champ à compresser permet de refléter l'écart entre les valeurs, ainsi plus la déviation au sein des valeurs d'un domaine sera grande, plus son impact (potentiel) sur les ratios de compression des algorithmes sera grand. On met en évidence cette relation sur la figure 3.20, sur quelques domaines de la simulation ExaMilkyWay pour le champ de densité. On remarque clairement les pics du ratio lorsque la déviation diminue, et inversement. En fonction des données à compresser, cette propriété peut aider les utilisateurs à orienter le choix de l'algorithme à utiliser. Dans le cas du PCP il s'agit plutôt d'utiliser la version sur 5 bits (PCP5) (ou prochainement la version adaptative qui permettra de sélectionner automatiquement le meilleur encodage).

Compte tenu des volumes de données de la simulation Coda, la conversion s'est faite directement en une base HDep avec les champs compressés, en simple précision, avec l'algorithme PCP4 *avec perte*, $n_{ovrr} = 16$ soit une erreur absolue $\leq 3.9\ 10^{-3}$. On notera que la base de données fait 2.6 To, répartis sur 131072 domaines . On traite, lors du test, 1 domaine sur 64, ce qui fait un échantillon de 2048 jeux de données. Cette approche permet d'obtenir une approximation des ratios de compression parce que la matière est répartie dans l'ensemble de la boite de simulation. La moyenne d'une sous-partie, suffisamment grande, de la liste des domaines, est donc une bonne approximation. En revanche, cela donnerait un résultat significativement différent avec une simulation *zoom* comme

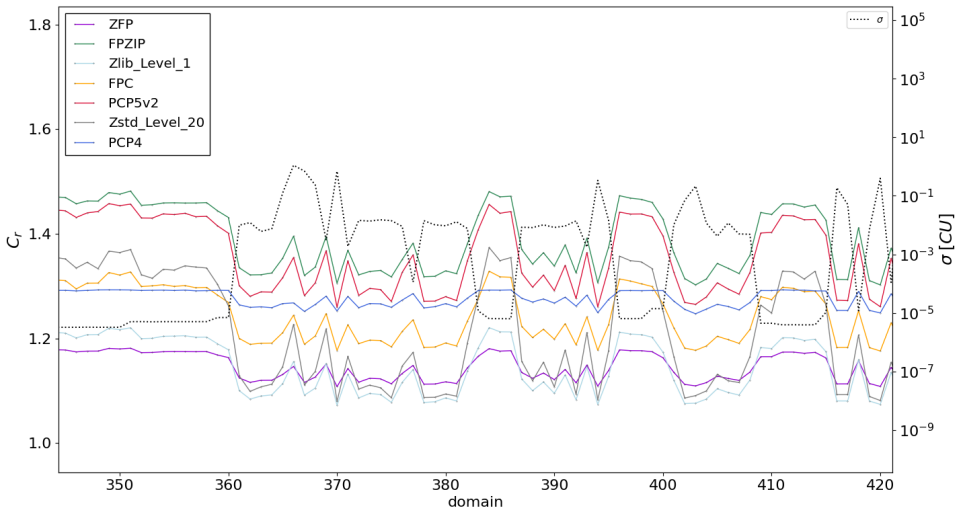

FIGURE 3.20 – Impact de la dispersion du champ scalaire (ici la densité) sur le ratio de compression pour les différentes bibliothèques testées. On présente ici quelques domaines de la simulation `ExaMilkyWay`.

`ExaMilkyWay`, à cause de l'inhomogénéité de l'information, comme on peut le voir sur la figure 3.19.

On rapporte dans le tableau 3.14 la valeur moyenne du ratio de compression $\bar{C}_r$, la déviation standard σ et le ratio cumulé C_r^* pour l'algorithme `PCP4`. On constate que pour le champ de densité, il n'y a pas d'intérêt a priori à utiliser le `PCP5` de même que pour la composante V_x du champ de vitesse, parce que le ratio moyen est à environ deux sigma de la limite théorique de 1.829. On peut donc s'attendre à un gain marginal sur quelques cellules avec `PCP5`. En revanche, la pression sature complètement le ratio théorique, il faudrait donc augmenter l'encodage. On notera qu'on ne fait pas de test avec le `PCP5` parce que les données ont été compressées avec $n_{ovrr} = 16$, donc l'algorithme serait faussé. En effet, le nombre de LZ (limité à 31) permettrait très souvent d'aller plus loin à cause des bits de poids faibles mis à 0. On notera que l'algorithme `PCP4`, s'arrête juste avant, laissant 1 bit entre les deux parties (*lossless* et *lossy*) (dans les cas limites). Enfin, on notera que cette simulation a été faite en grille fixe 8192^3, et même si le modèle `lightAMR` peut être utilisé ainsi que les algorithmes de compression, il existe d'autres modèles de données peut-être plus adaptés pour la description des données. De plus, si les données sont décrites sur une grille régulière, cela permettrait d'exploiter aux mieux les bibliothèques telles que `ZFP`, optimisée pour les données multidimensionnelles et ainsi améliorer les performances des compressions.

Algorithme	ρ			v_x			P_t		
	$\bar{C}_r$	σ	C_r^*	$\bar{C}_r$	σ	C_r^*	$\bar{C}_r$	σ	C_r^*
`PCP4`	1.402	0.017	1.401	1.636	0.114	1.628	1.829	0.001	1.829

TABLE 3.14 – Test de compression de l'algorithme `PCP` sur les données `Cosmic Dawn III` pour les différents champs. $\bar{C}_r$ est la valeur moyennée du ratio de compression sur les domaines traités, à savoir 1 domaine sur 64 (2048 valeurs) et σ sa dispersion. C_r^* est le ratio cumulé sur l'ensemble des domaines traités.

3.8.6 Compression *avec perte*

La compression sans perte ne permet pas d'atteindre des ratios élevés comme on a pu le constater sur les résultats de la section précédente. Le maximum obtenu est de $C_r = 2.03$ pour la bibliothèque `FPZIP` sur la simulation `ExaMilkyWay` sur les champs scalaire en simple précision. Les méthodes *avec perte* permettent d'améliorer le ratio de compression. On donne dans le tableau 3.15 la valeur du ratio de compression, cumulée sur tous les domaines, pour les champs scalaires en double précision de la simulation `Orion`. `PCP4` et `FPZIP` ont tous les deux une erreur

relative fixe indépendamment du champ scalaire, contrairement à ZFP où l'erreur peut varier de plusieurs ordres de grandeurs. En terme de ratio, FPZIP est comparable à PCP4 à une erreur similaire. L'encodage entropique de FPZIP ne permet donc pas un gain significatif. Compte tenu des ratios et des erreurs obtenus avec ZFP, il est plus intéressant de réduire le champ en simple précision et d'utiliser la compression sans perte. Enfin, les vitesses de compression (non données ici par souci de brièveté) sont similaires à celle du tableau 3.12, pour la compression sans perte.

Algorithme	ρ		v_x		P_t	
	C_r	Err_{max}	C_r	Err_{max}	C_r	Err_{max}
PCP4 $N_{ovrr} = 16$	1.767	7.28×10^{-12}	1.752	7.28×10^{-12}	1.763	7.28×10^{-12}
PCP4 $N_{ovrr} = 24$	2.268	1.86×10^{-9}	2.244	1.86×10^{-9}	2.622	1.86×10^{-9}
PCP4 $N_{ovrr} = 31$	3.016	2.38×10^{-7}	2.974	2.38×10^{-7}	3.005	2.38×10^{-7}
FPZIP $N_m = 34$	3.004	2.38×10^{-7}	2.928	2.38×10^{-7}	2.97	2.38×10^{-7}
FPZIP $N_m = 42$	2.186	9.31×10^{-10}	2.145	9.31×10^{-10}	2.168	9.31×10^{-10}
ZFP $N_m = 34$	1.872	4.97×10^{-4}	1.865	5.43×10^{-2}	1.864	7.6×10^{-4}
ZFP $N_m = 42$	1.517	3.08×10^{-6}	1.512	1.92×10^{-5}	1.512	4.97×10^{-7}

TABLE 3.15 – Tableau des ratios de compression pour les bibliothèques PCP, ZFP et FPZIP avec différentes configurations pour l'erreur autorisée, pour les champs scalaires en **double précision**. L'erreur Err_{max} correspond à l'erreur relative point à point maximale sur l'ensemble des jeux de données (tout domaine confondu).

Il est intéressant de regarder les ratios par domaine, cela permet de se rendre compte plus facilement de la variabilité de l'erreur. Les algorithmes PCP4 et FPZIP, sont quasiment constants par construction, alors que ZFP peut varier énormément en fonction des domaines. Néanmoins, peu importe la bibliothèque utilisée, il est impératif d'évaluer l'impact de la compression avec perte sur les analyses qui seront effectuées.

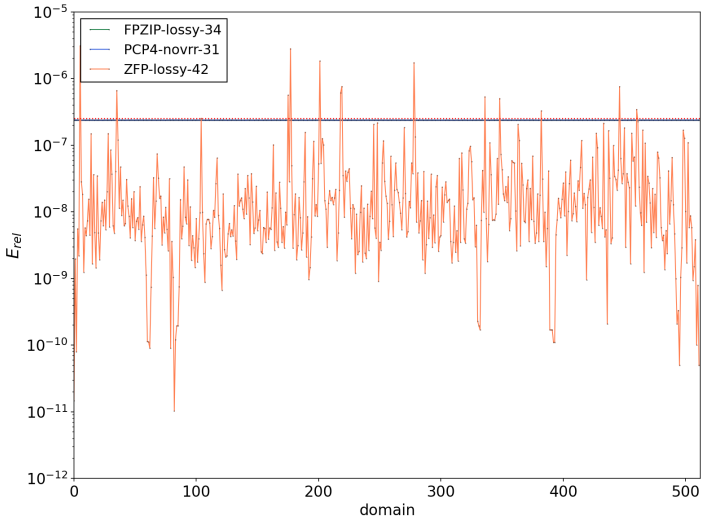

FIGURE 3.21 – Erreur de décompression pour les algorithmes de PCP, FPZIP et ZFP, sur le champ de densité double précision de la simulation Orion. L'erreur demandée est de $2\,10^{-9}$. FPZIP et PCP4 respectent parfaitement l'erreur. ZFP en orange, fluctue en-dessous et au-dessus de la limite.

En effet, la méthode d'arrondi systématique de FPZIP vers zéro peut conduire à une perte d'énergie qui peut introduire un biais dans les analyses de données, [Baker et al., 2016]. Si on prend un domaine quelconque de la simulation Orion, on peut tracer la distribution de l'erreur relative (point à point sur les valeurs du domaine) sur la figure 3.22, pour FPZIP et PCP4. La distribution du PCP4 est centrée autour de 0, comme on l'a mentionné précédemment et on rajoute $2^{N_{ovrr}-1}$ lors de la phase de décompression. FPZIP en revanche a une erreur point à point strictement positive. En fonction des analyses, il peut donc y avoir un impact sur les résultats.

Bien que le calcul du PSNR soit une métrique couramment utilisée, il est dépendant de l'intervalle de valeur. Sur une simulation, comme Orion par exemple, l'intervalle des valeurs de la densité varie entre $[0.387269, 8.00242e+$

FIGURE 3.22 – Distribution des erreurs lors de la décompression du champ de densité double précision, sur un domaine de la simulation `Orion`. A gauche, les erreurs avec l'algorithme `PCP` et à droite avec l'algorithme `FPZIP`.

10], soit 11 ordres de grandeur. En conséquence, les valeurs obtenues des `PSNR` sont élevées. En effet, les `PSNR` obtenus avec une erreur de compression avec `PCP4` de 2.38 10^{-7} sont de l'ordre de 154 en moyenne avec un déviation de standard de 3.1.

3.8.7 Réduction globale du volume de données

On présente ici des résultats globaux sur la réduction du volume de données pour les différentes simulations. Sur la figure 3.23, on quantifie la réduction des volumes de données pour la partie `AMR` et la partie contenant les grandeurs scalaires flottantes (`HYDRO`), pour un *snapshot* de la simulation `Extreme-Horizon`, en fonction des différents algorithmes. Les deux barres de gauche représentent respectivement, la description `AMR` de la version native, $\sim 500\,Go$, et la description avec le modèle `lightAMR`, $\sim 850 Mo$ (compression `CPS52`). Un premier facteur de l'ordre de 12 est gagné sur la description en arbre du maillage, puis la compression `CPS52` permet de gagner encore un facteur supplémentaire d'environ 40 sur le volume des données. Comme on peut le voir, la description du maillage ne représente désormais plus qu'une partie négligeable du volume total.

Les autres barres représentent les données des champs scalaires, en bleu foncé pour les données d'origine et en couleurs pour les données `lightAMR`. Les volumes d'origine représentent $\sim 2.3\,To$ en double précision (13 champs) et $\sim 1.2\,To$ en simple précision. Les barres en vert foncé, montre le gain apporté par l'élagage de l'arbre, $\sim 13\%$, sans compression et donc indépendant de la précision. On précise que les légères différences par rapport aux chiffres de le tableau 3.3 sont liés aux arrondis et aux légers surcoûts de certaines méta-données.

Les barres suivantes concernent les données après application de l'algorithme de compression `PCP4` *sans perte*, pour les barres en vert clair, et orange, orange foncé et rouge pour la version *avec perte*. On notera que les pourcentages affichés sont les réductions par rapport au volume d'origine et non par rapport à la barre précédente. Toutefois, tous les algorithmes de compression ont été appliqués après élagage de l'arbre. Ainsi, la compression *sans perte*, pour les grandeurs double précision, permet un gain de 29% du volume, il faut donc comprendre ici $\sim 13\%$ lié à l'élagage et $\sim 16\%$ lié à la compression. On teste différentes valeurs du paramètre N_{ovrr} pour la compression avec perte. Le gain pour les grandeurs doubles est de 11% par rapport à la compression avec perte, et augmente de 11% à chaque fois qu'on retire 8 bits de plus. Compte tenu de la méthode utilisée, ce taux est naturellement constant. Pour la version simple précision, on ne teste que les valeurs de N_{over} de 8 et 16. Au-delà de 16 bits, compte tenu d'un encodage du LZ sur 4 bits, on risquerait d'enlever plus de 31 bits, ce qui est interdit par la représentation utilisée. Finalement, le gain maximum est donc de 85% pour cette simulation pour l'`AMR` et l'`HYDRO`. On discutera au chapitre suivant de l'impact de ce type de configuration (*downcasting* + compression *avec perte*) sur les résultats d'analyses.

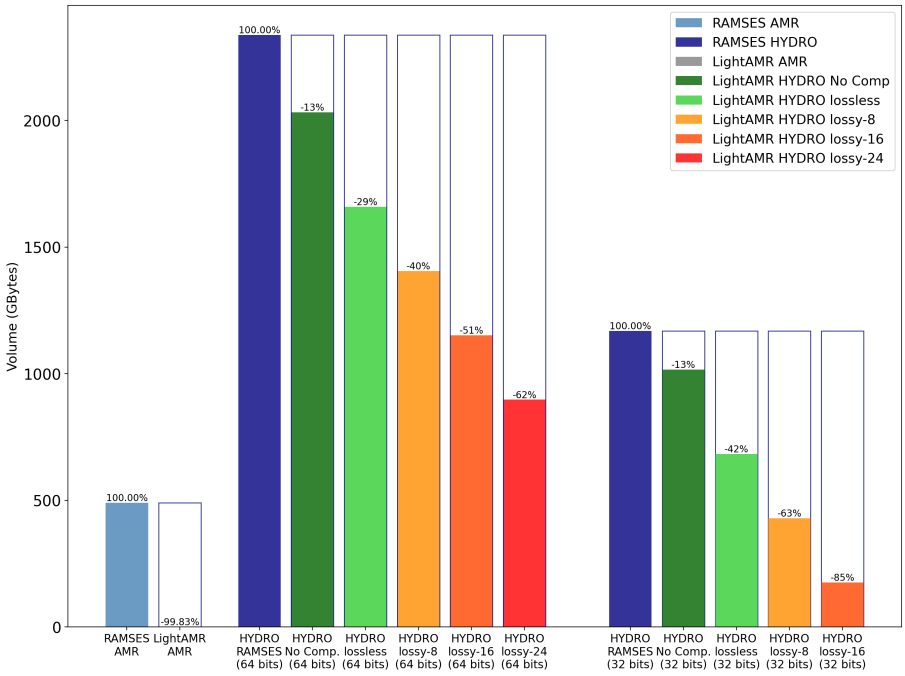

FIGURE 3.23 – Récapitulatif du volume de donnée `AMR` et `HYDRO` pour la simulation `Orion` en fonction de la précision et de la méthode de compression utilisée (*sans perte* ou *avec perte*) pour les données flottantes.

3.9 Conclusion et perspectives d'évolution

On a présenté un modèle de données très compact destiné à la description de maillages `AMR` en arbre, baptisé : `lightAMR`. Ce modèle permet de conserver l'information hiérarchique des cellules, de manière compacte en proposant une description sous la forme de tableaux booléens. L'ordonnancement des cellules se fait de haut en bas, niveau par niveau. Les données flottantes, associées aux cellules, doivent bien évidemment respecter le même schéma. On a également décrit un algorithme d'élagage de la description de l'arbre qui permet de retirer la redondance d'information en se basant sur l'appartenance des cellules au processus `MPI` courant. De plus, on a proposé un algorithme de compression du type *Run-Length-Encoding* pour compresser les données de description du maillage, ainsi qu'un algorithme compression des données scalaires flottantes par *Delta-Encoding*. Deux versions de ce dernier ont été décrites : *sans perte* et *avec perte*.

Grâce au développement d'un convertisseur de données, on a pu tester ce modèle en convertissant des données de différentes simulations effectuées avec le code `RAMSES`, de petites simulations de type *zoom*, à de grandes simulations cosmologiques. Ainsi, on a montré que l'élagage de l'arbre, dont l'efficacité varie en fonction des simulations, est une première étape importante pour réduire le volume de données. En effet, entre 11% et 48%, du volume total peut être retiré sur les simulations testées. L'exploration de la compression de données, et l'utilisation de notre algorithme `CPS52` a montré que désormais la description du maillage n'occupe plus qu'une partie négligeable du volume total de données, alors qu'avec la description d'origine, il pouvait occuper jusqu'à plus de 20%. Des benchmarks de l'algorithme `CPS52`, sur les données de simulation testées, ont montré qu'il permet d'obtenir de meilleurs ratios de compression que les bibliothèques `LZ4`, `ZLib` ou `Snappy` avec une vitesse de l'ordre de 1 Go/s.

Néanmoins, la compression `CPS52` utilise un encodage sous-optimal. Une évolution possible consiste donc à affiner l'encodage soit par l'utilisation de 6 bits au lieu de 8, offrant ainsi un gain de 20%, soit par le biais de l'utilisation d'un encodage entropique pour encoder les symboles les plus fréquents sur moins de bits que les symboles les plus rares. Le gain total estimé avec ces optimisations est de l'ordre de 40% à 50%.

Ensuite, avec l'algorithme de compression `PCP`, pour les données scalaires flottantes, on tire profit de la topologie du maillage hiérarchique pour mettre en place une compression par prédiction de valeur (*Delta* compression). La version *sans perte* de cette algorithme permet d'obtenir des ratios de compression modeste de l'ordre

1.12 - 1.23 en double précision, en fonction des champs scalaires et des simulations. On a mis en évidence le double gain en terme d'espace de stockage de la réduction de précision des données (vers du simple précision, 32 bits). En effet, en plus de facteur 2, les ratios de compression sont plus élevées, 1.3 - 1.9. On a comparé notre algorithme à des bibliothèques spécialisées dans la compression de données de simulation numériques : `ZFP`, `FPZIP`, `FPC`. Les benchmarks effectués sur différents champs scalaires ont montré que le ratio de compression et/ou la vitesse de compression de notre algorithme surclassent ces bibliothèques. On notera, cependant, que la fonction de prédiction de Lorenzo, utilisée notamment par `FPZIP`, permet d'obtenir un ratio de compression similaire ou très légèrement supérieur. Néanmoins, la vitesse de compression de cette bibliothèque, utilisant un encodage entropique, est jusqu'à 30 fois plus lente. En effet, le `PCP` permet d'obtenir des vitesses de compression comprise entre 1 Go/s et 3 Go/s avec une implémentation séquentielle.

La force de la fonction de prédiction utilisée par l'algorithme `PCP` est aussi sa faiblesse. En effet, bien que celle-ci soit parfaitement adaptée au modèle `lightAMR`, elle est spécifique à celui-ci et implique de fournir à l'algorithme de compression le tableau de description de l'arbre. Contrairement à la fonction de prédiction de Lorenzo (ici en `1D`) qui est universelle et permet d'obtenir des résultats presque équivalents. En effet, bien que cette fonction de prédiction présente une erreur moyenne plus élevée que la prédiction `PCP` et ne permette pas de suivre les variations des champs, l'erreur reste relativement faible. On suppose donc que la différence sur les ratios de compression, entre `PCP` et `FPZIP`, est liée principalement à la différence d'encodage. Il serait donc particulièrement intéressant de tester différentes méthodes d'encodage pour notre algorithme et notamment les méthodes d'encodage entropique. En effet, il y a un gain potentiel en terme de ratio dans l'approche cellule par cellule plutôt qu'avec un lissage par paquet des `LZ`. Cependant, la littérature montre qu'il est difficile d'atteindre des ratios de compression supérieur à 2 avec les méthodes sans perte.

L'extension *avec perte* de l'algorithme `PCP` se base sur une approche très simple, sans impact sur la vitesse de compression, qui consiste à supprimer des bits de poids faibles de la mantisse, similaire à la méthode utilisée par `FPZIP`. Contrairement aux algorithmes par décomposition ou transformation telle que l'approche utilisée par `ZFP`, cette méthode permet d'obtenir une erreur "point à point" (ou "valeur à valeur") bornée et prédictive, ce qui permet d'encourager l'utilisation de la compression avec perte auprès de utilisateur. De plus, contrairement à `FPZIP`, lors de la décompression du champ scalaire, l'erreur est centrée autour de 0, ce qui donne un atout considérable à notre approche. En effet, une erreur centrée permet d'éviter l'accumulation des erreurs lors des analyses à partir de champs décompressés, comme nous le verrons au chapitre 4. Les benchmarks de la compression *avec perte* ont montré la supériorité de notre approche en terme de vitesse et de ratio de compression par rapport aux bibliothèques de la littérature. Néanmoins, bien que cette version du `PCP`, permette d'atteindre des ratios de ~ 3 pour une erreur relative maximale de 10^{-7}, le gain n'est pas significatif. Il est important de noter que retirer des bits de poids faibles en plus des bits de poids fort permet de réduire drastiquement l'intervalle des valeurs possibles pour les bits restants. En conséquence, on s'attend à ce que l'apport d'un encodage plus complexe permette d'améliorer les ratios de compression, au prix d'une vitesse de compression réduite.

Enfin, on a discuté brièvement de la compatibilité de notre modèle avec la nouvelle structure de données de `VTK` dédiée à la visualisation de données `AMR` en arbre : les `HyperTreeGrid`. Pour le lecteur curieux de tester cette nouvelle structure, on donne en annexe A, un code `Python` complet de démonstration (non optimisé) qui permet de construire un `vtKHyperTreeGrid` à partir de données au modèle `lightAMR`. On prend comme exemple la grille `2D` de la figure 3.1. Le code produit un fichier *example_ 2D.htg* qu'il est possible d'ouvrir avec `Paraview` version >= 10. On notera que le code nécessite d'avoir les bibliothèques `Numpy` (version >= 1.24) et `VTK` (version >= 9.2). Elles peuvent être installées avec la commande `PIP`, e.g. `pip install vtk==9.2.6`, (on recommande l'utilisation d'un environnement virtuel).

Chapitre 4

Stratégies pour l'analyse des données

L'objectif d'une simulation numérique est de produire des données brutes à partir des modèles physiques implémentées. Le post-traitement vise à étudier ces modèles par l'analyse de ces données brutes. En conséquence, pour préparer le post-traitement, le formatage des données aux sein des fichiers et les modèles de données utilisés sont deux points essentiels pour faciliter le développement et la maintenance des outils d'analyse (cf. chapitres précédents). Par exemple, un code comme `RAMSES` qui peut produire des centaines de milliers de fichiers par répertoire, ne permet pas aux outils d'analyses d'atteindre de bonnes performances d'`E/S`. Grâce à l'intégration de `Hercule` et au modèle `lightAMR` ce problème est désormais résolu. Cependant, puisque le modèle de données a été changé, les outils existants capables d'analyser le format natif de `RAMSES`, tels que `Yt`, `Pymses`, etc., ne sont plus compatibles. De plus, lors de la mise en place du plan de gestion de données de la simulation `ExaMilkyWay` (voir chapitre 6, il était clair que malgré les réductions significatives apportées par le modèle `lightAMR` et la compression de données, la quantité d'information à analyser resterait très importante, de l'ordre de plusieurs dizaines de milliards de cellules. Pour analyser cette grande quantité d'information, des algorithmes de filtrage géométrique (région d'intérêt) ne suffisent pas. On se tourne donc naturellement vers du `High Performance Data Analysis` (`HPDA`). L'objectif de ce chapitre est donc d'expliquer la stratégie et les développements que j'ai mis en place pour pouvoir analyser les grands volumes de données produits par la simulation `ExaMilkyWay`, à savoir environ 25 To de données cumulées ($>$ 40 milliards de cellules par analyse).

4.1 Post-traitement *In-Situ* ou *In-Transit*, une réelle nécessité ?

Avant de détailler la stratégie et l'outil d'analyse, il est important de rappeler les trois grandes approches pour analyser les données provenant d'un code de simulation : *post-mortem*, *in-situ* et *in-transit*. La méthode d'analyse *post-mortem* est l'approche classique (qu'on utilisera ici) qui consiste à analyser les données **après** leur écriture sur disque. On notera que la simulation peut continuer de tourner. Avec cette approche, on retrouve les problématiques d'`E/S` qu'on a évoqué précédemment, à savoir : vitesse d'écriture et espace stockage. Les approches *in-situ* et *in-transit* ont été proposées dans l'objectif de pallier les problèmes des `E/S` de manière générale, [Ross et al., 2008]. Dans la première approche, le code suspend son exécution, et donc l'avancement de la simulation, afin de déclencher une étape d'analyse. On profite ainsi d'avoir toutes les données en mémoire et des ressources de calculs pour faire les analyses sans avoir besoin d'effectuer des opérations d'`E/S`, ou de transférer les données à travers le réseau. Cette méthode est relativement simple à mettre en oeuvre mais implique d'implémenter directement les analyses dans l'application, [Ohno and Kageyama, 2021]. Il est essentiel de noter que les algorithmes d'analyses sont généralement exploratoires et souffrent assez souvent d'une moins bonne scalabilité que l'application elle-même, en conséquence une étape d'analyse *in-situ* peut perturber la bonne scalabilité d'une application et même devenir un point chaud en terme de temps de calcul, [Friesen et al., 2016]. A l'opposé, il y a l'approche *in-transit* qui transfère les données sur le réseau et permet ainsi à la simulation de poursuivre son exécution de manière asynchrone. Cette approche est plus compliquée à mettre en place, [Sewell et al., 2015], et peut nécessiter de séparer les ressources de calculs en deux groupe distincts : un groupe pour la simulation et un autre, généralement plus petit, pour l'analyse de données, [Bennett et al., 2012]. On notera également que cette approche peut rapidement devenir problématique pour les grandes simulations vis-à-vis du volume de données à transférer à travers le réseau. Il y a un également des problèmes potentiels d'équilibrage au sein des ressources d'analyse. En outre, les ressources doivent être définies et allouées au lancement de la simulation, et sont donc comptabilisées dans le budget horaire de l'utilisateur même si

leur utilisation n'est que périodique.

De plus, la plus-value de l'analyse *à la volée*, qu'elle soit *in-transit* ou *in-situ* est parfois limitée et sert principalement l'intérêt du *monitoring* de la simulation plus que de l'analyse stricto sensu. Il faut également garder à l'esprit, que lorsqu'on soumet une requête d'exécution de job sur un centre de calcul, celui-ci passe par une queue de soumission avant d'être exécuté. Pour faire de l'analyse *à la volée*, il faut donc être devant un ordinateur et disponible au moment où la simulation s'exécute. Enfin, comme soulevé par [Friesen et al., 2016], cette méthodologie d'analyse permet finalement d'échanger de l'espace de stockage contre des heures de calculs. Il faut donc être prudent quant aux heures consommées et au moment précis des protections/reprises, si certaines analyses complémentaires sont nécessaires a posteriori. Il y a donc un besoin d'adaptation important de la communauté vis-à-vis de la gestion des analyses, comme mentionné dans le rapport de l'`EESI`, [a], pour pouvoir passer aux analyses *à la volée*.

Durant ma thèse, j'ai eu l'opportunité de travailler sur l'ensemble de la chaîne de vie des données. Du développement au sein d'un code de simulation, en passant par l'optimisation des `E/S` et des modèles de données, mais aussi et surtout en utilisant mes outils dans le contexte de la simulation `ExaMilkyWay`. De plus, j'ai développé cet outil d'analyse qui permet d'exploiter le modèle `lightAMR` et de produire des données destinées à la communication scientifique autour de l'étude de la cascade de turbulence. Cette opportunité m'a permis de comprendre les enjeux, mais aussi les difficultés du traitement de grands volumes de données, qui est un potentiel point bloquant pour mettre en place une méthode d'analyse *à la volée*. Cet outil a été développé en vue du post-traitement *post-mortem* des données (après écriture sur disque). Néanmoins, il reste compatible avec des approches *in-transit*, qui pourraient être exploitées avec des outils tel que `PaDaWan`, [Capul et al., 2018], qui est un package `Python` qui permet de récupérer le flux d'écriture de `Hercule` pour stocker les données en mémoire sur des noeuds de calcul. Mais ce fonctionnement sort du cadre de ces travaux.

4.2 Objectifs et caractéristiques de l'outil d'analyse

Cet outil a été développé pour répondre à trois catégories de besoins : tests, conversion et traitements des données. Le point commun de ces besoins est la relecture de données volumineuses, décrite avec le modèle `lightAMR` à des fins d'analyse. On rappelle également que ce modèle de données n'est pas lié à la bibliothèque `Hercule`, en conséquence d'autres types de lecteurs peuvent être implémentés pour lire les données. Cette approche a d'ailleurs été faite de manière expérimentale avec des données `lightAMR` au format `HDF5`. Quel que soit le lecteur, afin de tirer profit des `E/S` parallèles, il a été nécessaire d'utiliser une couche de parallélisme avec l'interface `MPI`. L'architecture des machines de calcul actuelles est optimisée pour des codes hybrides puisque le nombre de coeurs par noeuds de calcul a tendance à augmenter et la mémoire par coeur à diminuer. Afin de mieux exploiter ce type de ressources, une deuxième couche de parallélisme a été développée avec `OpenMP` pour bénéficier d'un outil à parallélisme hybride pouvant ainsi déployer plusieurs *threads* par processus `MPI`.

On retrouve trois approches principales utilisées lors du développement des outils d'analyse de données. La première est d'utiliser directement un langage non compilé, tel que `Python`. Les avantagea de cette méthode sont bien sûr la souplesse, la vitesse de développement, et la possibilité de de modifier les scripts rapidement pour tester de nouvelles analyses. Néanmoins, lorsque le volume de données devient significatif et/ou les analyses de plus en plus complexes, il devient difficile de maintenir ce type d'approche pour des raisons de performances. Pour pallier, cette problématique, certains outils tel que `Pymses`, délèguent, via une interface, les calculs à un langage compilé, généralement le `C` ou `C++`. C'est le cas de `Pymses` par exemple, qui interface du code `C` pour la lecture de données et certaines analyses, comme le lancer de rayons. Se pose alors la question importante de gestion mémoire. En effet, il faut faire le choix de la couche logicielle qui portera les données. Dans le cas de `Pymses`, c'est la couche `Python`. Il s'agit de la deuxième approche, un outil hybride, `Python/C`. Les limites de cette approche sont principalement la difficulté de la mise en place de l'interface et la gestion de la mémoire, même s'il existe de plus en plus d'outil pour créer automatiquement une interface, e.g. :`Pybind11`, `Boost`. La troisième approche, utilisé ici, est de développé un code entièrement dans un langage compilé, pour produire des données brutes réduites et d'utiliser, dans un second temps, une interface avec des scripts `Python`, par exemple, pour produire les résultats finaux, e.g. : carte 2D sous forme d'image avec unités, ajustement des couleurs, etc. Les limites de cette approche sont liées à la lourdeur des

a. http://www.eesi-project.eu/at-works/roadmaps/

développements, la compilation du code, en autres.

Cet outil d'analyse s'utilise donc principalement avec l'aide d'une fonction *main* et de fichier de configuration pour définir les analyses à effectuer sur les données. Or comme on l'a mentionné précédemment, l'analyse des données est un processus exploratoire par nature et nécessite un outil qui permet d'apporter beaucoup de souplesse tout en garantissant d'excellentes performances. Le problème avec un code compilé devient donc rapidement la lourdeur de la nécessité de modifier cette fonction *main* pour ajouter de nouvelle opération ou étapes d'analyses, même si les fonctions sont déjà disponibles. Il est bien évidemment possible d'apporter de la souplesse avec un fichier de configuration plus riche, mais c'est une méthode qui devient rapidement compliquée à gérer. Un objectif, en cours de développement, est donc d'interfacer les classes et les fonctions, pour converger vers un outil beaucoup plus facile à manipuler et apporter la souplesse nécessaire à l'exploration et l'analyse de données. L'interfaçage est fait à l'aide de la bibliothèque `Pybind11` qui permet de facilement faire la couche d'interface `Python` ↔ `C++`. Le *pipeline* d'analyses devra donc, à terme, se rapprocher de celui de la figure 4.1. Le challenge de cette approche est de maintenir de bonnes performances et le parallélisme hybride de l'application, tout en offrant une interface simple et riche.

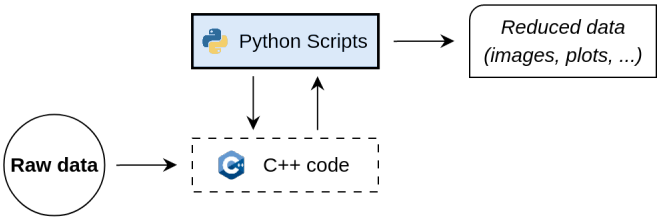

FIGURE 4.1 – Future *pipeline* d'analyse grâce à l'interfaçage du code `C++` bas niveau avec `Pybind11`, tout en maintenant le parallélisme hybride.

4.3 Structure de données et parallélisme

Comme on l'a mentionné précédemment, cet outil utilise un parallélisme à deux couches : mémoire distribuée (`MPI`) et mémoire partagée (`OpenMP`). Avant de discuter des structures de données, la liste des domaines à traiter est d'abord évaluée. En effet, cette liste représente soit l'ensemble complet des indices des domaines de la simulation $\{0, .., n\}$ (pour une simulation produisant n domaines), soit un sous-ensemble des domaines, sélectionné en fonction d'une région définie par l'utilisateur, voir section 4.5. Cette liste d'indice de domaine est ensuite répartie entre les P processus utilisés pour l'analyse (mémoire partagée). Ensuite, chaque processus gérera de manière indépendante, les domaines et exploitera le parallélisme à mémoire partagée (*multithreading* `OpenMP`) pour traiter les données des différents domaines, voir figure 4.2. On notera que la lecture des données est effectuée, en parallèle, par chaque processus `MPI`, qui demandera à la bibliothèque d'`E/S`, ici `Hercule`, les données de ses domaines. En conséquence, plus le nombre de processus `MPI` augmente, plus la charge `E/S` sera répartie.

Ensuite, il faut définir la structure des données utilisées par les processus pour les analyses. Bien que le modèle `lightAMR` soit très compact pour décrire le maillage, à des fins de stockage ou de transfert réseau, il est nettement moins efficace pour le calcul. En effet, par exemple, l'utilisation du *multithreading* implique nécessairement de pré-calculer un tableau d'indirection pour pouvoir itérer sur les cellules et accéder aux données. Cette indirection a un coût sur les performances puisqu'elle ne permet pas de bénéficier pleinement des avantages apportés par le système de cache des `CPU`. Il a donc fallu choisir une structure de données différente pour le calcul. Avant d'expliciter les choix techniques effectués, il est important de regarder ce qui a été proposé dans la littérature pour gérer les maillages `AMR` en post-traitement.

En 2017, une extension des modèles de données de `VTK` est proposée et mise en place sous la forme d'une classe `C++` du nom de `vtkHyperTreeGrid`, qu'on a déjà mentionné au chapitre précédent, destinée à la visualisation et à l'analyse de données `AMR` (en arbre). J'ai donc commencé par l'utiliser pour évaluer son intérêt dans le cadre du post-traitement de données (hors visualisation). Premièrement, il est important de noter que pour exploiter cette structure, il est nécessaire de déployer ou d'avoir accès au package `VTK` (bibliothèque compilée et *headers*). Il

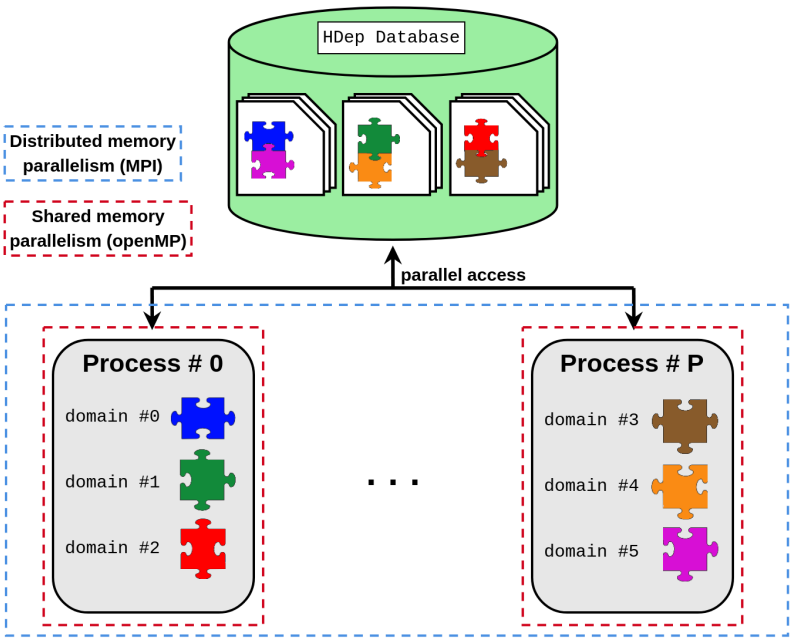

FIGURE 4.2 – Parallélisme hybride de l'outil d'analyse. Les processus `MPI` lisent les données des domaines qui leur ont été attribués dans la base `Hercule`. Les domaines sont traités de manière indépendante par les *threads* `OpenMP`.

s'agit d'une bibliothèque qui a de nombreuses dépendances et qui évolue constamment. En terme de déploiement, cela peut soulever de nombreux problèmes de dépendances **et** de versions pour les utilisateurs. Par exemple, le formalisme des données portées par les `vtkHyperTreeGrid` a évolué (et continue d'évoluer), rendant ainsi le code totalement incompatible d'une version à l'autre de `VTK`. Il s'agit d'un problème rencontré durant le développement de cet outil qui m'a poussé à m'orienter vers une autre solution. Deuxièmement, l'approche multiprocessus avec le modèle `MPI` nécessite d'utiliser une surcouche `MPI` mise en place par `VTK` ce qui rend l'outil totalement dépendant de la bibliothèque. De plus, l'utilisation des `HyperTreeGrid` dans ce contexte (multiprocessus) n'est pas triviale et certains filtres ne fonctionnent pas correctement, par exemple, si une même cellule est présente sur plusieurs processus, ce qui est le cas avec la description de `RAMSES`. Troisièmement, la structure et les curseurs proposés, voir [Harel et al., 2017b], ne permettent pas d'avoir une vision claire des possibilités d'exploitation du parallélisme par tâches (*multithreading*). Une des raisons est notamment qu'avec la structure en arbre, les curseurs permettant de se déplacer dans celle-ci sont construits de manière à être utilisé principalement par des algorithmes récursifs. Par exemple, la construction d'un *hyperTreeGrid* n'est pas possible avec plusieurs tâches et travailler exclusivement sur l'enveloppe feuille ne peut se faire qu'en parcourant l'arbre de haut en bas, etc. De plus, l'accès aux données du maillage ne se fait qu'au travers de l'`API` disponible et rend donc impossible (sans être intrusif dans le code source du package) de prendre certains raccourcis dans les algorithmes, notamment pour les méthodes de *flou gaussien* ou *lancer de rayons*, détaillées par la suite. Finalement, cette structure est incompatible avec l'utilisation de `GPU`, il faut passer par une structure temporaire, comme cela a été fait pour du lancer de rayon sur `GPU` par [Roche and Dubois, 2020]. Néanmoins, les *vtkHyperTreeGrid* restent une approche très performante pour de la *visualisation* des données `AMR` au sein, par exemple, de *Paraview*.

Il existe d'autres approches, comme par exemple, les tables de hachage, utilisées notamment dans le code de calcul `Dyablo`, [Durocher and Delorme, 2024] ou encore par [Tumblin et al., 2015]. Bien qu'il s'agisse de codes de calculs et non d'analyse de données, cette structure est intéressante parce qu'elle est performante et très simple d'utilisation. On notera également qu'elle est compatible (pour certaines implémentations) avec l'utilisation de `GPU` (offrant une possible évolution vers ce type d'architecture pour l'outil que l'on présentera ici). On utilisera donc ce type de structure comme base du parallélisme en mémoire distribué. Ensuite pour le parallélisme par tâche, il y a, en fait, deux options possibles. Soit chaque processus construit la liste des cellules, de tous ses domaines, dans cette structure de base en mémoire partagée, soit il faut rajouter un deuxième niveau, afin de pouvoir gérer les domaines séparément. Bien que la première option soit intéressante, elle offre moins de dynamisme, pour supprimer un domaine par exemple. De plus, lors de la construction de la structure (et de l'insertion des cellules), elles impliquent d'utiliser une table de hachage concurrente afin que plusieurs *threads* puissent écrire simultanément. On notera

```
template<class Mesh_t>
class Mesh{
        std::unordered_map<int, Mesh_t> _domains;
        [...]
};

class HMesh {
        robin_hood::unordered_map<Hashmap_Key, Hashmap_Value, Hash_Function> _cells;
        [...]
};

class VMesh{
        std::vector<Cell> _cells;
        [...]
};
```

FIGURE 4.3 – Exemple d'implémentation du conteneur global permettant de stocker le maillage par domaine.

également, que lorsque le nombre de cellule devient très grand, le risque de collision peut augmenter, réduisant ainsi les performances. C'est pour ces raisons que j'ai choisi la deuxième option. Ainsi, chaque processus aura une table de hachage comme structure principale, dont la clé sera l'identifiant du domaine (un numéro unique) et la valeur une sous-structure qui contiendra, pour chaque domaine, la liste des cellules, voir sur l'extrait 4.3. On notera de plus que cette approche permet de conserver l'information du domaine d'origine des cellules, à moindre coût. Par la suite, on appellera *conteneur global* cette structure de base contenant les cellules des domaines, référencée sous le terme de `Mesh` dans l'exemple sur la figure 4.4. Il est important de préciser qu'une partie de ce dynamisme est également possible parce que `Hercule` offre un accès facile aux données domaine par domaine, et que `lightAMR` est construit et auto-portant par domaine, voir chapitre 3.

Pour la deuxième structure, sa définition n'est pas figée et dépendra des analyses demandées par l'utilisateur. Par exemple, pour produire une image par la méthode du *flou gaussien* (voir section 4.7.2), un histogramme `1D` ou `2D`, on utilisera plutôt un vecteur de cellules, car cette structure sera plus performante compte tenu des opérations nécessaires. En revanche, pour faire du *lancer de rayon*, on construira plutôt une table de hachage des cellules (voir section 4.7.3). Grâce à l'utilisation des *templates* `C++`, les structures de données peuvent facilement être étendues ou adaptées, voir 4.4. Au sein du conteneur global, les domaines sont différenciés, donc on notera que la construction de cette deuxième structure peut être effectuée par plusieurs *threads* simultanément sans condition d'accès bloquante. Dans cette structure, on stocke les informations des cellules feuilles extraites depuis la description au modèle `lightAMR`, c'est-à-dire ses coordonnées logiques et son niveau `AMR`. On constatera que les cellules parents ne sont pas stockées sauf si toutes les cellules enfants sont des feuilles. Dans ce cas, exceptionnellement, on rajoute la cellule parent. Un booléen est donc nécessaire pour savoir si la cellule stockée est une feuille ou une cellule parent "terminale", qu'on appellera par la suite *noeud terminal*. Ces cellules correspondent à l'enveloppe feuille de l'arbre du domaine à traiter, qui peut être filtrée à la construction par une région d'intérêt, comme on le montre à la section 4.5. Il est important de noter que la somme des cellules de l'enveloppe feuille de tous les domaines de la simulation doit représenter la totalité du volume de la boite de simulation. Enfin, dans le cas d'une table de hachage pour les cellules, on en utilise une non ordonnée de type "robin", plus performante [b] que celle disponible dans la bibliothèque standard du `C++`.

Afin de réduire la consommation mémoire, en plus de l'utilisation de *noeuds terminaux* permettant de stocker 1 cellule au lieu des f_d^{DIM} cellules enfants, une approche `LOD` (*Level-Of-Detail* a été implémentée lors de la construction de la structure secondaire : le vecteur ou la table de hachage des cellules des domaines. En effet, l'utilisateur a la possibilité de définir un niveau `AMR` maximal pour le post-traitement. Ainsi, lorsque l'algorithme arrive à ce niveau, si une cellule est parent, il la considère comme une cellule feuille et la rajoute dans la structure, si et seulement si, elle appartient au domaine courant. Ces deux méthodes complétées par l'utilisation du pavage cubique, voir section 4.5, et éventuellement le *batch-mode*, voir section 4.4, constituent les points clés pour optimiser la consommation mémoire, les performances de calcul et les `E/S` de l'outil d'analyse.

b. https://github.com/martinus/robin-hood-hashing

Pour les données hydrodynamiques, qui constituent le plus gros du volume mémoire, les champs scalaires sont lus et stockés tels quels (suivant l'ordre du modèle `lightAMR`). En conséquence, l'accès aux données hydrodynamiques doit nécessairement se faire par une indirection en stockant, en plus, par cellule, l'indice dans le tableau de ces données. On notera que recopier dans une autre structure les données évite un pic mémoire , mais implique de conserver toutes les données du domaine du champ(s) scalaire(s) chargé(s), y compris les données des cellules parents, soit 12.5% de surplus. Dans le cas d'un filtrage géométrique du maillage, on conserve également les données des cellules filtrées, ce qui peut constituer un surcoût important. Il s'agit d'une optimisation dont on discutera dans les perspectives d'évolution. On donne sur l'extrait 4.4 la structure utilisée dans le cas d'une table de hachage et dans le cas d'un vecteur. On notera que cette indirection peut également avoir un impact sur le cache des `CPU`, mais elle permet d'avoir beaucoup de dynamisme, notamment pour l'ajout de champ dérivé, calculé depuis des grandeurs lues, ou pour supprimer une grandeur qui n'est plus nécessaire, le tout sans avoir besoin de recompiler l'outil.

```
struct Cell{
        Logical_Pos_t coord;
        uint32_t dataIndex;
        uint8_t level;
        bool isTerminal;
};
```

```
struct Hashmap_Key{
    Logical_Pos_t coord;
    uint8_t level;
};

struct Hashmap_Value{
    uint32_t dataIndex;
    bool isTerminal;
};
```

FIGURE 4.4 – A gauche, la structure de base utilisée dans le conteneur de type *vecteur* et à droite les deux structures "clé-valeur" pour le conteneur de type *hashmap*.

Les données au modèle `lightAMR` lues, ainsi que les particules (le cas échéant) sont stockées dans des structures à part, respectivement `DataHydro` et `DataParts`, avec le même fonctionnement de container global sous forme d'une table de hachage par domaine. On donne par souci de clarté de la structure, sur la figure 4.5, une vue schématique de l'architecture du code.

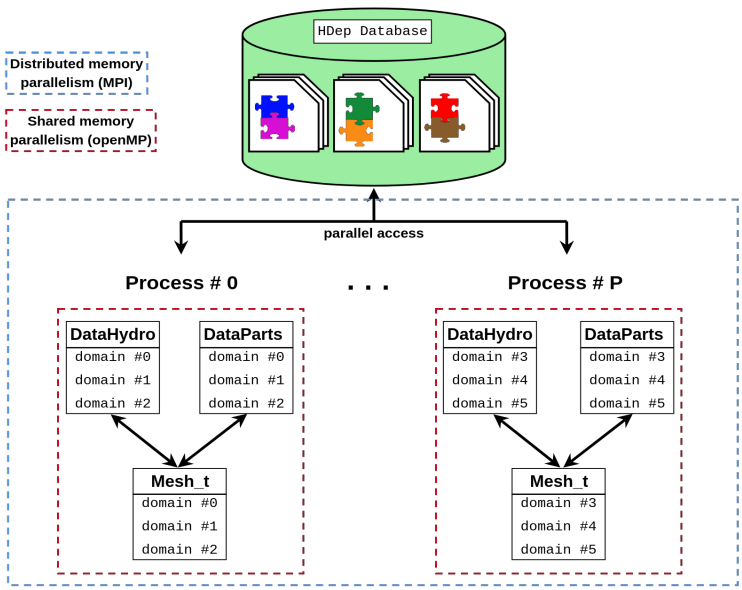

FIGURE 4.5 – Schéma de l'architecture parallèle et des structures du code. Les éléments `DataHydro`, `DataParts` et `Mesh_t` sont des objets `C++`. Les deux premiers portent les données lues depuis la base `Hercule`, et le troisième est construit au début de la phase d'analyse.

L'implémentation du parallélisme par tâche devient alors plus naturelle, voir exemple 4.6. On notera

l'utilisation du mot-clé `OpenMP` *dynamic* pour une répartition plus optimale de la charge entre les tâches, voir section 4.9. Bien que cet exemple reste simple, dans la plupart des cas, les analyses peuvent être effectuées domaine par domaine. En effet, c'est ce type d'approche qui est utilisé pour la construction de l'enveloppe feuille, la décompression de la grille `AMR`, la décompression des données flottantes, le calcul d'histogrammes 1D ou 2D, calcul des champs dérivés, etc. La seule exception est l'algorithme de lancer de rayons qu'on détaillera à la section 4.7.3. Néanmoins, il faut être prudent quant aux potentielles conditions d'accès bloquantes (*race condition*) : c'est le cas notamment lorsque les données issues de plusieurs domaines contribuent à un même résultat.

```
size_t getNumberOfElements() const {
    size_t total = 0;

    #pragma omp parallel for reduction(+:total) schedule(dynamic)}
    for( size_t idom=0; idom<numberOfDomains; ++idom ){
            const auto & md = _mesh.at( _domainsID[ idom ] );
            total += md.getNumberOfElements();
    }
    return total;
}
```

FIGURE 4.6 – Exemple du calcul du nombre de cellules en *multithread* avec `openMP` sur le conteneur global d'un processus `MPI`.

4.4 Le *batch-mode*

Il arrive régulièrement que le volume de données à traiter soit plus grand que la quantité de mémoire disponible. Bien évidemment, lorsque l'outil d'analyse a une bonne scalabilité, une option simple consiste à demander l'accès à plus de ressources. En effet, dans la plupart des cas, les analyses s'effectuent sur un centre de calcul, il suffira donc simplement de diviser le nombre de domaines à traiter par le nombre de processus `MPI` et on ajustera le nombre de tâches `openMP` en fonction du nombre de domaines à traiter par processus. Si les besoins en mémoire sont importants, il suffit alors soit d'augmenter le nombre de processus `MPI`, soit d'augmenter le nombre de tâches. Il faut bien sûr faire attention à la scalabilité en tâche et processus des analyses. Néanmoins, on notera que rajouter un processus `MPI`, dans notre cas, est généralement plus intéressant que rajouter une tâche `openMP`, car cela permet de mieux répartir les `E/S`. Il y a un optimum pour avoir un bon équilibre en processus et en tâche par processus, en fonction des données et des analyses.

Néanmoins, il existe des approches appelées *algorithmes de mémoire externe* (*External Memory Algorithm*), qui permettent de résoudre ce type de problèmes. Certaines méthodes sont particulièrement avancées et nécessitent de mettre en place un système de mémoire tampon qui sera utilisé par l'algorithme, [Vitter, 2001]. D'autres sont nettement plus simples et consistent à découper les données en paquets. C'est le choix d'implémentation que j'ai fait, car il permet de répondre à la plupart des analyses. En effet, la liste des domaines à traiter est découpée en paquets, appelés *batch*, qui sont traités les uns à la suite des autres, et les résultats sont accumulés. Cette approche permet par exemple de s'affranchir de l'utilisation de processus `MPI` lorsque les analyses sont effectuées sur une machine locale.

Cependant, ce type d'approche n'est pas toujours facilement utilisable. Il est par exemple difficile de faire un histogramme d'une grandeur physique si on ne connaît pas les bornes de cette grandeur, ou tout simplement s'il y a une dépendance des résultats entre un *batch* et le suivant. Cette méthode est particulièrement bien adaptée lors du calcul de cartes `2D`, par la méthode de flouttage gaussien ou de lancer de rayon, parce que les résultats peuvent être facilement accumulées par le biais d'un opérateur global (`SUM`, `MAX`, `MIN`, etc.) d'un *batch* à l'autre.

Prenons l'exemple d'une analyse qui a pour objectif de produire une carte par lancer de rayons d'une sous-région d'une simulation (ici `ExaMilkyWay`), composée de 5120 domaines. On fait le choix d'utiliser un traitement par paquet de 512 domaines. Pour cette analyse, on utilise un ordinateur de bureau qui dispose de 16

coeurs et de 32 Gigaoctets de mémoire vive uniquement. On utilise donc un seul processus `MPI` et 16 *threads*. On notera que dans cette configuration, utiliser plusieurs processus `MPI` n'est pas intéressant, parce qu'on ne dispose pas de système de fichiers parallèles, donc les lectures sur plusieurs processus ne pourront pas, d'un point de vue *hardware*, se faire en parallèle. L'analyse accumulera donc les cartes sur 10 *batch*. On trace avec `Vtune` (package *OneAPI de intel*) le déroulement de l'exécution, ce qui permet d'obtenir le graphe du temps `CPU` sur la figure 4.7. On y voit très clairement les différents *batch* avec la succession de temps de lecture, de décompression et de création du maillage (structure *HMesh* mentionnée précédemment) et du lancer de rayon. Toutes ces opérations se déroulent en multi-tâches. Le temps de cette analyse est d'environ 71.4 secondes sur un `CPU` (*AMD Ryzen 7 5700X*).

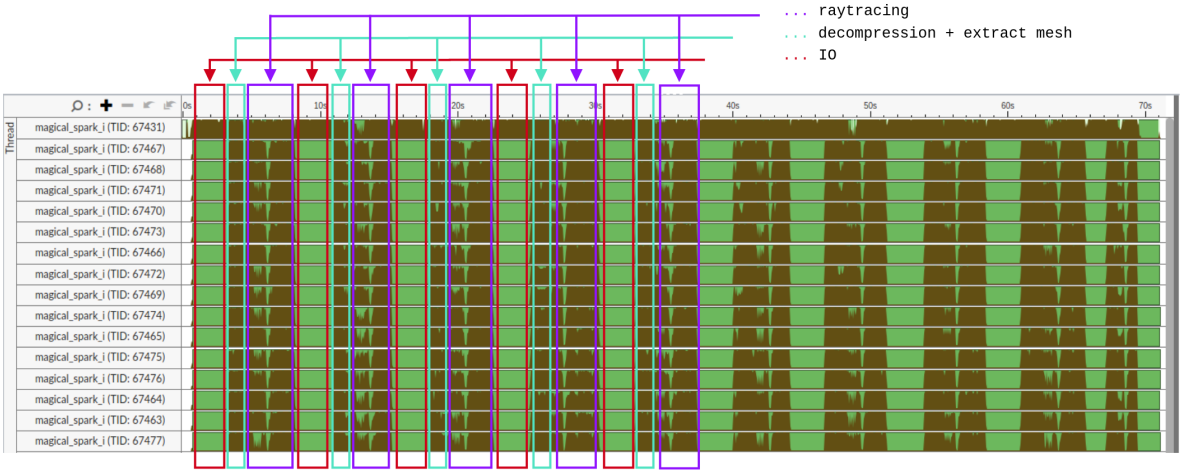

FIGURE 4.7 – Analyse du temps d'exécution avec l'outil `Vtune` (Intel OneAPI) pour la production d'une image par lancer de rayons de 5120 domaines de la simulation `ExaMilkyWay`. On met en évidence en rouge les régions de temps d'exécution "perdu" par la lecture des données du batch. En bleu les étapes de décompression et construction des structures de calcul, en mauve le calcul de la carte.

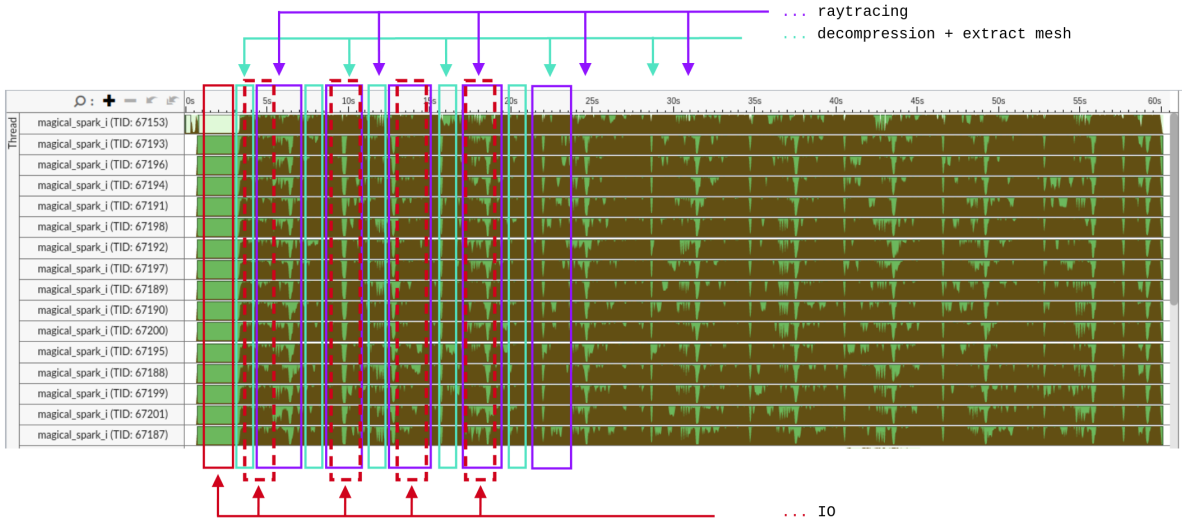

FIGURE 4.8 – Analyse du temps d'exécution avec l'outil `Vtune` (Intel OneAPI) pour la production d'une image par lancer de rayons de 5120 domaines de la simulation `ExaMilkyWay`. On met en évidence en rouge les régions de lecture de données, exécutée en arrière plan par un processus tiers, pendant le calcul de la carte sur les domaines précédents. En bleu les étapes de décompression et construction des structures de calcul, en mauve le calcul de la carte.

Il est intéressant de noter les temps d'attente des 15 *threads* lorsque le processus principal lit les données du *batch*. Ce temps d'attente est du temps perdu qu'il est possible de réduire. En effet, en utilisant les fonctionnalités

du C++20, notamment les exécutions asynchrones, on peut superposer les temps de lecture des données et les temps de calculs. Pour cela, un *thread* externe est utilisé et se chargera de rendre les données disponibles au moment où les analyses en auront besoin. Il s'agit d'une méthode expérimentale, qu'il faudrait approfondir, mais qui permet tout de même d'obtenir des résultats intéressants. On obtient ainsi le profil de la figure 4.8. Le temps total d'exécution est réduit d'environ 15%, et on voit clairement que désormais les opérations se superposent et l'utilisation globale du CPU est bien meilleure.

4.5 Le pavage cubique minimal

Afin d'optimiser les données à lire, il est commun de trouver des approches de type Fast Query [Chou et al., 2011a], [Chou et al., 2011c], [Chou et al., 2011b]. Cette méthode utilisée par [Byna et al., 2012] permet de visualiser et analyser un trillion de particules. Le problème de cette approche est que les données ont été écrites avec HDF5 avec l'approche single-shared-file, produisant ainsi un unique fichier de 30 Téraoctets. Pour accélérer les analyses, ils ont donc utilisé un système d'indexation, qui nécessite de relire les données pour ensuite écrire un fichier d'indexation, pour pouvoir faire des accès rapides à des données avec une région d'intérêt dans un deuxième temps, ROI (*Region Of Interest*), et ainsi accélérer significativement les analyses et les E/S. On propose une approche dans le même esprit avec l'exploitation du pavage cubique minimal, PCM, pour sélectionner les données à lire dans une ROI et donc accélérer les traitements et les E/S. Ainsi, sa première fonction sera de permettre de définir l'indice des domaines qui vont intersecter la ROI, définie par l'utilisateur (région disponible : sphérique, parallélépipèdique, ellipsoïdale). On notera que ce filtrage des domaines est effectué en amont des lectures. La liste filtrée est ensuite répartie entre les processus MPI. Encore une fois, l'accès aux domaines en parallèle grâce à Hercule est un atout.

En géométrie, un pavage de l'espace est l'action de remplir l'espace par des polyèdres sans chevauchement. On s'intéresse ici à un espace en 2 ou 3 dimensions. L'objet géométrique utilisé le plus simple est généralement le cube. On notera également que la plupart du temps, on utilise un même polyèdre pour paver l'espace, par exemple : un cube de même dimension. Dans notre cas, on parle de pavage cubique minimal (PCM) parce qu'on utilisera un minimum de cubes, de tailles différentes, pour paver l'espace défini par l'intervalle de la courbe de Hilbert, c'est-à-dire un domaine de simulation. En effet, on construit ce pavage par domaine pour être compatible avec l'approche de traitement *par domaine* de l'outil. On notera, en conséquence, que ce calcul est effectué en mémoire partagée (par domaine). De plus, on impose de respecter la description en arbre, et donc la taille des carrés / cubes utilisés. En tenant compte des deux contraintes de construction du PCM, par domaine et suivant l'arbre AMR, le pavage est minimal comme on peut le voir sur la figure 4.9.

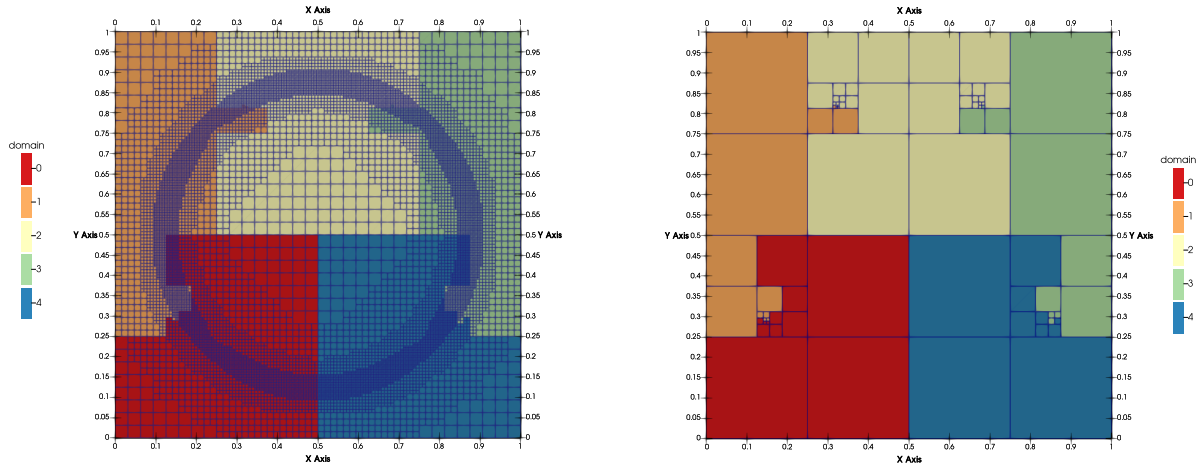

FIGURE 4.9 – Décomposition de domaines sur 5 processus MPI, avec affichage de la grille AMR, pour un cas test d'un blast 2D, figure de gauche. Sur la figure de droite, on donne le pavage cubique minimal calculé pour chaque domaine.

Deux méthodes ont été implémentées pour calculer le PCM. La première méthode se base sur la décomposition de domaines et la valeur des clés de Hilbert aux frontières. Il s'agit de la même approche que celle proposée par [Chapon, 2011]. La deuxième méthode consiste à appliquer un algorithme d'élagage similaire à celui

décrit au chapitre 3 pour la suppression de redondance d'information dans la description de la grille AMR au modèle lightAMR. En effet, il suffit simplement de changer la condition d'*élagage* par *"si toutes les cellules enfants appartiennent au domaine courant" alors* on change la valeur du raffinement de la cellule parent. Ces deux méthodes donnent exactement le même pavage cubique. L'avantage de la première méthode est qu'elle ne nécessite que très peu d'information et donc peu de lecture. En effet, elle nécessite uniquement les clés de Hilbert (on se base ici sur la méthode de décomposition de RAMSES) c'est-à-dire qu'il suffit de lire un tableau de $n+1$ indices 64 bits (ou 128 bits). Il est important de noter que lorsque la grille de raffinement dépasse le niveau AMR 16, une valeur 64 bits ne suffit plus pour stocker la clés de Hilbert, et la quadruple précision est nécessaire. Le problème de la quadruple précision est qu'elle n'a pas été normalisée par une norme IEEE, en conséquence, l'écriture d'un tableau quadruple précision en Fortran et sa relecture en C ou C++ pose des problèmes de compatibilités et un réarrangement des octets est nécessaire. Une fois le pavage obtenu sous la forme d'une liste de cellules, il est très rapide de calculer l'intersection de chacune des cellules, qu'on appellera "bloc" pour éviter la confusion (avec les cellules du maillage), du pavage du domaine avec une ROI, et ainsi pré-définir rapidement la liste des domaines à traiter, voir figure 4.10. Le gain apporté par le pavage cubique est considérable sur les E/S puisqu'il permet d'éviter de lire des données inutiles. On verra à la section 4.7.3 qu'il peut aussi être utilisé comme une structure d'accélération pour améliorer drastiquement les performances de l'algorithme de lancer de rayons.

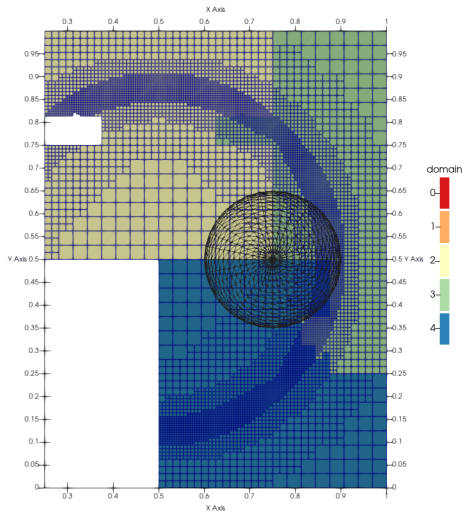

FIGURE 4.10 – Filtrage des domaines à l'aide du pavage cubique minimal et d'une région d'intérêt sphérique de rayon 0.2, centrée en $\{0.75, 0.5, 0.0\}$ dans la boite de simulation 2D. Les données du domaines 0 et 1, ne seront pas lus.

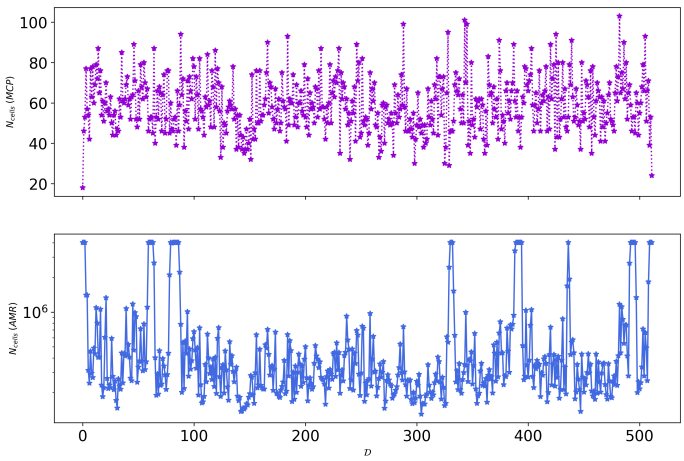

FIGURE 4.11 – Nombre de cellules par domaine dans le pavage cubique minimal de la simulation Orion (graphique du haut) comparé au nombre de cellules total par domaine (du modèle lightAMR), graphique du bas.

On donne à titre indicatif, sur la figure 4.11, le nombre de cellules par domaine du `PCM` et le nombre de cellules `AMR` pour la simulation `Orion`. Alors que le nombre de cellules oscille entre $\sim 1.3 \cdot 10^5$ et $\sim 4 \cdot 10^6$, le nombre de blocs du `PCM` ne varie qu'entre 18 et 103. Il s'agit d'un atout considérable apporté nativement par l'`AMR`, qui peut être utilisé comme une structure d'accélération pour le lancer de rayons, comme on le verra à la section 4.7.3.

4.6 Filtrage du maillage

Comme mentionné précédemment, le filtrage du maillage par une `ROI`, permet de réduire la consommation mémoire utilisée par les structures contenant les cellules d'un domaine et les temps de traitements. En effet, lors de la construction de l'enveloppe feuille, par domaine, les cellules en dehors de la région ne seront pas considérées, comme on le montre sur la figure 4.12. On notera que dans le filtrage avec le maillage, on a des cellules qui n'intersectent pas, a priori, la région de la sphère, e.g. : les cellules en $(0.6, 0.6)$. C'est dû à l'utilisation des *noeuds terminaux* pour le calcul de l'intersection, alors que l'affichage se fait, ici, en décrivant les cellules des *noeuds terminaux*. Les traitements ne s'appliqueront donc qu'à ces cellules. On notera que les données hydrodynamiques des cellules filtrées du domaine 2,3,4 sont malgré tout stockées en mémoire, comme mentionné à la section 4.3. Il est important de noter que le filtrage géométrique peut poser des problèmes avec l'algorithme utilisé pour le lancer de rayon, voir section 4.7.3.

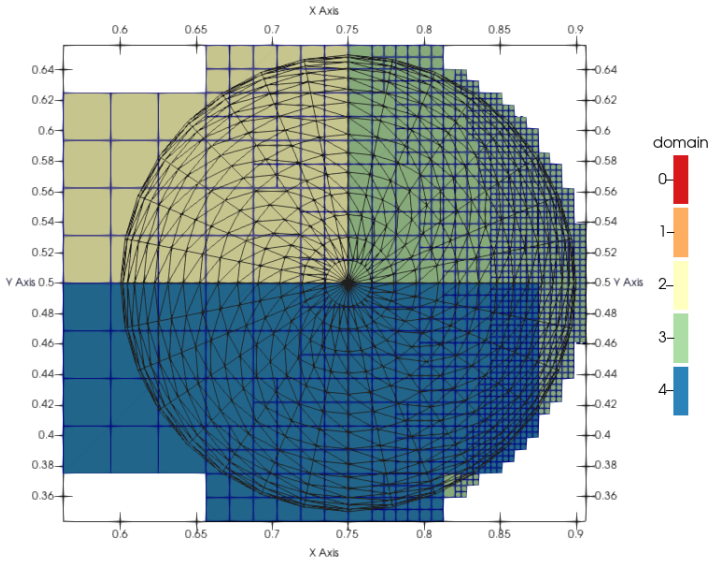

FIGURE 4.12 – Filtrage d'un maillage `2D` à la construction des structures dans le conteneur global pour chaque domaine par un `ROI` sphérique défini par l'utilisateur.

4.7 Production de carte 2D

Dans cet outil, on définit une carte 2D à partir d'un plan $\mathcal{P}$, d'un nombre de pixels et d'une extension spatiale. En effet, la position et l'orientation de la carte sont définies par le plan $\mathcal{P}$, c'est-à-dire un point `3D`, assimilé au centre de la carte $\mathcal{O}$, et une normale $\vec{n}$. L'extension spatiale est la région couverte par la carte dans la base orthonormée du plan, définie par les vecteurs $\{\vec{u}, \vec{v}, \vec{n}\}$. Cette extension est donc définie par un couple de valeurs $\{x_e, y_e\}$, dans l'intervalle $[0, 1]$, suivant les vecteurs $\vec{u}, \vec{v}$. Le nombre de pixels est un couple de valeurs entières, $\{i, j\}$ qui permet de discrétiser cette extension, en élément unitaire qu'on appellera par abus de langage un *pixel*. Les différentes méthodes détaillées par la suite, appliqueront donc des traitements qui ont pour but de définir une valeur pour chacun des pixels.

Dans les trois algorithmes de production de carte `2D`, on utilisera cette norme pour définir une carte. De plus, pour chaque algorithme, on utilisera un `operateur` qui permet de définir l'opération appliquée pour calculer

la valeur d'un pixel. Certains opérateurs peuvent être spécifiques à un algorithme donné. Par exemple, dans le cas de la méthode par *lancer de rayon*, l'opérateur `max` permet de prendre la valeur maximale lors de la mise à jour de la valeur du pixel.

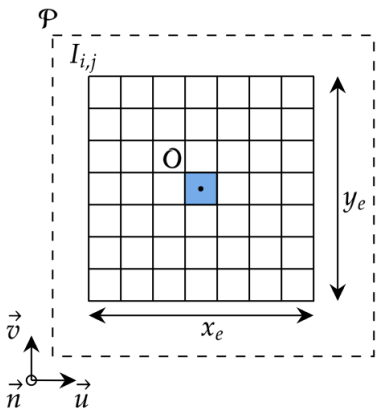

FIGURE 4.13 – Norme utilisée pour définir une carte `2D` dans l'outil d'analyse à partir d'un plan $\mathcal{P}$ dont l'utilisateur renseigne la normale n et le centre $\mathcal{O}$. Le couple (x_e, y_e) est l'extension spatiale entre $[0, 1]$ et I est la carte `2D`, discrétisée par le nombre de pixels (i, j) suivant les 2 directions.

4.7.1 Carte par la méthode du "Slicing"

Une tranche (ou *slice*) permet de produire une carte `2D` à partir de l'intersection du plan $\mathcal{P}$ avec les cellules du maillage. Pour ce type d'algorithme, parcourir l'ensemble des cellules (par domaine) et calculer l'intersection avec le plan et ainsi déterminer le pixel qui devra être mis à jour, est extrêmement coûteux. Typiquement, N_{cell} opérations de calcul d'intersections seraient nécessaires, et même avec les réductions apportées par le filtrage géométrique et l'utilisation du *multithreading* sur les domaines, ce type d'approche n'est pas envisageable. En effet, la très grande majorité du temps de calcul est passée dans l'évaluation d'intersection inutile. En conséquence, l'utilisation d'une structure de données de type `VMesh` (sous forme de vecteur) mentionnée précédemment n'est pas optimale. En revanche, il est possible de prendre le problème à l'envers et projeter le centre des pixels en coordonnées `3D` dans la structure contenant le maillage. Pour ce type d'opération, les tables de hachage sont naturellement la structure la plus adaptées.

On notera que compte tenu de la stratégie de traitement avec un conteneur global, contenant les domaines, il y a deux options possibles. La première consiste à rechercher dans l'ensemble des domaines du conteneur et donc faire $n_{domains}$ opérations de recherche dans la table de hachage des cellules, pour chaque coordonnée `3D` des pixels. Bien que cette opération soit plus intéressante que celle mentionnée précédemment, en parcourant toutes les cellules d'un vecteur, elle reste très coûteuse. D'autant plus que le niveau `"AMR"` du point `3D` n'est connu, il faut donc faire une recherche pour tous les niveaux stockés dans la table de hachage, et cela revient à faire finalement $n_{domains} \times n_{level}$ opérations. On remarquera que les coordonnées logiques doivent être adaptées en fonction du niveau. La seconde option, celle qui est implémentée, est de calculer l'indice de Hilbert du point `3D` au niveau l_{max} de la simulation, ce qui permet de déterminer l'indice du domaine auquel appartient le point, grâce à la liste des clés de Hilbert des frontières de domaines. Ensuite, il suffit de faire une recherche dans la table de hachage du domaine correspondant. On notera qu'ici il faut déplacer le parallélisme `OpenMP`, non plus sur les domaines mais sur la liste des pixels à projeter. De cette manière, l'algorithme sera plus performant et il n'y aura pas de concurrence pour l'écriture de la valeur d'un pixel.

On précise que cet algorithme serait plus performant si la structure globale contenait directement les cellules de l'enveloppe feuille et non pas une sous-structure par domaine. Ceci pourrait même permettre un rendu temps réel de *slices*, avec suffisamment de processus / *threads*, par rapport au nombre de cellules. Un petit prototype avec un rendu, en Python, a montré la possibilité du temps réel avec cette dernière méthode par rapport à celle

implémentée.

4.7.2 Carte par la méthode du flou gaussien

Le flou gaussien est une méthode issue du traitement d'image qui permet de lisser la valeur d'un pixel sur plusieurs pixels alentours pour atténuer des imperfections ou réduire le bruit. Pour appliquer cette méthode avec des données AMR 3D, il y a plusieurs étapes : projection, transformée de Fourier, convolution, transformée de Fourier inverse, sommation, [Labadens et al., 2012]. Soit l'enveloppe feuille $\mathcal{E}$ du maillage, et les coordonnées de l'image I suivant la convention définie précédemment. On définit autant d'images que de niveaux AMR, notés I_l, dans la liste de cellules tous domaines confondus. La première étape consiste alors à projeter parallèlement, dans l'image I_l, le centre des cellules de niveau l, et ainsi déterminer le pixel auquel la cellule contribuera, voir schéma 4.14. On notera qu'il n'y a pas de direction de projection privilégiée, contrairement au lancer de rayon. C'est à cause de cette étape de projection du centre des cellules qu'il est préférable d'utiliser une structure de l'enveloppe feuille sous la forme d'un vecteur, plutôt que d'une table de hachage.

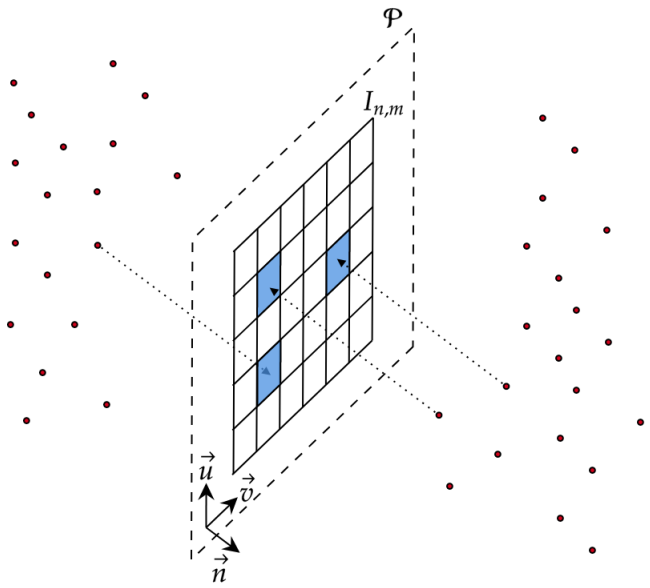

FIGURE 4.14 – Schéma de la méthode de projection du centre des cellules pour l'algorithme de *flou gaussien*. On se place ici dans le cas d'un image à un niveau l.

Afin de tenir compte de la contribution des cellules AMR aux différents niveaux, on applique un flou avec un noyau gaussien 2D de taille différente en fonction du niveau. La formule d'une gaussienne en 2D est donnée par la formule 4.1, où σ_l est la taille du noyau au niveau AMR l. On notera que le noyau est symétrique, c'est-à-dire que $\sigma_x = \sigma_y = \sigma_l$. L'utilisation d'un noyau inversement proportionnel au niveau des cellules permet, au travers de la transformée de Fourier, de lisser les niveaux de manière variable. En effet, les niveaux AMR élevés seront étalés sur plus de pixels que les niveaux AMR les moins élevées où la résolution est faible. Ainsi, cette méthode permet de faire ressortir les caractéristiques principales sur la carte produite. Par défaut, $\sigma_0 = 1.$, mais il est possible d'adapter le niveau de "flou" en augmentant ou en diminuant cette valeur. Par exemple, en définissant un $\sigma_0 = 0.5$, l'image sera moins floue et on verra mieux apparaître les structures données par les hauts niveaux AMR (petites cellules). Inversement, $\sigma_0 = 2.0$, augmentera le lissage des contributions.

Contrairement aux méthodes dans le traitement d'image, où les noyaux sont généralement de petites matrices, e.g. 3×3, 9×9, ici les noyaux sont des cartes à part entière, de la même taille que l'image finale. En effet, cela permet d'effectuer le flou gaussien en une seule passe grâce à la convolution dans l'espace de Fourier. On notera que les cartes des noyaux sont normalisées, ainsi on n'ajoutera pas d'information lors de la convolution. La deuxième étape consiste donc à produire les cartes des noyaux et effectuer leur transformée de Fourier (TF), ainsi que celle des cartes issues de la projection des cellules. On utilise une bibliothèque optimisée dédiée aux calculs de

`TF` : `FFTW3`[c]. La troisième étape consiste à convoluer des cartes de fréquence des cellules projetées par celles des noyaux, par niveau. La quatrième étape est celle des `TF` inverses, par niveau également.

$$G^l(x,y) = \frac{1}{2\pi\sigma_l^2}\exp\left(-\frac{x^2+y^2}{2\sigma_l^2}\right) \tag{4.1}$$

Finalement, la dernière étape consiste à sommer toutes les cartes des différents niveaux pour obtenir l'image finale. On donne sur la figure 4.15, l'exemple sur les données de la simulation `ExaMilkyWay`, des différentes étapes de cette méthode. Sur la première ligne, de gauche à droite, on a respectivement les contributions des cellules de niveau 11, 12 et 13, pour le champ de densité. On voit par exemple, que les cellules de niveau 13 sont principalement au centre de la galaxie. La ligne suivante correspond aux `TF` de ces cartes. Et la troisième ligne donne le résultat de la `TF` inverse après convolution par la carte des fréquences des noyaux gaussiens. Finalement, le résultat final après sommation des différentes cartes est donné sur la figure 4.16.

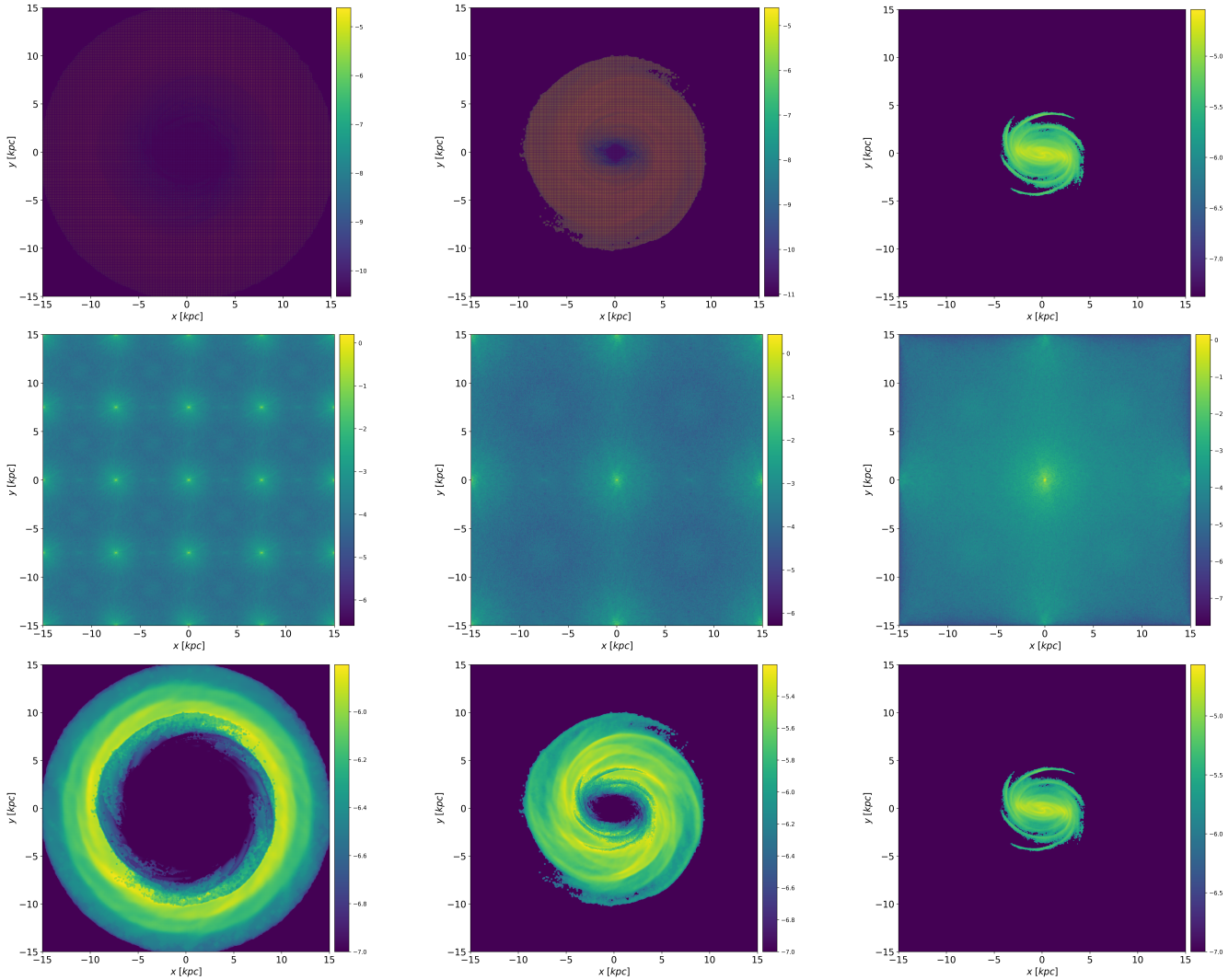

FIGURE 4.15 – Étapes de la méthode de flou gaussien adaptative par niveau, sous la forme de carte, pour le champ de densité de la simulation `ExaMilkyWay`, après relaxation, voir 6. Le plan est de normale $\vec{n} = (0.0, 0.0, 1.0)$ et de centre $(0.5, 0.5, 0.5)$ pour une région carrée de 30 *kpc* de côté, produisant une image de 2048×2048 pixels. On affiche les cartes uniquement pour les niveaux 11, 12 et 13, de gauche à droite. Sur la 1ère ligne, les projections parallèles du centre des cellules (contribution du champ de densité) dont on donne sur la 2ème ligne les cartes de fréquences. Sur la troisième ligne, les cartes après convolution. On se place ici en unité code.

c. https://www.fftw.org/

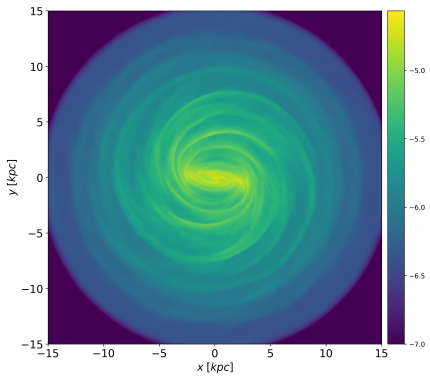

FIGURE 4.16 – Résultat du flou gaussien adaptatif sur la simulation `ExaMilkyWay` après sommation des cartes des différents niveaux. On se place ici en unité code.

Le désavantage principal de cette méthode est d'être particulièrement consommatrice de mémoire, surtout avec des cartes hautes résolution. En effet, par exemple, pour une `ROI` contenant des cellules avec un niveau compris entre $[10, 16]$, il faudra ainsi définir 7 images. Si l'utilisateur souhaite faire une image carrée haute résolution de 8192×8192 pixels, il faut ainsi allouer ~ 1.8 Go de mémoire rien que pour les données brutes des images (en simple précision), par processus `MPI`.

4.7.3 Algorithme de lancer de rayons

Le lancer de rayon est une méthode extrêmement populaire dans les jeux vidéos pour faire du rendu d'image ou de scène. Depuis 2019, il existe même des instructions matérielles spécifiques pour accélérer les rendus et faire du lancer de rayons d'image haute résolution en temps réel. On peut citer l'exemple des célèbres cartes `GPU RTX` de chez `Nvidia`. La technologie `RTX` (*Ray Tracing Texel eXtreme*) permet une accélération matérielle et logicielle, grâce à des coeurs dédiés optimisés pour accélérer les opérations mathématiques nécessaires pour simuler la propagation d'un rayon, et notamment la traversée de structures hiérarchiques. Dans la littérature, depuis les 5 dernières années, on retrouve de nombreux travaux utilisant des `API` "constructeur" pour accélérer le rendu de maillage `AMR` en arbre. En effet, par exemple, [Wang et al., 2020] utilise l'`API Ospray` de `Intel` dédiés au lancer de rayon. Ils sont capables de faire du rendu volumique par le biais d'isosurface, sur une station de travail avec 72 coeurs avec 3 *To* de `RAM`, d'une simulation de 625 millions de cellules `AMR` réparties sur 4 niveaux. En utilisant le même jeu de données, [Wald et al., 2021] utilise l'`API Optix` de `Nvidia` pour faire du rendu temps réel sur `GPU` avec une station disposant de deux cartes `Nvidia RTX 8000` de 48 *Go* chacune.

Néanmoins, le premier point bloquant pour exploiter ces deux approches dont les codes sont accessibles en ligne : `TB-AMR`[d], `OwlExaBrick`[e] est l'installation et la compilation des codes. En effet, l'installation des dépendances, par exemple, les drivers `Nvidia` n'est pas toujours une tâche aisée. Deuxièmement, il reste (l'éternel) problème de compatibilité des formats de données. Par exemple, pour exploiter les `ExaBricks` de [Wald et al., 2021], il faut passer par une première étape de transformation des données dans un format proche du modèle *non structuré*, puis construire les `exabricks` utilisées comme structure d'accélération (au-dessus du maillage). Si cette première étape est possible pour les volumes de données de quelques gigaoctets, le problème devient bloquant pour une simulation comme `ExaMilkyWay`. D'autre part, il faut noter que les jeux de données utilisés pour les tests contiennent au maximum 4 niveaux `AMR`, il serait intéressant de voir la scalabilité de ces approches avec plus de niveaux, notamment en terme de consommation mémoire et performances vis-à-vis des structures d'accélération utilisées. Enfin, la scalabilité de ces approches, avec plusieurs processus `MPI`, pour gérer de plus gros volumes de données, n'est pas détaillée.

Ce que l'on propose ici, c'est une approche similaire sur certains aspects, mais dont le but principal n'est pas de faire du rendu en temps réel d'isosurface (ou de la visualisation `3D`). On s'affranchit donc des problématiques d'interpolation aux frontières de niveaux `AMR`. De plus, un avantage significatif de cette méthode est d'être compa-

d. https://github.com/ethan0911/TB-AMR/tree/NASA_TAMR
e. https://github.com/owl-project/owlExaBrick

tible avec l'approche "par domaine" de l'outil d'analyse et donc d'avoir une bonne scalablilité avec l'application. Si le volume de données augmente, il suffit de rajouter des *threads* ou des processus `MPI`. On reviendra sur ce point par la suite. Notre objectif ici est de produire des cartes par lancer de rayon depuis un "observateur" qui est le plan $\mathcal{P}$ discrétisé, comme on l'a vu précédemment. Le point de départ de tous les rayons est le centre en coordonnées `3D` cartésiennes des pixels. Il est important de noter que les rayons se propagent depuis $\mathcal{P}$ suivant sa normale $\vec{n}$, et rentre donc dans la catégorie des méthodes de type *raycasting*. On notera qu'on ne tient pas compte de l'opacité des matériaux, le rayon se propagera donc à travers l'ensemble des cellules sans limite de profondeur. Avec cette méthode, on pourra donc produire des cartes de niveau minimum ou maximum le long d'une ligne de visée, calculer une grandeur par intégration le long des rayons (carte de densité de surface, vitesse pondérée, etc.). Ces cartes sont indispensables pour les analyses que l'on fera de la simulation `ExaMilkyWay` au chapitre 6.

Afin d'accélérer le traitement des rayons, on met en place plusieurs stratégies et optimisations. En effet, le nombre de rayons peut atteindre rapidement plusieurs dizaines de millions. Typiquement, les cartes `2D` que l'on produira par la suite seront constituées de 8192^2 pixels, soit un peu plus de 67 millions de rayons. Calculer les intersections pour chacun des rayons avec toutes les cellules du maillages, même sur une `ROI` de petite taille, n'est pas une option. En effet, à la fin de cette section, on montrera quelques tests de performances, pour produire des cartes 8192^2 dont le maillage contient $31 \cdot 10^9$ cellules (noeuds terminaux plus feuille), ce qui fait environ $\sim 2 \cdot 10^{18}$ intersections. L'objectif principal de tous les algorithmes de lancer de rayon est de diminuer la quantité d'intersection à calculer. Il existe de très nombreuses approches, parmi lesquels on retrouve la construction de structures d'accélération, appelées `BVH` (*Bounding Volume Hierarchy*). Ce type de structure permet de regrouper géométriquement les régions pour diminuer le nombre d'intersection à calculer, voir figure 4.17. Sur cette exemple, il suffit de calculer récursivement si un rayon intersecte la région $\mathcal{A}$, et de descendre dans la structure en itérant sur les régions contenues dans `A`, etc. On notera que cette structure peut se représenter sous la forme d'un arbre. Les bibliothèques comme `Optix` ou `Ospray` utilisent des `BVH` en interne.

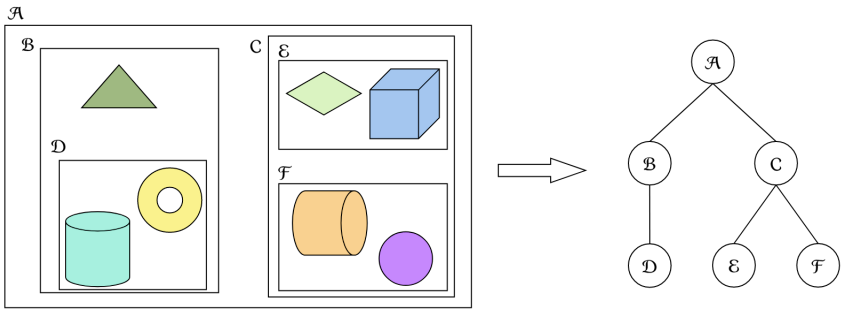

FIGURE 4.17 – Exemple de structure d'accélération, de type `BVH` regroupant les objets en fonction de leur position dans l'espace afin de diminuer le nombre d'intersections à calculer par l'algorithme de lancer de rayons.

On voit naturellement qu'un maillage `AMR`, dans sa description en arbre, au modèle `lightAMR`, par exemple, pourrait être utilisé nativement comme `BVH` pour éviter le calcul de l'intersection avec toute la descendance d'un noeud, s'il n'est pas intersecté par un rayon. Afin d'être plus performant et d'éviter de se déplacer dans l'arbre, on utilisera plutôt le `PCM` (voir section 4.5), sous la forme d'une liste cellule, comme `BVH`. Avec l'utilisation du pavage cubique, dans l'exemple que l'on a mentionné précédemment, on gagne environ 4 ordres de grandeur sur le nombre d'intersections à calculer, le réduisant ainsi, à environ $2 \cdot 10^{14}$. Pour aller plus loin, il est possible de tirer profit du fait que les rayons se propagent parallèlement au plan, suivant sa normale. Ainsi, il est possible, comme on le montre sur la figure 4.18, de pré-calculer pour chaque bloc du pavage l'intervalle de coordonnées (i_{min}, i_{max}) et (j_{min}, j_{max}) des pixels qui intersecteront le bloc. On notera qu'on approxime le bloc par une sphère parfaite circonscrite. Il est compliqué d'évaluer le gain, malgré tout conséquent, de cette approche, parce qu'elle dépend de la taille de l'image, de la taille des blocs, dépendante du niveau `AMR` et de la `ROI`. On peut raisonnable estimer, après quelques tests, que le gain est d'environ 4 ordres de grandeurs.

Une fois que la liste des rayons est déterminée, on évite de calculer pour chaque rayon l'intersection avec toutes les cellules contenues dans le bloc. On exploitera la structure sous forme de table de hachage des cellules du domaine. En effet, connaissant le point d'intersection du rayon avec le bloc du pavage cubique (et donc la face d'en-

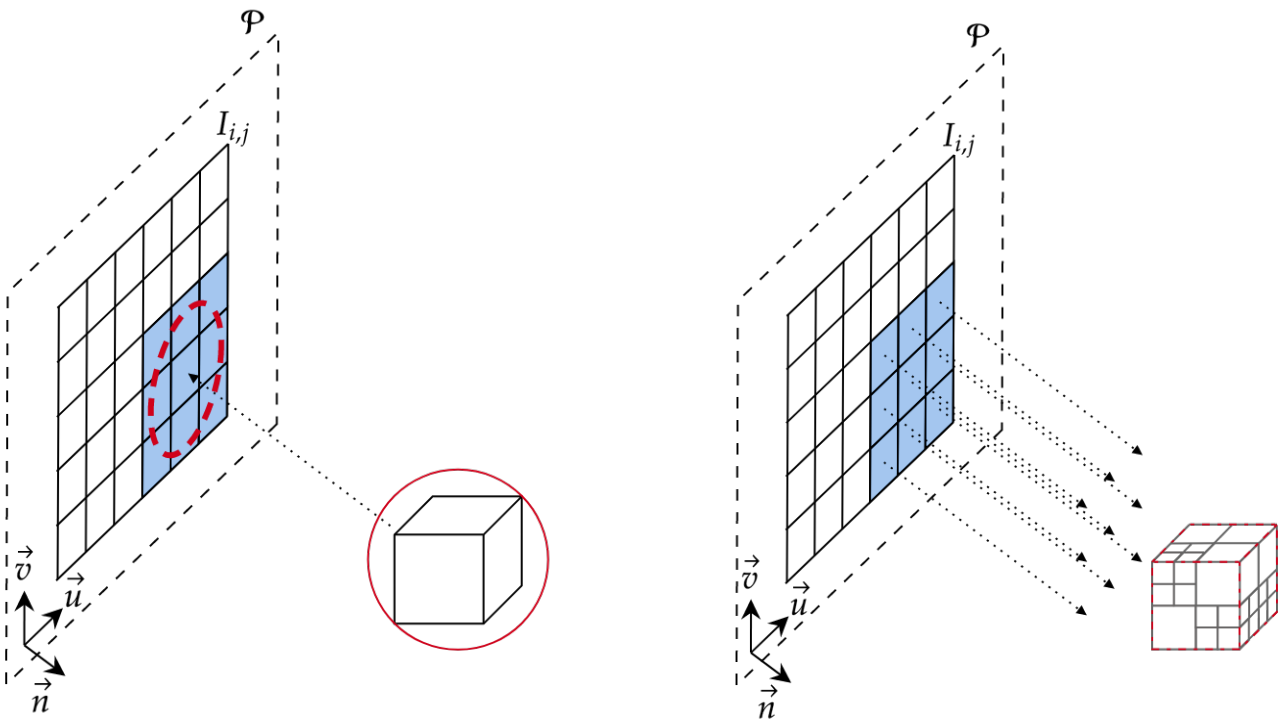

FIGURE 4.18 – A gauche, le calcul de la liste des rayons (issu des pixels) qui vont intersecter le bloc du `PCM`, lors de la projection parallèle suivant la normale du plan $\mathcal{P}$. Le bloc est approximé par une sphère parfaite circonscrite au bloc. A droite, on ne calcule que les intersections de ces rayons avec le bloc.

trée du rayon), on pourra faire une recherche dans la table de hachage de la cellule feuille (ou du noeud terminal), qui sera le point d'entrée du rayon. On note que les coordonnées logiques et le niveau `AMR` du bloc sont connus. Ensuite, il suffira de calculer le point de sortie du rayon (et donc la face) pour calculer les coordonnées de la cellule voisine à chercher dans la table. Ainsi, on se déplacera le long du rayon jusqu'à sortir du bloc, voir figure 4.19. Il y a plusieurs points importants à relever. Premièrement, cette approche implique donc d'utiliser une structure dans le conteneur global sous la forme de table de hachage (qui sera en lecture seule, facilitant le *multithreading*). Deuxièmement, le fait de se déplacer le long du rayon, par recherche des cellules voisines, permet de s'affranchir du problème d'échantillonage, [Wang et al., 2020]. Certains algorithmes font de l'échantillonnage à intervalle régulier le long du rayon, mais avec les maillages adaptatifs, cette approche n'est pas adaptée et génère soit un sur-échantillonnage, soit un sous-échantillonnage. Troisièmement, on a mentionné à la section 4.5 que la balance 2 :1 (pas plus d'un niveau entre deux cellules voisines) était respectée par le `PCM` au sein des blocs. Cette propriété permet d'accélérer la recherche des cellules dans la table de hachage en se déplaçant le long du rayon. En effet, après avoir déterminé les coordonnées et le niveau de la cellule d'entrée, les cellules suivantes auront au plus un niveau d'écart. On rappelle que les clés de recherche sont composées des coordonnées logiques et du niveau (le niveau est une inconnue).

Enfin, on a mentionné à la section 4.6, que le filtrage géométrique du maillage pouvait impacter l'algorithme de lancer de rayon. En effet, le fait de se déplacer le long du rayon de cellule en cellule implique nécessairement qu'il ne peut pas y avoir de trou au sein du bloc. Il est donc conseillé de faire un filtrage géométrique large pour éviter les artefacts. D'autre part, la *contribution* à la valeur du pixel apportée par une cellule intersectée par le rayon est paramétrée depuis le fichier de configuration. Elle peut être du type : `add`, `min`, `max`, `integration`, correspondant respectivement à une addition de la valeur du champ scalaire (il s'agit alors d'un équivalent à une fonction de *binning*), une opération `min` ou `max` (entre la valeur du pixel et d'un champ scalaire de la cellule), une intégration par la longueur du rayon dans la cellule traversée.

On rapporte sur la figure 4.20 des tests de performances de l'algorithme sur les données de la simulation `ExaMilkyWay` à haute résolution ($t = 19\ Myr$, voir chapitre 6). Ces tests sont effectués au `TGCC` sur la partition `ROME` disposant de 2 `CPU AMD EPYC` de 64 coeurs à 2.6 Ghz. On produit 6 cartes (avec différents types de contribution), chacune de 8192^2 pixels dans une `ROI` contenant tout le disque galactique ($24\ kpc \times 24\ kpc$), soit environ 31 10^9

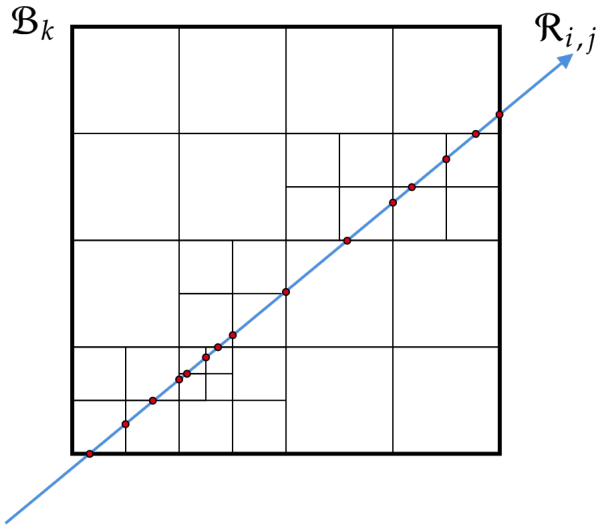

FIGURE 4.19 – Intersection d'un rayon issu d'un pixel (i, j) avec un bloc $\mathcal{B}_{\|}$ du pavage cubique. Les cercles rouges sont les intersections avec l'ensemble des cellules du pavage.

cellules (noeud terminaux plus feuilles). On utilise 48 processus `MPI` et on fera varier le nombre de *threads* entre 1 et 24. On précise que l'algorithme parcourt tous les domaines à traiter et accumule la contribution de chacun d'entre eux à l'image finale. On tire également profit de l'alignement des blocs du `PCM` avec les axes pour utiliser une méthode de calcul de l'intersection cube-rayon efficace : rayon-`AABB` (Axis-Aligned Bounding Box), [Williams et al., 2005].

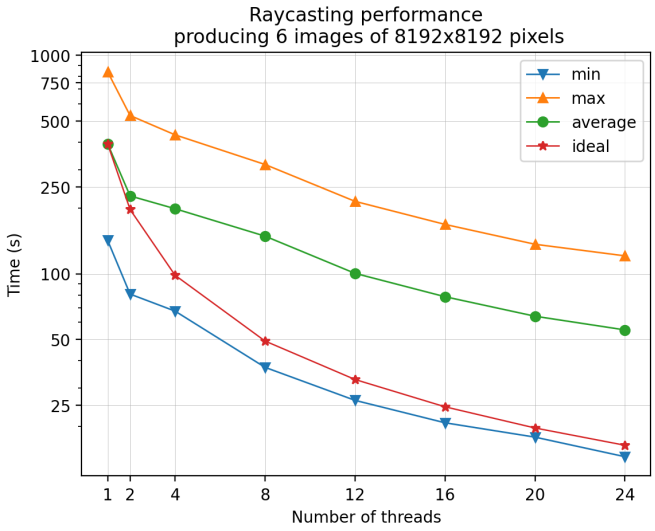

FIGURE 4.20 – Performance de l'algorithme de lancer de rayon, pour produire 6 images de 8196^2 pixels de la simulation `ExaMilkyWay` à haute résolution sur 48 processus `MPI`. On fait varier le nombre de tâches `openMP` par processus pour produire 6 images. Le nombre de cellules à traiter est de $\sim 31\ 10^9$ cellules.

Sur ce graphique on constate que l'écart entre le processus `MPI` le plus lent (courbe orange) et le plus rapide (courbe bleue) peut être significatif, et reste constant à partir de 8 *threads*. Contrairement à l'exemple qu'on a donné à la section 4.3, le parallélisme par tâche, est mis au niveau de l'intervalle des pixels au lieu des domaines à traiter. L'intérêt est d'éviter d'avoir de la concurrence en écriture pour mettre à jour les valeurs dans la carte. En effet, plusieurs blocs ou domaines pouvant contribuer aux mêmes pixels. La scalabilité en *threads* de l'algorithme n'a pas encore fait l'objet d'une optimisation. Néanmoins, il est possible de déplacer le parallélisme et grâce à des

pré-calculs, limiter ou retirer la concurrence d'accès en écriture, quitte à augmenter la consommation mémoire. Enfin, on notera que pour produire 6 cartes, il faut compter un peu plus de 2 minutes à 20 *threads*, ce qui permet largement d'avoir le dynamisme nécessaire dans la phase d'analyse. Enfin, il faut compter environ 3min30 au total (20 ou 24 *threads*) pour obtenir le résultat, toutes opérations confondues : lecture, décompression, construction du `PCM`, production des cartes, écriture des cartes avec `HDF5`.

4.8 Impact du *downcasting* et de la compression *avec perte*

Dans cette section, on propose de regarder l'impact de la compression de données *avec perte* ainsi que du *downcasting* sur des images produites par lancer de rayons. Pour ce faire, on produit deux bases de données `HDep`. Une contenant les champs scalaires en double précision sans compression et une base de données avec les champs en précision réduite (32 bits) et avec de la compression *avec perte* avec une erreur absolue maximale de $2.44 \cdot 10^{-4}$ ($n_{ovrr} = 12$). Sur la figure 4.21 de gauche, on montre l'image produite à partir du champ de densité sans compression par la méthode de lancer de rayons. Cette image permet d'avoir une idée du contenu. Sur l'image de droite, on montre l'erreur relative sur les données brutes entre la carte `2D` issue de la densité non compressée et la carte `2D` issue du champ de densité réduit en simple précision et compressée *avec perte*. L'erreur fluctue donc entre $[-0.024, +0.024]$ %. Afin de mieux visualiser l'impact de la compression, l'erreur relative est multipliée par un facteur 10^6. Comme présenté au chapitre précédent, l'erreur est donc centrée autour de zéro. On constate que dans les zones à faible contraste de densité l'erreur est faible et proche de 0 alors que dans les zones à fort contraste, l'erreur fluctue entre les bornes.

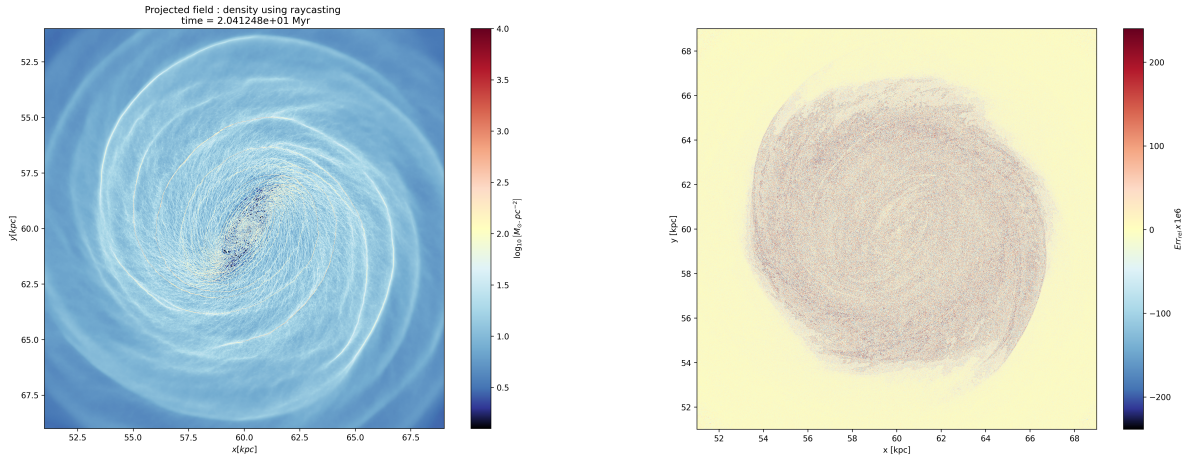

FIGURE 4.21 – A gauche, une carte de densité surfacique de la simulation *ExaMilkyway* à $t \simeq 20\ Myr$ à partir de données double précision sans compression par la méthode de lancer de rayon. A droite, l'erreur relative (multipliée par 10^6 pour plus de contraste) entre deux cartes : densité double précision sans compression et densité simple précision compressé avec perte avec une erreur relative $<= 2.4\,10^{-4}$.

On peut faire les même types de tests avec les différentes analyses disponibles : flou gaussien, histogrammes `1D` ou `2D`, calculs de grandeurs dérivées, spectres de puissance de carte `2D`, etc. Par souci de brièveté, on n'inclura pas ici tous les résultats pour tous les types d'analyses. Les tests effectués dans le cadre de ces travaux de thèse ont montré que pour toutes les analyses qui sont faites au chapitre 6, l'impact de l'erreur de compression est négligeable et n'impacte pas les conclusions scientifiques obtenues.

4.9 Performances

4.9.1 Équilibrage de charge

Compte tenu de l'approche de traitement proposé par l'outil, l'équilibrage intervient aux deux niveaux du parallélisme. En effet, il y a d'un côté l'équilibrage de la liste des domaines à traiter par processus `MPI`, et de

l'autre côté le nombre de cellules par domaine. Si on prend les données d'une simulation comme `Extreme-Horizon` ou `ExaMilkyWay`, il peut y avoir un facteur ~ 10 de différence sur le nombre de cellules entres les domaines. Néanmoins, puisque les analyses sont généralement effectuées avec beaucoup moins de processus que la simulation, les domaines s'équilibrent relativement bien. Par exemple, un post-traitement dans le but de produire une carte entière de la simulation `Extreme-Horizon` par lancer de rayons sur 8 processus `MPI` s'équilibre assez bien, puisque chaque processus devra gérer environ $2.5\ 10^9$ cellules. On notera néanmoins que lorsqu'on utilise des régions d'intérêt et donc un traitement sur une sous-partie des domaines, un déséquilibre entre les processus peut apparaître.

Au sein des processus, le déséquilibre des domaines implique d'utiliser la directive `openMP` *dynamic* afin de laisser la possibilité à la bibliothèque de gérer la répartition des domaines entres les *threads* de façon dynamique durant l'exécution. Bien que cette approche très simple fonctionne particulièrement bien dans la plus part des analyses : calcul de grandeur dérivée, histogramme 1D/2D, accumulation de grandeur par secteurs, etc. ; elle peut être source de ralentissement sur certains algorithmes plus compliqués.

4.10 Visualisation 3D

Durant le chapitre sur le modèle `lightAMR`, on a discuté de la compatibilité du modèle avec la structure de données proposée pour la gestion de l'`AMR` en arbre par le package de visualisation `VTK`. On a évoqué notamment les problèmes liés au découpage de domaines et aux arbres non complets par domaine produits par `RAMSES`. De plus, l'outil d'analyse proposé ici, a pour vocation d'être un outil d'analyse des données uniquement et non pas de visualisation des données. Néanmoins, la visualisation en `3D` des données permet de mieux les appréhender. L'objectif principal des `vtkHyperTreeGrid` est de combler le manque de structure disponible, en visualisation, pour gérer les (gros) maillages `AMR`. J'ai donc implémenté la construction de `vtkHypertreeGrid` au sein de l'outil d'analyse. En effet, il est possible, à partir des données au modèle `lightAMR` ou de structures contenant l'enveloppe feuille, de reconstruire un `vtkHypertreeGrid`. Il est alors possible d'utiliser les fonctions d'`E/S` disponibles dans le package `VTK` pour exporter au format `XML` (`vtkHyperTreeGridXMLWriter`) la structure et ainsi de produire un fichier `.htg` compatible avec `Paraview`. Cela permet d'étendre les fonctionnalités disponibles pour les utilisateurs.

Cette fonctionnalité de conversion permet donc de tirer partie de toutes les options de pré-filtrage (filtrage des domaines, filtrage du maillage) pour extraire une zone à visualiser, tout en bénéficiant des développements faits au sein de `Paraview` pour la visualisation. Il faut bien évidemment tenir compte des problématiques évoquées concernant le découpage de Hilbert au niveau le plus fin fait par `RAMSES`. De plus, la structure reconstruite est donc crée depuis une cellule racine unique. A l'avenir, j'ai prévu d'étendre cette reconstruction afin de déterminer un niveau de base à partir duquel construire l'`HyperTree`, et avoir une structure plus proche de ce qui est attendu. En effet, les filtres appliqués sur les `vtkHyperTreeGrid` tels que *clipping* (découpage d'une région), *slicing* (extraction d'un plan dans le maillage), *threshold* (filtrage par valeur), etc, sont optimisées pour gérer une collection d'arbres issus de cellules définies sur une grille régulière (voir section sur la compatibilité du modèle `lightAMR` au chapitre précédent et les notions de `HyperTreeGrid`, `HyperTree`).

On montre le résultat obtenu sur la figure 4.22, d'une région ellipsoïdale issue d'une base `HDep` de la simulation `ExaMilkyWay`, au début de la montée en résolution, dont un `vtkHyperTreeGrid` a été construit et exporté suivant le modèle de données `htg` au format `XML`. Le fichier a ensuite été relu avec `Paraview` (version 5.11). Après un filtrage par valeur (`vtkHyperTreeGridThreshold`) sur le niveau `AMR`, on ne conserve que les niveaux dans l'intervalle $[13, 17]$. On rappelle que la structure a été construite depuis le niveau 0. On applique alors deux filtres `vtkHyperTreeGridSlicer`, afin d'afficher deux coupes dans le maillage, dans le plan (x, y) et une dans le plan (x, z). On affiche alors le niveau `AMR` des cellules.

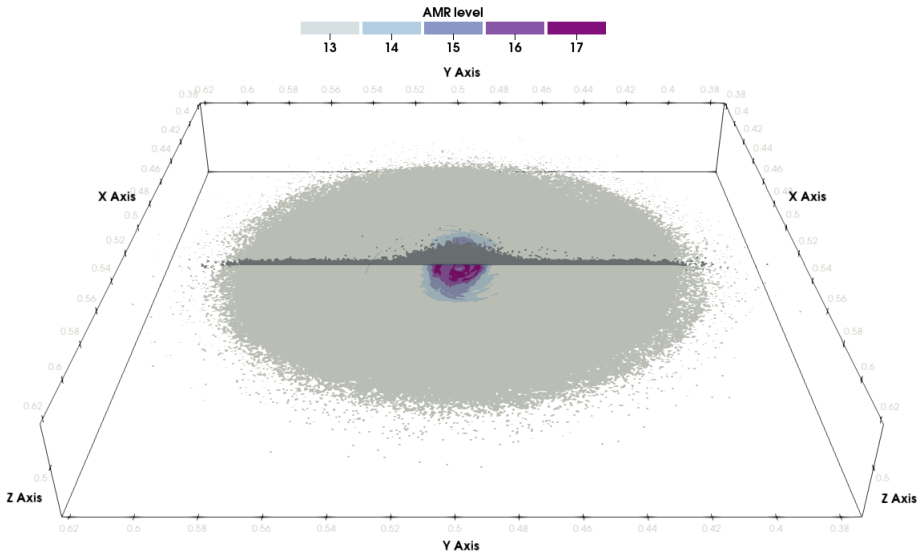

FIGURE 4.22 – Visualisation `3D` avec *Paraview* de coupes dans le plan équatorial et vertical d'une galaxie simulée dont le maillage est décrit avec un *vtkHyperTreeGrid*.

Il est très important de noter que le maillage contient environ $\sim 235 \cdot 10^6$ cellules et ne pèse que 2.5 *Go*. Grâce, au `vtkHyperTreeGrid`, ce maillage a pu être visualisé sur un ordinateur de bureau (Intel(R) Core(TM) `i5-7200U CPU` @ 2.50GHz, 16 *Go* de RAM). Son équivalent *non structuré* ne serait tout simplement pas visualisable sans utiliser des *fat node* de visualisation sur un cluster de calculs. Les `vtkHyperTreeGrid` constituent donc un atout important pour la visualisation de données `AMR`. Néanmoins, il y a encore certaines limites avec cette approche (conversion et visualisation). Premièrement, il n'y a pas de gestion d'`E/S` parallèles au sein des fonctions d'écriture de `VTK` pour ce format. Il faut donc faire la conversion avec un seul processus. Cependant, puisque l'outil d'analyse utilise un parallélisme hybride, il suffit d'utiliser plusieurs *threads*. Ainsi, la lecture de données sera purement séquentielle, mais la sélection des domaines, la décompression des données et l'extraction de l'enveloppe feuille se feront avec plusieurs *threads*. On notera cependant que la construction du `vtkHyperTreeGrid` ne peut être faite que de manière séquentielle et constituera donc la plus grosse partie du temps de conversion. Deuxièmement, les filtres disponibles (*slice*, *threshold*, *Clip*, etc ;) ne tirent pas profit du *multithreading*. En fonction de la taille de l'arbre, ces filtres peuvent être très coûteux en temps de calcul. Il est important de noter que compte tenu des fonctionnalités de déplacement dans l'arbre avec les curseurs, l'utilisation de *multithreading* avec un *hyperTreeGrid* défini depuis une unique cellule au niveau 0, n'est pas évident. Enfin, on notera que des améliorations et de nouveaux filtres sont en cours de développement au sein de `Kitware` avec ses partenaires académiques, dont pourront bénéficier les utilisateurs. Par exemple, les iso-contours sur les `HyperTreeGrid` sont en cours de développement et permettront de faire du rendu volumique.

4.11 Partage de données et science ouverte

La base de données `Galactica`, développée au `CEA`, est composée d'une suite logicielle cohérente permettant de facilité la diffusion de données scientifiques issues de simulations numériques en astrophysique. Elle est constituée de trois organes essentiels : une application web `Galactica`[f], un package `Python astrophysix`[g] et un package `Python Galactica-terminus`[h]. L'application web est l'interface principale, visible par les utilisateurs. A partir de cette application, on peut naviguer parmi les différents projets disponibles, accéder aux méta-informations (code utilisé, paramètres, modèles physiques, etc.) mais également à des données réduites mis en ligne par les porteurs du projet (images, graphiques, catalogues d'objets). Les utilisateurs authentifiés peuvent soumettre des requêtes de traitement sur des données brutes de simulations. Les traitements sont prédéfinis et rendus disponibles

f. http://www.galactica-simulations.eu/db/
g. https://pypi.org/project/astrophysix/
h. https://pypi.org/project/galactica-terminus/

par les porteurs de projet. Par exemple, si une méthode de production de carte `2D` est disponible, un utilisateur peut demander à faire une carte avec certains paramètres précis parmi une liste d'options et de paramètres eux aussi rendus disponibles.

Le package `astrophysix` permet de faciliter l'importation et le téléversement automatique de projets scientifiques sur la base de données de `Galactica`. Ainsi, les astrophysiciens peuvent documenter, hors ligne, leur projet dans une *étude astrophysix*. A l'exécution du script, un fichier au format `HDF5` standardisé est produit et pourra être téléversé sur l'application web qui déploiera automatiquement de nouvelles pages (ou mettra à jour le contenu de pages existantes) relatives au projet et aux simulations et/ou aux résultats associés. Un script de quelques lignes suffit à créer un projet de simulation numérique pour le publier sur `Galactica`.

Le package `galactica-terminus` permet quant à lui de déployer un serveur de traitement de données brutes de simulations appelé `Terminus`. On a vu que les données réduites (images, courbes, spectres, catalogues, etc) sont hébergées directement sur des disques de stockage du serveur web `Galactica` et n'occupent que quelques gigaoctets. Les données brutes de simulations sont elles, beaucoup plus volumineuses, de l'ordre de quelques dizaines de gigaoctets à plusieurs pétaoctets en fonction des simulations. Pour diffuser ces données massives et intransférables sur le réseau et en faire bénéficier l'ensemble de la communauté scientifique, une architecture de traitement de données distribué et asynchrone a été mise en place autour de `Galactica`. Si l'application web, accessible depuis un navigateur internet, constitue le `frontend` de la base de données `Galactica`, les serveurs `Terminus` constituent son `backend`. Il s'agit d'une fine surcouche basée sur le paradigme de gestion de tâches distribuées et asynchrones `Celery`[i]. Ainsi, de multiples instances de ces serveurs peuvent être déployées au plus près des données, au sein des instituts de recherche en astrophysique qui les hébergent. Tous ces serveurs communiquent avec l'application web `Galactica` pour permettre de générer à distance, directement depuis l'interface web de `Galactica`, des requêtes de traitement "à la demande", répondant aux besoins spécifiques des utilisateurs.

4.11.1 Contribution au package *galactica-terminus*

Les instances des serveurs `Terminus` visent à être déployées sur n'importe quelle machine disposant de ressources de stockage et de calcul, et hébergeant des données de simulations à diffuser. Les machines ciblées vont des grands centres de calculs, aux stations de travail des physiciens, en passant par des clusters d'analyse des instituts de recherche en astrophysique. On notera également que ce serveur peut être intégré à un environnement de soumission de *job* de type `SLURM`.

Pour des raisons de sécurité, les serveurs de traitement de données distants ne sont pas directement connectés au serveur de l'application web mais communiquent avec lui via un système de routage de messages de type `RabbitMQ`[j], implémentant le protocole `AMQP` et dont la scalabilité a été largement démontrée par une large communauté d'utilisateurs. Ainsi, lorsqu'un utilisateur soumet une requête d'analyse d'un jeu de données, l'application web envoie un *message* contenant les informations de la requête (jeux de données cibles, type de traitement, paramètres du traitement, etc.) au serveur `Terminus` correspondant au traitement à effectuer. De son côté, le serveur de traitement `Terminus` dépile les requêtes reçues dans sa boîte de messagerie et lance les *jobs* d'analyse avec les paramètres souhaités. Durant ces processus, des notifications sont envoyées à l'application web pour informer l'utilisateur de l'état d'avancement de la requête. Une fois les analyses terminées, les résultats sont transférés à l'application web qui les rend disponibles à l'utilisateur et l'informe par mail du bon déroulé de sa requête. On notera que compte tenu du fait que les résultats sont transférés par le réseau à l'application web et stockés temporairement (un système de nettoyage périodique permet de libérer de l'espace disque), les résultats des *jobs* d'analyses ne doivent pas être trop volumineux. Il s'agit généralement de données réduites à haute valeur ajoutée : cartes 2D, spectres, cube de données 3D, graphiques 1D, etc.

Avec ces serveurs *Terminus*, tous types de services de traitement de données peuvent être proposés afin de générer des produits à haute valeur ajoutée pouvant intéresser la communauté. La seule condition à respecter est que les services d'analyse doivent pouvoir être exécutés depuis une couche d'appel `Python`. On notera qu'il est possible de rendre disponible des bibliothèques d'autres langages tels que le C/C++, si celle-ci dispose d'une interface

i. https://docs.celeryq.dev
j. https://www.rabbitmq.com/

Python. Avec ce type d'approche et le mode de fonctionnement des serveurs Terminus, il est possible d'exploiter les données de simulations issues de n'importe quel type de code de calcul, peu importe le format des données. En effet, l'utilisateur de Galactica n'a pas besoin de connaître le format de données puisque c'est le responsable de la simulation, qui lui maîtrise le format de données, qui met à disposition de la communauté ses propres scripts d'analyse. La généricité et la grande versatilité de cette architecture de traitement de données distant permet de couvrir tous les besoins de la communauté scientifique qui requiert d'accéder à des modèles théoriques issus de simulations numériques en astrophysique, en accord avec les principes de *Science Ouverte*.

Durant mes travaux de thèse, j'ai effectué le portage en Python3 d'un premier prototype de serveur Terminus initialement développé en Python2.7, j'ai développé le système de configuration et d'installation d'un serveur Terminus pour faciliter son déploiement au sein des instituts de recherche en astrophysique et j'ai largement contribué à la rédaction de la documentation en ligne (https://galactica-terminus.readthedocs.io/) disponible pour aider à son installation et son utilisation.

4.12 Conclusion et perspectives d'évolutions

L'outil d'analyse a été développé pour traiter de manière efficace des données au format lightAMR. On a détaillé l'approche hybride du parallélisme où chaque processus MPI gère une collection de domaines à traiter, pouvant être pré-filtrée grâce au pavage cubique minimal, réduisant ainsi la charge E/S. Les structures de données ont été pensées de manière à pouvoir traiter la liste des domaines d'un processus avec un parallélisme en mémoire partagée avec les directives OpenMP. En effet, cette liste sera répartie dynamiquement entre les différents *threads* qui vont ainsi traiter les domaines séparément. On notera que cette approche peut être limitée en terme de scalabilité si les domaines à traiter deviennent très volumineux. En effet, le nombre de domaine par processus chutera, et donc la scalabilité en nombre de *threads* deviendra bloquante. Une solution consiste à construire, directement dans le conteneur global, la liste complète des cellules de tous les domaines à traiter, plutôt que de la séparée par domaine. Il faudra ensuite déplacer le parallélisme OpenMP pour traiter cette liste en mémoire partagée.

On a détaillé le système de *batch* implémenté qui permet d'effectuer la plupart des analyses avec des ressources de calcul limitées et montré qu'il est possible de faire du recouvrement des E/S avec la phase de calcul, grâce à l'utilisation de fonctionnalités avancées en C++. Enfin, on a détaillé 3 méthodes de production de carte 2D : tranche, flou gaussien et lancer de rayon. On a vu que le pavage cubique minimal pouvait être utilisé comme structure d'accélération pour réduire significativement le nombre d'intersections à calculer dans l'algorithme lancer de rayon et ainsi optimiser ses performances. On notera cependant qu'on ne tire pas profit de potentielles accélérations matérielles comme celles apportées par des bibliothèques dédiés Optrix ou Ospray, qui pourraient encore améliorer les performances de l'algorithme, même si son objectif n'est pas de faire du rendu en temps réel. On notera également qu'on ne tient pas compte de l'opacité des matériaux traversés ou de la longueur d'intégration des rayons. Ces fonctionnalités font parties de futures améliorations.

Enfin, on rappelle que l'outil d'analyse développé est écrit en C++ et que les données produites sont brutes (en unité "code") et impliquent donc une deuxième phase de traitement haut niveau (script python par exemple). Afin d'apporter plus de souplesse et de dynamisme dans les phases exploratoires des analyses, une interfaçage avec la bibliothèque Pybind11 est en cours de développement.

D'autre part, cet outil a été développé dans l'objectif du traitement de grands volumes de données post-mortem, i.e. : après écriture des données sur disque. Néanmoins, les fonctions de lectures sont totalement séparées des fonctions d'analyses, il serait donc relativement facile de l'incorporer dans un écosystème plus complexe pour pouvoir proposer un traitement "à la volée" des données et ainsi s'inscrire dans une approche *in-transit*. De plus, la cible matérielle de cet outil est le CPU. Il existe des *framework* tel que Kokkos, [Trott et al., 2022], qui permettent de faciliter les optimisations bas niveau et le portage des codes sur d'autres architectures tel que le GPU. Dans cet outil, les structures de données sont également séparées dans un module à part, il est donc envisageable de pouvoir y intégrer ce type de bibliothèque. Néanmoins, Kokkos est particulièrement efficace lorsqu'il faut parcourir de très grands tableaux continus en mémoire. Avec l'approche par domaine qu'on propose ici, il n'est pas possible de remplacer les directives OpenMP par des boucles Kokkos. En effet, les noyaux sont const, i.e. : sans modification des structures. Il faudrait donc, comme on l'a mentionné précédemment, accumuler la liste complète des cellules à traiter plutôt que la séparer par domaine. Enfin, on notera qu'afin de pouvoir cibler à la fois une architecture à base de CPU ou

de `GPU`, le *framework* de `Kokkos` se base sur la cible la plus contraignante, le `GPU`. En conséquence, cette approche apporte plus de rigidité dans les développements. Par exemple, les allocations de mémoire doivent être faites en amont, et donc il peut être nécessaire d'effectuer, souvent, de nombreux pré-calculs. On notera, cependant, que pour l'instant, il n'y a pas de besoin réel en terme de gain de performance qui justifierait l'investissement nécessaire pour ces développements, ni l'utilisation de `GPU` pour les analyses.

Troisième partie

Simulation ExaMilkyWay

Chapitre 5

L'environnement galactique

Dans ce chapitre, on détaillera les différentes structures et processus astrophysiques à l'échelle d'une galaxie. L'objectif est d'introduire les notions importantes qui seront utilisées ou modélisées au chapitre suivant. On commencera par introduire les galaxies, le milieu interstellaire ainsi que sa dynamique, et le rôle important des étoiles. On s'intéressera de plus près aux étoiles et plus précisément aux processus d'activation ou de régulation de leurs formations, ainsi que leurs rétro-actions dans le milieu qui les entoure. Dans un deuxième temps, nous introduirons la turbulence dans le milieu interstellaire.

5.1 Les galaxies

Les galaxies sont des objets astrophysiques composés de matière visible tels que du gaz, de la poussière et des étoiles, mais également de la matière noire, invisible. Cette dernière, dont les premiers indices d'existence remontent à 1933 lors de calculs de masses, n'est détectée, encore à l'heure actuelle, qu'au travers de ses effets gravitationnels. Elle représente pourtant environ 90% de la masse de notre galaxie, La Voie Lactée. Elle forme ainsi la base du potentiel gravitationnel dans lequel évolue la matière visible au sein de la galaxie, dans le milieu interstellaire. C'est cette matière visible, par le biais de la formation stellaire et des processus de rétroaction sur le milieu interstellaire, qui dessinera l'aspect des galaxies. La morphologie des galaxies est classifiée par la séquence de Hubble. Dans cette classification, on retrouve notamment des galaxies elliptiques et lenticulaires et les galaxies spirales avec ou sans barre. On s'intéressera plus particulièrement aux spirales barrées dont fait partie La Voie Lactée. Ce type de galaxie représente environ les 2/3 des galaxies spirales. On donne un exemple la galaxie `M83` vue par le télescope spatiale Hubble, sur la figure 5.1.

FIGURE 5.1 – Galaxie `M83`, spirale barrée vue par le télescope spatiale Hubble, crédit : `NASA/ESA/HTT`.

La barre est en fait une structure stellaire à laquelle le gaz en rotation répondra. En effet, le gaz formera une onde de densité en rotation associée à une perte de moment angulaire dans les régions centrales du disque. Parmi les processus à l'origine de cette perte de moment, il y a les instabilités gravitationnelles induites par les bras spiraux et la barre naissante, qui vont transférer le moment vers les régions extérieures du disque, [Bournaud et al.,

2010]. Ou encore, la friction dynamique de la barre avec le halo de matière noire, [Athanassoula and Misiriotis, 2002]. Les barres vont donc influer sur la dynamique du milieu mais aussi sur la formation stellaire en déplaçant de larges quantités de gaz soit vers le centre galactique, soit vers l'extérieur. En effet, lorsque la barre arrive vers le gaz des régions extérieures qui se déplace plus vite, la gravité générée par la barre, le ralentira et le fera tomber vers le centre. Grâce au processus que l'on décrira dans les sections suivantes, des pics de formation stellaire peuvent ainsi avoir lieu dans les régions centrales.

5.2 Milieu interstellaire

La matière noire, non collissionnelle, est responsable de la plus grande partie de la masse des galaxies et permet de définir le champ gravitationnel dans lequel évoluera la matière baryonique qui ne représente qu'environ $\sim 10\%$ de la masse. Au sein d'une galaxie, on retrouve de nombreux processus agissant sur de très grandes échelles spatiales et temporelles. Parmi les processus les plus impressionnants et impactants, on retrouve la formation des étoiles et leurs rétro-actions dans le milieu qui les entoure. Pour mieux appréhender ces processus, il faut s'intéresser au milieu dans lequel baigne les étoiles : le milieu interstellaire (`MIS`). Plus précisément, le `MIS` fait référence à la matière contenue dans l'espace entre les étoiles. Cette matière est constituée à 99 % de gaz et 1% de poussière. Le gaz, dont la composition est essentiellement de l'hydrogène ($\sim 73\%$) et de l'hélium ($\sim 25\%$), est très diffus avec un densité moyenne de l'ordre de quelques atomes par centimètre cube. Néanmoins, le `MIS` est loin d'être un milieu homogène. Le gaz qu'il contient est soumis à différents processus de chauffage ou refroidissement. Ainsi, il peut se retrouver dans différentes phases en température et en densité [Mihalas and Binney, 1981], [Kulkarni and Heiles, 1988] :

- **ionisé très chaud** (`HIM`, *Hot Ionized Medium*), $T \simeq 10^6\,K,\ \rho \simeq 10^{-3}\,H \cdot cm^{-3}$, produit par les explosions de supernovae, fortement hors équilibre, avec un temps caractéristique de refroidissement long, [McKee and Ostriker, 1977],
- **ionisé chaud** (`WIM`, *Warm Ionized Medium*), $T \simeq 8 \cdot 10^3\,K,\ \rho \simeq 2 \cdot 10^{-1} - 5 \cdot 10^{-1}\,H \cdot cm^{-3}$, photo-ionisé par les étoiles massives `OB`, [Ferguson et al., 1996],
- **neutre chaud** (`WNM`, *Warm Neutral Medium*), $T \simeq 6 \cdot 10^3 - 1 \cdot 10^4\,K,\ \rho \simeq 2 \cdot 10^{-1} - 5 \cdot 10^{-1}\,H \cdot cm^{-3}$, associé à une des phases stables du gaz atomique d'hydrogène. Un des traceurs les plus importants pour mettre en évidence cette phase (et la suivante) est donc la raie d'absorption à 21 cm de l'hydrogène `HI`,
- **froid neutre** (`CNM`, *Cold Neutral Medium*), $T \simeq 50 - 100\,K,\ \rho \simeq 2 \cdot 10^1 - 5 \cdot 10^1\,H \cdot cm^{-3}$, dans ce deuxième état stable du gaz d'`HI`,
- **moléculaire** (`MM`, *Molecular Medium*), $T \simeq 10 - 20\,K,\ \rho \simeq 10^2 - 10^6\,H \cdot cm^{-3}$.

Il faut voir ces différentes phases comme des états possibles du gaz à un instant t. En réalité, le gaz change d'état pour différentes raisons : ionisation par des photons d'étoiles, refroidissement par radiation, recombinaison d'ions et d'électrons, recombinaison d'hydrogène en H_2, etc. Le gaz est donc soumis en permanence à des processus de refroidissement ou de chauffage. Typiquement, les observations ont estimé qu'une quantité non négligeable de la masse du gaz est située entre les deux états stables `WNM` et `CNM`, [Heiles and Troland, 2003], [Roy et al., 2013]. Des simulations numériques à haute résolution ont d'ailleurs confirmé ces résultats, e.g. [Hennebelle and Audit, 2007]. On retrouve parfois, une autre phase mise à part dans cette classification : les régions `HII`. On les différencie du `WIM`. Il s'agit de bulle de gaz chaud ionisé que l'on peut retrouver autour des étoiles jeunes et chaudes. Le `MIS` peut donc être vu comme un milieu continu avec des larges fluctuations en densité et en température. Dans notre galaxie, on notera que le gaz ionisé (`HIM` et `WIM`), bien que représentant la majorité du volume, ne compte en fait que pour une petite partie de la masse de gaz, environ 25%, [Draine, 2011]. La plupart de la masse est portée par les phases neutres et la phase moléculaire, bien qu'elle ne représente qu'une petite fraction du volume total, de l'ordre de 1 % à 2 %.

En fonction de l'état du gaz, différentes méthodes d'observation sont utilisées et permettent de mettre en évidence des structures et/ou une répartition particulière du gaz. Le `WNM` est observé grâce à la raie à 21 cm en absorption de l'hydrogène et il tend à être distribué de manière relativement uniforme dans le milieu interstellaire. Le `CNM` est observé également grâce à la raie à 21 cm mais en émission. Sa répartition est plus discrète et sous forme de grumeaux ou de filaments. Pour la phase moléculaire, `MM`, c'est principalement grâce aux raies de transitions électroniques qu'on peut l'observer. En effet, l'hydrogène H_2, bien que majoritaire dans ces régions, [Hollenbach

et al., 1971], est une molécule symétrique et donc très difficile à observer à ces températures. Heureusement, il est désormais possible de détecter de nombreuses autres molécules puisqu'on a détecté environ 200 molécules au sein du MIS de la voie Lactée, [McGuire, 2018]. Généralement, on utilise la deuxième molécule la plus abondante, le CO, comme *proxy* pour mettre en évidence ces régions. On constate ainsi que le gaz moléculaire est disséminé en zones denses dont les plus grandes, dénommés nuages moléculaires géants (GMC), ont des masses allant de $\sim 10^2\, M_\odot$ à $\sim 10^5\, M_\odot$, et des tailles variables de 1 *pc* à plus de 100 *pc*, [Draine, 2011].

5.3 Les nuages moléculaires

Différentes relations ont été proposées et permettent de mettre en évidence la présence de formation stellaire dans les nuages moléculaires, comme par exemple, l'abondance de CO et la quantité totale de luminosité infra-rouge, [Solomon et al., 1997]. De même, la relation de *Kennicutt–Schmidt* permet de relier la densité de surface de gaz moléculaire à la densité de surface du taux de formation d'étoiles, (SFR en anglais), [Kennicutt Jr, 1998], [Bigiel et al., 2008]. Il est très important de noter les écarts possibles des valeurs de SFR reflètent des taux d'efficacité de formation stellaires (SFE) variables en fonction des nuages et des galaxies, [Sun et al., 2023]. On peut définir cette efficacité comme le rapport entre la masse de gaz moléculaire disponible et le taux de formation d'étoiles. Puisque les nuages moléculaires peuvent être reliés à la formation d'étoiles, ils sont donc sujets à une attention particulière lors des campagnes observationnelles et des études théoriques.

Les campagnes d'observations ont montré que ces nuages sont principalement concentrés dans de larges zones ou dans des segments de bras spiraux, [Elmegreen, 1985]. En effet, à l'échelle galactique, les forces de marées viennent contre-balancer la formation de nuages moléculaires denses. On verra ainsi apparaître des GMC dans les régions où les perturbations donneront l'avantage à la gravité, par exemple dans les bras spiraux. A l'échelle de ces nuages, c'est probablement la turbulence, que l'on abordera par la suite, accompagnée par le champ magnétique, qui vont principalement s'opposer à l'effondrement des nuages en plus petites structures : les coeurs denses pré-stellaires. A cette dernière échelle, la gravité continue d'agir et s'oppose alors à la pression thermique, qui devient la composante dominante, jusqu'à ce que la densité devienne suffisamment grande pour former des étoiles, supérieure à $10^5\ cm^{-3}$, [Larson, 2003]. A l'intérieur de ces nuages, on retrouve des sous-structures filamentaires ou des amas de manière hiérarchique sur plusieurs échelles, e.g. [Williams et al., 2000]. Ces structures peuvent former des systèmes stellaires ou groupes entiers d'étoiles, [Lada and Lada, 2003], [Larson, 2003]. La gravité joue donc un rôle essentiel, à plusieurs échelles, dans la formation stellaire. Néanmoins, elle est peut être contre-balancée par différentes forces à ces échelles, comme on le verra à la section suivante.

L'échelle caractéristique des GMC est de l'ordre de $\sim 10 - 100\, pc$ pour des densités supérieures à $100\ H \cdot cm^{-3}$ et des températures de l'ordre de quelques dizaines de Kelvins. Les étoiles se forment donc lorsque que l'équilibre entre les forces d'effondrement et les forces de pression n'est plus maintenu. On peut caractériser le temps d'effondrement, aussi appelé *temps de chute libre* (ou *free-fall time* en anglais), noté $t_{\rm ff}$, comme le temps caractéristique d'effondrement d'une sphère de gaz homogène. On fait l'hypothèse implicite que la sphère ne subit aucune force qui s'opposerait à son effondrement gravitationnel. Alors le temps est donné par la relation 5.1, [Binney and Knebe, 2002].

$$t_{\rm ff} = \sqrt{\frac{3\pi}{32 G \rho}} \quad \propto \rho^{-1/2} \tag{5.1}$$

En 1902, Jeans propose que la stabilité de cette sphère de gaz de densité homogène peut être définie par une longueur caractéristique, désormais notée λ_j, à partir de sa densité initiale ρ_0, de la vitesse du son c_s dans le gaz, et de la constante gravitationnelle G :

$$\lambda_j = c_s \left(\frac{\pi}{\rho_0\, G} \right)^{1/2} \tag{5.2}$$

De plus, à partir de cette longueur caractéristique, on peut également définir la *masse de jeans*, M_j, de cette sphère de diamètre λ_j :

$$M_j = \frac{4\pi}{3}\left(\frac{\lambda_j}{2}\right)^3 = \frac{\pi}{6}c_s^3\left(\frac{\pi}{G}\right)^{3/2}\rho_0^{-1/2} \tag{5.3}$$

Ainsi, un nuage de gaz, modélisé par une sphère parfaite, avec un rayon supérieur à la longueur λ_j, sera instable et s'effondrera sous sa propre gravité. C'est ce processus d'effondrement qui est à l'origine de la formation d'étoile. Enfin, on notera que la vitesse du son dans le gaz est défini à partir de la pression P, de la densité ρ et du coefficient adiabatique γ, par l'équation 5.4. On notera qu'on se place ici dans le cas d'un gaz parfait. De plus, on supposer par la suite que le gaz est monoatomique avec un coefficient adiabatique $\gamma = 5/3$.

$$c_s = \sqrt{\frac{\gamma P}{\rho}} \tag{5.4}$$

D'un point de vue théorique, il a été proposé différentes quantités pour caractériser ces nuages. Par exemple, le temps de déplétion exprimé comme l'inverse de l'efficacité de formation d'étoiles, $t_{dep} = SFE^{-1}$, peut être interprété comme le temps caractéristique pour vider le réservoir de gaz disponible en formant des étoiles compte tenu d'un taux de formation stellaire (`SFR`, *Star Formation Rate*) donné, voir section suivante. Ce temps caractéristique a été évalué par l'observation. Il est généralement de l'ordre de $\sim 1 - 10\,Gyr$, dans les galaxies proches observées et peut varier en fonction de leurs caractéristiques, [Bigiel et al., 2008], [Bigiel et al., 2011], [Tacconi et al., 2013]. Sous certaines conditions, le temps de déplétion peut être inférieur à 0.1 Gyr, lors de pics élevés de formation stellaire induits, par exemple, par la fusion de galaxies, [Chapon, 2011], [Saintonge and Catinella, 2022]. Enfin, la structure interne des nuages moléculaires étant partiellement hiérarchique, des modèles fractales permettent d'approximer certains aspects de leur structure, [Elmegreen and Falgarone, 1996]. Ces aspects "fractal" suggèrent que la forme de ces nuages puisse être induite par de la turbulence, [Falgarone and Phillips, 1991], voir section 5.6.

5.4 La formation stellaire

A la section précédente, on a mentionné que le temps de déplétion était inversement proportionnel à l'efficacité de formation stellaire et donc au taux de formation stellaire `SFR`. Ce taux de formation s'exprime comme étant la variation de la masse d'étoile (formée) (M_*) par unité de temps.

$$SFR = \frac{dM_*}{dt} \tag{5.5}$$

En 1955, Salpeter proposa d'utiliser une loi de puissance pour définir la distribution des étoiles nouvellement formées en fonction de leurs masses initiales dont il estima la pente, grâce aux données observationnelles, [Salpeter, 1955]. On fera référence à cette courbe avec l'acronyme `IMF` (*Initial Mass Function*). Ainsi, Salpeter estima le paramètre α de l'équation 5.6 pour les étoiles allant de 0.4 $M_\odot$ à 50 $M_\odot$ (ici A est une constante de normalisation) :

$$\frac{d\,n}{d\,M} = A\,M^{-\alpha} \tag{5.6}$$

On notera que cette distribution suggère que les étoiles peu massives, dont la masse est inférieure à 1 $M_\odot$, sont donc plus nombreuses. Bien que la loi de puissance soit communément admise pour les étoiles massives ($M > 1\ M_\odot$), elle est encore discutée pour les étoiles ayant une masse inférieure à celle de notre soleil. Par exemple, [Kroupa, 2001] proposa un découpage sous forme de plusieurs lois de puissance pour les masses inférieures à 1 $M_\odot$.

Les observations ne donnent accès qu'à des quantités intégrées le long de la ligne de visée de l'observation, telle que la densité de colonne du gaz, ou la densité de colonne de la `SFR`. En 1989, Kennicutt–Schmidt enrichissent une relation initialement proposée par Schmidt en 1959, permettant de relier la densité de surface de formation d'étoile, Σ_{SFR}, à la densité de surface du gaz Σ_{gas} par le biais d'une loi de puissance telle que $\Sigma_{SFR} \propto (\Sigma_{gas})^n$, avec $n \simeq 1.4$, [Kennicutt, 1989]. On donne sur la figure 5.2, cette relation mise à jour avec les données observationnelles récentes, [Bigiel et al., 2008]. On notera que cette relation a mis en évidence que les galaxies convertissent leur gaz en étoiles près de 100 fois plus lentement que ce que le temps d'effondrement gravitationnel caractéristique des nuages de gaz observés permet de prévoir. On peut le voir également comme une différence significative entre

le temps caractéristique de déplétion et le temps caractéristique d'effondrement : $\tau_{dep} \gg t_{ff}$. En conséquence, il doit donc y avoir des processus physiques qui s'opposent à la formation stellaire, en s'opposant à l'effondrement gravitationnel. Parmi les processus envisagés, il y a ceux issus de la rétro-action des étoiles, qu'on abordera à la section suivante. On notera que l'échelle spatiale caractéristique de la formation d'étoile est de l'ordre de $\sim 10^{-7}\ pc$ pour des densités de l'ordre de $\sim 10^{24}\ H \cdot cm^{-3}$.

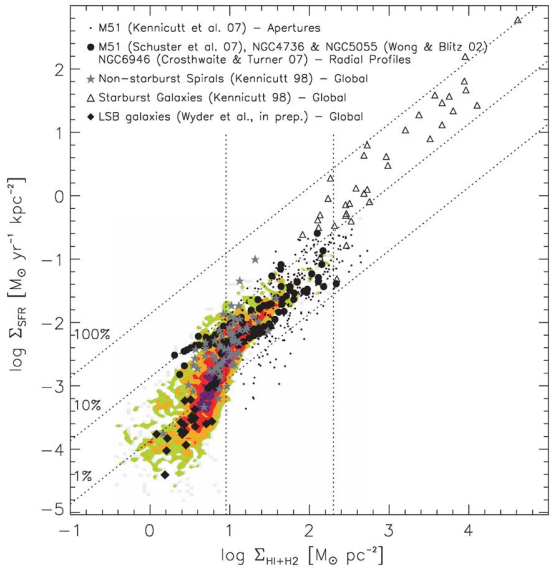

FIGURE 5.2 – Densité de surface de la formation stellaire, Σ_{SFR}, en fonction de la densité de surface du gaz atomique et moléculaire, Σ_{HI+HII}, [Bigiel et al., 2008].

La pente de cette relation peut varier en fonction des intervalles de densité de surface observable. En effet, à partir de l'échelle galactique, donc supérieure à 1 kpc, il est possible d'identifier deux voire trois régimes distincts de formation stellaire, [Kennicutt and Evans, 2012]. Par exemple, à basse densité, la pente est plus élevée qu'à haute densité. Aussi, une galaxie peut être dans l'un ou l'autre de ces régimes, mais également inclure plusieurs régimes. On notera également que cette relation, liée à l'observation, est valable pour un instant t, [Bigiel et al., 2008]. De plus, elle ne permet pas de tenir compte de la densité relative des nuages par rapport au plan galactique et de la possible variation du temps de chute libre impacté par la dynamique galactique, [Girichidis et al., 2020].

Enfin, il est important de préciser qu'il existe une classification à base de lettre (`OBAFGKM`), dite classification de Morgan-Keenan, des étoiles en fonction de leurs propriétés spectrales (ou leur masse), [Habets and Heintze, 1981]. De plus, on notera que le temps et la fin de vie d'une étoile dépend de sa consommation de ses réserves d'hydrogène par les processus thermonucléaires et donc de sa masse initiale, d'où l'importance de l'`IMF`. Ainsi pour les étoiles les moins massives, de classe `G` (comme notre soleil par exemple), le temps de vie est de l'ordre de $\sim 10\ Gyr$, et les plus massives, de classe `O`, de l'ordre de $\sim 10\ Myr$. Durant leur vie, les étoiles vont interagir avec l'environnement qui les entoure. On s'intéressera ici aux rétro-actions des étoiles sur le `MIS`.

5.5 Rétroaction stellaire sur le milieu environnant

En l'absence de rétroaction, la matière visible s'effondrerait rapidement par le biais d'une cascade de refroidissement (*cooling catastrophe*). En effet, en s'effondrant la matière devient plus dense et se refroidit efficacement par émission de rayonnement, sa pression thermique faiblissante ne permet pas de contrer la poursuite de l'effondrement gravitationnel. La formation d'étoile serait alors très efficace produisant une masse stellaire plusieurs ordres de grandeur plus élevé que ce qui est observé, en plus de consommer trop rapidement tout le gaz disponible, e.g. [Keres et al., 2009]. La rétroaction des étoiles, déjà mentionnée à la section précédente, est donc extrêmement importante pour réguler cet effondrement gravitationnel en réchauffant le gaz mais aussi en soufflant les coeurs denses, diminuant ainsi le la fraction de gaz consommée et de fait, la formation stellaire. Ces phénomènes de rétroaction stellaires sont issus de processus physiques variés parmi lesquels on retrouve notamment : les vents,

les supernovae, la photoionisation. Ils permettent de déposer, transférer ou injecter de l'énergie, de la quantité de mouvement, de la masse mais aussi d'enrichir la composition chimique du MIS. La formation stellaire joue donc un rôle essentiel dans l'évolution de la structure et de la composition du MIS.

Ces différents processus agissent à des échelles différentes. Les vents stellaires sont des flux de particules chargées émis par les étoiles, injectant du gaz, du moment et de l'énergie dans le MIS. L'influence de ces vents se limite à quelques parsecs. La photoionisation résulte de l'émission de photons suffisamment énergétiques pour ioniser le milieu environnant de l'étoile. Les étoiles massives, de type O, injectent une énergie tellement importante, que le gaz d'hydrogène sera ionisé autour d'elles, formant ainsi des régions HII. On notera que ces régions peuvent atteindre une dizaine de parsec, en fonction de la source centrale, et une température de l'ordre de 10^4 K. Les supernovae, SNe, constituent la fin de vie d'une fraction des étoiles de l'IMF, étoiles de type O. Il s'agit de la rétroaction la plus violente, suspectée de jouer un rôle principale dans la régulation de la formation stellaire.

Ainsi, les étoiles dont la masse initiale est supérieure à 8 $M_\odot$ auront tendance à exploser violemment, réinjectant une quantité significative d'énergie, de moment mais aussi des éléments lourds, tels que des métaux, dans leur environnement. La rétroaction d'une SNe peut se décomposer en trois étapes : l'expansion libre, l'expansion adiabatique (Sedov-Taylor), le refroidissement, [Sedov, 1959]. Durant la première phase, l'éjecta se déplaçant à haute vitesse, crée une onde choc qui souffle le gaz avoisinant. La masse de gaz soufflé est inférieure à la masse de l'éjecta. Dans la deuxième phase intervient lorsque la masse de gaz soufflé devient supérieur à l'éjecta. L'énergie cinétique de l'explosion initiale est alors transférée au gaz qui est chauffé par l'onde de choc, et l'expansion de l'enveloppe continue de manière adiabatique. Dans cette phase, l'évolution est déterminée uniquement par l'énergie initiale de l'explosion, la densité du gaz du MIS et le temps depuis l'explosion. Finalement, lors de la phase de refroidissement, l'énergie du gaz est irradiée efficacement. On notera que durant ces explosions, du gaz est expulsé à des vitesses supersoniques $6000 - 7000$ km/s, et l'énergie injectée par la SNe, dans le MIS, est indépendante de la masse de l'étoile. Elle est estimé à environ 10^{51} erg, [Janka, 2012]. Il est important de noter, par rapport au modèle de rétroaction qu'on introduira au chapitre suivant, que dans la deuxième phase, la densité du milieu joue un rôle sur l'efficacité des SNe, [Iffrig and Hennebelle, 2015].

En réalité, d'autres effets peuvent réguler la formation stellaire en empêchant ou favorisant l'effondrement des nuages, tel que le passage d'onde de densité (bras spiraux ou barre centrale) ou encore la turbulence.

5.6 La turbulence

La turbulence est un phénomène physique dans lequel les fluctuations irrégulières, mais néanmoins hiérarchiques, des grandeurs telles que la vitesse ou la pression par exemple, couvre une large gamme d'échelles. En 1883, l'expérience de Reynolds de l'écoulement d'un fluide à travers un tuyau a mis en évidence deux régimes distincts. Afin de caractériser ses écoulements, très différents, il définit une grandeur sans dimension, désormais appelé nombre de Reynolds, noté R_e. Lorsque $R_e \ll 1$, l'écoulement est dit laminaire et lorsque $R_e \gg 1$, turbulent. Dans cette dernière, on voit apparaître des *tourbillons* de différentes tailles. D'un point de vue théorique, si on considère l'écoulement d'un fluide newtonien incompressible, l'équation qui permet de décrire le mouvement de fluide, dit de Navier-Stokes, s'écrit :

$$\rho\left(\frac{\partial v}{\partial t} + (\vec{v}\cdot\vec{\nabla})\,\vec{v}\right) - \nu\Delta\vec{v} = \rho\vec{g} - \vec{\nabla}P \tag{5.7}$$

On notera que les termes non linéaires introduits par $(\vec{v}\cdot\vec{\nabla})\,\vec{v}$ sont responsables du développement de la turbulence et de la difficulté de résoudre cette équation. En effet, ces termes sont associé à l'advection du fluide, et le terme $\nu\Delta\vec{v}$ à la viscosité, ν étant la viscosité cinématique. Le nombre de Reynolds R_e, sans dimension, peut se définir par le rapport du terme d'advection sur le terme de viscosité, [Reynolds, 1895].

$$R_e = \frac{\|\,\rho\left(\vec{v}\cdot\vec{\nabla}\right)\vec{v}\,\|}{\|\,\nu\Delta\vec{v}\,\|} \simeq \frac{LV}{\nu} \tag{5.8}$$

Les grandeurs L et V sont respectivement la longueur et la vitesse caractéristique du fluide. Lorsque l'écoulement est turbulent, les valeurs de gradient de vitesse sont élevées et l'énergie cinétique de l'écoulement

s'étale sur de grandes échelles. Dans le milieu interstellaire, par exemple, le nombre de Reynolds est très grand, il s'agit donc d'un milieu turbulent avec de grandes valeurs du gradient de vitesse. En conséquence, il est intéressant d'utiliser des métriques associées telles que la dispersion de vitesse pour étudier les phénomènes turbulents, comme on le verra au chapitre 6.

Richardson propose en 1920, une théorie phénoménologique de la turbulence incompressible en proposant un point de vue énergétique, sous la forme d'une cascade d'énergie. En effet, son hypothèse fondamentale se base sur une hiérarchie de tourbillons imbriqués les uns dans les autres. Cette imbrication permettant un transfert d'énergie des grandes échelles où elle est injectée vers les petites échelles où elle est finalement dissipée par la viscosité du fluide. En 1968, la théorie de Kolmogorov, [Kolmogorov, 1968], permet de formaliser la cascade de turbulences en proposant deux hypothèses fondamentales.

On considère un fluide homogène dont les fluctuations aux petites échelles sont homogènes, isotropes, et dont la statistique est indépendante des mouvements aux grandes échelles et stationnaire. L'hypothèse (`H1`) dit que pour les très grands nombres de Reynolds, les propriétés statistiques aux petites échelles sont entièrement déterminées, de manière unique et universelle, par la dissipation d'énergie, ϵ, et la viscosité, ν. L'hypothèse (`H2`) stipule que la statistique aux grandes échelles (grand nombre de Reynolds) dépend uniquement de la dissipation d'énergie ϵ, la dépendance à la viscosité étant négligeable.

En utilisant la décomposition de Fourier du champ de vitesse, on peut caractériser l'évolution de l'énergie entre les fréquences d'oscillations des tourbillons aux différentes échelles. Ainsi, on met en évidence le spectre d'énergie dont la théorie de Kolmogorov permet de définir différentes régions, voir figure 5.3. Il est important de noter que les tourbillons aux grandes échelles ont une fréquence faible, et les plus petits tourbillons, aux petites échelles, une fréquence élevée. Sur ce diagramme, les fréquences inférieures à k_0 caractérisent la zone d'injection de l'énergie qui dépend du contexte. La fréquence k_η marque l'échelle η de transition entre l'échelle de transfert d'énergie et l'échelle de dissipation d'énergie par les termes visqueux. En effet, pour $k < k_\eta$ les grands tourbillons se fragmentent pour former des plus petits tourbillons, l'énergie est donc transférée des grands tourbillons vers les plus petits en cascade jusqu'à l'échelle de dissipation. Pour $k > k_\eta$, les tourbillons seront dissipées par la viscosité du fluide.

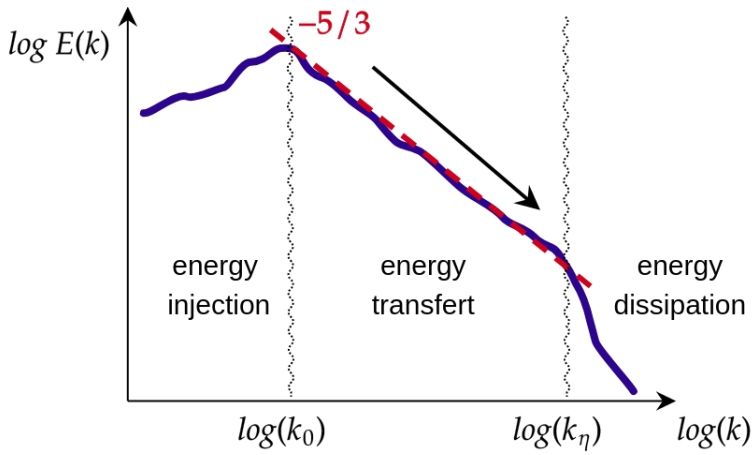

FIGURE 5.3 – Schéma de la cascade énergétique de la turbulence et les trois zones : injection, transfert, dissipation. L'échelle η de Kolmogorov marque la transition vers $R_e \gg 1$, à $log(k_\eta)$ où $R_e = 1$.

On définit les fréquences k_0, k_l, k_η correspondant respectivement à différentes échelles l_0 dans le régime d'injection d'énergie, l_l dans le régime de transfert d'énergie et l_η dans le régime de dissipation, telles que $l_0 > l_l > l_\eta$. On peut définir à partir de ces différentes échelles et des vitesses caractéristiques associées, un temps caractéristique de retournement d'un tourbillon à l'échelle l,

$$\tau_l \sim \frac{l}{v_l} \tag{5.9}$$

et un temps de dissipation grâce à la viscosité,

$$\tau_\eta \sim \frac{l_\eta^2}{\nu} \tag{5.10}$$

et donc à l'échelle η de dissipation on a

$$\frac{l_\eta}{v_\eta} \sim \frac{l_\eta^2}{\nu} \tag{5.11}$$

par analyse dimensionnelle, on peut donc montrer que le taux de transfert d'énergie ϵ est tel que $\epsilon \sim v_l^3/l$. Le taux de transfert étant supposé constant dans la partie inertielle, on a donc $v_l \propto l^{1/3}$, augmentant ainsi avec l'échelle. A partir de là, il est possible de déduire la loi de Kolmogorov pour le spectre de l'énergie, en passant par l'espace de Fourier. En effet, en définissant le spectre de puissance tel que $P(k) = \hat{G}(k)\,\hat{G}(k)^*$ avec $\hat{G}(k)$ la transformée de Fourier d'une fonction $G(\mathbf{r})$ dans l'espace réel. Par convention, dans la littérature on exprime le spectre d'énergie à une dimension sous la forme $E(k)\,dk$ comme étant égale à la moyenne du spectre de puissance dans toutes les directions (sur des sphères de rayon dk). On a donc pour D le nombre dimension, la relation 5.12

$$E(k)dk = P(\mathbf{k})\,dk^D \tag{5.12}$$

On obtient ainsi la loi de puissance dans la zone inertielle sous la forme

$$E(k) \propto \epsilon^{2/3}\, k^{-5/3} \tag{5.13}$$

Ainsi, pour le cas tridimensionnel $D = 3$, on a : $E(k) = 4\pi\, k^2\, P(k)$ et donc $P(k) \propto k^{-11/3}$.

En astrophysique, aux grandes échelles dans une galaxie, la phase `WIM` et la phase qui domine en terme de volume. Compte tenu de la définition du nombre de Reynolds qui implique le rapport des forces d'inertie sur les forces visqueuses, et de la vitesse de rotation typique, 200 km/s dans notre galaxie à quelques kiloparsec du centre, on s'attend naturellement à avoir un nombre de Reynolds élevé, de l'ordre de 10^6. Le `MIS` est donc un milieu fortement turbulent, e.g. [Utomo et al., 2019]. Néanmoins, la caractérisation et la compréhension de la turbulence du `MIS` est loin d'être une tâche facile. En effet, comme on l'a vu précédemment, le milieu est auto-gravitant, compressible, de nature supersonique, le champ magnétique est non nul, etc. De plus, il y a de nombreux processus susceptibles d'injecter de l'énergie (voir sections précédentes), ainsi la caractérisation de l'échelle intégrale (échelle d'injection) est difficile a définir. De même, l'échelle de dissipation est variable. Au sein du gaz très froid, par exemple, elle peut être de l'ordre du libre parcours moyen des atomes d'hydrogène. On notera également que l'origine de cette turbulence interstellaire reste incertaine car elle peut venir des effets de rétro-action stellaires et/ou des instabilités gravito-hydrodynamiques à plus grande échelle, e.g. [Bournaud et al., 2010], [Renaud et al., 2013], voire même par accrétion de gaz externe à la galaxie, [Gabor and Bournaud, 2013].

Enfin, l'écoulement supersonique dans du gaz fortement compressible induit des fortes perturbations de densité. Les ondes de chocs générées offrent ainsi une autre possibilité de dissipation de l'énergie ou de transfert entre différentes échelles. Avec ces processus on s'éloigne ainsi de la cascade de turbulence typique d'un fluide incompressible de Kolmogorov. L'impact sur le spectre de puissance reste néanmoins relativement faible. En effet, en supposant qu'un choc parfait puisse être représenté par une fonction échelon (*step function*), on peut montrer que sa transformation de Fourier est k^{-2}. Par le biais d'une intégration en `3D` sur un ensemble de chocs qu'on suppose aléatoire, on a donc $E(k)\,dk \propto k^{-2}\,dk$ et donc pour le cas tridimensionnel $D = 3$, $P(k) \propto k^{-4}$ comme proposé par Burger(1974). Dans le cas du `MIS`, partiellement compressible, on peut s'attend donc à avoir des spectres de puissance entre $-11/3$ et -4, dans la simulation qui sera effectuée au chapitre suivant.

5.7 Compressibilité

Comme on l'a vu et mentionné précédemment, le `MIS` soumis à de nombreux processus qui vont agir sur le gaz en le compressant et le dilatant. La décomposition de Helmholtz permet de décomposer un champ vectoriel suivant deux composantes : compressible (ou irrotationnelle) et solénoïdale. Il est intéressant de s'intéresser à la décomposition du champ de vitesse dans le cas de la turbulence. En effet, la composante compressible aura tendance à créer des zones de surdensités et des zones vides, générant ainsi de large contraste en densité. A l'opposé, la composante solénoïdale (ou vorticité) aura tendance à mélanger le milieux sans modifier la densité.

5.8 Stabilité d'un disque

Pour conclure ce chapitre et faire une transition avec la simulation numérique, on propose de s'arrêter un instant sur les propriétés de stabilité d'un disque en rotation. En effet, un disque galactique en rotation est un système en quasi-équilibre entre plusieurs acteurs : la gravité, la pression thermique et turbulente, la rotation. [Toomre, 1964] proposa de définir un critère de stabilité d'un disque stellaire par la relation 5.14 à partir de la fréquence épicyclique κ, de la dispersion de vitesse σ_*, la constante gravitationnelle G et la densité de colonne.

$$\mathcal{Q}_* = \frac{\kappa\,\sigma_*}{3.36\,G\,\Sigma_*} > 1 \tag{5.14}$$

On peut étendre cette relation au disque de gaz en tenant compte de la vitesse du son c_s déjà introduite précédemment. Il est intéressant de noter que la pression thermique, au travers de c_s, joue le même rôle stabilisateur dans le critère $\mathcal{Q}_g$ que la dispersion de vitesse dans $\mathcal{Q}_*$.

$$\mathcal{Q}_g = \frac{c_s\,\kappa}{\pi\,G\,\Sigma_*} > 1 \tag{5.15}$$

Chapitre 6

Simulation `ExaMilkyWay`

Dans ce chapitre, on va commencer par remettre en contexte le sujet d'étude et la simulation par rapport à la littérature, ce qui nous permettra de définir le dimensionnement du calcul et par la suite le plan de gestion de données pour la simulation. Avec le dimensionnement on discutera des problèmes rencontrés lors des premiers tests à grand nombre de processus `MPI` et les solutions apportées, qui ont permis de réduire la consommation mémoire et d'économiser des heures de calculs sur l'allocation du projet. Ensuite, on détaillera la simulation et les premiers résultats obtenus.

6.1 Vue d'ensemble

L'objectif principal de ces travaux de thèse est l'étude du développement de la turbulence interstellaire dans une galaxie isolée du type de la Voie Lactée, ainsi que son rôle dans l'efficacité ou plutôt l'inefficacité observée de la formation d'étoiles, comparé au rôle de la rétroaction des étoiles jeunes. La vitesse et l'efficacité de la formation d'étoile dans les galaxies, à tous les âges de l'Univers, est un sujet d'étude très actifs de l'astrophysique moderne. Comme on l'a vu précédemment, plusieurs mécanismes peuvent impacter la formation stellaire. Néanmoins, le mécanisme principal de cette régulation reste un sujet très débattu et trois hypothèses principales ont émergé : le faible rendement de la formation d'étoile, la rétroaction stellaire et la pression turbulente. La première hypothèse semble aujourd'hui exclue puisque l'efficacité, à l'échelle de la formation d'une étoile individuelle, de la conversion des coeurs denses semble être d'au moins 50% [Elmegreen, 2002], [McKee and Ostriker, 2007]. La rétroaction permet aux étoiles massives de détruire les nuages denses et d'éjecter une grande partie des réservoirs de gaz hors des galaxies, [Hopkins et al., 2014]. La pression turbulente du milieu interstellaire peut dans certains cas compresser les nuages de gaz et donc favoriser la formation stellaire mais plus généralement permet de stabiliser les nuages contre leur effondrement/fragmentation gravitationnel en coeurs pré-stellaires ([Kraljic et al., 2014], [Renaud et al., 2012], [Colman et al., 2022]).

Dans ces travaux on propose d'évaluer les deux dernières hypothèses. La génération de la turbulence interstellaire et son impact sur la formation de nuage dense pré-stellaire sont des problèmes multi-échelles et multi-physiques par nature. En effet, ils font intervenir des instabilités gravitationnelles depuis les grandes échelles (bras spiraux, barre centrale), de l'hydrodynamique du gaz ainsi que son chauffage/refroidissement aux moyennes échelles, et la formation stellaire ainsi que sa rétroaction au plus petites échelles. Il s'agit donc de résoudre des échelles allant de quelques dizaines kiloparsec à quelques parsecs. Les densités de gaz peuvent varier de l'ordre de 1 H/cm^3 en moyenne dans la galaxie, à l'échelle de quelques kpc jusqu'à $\simeq 10^7$ H/cm^3 dans les coeurs denses de quelques pc, [McKee and Ostriker, 2007]. Afin d'étudier ces processus, il est nécessaire d'utiliser les méthodes de maillages adaptatives requises et capables de gérer ces aspects multi-échelles. Les besoins d'une grande gamme d'échelles spatiales et de densité sont tels que les études numériques existantes se limitent généralement à :

et/ou, des régions sous-galactiques de quelques parsecs à quelques kiloparsecs manquant donc les mécanismes d'injection d'énergie des grandes échelles qui semblent jouer un rôle majeur [Colling et al., 2018], [Brucy et al., 2020], [Colman et al., 2022],

et/ou, des modèles très limités de rétroaction n'incluant qu'une injection du moment par les supernovae et négligeant ainsi la pression radiative et le chauffage du gaz par les jeunes étoiles, mécanisme potentiellement dominant, [Bournaud et al., 2010], [Murray et al., 2011],

et/ou, des modèles thermiques très simplifiés du gaz interstellaire sous forme d'une équation d'état du type $T = f(\rho)$ ne permettant pas d'introduire une dynamique de chauffage/refroidissement, [Teyssier et al., 2010], [Renaud et al., 2013],

et/ou, des durées physiques très courtes de l'ordre de quelques millions d'années ne permettant pas de couvrir les cycles de formation et destructions des nuages moléculaires denses,

et/ou, une résolution élevée dans les régions très denses mais limitée à quelques dizaines de parsec dans les régions peu dense qui sont pourtant cruciales pour la propagation multi-échelle de la turbulence,

Plusieurs raisons sont à l'origine de ces limitations. On retrouve, notamment, la quantité des ressources disponibles en mémoire, en nombre d'unité de calculs, en espace de stockage, en nombre d'heures de calculs alloués, etc. On notera que la bonne scalabilité du code utilisé est également un facteur qui peut être limitant. Dans notre cas, avec `RAMSES`, on tire profit des développements détaillés aux chapitres précédents afin de lever les points bloquants du code autour des `E/S`. Ainsi, on réalise une simulation numérique avec les caractéristiques suivantes :

- une galaxie isolée *entière* de type Voie Lactée,
- une résolution spatiale élevées de l'ordre du parsec donc dans un grande partie du volume du disque, y compris dans les régions les moins denses pour avoir une résolution fine du développement de la turbulence à grande échelle,
- un modèle thermodynamique de chauffage/refroidissement du gaz,
- un modèle de formation stellaire,
- un modèle de rétroaction cinétique des étoiles massives,
- couvrir plus d'un cycle ($\sim 10\ Myr$) de formation / rétroaction des étoiles massives à haute résolution.

On prend comme conditions initiales pour les particules de vieilles étoiles, les positions et vitesses mises à jour par la mission `GAIA` (`RC2`) pour la Voie Lactée et un profil exponentiel pour le disque de gaz. De plus, on considère une répartition sphérique du halo de matière noire. On détaille ces aspects à la section 6.5. La simulation va se dérouler en deux grandes étapes, voir figure 6.1. La première étape, à basse résolution spatiale entre 117.2 pc, pour les cellules les plus grossières et jusqu'à 14.7 pc dans les cellules les plus fines (niveau `AMR` $[10, 13]$), vise à former les grandes structures tout en conservant le gaz à une température plancher de 10^4 K. On notera qu'on choisit ici une boite de simulation de 120 kpc de côté. Dans la deuxième partie, la résolution va être augmentée progressivement jusqu'à atteindre $\sim 1\ pc$ (niveau `AMR` 17) tout en maximisant la fraction du volume de gaz résolue, compte tenu des ressources mémoires et en temps de calcul disponible. C'est dans cette seconde partie qu'on activera les modèles thermodynamiques, de formation stellaire et de rétroaction des étoiles massives, voir section 6.5.

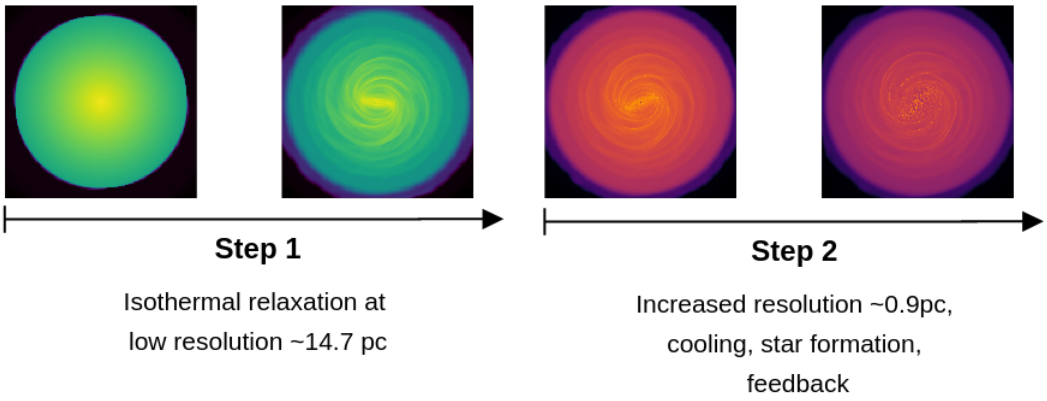

FIGURE 6.1 – A gauche la 1ère étape de la simulation avec $\Delta x_{min} \simeq 117\ pc$ et $\Delta x_{max} \simeq 14.7\ pc$. A droite, la deuxième étape avec le chauffage/refroidissement du gaz, la formation stellaire et sa rétroaction, $\Delta x_{max} \simeq 117\ pc$ et $\Delta x_{min} \simeq 0.9\ pc$

6.2 Dimensionnement du calcul

Pour dimensionner le calcul il faut tout d'abord tenir compte du matériel qui sera utilisé. La simulation `ExaMilkyWay` va être exécutée au `TGCC` (Très Grand Centre de calcul du CEA) sur la partition `ROME`. Chaque noeud

de cette partition dispose de 2 CPU AMD Epyc 7H12 disposant chacun de 64 coeurs physiques et de 256 Go de mémoire vive. Sur ce type de noeud, il est possible de lancer des *jobs* avec trois types de configurations différentes, qu'on appellera par abus de langage : openMP, MPI ou hybride openMP + MPI. Lorsque l'utilisateur fait une réservation, il doit renseigner le nombre de noeuds n_n et/ou le nombre de tâches n_t par noeud (sous entendu, processus MPI) et/ou le nombre de coeurs par tâche n_c (sous entendu tâche openMP). Par défaut, $n_c = 1$ et donc un processus MPI est rattaché à un coeur de calcul. Le nombre de noeuds peut en fait être déterminé par la relation $n_t * n_c/128$ (car chaque noeud dispose de 128 coeurs), sauf en cas de demande d'exclusivité sur les noeuds. On notera que ce nombre est arrondi à l'entier supérieur si besoin. L'imputation horaire sur le budget horaire de l'utilisateur se fait alors, dans la plupart des cas, suivant la formule $n_t * n_c * t/3600.0$, où t est le nombre de secondes d'exécution du job. On parle alors de consommation en *heures CPU*. Dans le cadre du projet ExaMilkyWay, 52 millions d'heures CPU ont été attribué par GENCI. D'autre part, chaque noeud disposant d'une mémoire de 256 *Go*, elle est répartie entre les coeurs de manière à ce qu'une tâche puisse avoir accès à $n_c * 256/128 Go$ soit 2 Go pour un fonctionnement par défaut avec $n_c = 1$.

Puisque RAMSES n'utilise qu'une seule couche de parallélisme, la configuration de base est donc généralement $n_t = N$, $n_c = 1$, et $n_n = n_t/128$, où N est le nombre de processus MPI (et le nombre de domaine) de la simulation. En conséquence, la mémoire totale disponible est donc de $N * 2$ Go. Puisque la consommation mémoire d'un code AMR peut varier et même croître strictement au cours de la simulation, il peut arriver que le code s'arrête à cause d'un manque de mémoire. Dans ce cas de figure, il y a deux options : augmenter le nombre de tâches, n_t, ou augmenter le nombre de coeurs par tâche, n_c. Malgré différentes tentatives par le passé, il n'est pas possible, avec RAMSES, de relancer une simulation en modifiant le nombre de processus MPI. La seule option est donc d'augmenter le nombre de coeur par tâche. Cette approche est généralement réservée au code hybride ce qui permet par exemple d'accorder le nombre de *threads* openMP avec n_c, ainsi tous les coeurs sont utilisés. (c'est ce qui est fait avec l'outil d'analyse par exemple). Dans le cas de RAMSES, seule la mémoire associée à ces n_c coeurs par tâche sera utilisée. Néanmoins, puisque ces coeurs supplémentaires sont réservés, ils seront pris en compte dans la comptabilisation des heures consommées. Ainsi, dépeupler, c'est-à-dire mettre $n_c > 1$ pour un code non hybride, implique un coût horaire augmenté.

Néanmoins, il est important de noter que RAMSES tire profit de la vectorisation. En effet, c'est un code qui appartient à la catégorie des logiciels de type *memory-bound*. Cela signifie que si la bande passante mémoire est augmentée, alors les performances du code seront augmentées. En conséquence, le dépeuplement n'induit un sur-coût horaire que de 30%. En effet, avec une configuration $n_c = 2$, le coût horaire d'une réservation à nombre de processus n_t égal, pour un même temps d'exécution, est doublé. Mais l'utilisation de coeurs supplémentaires permet à chaque processus de pouvoir bénéficier aussi de la passante mémoire de ces coeurs, en plus de la quantité de mémoire. Il faut aussi tenir compte du fait que le nombre de tâches MPI par noeud diminue, et donc le nombre n_n de noeuds augmente, ce qui permet de limiter la pression sur les cartes réseaux des noeuds. En conséquence, l'ajout de cette bande passante mémoire permet à RAMSES d'être plus rapide. Le temps d'exécution diminue et donc le sur-coût horaire n'est pas d'un facteur 2 mais de seulement $\sim 30\%$.

Pour dimensionner la simulation ExaMilkyWay, il faut tenir compte principalement de la quantité de mémoire qui sera nécessaire pour atteindre l'objectif initial, à savoir résoudre 90% de la masse du gaz à environ ~ 1 *pc* (niveau AMR 17). Compte tenu de la taille de boite nécessaire pour contenir le halo de matière noire, 120 *kpc*, on sait que le volume du disque de gaz la galaxie occupera $< 0.1\%$ du volume total de la boite. Sachant également que les 10% du gaz les moins denses occupent plus de 80% du volume du disque, on estime donc devoir résoudre environ 20% du volume à ~ 1 *pc* de résolution, ce qui nécessitera $\simeq 10^{11}$ cellules au niveau 17. Sachant que le coût mémoire d'une cellule dans RAMSES, qu'on peut estimer à environ 10 octets, représente une occupation mémoire de 70 *To*. Pour une architecture comme celle mentionnée précédemment, il faut donc compter 290 noeuds de calculs soit environ 37000 coeurs. Il faut en plus compter les particules (de l'ordre de 20 millions) et quelques sur-coûts mémoires : il est donc raisonnable d'utiliser 40000 coeurs pour la simulation. L'utilisation de 40,000 processus avec RAMSES soulève des problèmes d'E/S qu'on a évoqué et résolu avec l'intégration de Hercule et l'utilisation du modèle lightAMR. Cependant, la consommation mémoire du code augmente significativement avec le nombre de processus MPI, et peut nécessiter de dépeupler dès le début de la simulation. Des développements ont été nécessaires afin d'adresser cette problématique pour éviter le dépeuplement des coeurs le plus longtemps possible pour éviter le surcoût horaire. Ces développements sont détaillés à la section suivante et ont permis des gains mémoire significatifs.

Les premiers tests utilisant 40000 processus MPI ont révélé des problèmes de stabilité des communications MPI. En effet, la première partie de la simulation devant se dérouler à "basse" résolution pour former rapidement

les structures à grandes échelles, il y a peu de grilles dans les niveaux AMR ([10, 13]). La conséquence avec RAMSES est d'avoir beaucoup de communications inter-processus dans un laps de temps assez court pour synchroniser les niveaux entre les cycles d'itérations. Avec une configuration $n_c = 1$ (sans dépeuplement), sur 40000 MPI cela entraîne des instabilités aléatoires au niveau de la couche MPI probablement liées à une sursollicitation des cartes réseaux. Le temps de calcul des itérations est alors augmenté et fluctue anormalement (jusqu'à un facteur 2.5) et des crashs aléatoires ont lieu durant les réductions MPI de type allReduce. Augmenter le nombre de cellules afin d'augmenter la partie "calcul" entre les communications, comme par exemple, augmenter le niveau de raffinement minimum de 10 à 11, bien que pouvant potentiellement résoudre en partie le problème, produirait environ 214000 cellules par processus, en plus d'apporter une résolution fine dans toute la boite de simulation. Compte tenu des ambitions de la simulation en terme de résolution et de la valeur maximale atteignable du nombre de cellules par processus (hors dépeuplement) de 900000, on écarte cette option. On a également essayé changer de *backend* de communication, UCX, sur avis des experts du centre de calcul. Bien que cette solution permette de régulariser les pas de temps, elle induit un surcoût de $\sim 30\%$ des temps de calculs. En conséquence, il aurait fallu consommer la moitié de l'allocation horaire pour la première étape de la simulation.

Finalement, afin de résoudre ces problèmes, on choisit de réduire la fraction de la masse de disque résolue à ~ 1 *pc*, et ainsi réduire les besoins mémoires et donc le nombre de processus MPI. En effet, l'objectif initial était d'atteindre une résolution spatiale de l'ordre de 1 *pc* dans $\sim 90\%$ de la masse du disque. De plus, des développements annexes, détaillés à la section 6.4.1, sur les tableaux utilisés pour les communications, ont permis de réduire significativement l'empreinte mémoire du code et donc d'augmenter le nombre de cellules maximum par processus. In fine, la simulation a donc utilisé 16384 processus MPI et a permis une résolution spatiale de 85% de la masse du disque à 3.7 *pc* dont 50% à 1 *pc*, comme on le verra par la suite. De plus, afin de réduire la consommation des heures, des développements complémentaires ont été réalisés afin de pouvoir changer le nombre de processus MPI, ce qui a permis de faire la partie à basse résolution sur seulement 4096 processus et d'augmenter le nombre de processus pour la deuxième étape à haute résolution.

6.3 Plan de gestion de données

L'objectif principal de la mise place d'un plan de gestion de données, PGD, est d'organiser en avance la gestion des données au sein d'un projet, de faciliter le partage, l'inter-opérabilité et la reproductibilité des données issues de la recherche. Il s'agit en fait d'aider les scientifiques à converger vers le principe FAIR. Il existe des outils d'aide à la mise en place d'un PGD tel que le portail OPIDOR [a] qui propose notamment de nombreux exemples de PGD publics validés pour leur qualité et leur exhaustivité. Dans le cadre d'une simulation numérique, il est intéressant de développer les aspects suivant dans le plan :

1. production des données et/ou réutilisation de données existantes : méthodes, logiciels, sources,
2. format utilisé : données brutes, images, vidéos, etc, il est généralement préférable d'utiliser des formats standardisés et open source (si possible),
3. documentation et qualité : méta-données pour l'aide à l'exploration,
4. stockage et sauvegarde : volume des données, type de sauvegarde envisagé, protection des données,
5. partage : comment et quand les données seront disponibles, les méthodes d'accès (logiciel, base de données web, centre de calcul), référencement des données (attribution d'identifiant unique, DOI, ...),
6. gestion des ressources : responsabilité du stockage, budgets des ressources

Il s'agit d'une liste non exhaustive et principalement adaptée pour les simulations numériques. En effet, il n'existe pas de PGD complètement générique et ceux ci peuvent être spécifiques à certains domaines scientifiques. Par exemple, en fonction des champs applicatifs, des questions d'éthiques et d'anonymisation des données peuvent également être développées. Détailler un PGD est devenu une pratique courante, et même nécessaire, lors de la demande d'attribution de ressources de calculs. Dans le cadre la simulation ExaMilkyWay, voici un résumé du PGD qui a été établi :

1. Les données seront produites par le code RAMSES [Teyssier, 2002], avec les modules AMR, HYDRO, PM classiques ainsi que l'extension Merger pour les conditions initiales du disque de gaz. Les particules (matières noires, trou noir central et vieille étoiles) seront générées par le code PyGME, [Emsellem et al., 1994], [Monnet et al., 1992], et se baseront sur les données issues du catalogue GAIA DR2 et la mise à jour DR3,

a. https://dmp.opidor.fr/

2. trois types de données seront produits avec la bibliothèque d'`E/S Hercule` : donnés brutes pour les protections/reprises du code (format spécifique associé, base de données `hprot`), des données destinées à l'analyse au format `lightAMR`, [Strafella and Chapon, 2022], (base de données `hdep`) et des données pour la communication scientifique (image `PNG`, videos `AVI`, histogrammes, etc),
3. les détails techniques et scientifiques de la simulation seront détaillés dans le manuscrit de thèse de L. Strafella, et feront l'objet d'un article destiné à la communication scientifiques des résultats,
4. une partie des données de protection/reprise sera stockée de manière pérenne $\simeq 30\ To$ sur la partition `STORE` du `TGCC` à des points clés de l'étude : conditions initiales relaxées, avant activation de la formation stellaire, avant les premières explosions de supernovae, et pourront donc être ré-utilisées. Environ $\simeq 20\ To$ de données d'analyse seront stockées sur la partition `STORE`, où sera effectuée les premières analyses. Ces données pourront être rapatriées, au besoin, sur les machines associées au laboratoire pour des analyses plus détaillées,
5. la simulation, ses méta-informations et des données réduites (spectres, profiles, projection `2D`) seront référencées sur l'application web `GALACTICA` à l'issue du projet et pourront être explorées par la communauté scientifique grâce à l'écosystème de traitement mis à disposition par cette plateforme (via un serveur *terminus* ayant accès aux données d'analyse). De plus, le traitement des données au format `lightAMR` se fera par le biais de l'outil d'analyse qui sera reversé à la communauté.

6.4 Comment économiser des millions d'heures de calcul ?

On peut aborder le problème sous un autre angle, à savoir *comment faire un maximum de choses avec un volume horaire fixé ?* Dans le cas de la simulation `ExaMilkyWay`, sans les développements que l'on va présenter aux deux sections suivantes, les 52 millions d'heures de calculs `CPU` attribuées au projet n'auraient pas été suffisantes pour faire évoluer suffisamment la galaxie à haute résolution. Typiquement, la relaxation du disque de gaz depuis les conditions initiales aurait consommé probablement plus de la moitié des heures et il n'aurait pas été possible de faire suffisamment évoluer le disque à haute résolution avec la formation stellaire et la rétroaction des étoiles. Des développements ont donc été nécessaires pour réduire le nombre d'heures de calculs dans les deux phases de la simulation, afin de pouvoir faire évoluer le disque à haute résolution le plus longtemps possible, afin d'essayer de simuler plus d'un cycle de formations stellaires et réinjection de l'énergie de rétroaction aux grandes échelles.

La première approche a été de réduire sa consommation mémoire afin d'éviter de devoir dépeupler les coeurs de calculs (le plus longtemps possible). Pour cela, il a fallut modifier les tableaux tampons de communication inter-processus. Cette approche a permis de ne pas avoir à dépeupler la simulation `ExaMilkyWay` et d'augmenter le nombre maximum de grille par processus `MPI`. En effet, le gain mémoire apporté est d'environ 64 *Go* par noeud, soit 1/4 de la mémoire totale d'un noeud, sur la partition `ROME` de la machine `Irène` au `TGCC`. Le gain estimé en volume horaire est d'environ 15 millions d'heures.

La deuxième approche a été d'aller à l'encontre de ce qui a été dit au chapitre 3 et d'exploiter le format compact d'analyse comme point de sauvegarde pour relancer la simulation en changeant le nombre de processus `MPI`. Cette approche nous a permis de faire la première partie de la simulation (à basse résolution) à bas coût horaire, sur seulement 4096 processus `MPI` et de démarrer la phase haute résolution en portant la simulation sur 16384 processus `MPI`. Le gain estimé en volume horaire est d'environ 20 millions d'heures `CPU`. Au total, avec ces deux approches, c'est donc pratiquement 35 millions d'heures "économisées", soit les 2/3 de l'allocation du projet. On notera que les heures de calculs n'ont pas été réellement économisées (au sens non utilisées) mais plutôt consommées uniquement dans la phase à haute résolution permettant ainsi de les utiliser pour le coeur du projet d'étude.

6.4.1 Modification des tableaux tampons de communication

En 2014, une étude de consommation mémoire de `RAMSES` menée à l'`IDRIS` (Institut du développement et des ressources en informatique scientifique), a mis en évidence la surconsommation mémoire de la structure de base *communicator*, utilisée dans les tableaux tampons qui permettent d'échanger des données entres les différents processus `MPI`. En effet, la taille de cette structure rapportée par la fonction `sizeof` est de 424 octets (cette valeur est susceptible de varier en fonction des compilateurs et des machines). Pour comprendre les raisons de cette valeur, il faut d'abord introduire la notion de *padding*. Lorsqu'une structure contenant des types de taille différentes est

déclarée, le compilateur peut choisir de rajouter des octets pour aligner les différents types sur une taille commune pour des raisons de performances. Par exemple, une structure composée d'un entier 8 octets suivi d'un réel 4 octets se verra rajouter 4 octets pour que les deux variables soient alignés en mémoire sur 8 octets chacune. De plus, il est important de noter que le `Fortran` utilise une gestion différente des tableaux par rapport aux pointeurs en `C`, par exemple. En effet, un tableau Fortran, même dynamique (*allocatable*) est représenté plutôt comme un *objet* contenant ainsi plus d'informations qu'un simple pointeur, tels que la taille, le rang, les bornes, etc. Ces informations peuvent ainsi être récupérées par les fonctions intrinsèques telles que `size`, `ubound`, `lbound`, `shape`, etc. Néanmoins, ces informations complémentaires ont un coût mémoire. On donne dans l'extrait 6.1, la structure de base `communicator` utilisée dans `RAMSES`.

```
1  type communicator
2    integer                                    :: ngrid
3      integer                                    :: npart
4      integer,          dimension(:),   pointer :: igrid
5      integer,          dimension(:,:), pointer :: f
6      integer(kind=8), dimension(:,:), pointer :: fp
7      real(kind=8),     dimension(:,:), pointer :: u
8      real(kind=8),     dimension(:,:), pointer :: up
9  end type communicator
```

Extrait 6.1 – Structure de communication utilisée dans `RAMSES`

Dans le rapport de l'`IDRIS`[b], il est proposé de modifier cette structure. En effet, dans le code, cette structure `communicator` est utilisée comme type de plusieurs tableaux dont certains en 2 dimensions : `emission`, `reception`, `active_mg`, `emission_mg`. Ces tableaux ont pour dimensions `(ncpu, nlevelmax)`, ils vont donc croître rapidement avec le nombre de processus `MPI` et le niveau `AMR` maximum. Pour une simulation sur 16384 processus, avec 19 niveau `AMR`, cela représente une consommation mémoire d'environ 125 Mo par processus, par tableau. Avec un configuration en $n_c = 1$ (sans dépeuplement), sur la partition `ROME` du `TGCC`, cela implique une consommation par noeud de 64 Go sur les 256 Go disponibles. Pour une simulation sur 50000 processus (comme celle envisagée initialement pour la simulation `ExaMilkyWay`) c'est environ 1.5 Go par processus, soit plus de 80% la mémoire disponible sur un coeur (la mémoire théorique par coeur est de 2 Go mais en pratique seul 1.8 Go sont accessibles pour les applications). Sur ce type de configuration, le dépeuplement est donc nécessaire, avant même tout dimensionnement du nombre de grilles, pour pouvoir lancer la simulation.

Concrètement, ces tableaux sont utilisés pour envoyer et recevoir des données entres les processus `MPI`. D'un point de vue algorithmique, un processus ne communique pas avec l'ensemble des processus de la simulation mais avec un nombre limité de processus voisins. L'empreinte mémoire de la structure `communicator` peut donc être réduite significativement en utilisant une structure intermédiaire :

```
1  type point_comm
2    integer         , dimension(:),   pointer :: igrid
3    integer         , dimension(:,:), pointer :: f
4    integer(kind=8), dimension(:,:), pointer :: fp
5    real(kind=8)    , dimension(:,:), pointer :: u
6  end type point_comm
7
8  type communicator_light
9    integer                     :: ngrid
10   integer                     :: npart
11   type(point_comm), pointer :: pcomm
12 end type communicator_light
```

Extrait 6.2 – Structure de communication alléger

Le tableau `communicator_light` est utilisé comme type pour les tableaux `reception` et `active_mg`, qui conservent leur taille initiale `2D (ncpu, nlevelmax)`. Cependant, cette nouvelle structure ne pèse plus que 16 octets, soit un gain mémoire de $\simeq$ 96 %. On notera que lorsqu'une communication est nécessaire, il faudra

b. http://www.idris.fr/docs/docu/support-avance/ramses.html

commencer par allouer le pointeur `pcomm`, puis ses variables internes. Cette réduction permet un gain par tableau d'environ 121 Mo pour la configuration $\{16384, 19\}$ mentionnée précédemment. Il est possible d'appliquer le même type d'approche pour les tableaux `emission` et `emission_mg` pour tenir compte des communications point à point. En effet, au lieu d'allouer un tableau de taille `ncpu`, il est préférable d'allouer un tableau en fonction du nombre de processus avec lesquels il faut effectivement échanger des données. Plusieurs passages sont alors nécessaires pour déterminer le nombre de processus avec qui échanger, puis faire une allocation des structures et enfin transférer les données. On renvoie le lecteur vers le rapport de l'`IDRIS` pour plus d'informations. Ces développements n'étaient pas disponibles dans la version principale du code, il a donc fallu les implémenter. On notera que ces modifications ont fait l'objet d'une soumission et ont été intégrées à la branche principale de `RAMSES` pour que l'ensemble de la communauté des utilisateurs puisse, enfin, en bénéficier.

6.4.2 Changement du nombre de processus `MPI`

Comme on l'a mentionné précédemment, il n'est pas possible de modifier le nombre de processus `MPI` d'une simulation lors d'une protection/reprise avec `RAMSES`. Ce problème relève en fait de la capacité d'un code à reconstruire ses structures mémoires à l'identique à partir de données d'entrées mais également de la granularité du modèle de données utilisé. Prenons l'exemple, d'un code `AMR` que l'on souhaite relancer à partir des données issues d'une simulation réalisée sur N processus `MPI`, vers une simulation sur P processus, avec le cas simple tel que $P = 2N$. Si le code utilise un modèle de données avec une grande granularité, le bloc "unitaire" se situant au niveau du processus `MPI` par exemple (comme pour le modèle `lightAMR`), lors de la reprise, un des P processus devra lire au moins un bloc de données cohérent, c'est-à-dire les données d'au moins un des N processus. Puis potentiellement faire des communications `MPI` pour répartir les données entre les P processus. Certaines données seront lues par plusieurs processus. Les coûts de lecture et de stockage mémoire temporairement d'un ou plusieurs blocs peuvent rapidement devenir un point bloquant (et éventuellement les coûts de communications inter-processus). On fait remarquer qu'il est possible d'augmenter temporairement la mémoire par processus, le temps de la reprise (dépeuplement par exemple). En revanche, si la granularité du modèle est beaucoup plus fine, par exemple sous la forme d'une liste de cellules (coordonnées et niveau, modèle *non structuré* par exemple), le problème est plus simple. Admettons que chaque processus gère 1000 cellules, le nombre total de cellule sera donc de $1000 \cdot N$. Dans ce cas, si la bibliothèque `E/S` le permet, il suffit de relire $1000 \cdot N/P$ cellules lors de la reprise, soit 500 cellules par processus de reprise. Cette approche est facile à mettre en place avec un paradigme *single-shared-filed* avec une bibliothèque comme `HDF5`.

Dans le cas de `RAMSES`, les données sont volumineuses, réparties entre de nombreux fichiers dont la structure est complexe. Faire une reprise en changeant le nombre de processus `MPI`, compte tenu de ces problèmes, est loin d'être une tâche facile. En conséquence, les utilisateurs doivent impérativement estimer les besoins et dimensionner leur simulation en fonction des ressources mémoires et calculs qui seront nécessaires au moment le plus critique de leur simulation. Lorsque l'on utilise un code avec du maillage adaptatif où la consommation mémoire évolue au cours de la simulation, cet exercice peut devenir difficile. Comme mentionné précédemment, dans le cadre de la simulation `ExaMilkyWay`, faire la relaxation à basse résolution sur un grand nombre de processus `MPI`, dimensionnés en fonction des besoins de la partie à haute résolution, a posé de nombreux problèmes de stabilité des communications au niveau de la couche `MPI`. En conséquence, j'ai développé une extension sous la forme d'un *patch*, qui permet de changer le nombre de processus. En dehors de potentielles contraintes mémoire (qu'on évoquera), il n'y a pas de restriction sur le nouveau nombre de processus (plus grand, plus petit, etc).

Bien que le modèle `lightAMR` ne soit pas dédié aux protections/reprises, il reste néanmoins très compact et on l'utilisera donc pour pallier la limitation de la reprise d'une simulation avec un nombre différent de processus `MPI`. Comme mentionné précédemment, puisque le modèle `lightAMR` a une grande granularité (cohérence des données du modèle associées au processus), certaines données seront lues par plusieurs processus. On tire profit des performances apportées par l'intégration de la bibliothèque `Hercule` et par la réduction du volume de données du modèle `lightAMR`, pour limiter l'impact des `E/S`. Avec l'approche utilisée, d'un point de vue théorique, le code n'effectue pas une protection/reprise mais une initialisation. C'est-à-dire qu'il a fallu développer un module de conditions initiales qui relise les données d'une base `HDep`. Puisque l'objectif est de relancer le code dans un état identique, il faut faire attention à utiliser des données sans compression (ou avec compression *sans perte*), et contenant tous les champs hydrodynamiques, particules, etc.., nécessaires.

Lorsque RAMSES initialise le maillage et les grandeurs associées, il fait appel au module de conditions initiales qui est généralement personnalisé par les utilisateurs en fonction du sujet d'étude. Cette routine fournit une liste de n cellules de taille dx, donc les coordonnées sont dans l'intervalle $[0.0, boxlen]^{DIM}$, et demande en retour la valeur des variables conservatives pour chaque cellule. Dans un premier temps, il faut déterminer le domaine d'appartenance de chacune des cellules par rapport à la décomposition de domaine de la simulation d'origine, afin de déterminer le domaine contenant les données. Cela se fait par le calcul de la clé de Hilbert et d'une recherche de l'intervalle correspondant dans les listes des clés. Les cellules sont triées en fonction de cette liste de manière à ce que les points appartenant au même domaine soient traités ensemble. Il suffit ensuite de charger en mémoire les données du domaine correspondant et de projeter les coordonnées 3D du point dans la description du tableau de raffinement au modèle lightAMR.

A l'initialisation, le maillage de la simulation va donc être reconstruit progressivement, et au court des itérations et des équilibrages de charge faits par le code, les frontières des domaines vont bouger. On notera que la routine condinit d'initialisation est appelée pour un paquet de nvector cellules (nvector étant un paramètre d'optimisation valant généralement 32 ou 64). Afin d'éviter de relire plusieurs fois le même domaine au cours des appels à la fonction, on utilise une structure tampon qui permet de conserver en mémoire un certain nombre de lightAMR. Lorsque la structure est remplie, on retire le domaine le moins utilisé, selon le même principe qu'un cache LFU (*Least Frequently Used*), [Maffeis, 1993]. On notera qu'un effet de "clignotement" peut parfois arriver à la frontière de domaine. C'est-à-dire que le tableau tampon est rempli et qu'il y a besoin d'alterner entre deux domaines. On estime que ce type de problème est accentué significativement dans le cas de la simulation ExaMilkyWay où beaucoup de domaines sont concentrés dans un petit volume (on rappelle que le disque de gaz représente 0.2 % du volume de la boite). La taille du tampon mémoire est paramétrable depuis le fichier de configuration. Cette structure contient les données du maillage : tree, mask, ncpl, respectivement, le tableau de raffinement et d'appartenance, le nombre de cellule par niveau, les données hydrodynamiques dans le tableau scf, et un tableau d'indirections nécessaires pour l'algorithme de projection. On notera que le code initialise d'abord le maillage et les grandeurs hydrodynamiques, puis les particules (en adaptant le maillage si besoin). On donne dans l'extrait la structure en Fortran 90 à titre indicatif.

```
1 type wrs
2   integer :: domainID
3   integer(kind=1), dimension(:),   allocatable :: tree, mask
4   integer(kind=8), dimension(:),   allocatable :: ncpl
5   double precision, dimension(:,:), allocatable :: scf
6
7   integer(kind=4), dimension(:), allocatable   :: dataIndexes
8 end type wrs
```

Extrait 6.3 – Structure de communication allégée

On rapporte sur la figure 6.2, la courbe du temps d'initialisation pour la simulation ExaMilkyWay et la courbe de temps d'initialisation avec changement du nombre de processus avec la méthode détaillée dans cette section. Ce graphique est donné à titre indicatif et permet d'estimer l'ordre de grandeur des temps d'initialisation, il doit être interprété dans le contexte d'une galaxie isolée. Il faut également tenir compte des caractéristiques de la simulation tels que le nombre de niveau AMR, le niveau minimum, le niveau maximum, etc. Sur la courbe en orange, les points à 1024, 16384 et 40000 MPI sont des tests de changement de nombre de processus depuis des bases de données produites respectivement sur 512, 4096 et 16384 processus. Compte tenu de la complexité des calculs de projection d'un point 3D dans la description lightAMR, qui est similaire voire moins élevée que celle de l'initialisation de la galaxie isolée, la différence entre les deux courbes est principalement dûe à la lecture des données. On notera qu'on utilise une tampon pouvant stocker les données de 50 processus, compte tenu de la mémoire disponible sur les noeuds de la partition ROME, au format lightAMR pour le passage à 1024, 30 pour le passage à 16384 et 10 pour 40000.

Il est très important de noter que les simulations à 512, 4096 et 16384 ont été faites en sachant qu'il est désormais possible de changer le nombre de processus MPI, en conséquence, j'ai adapté la résolution des simulations. Typiquement, sur ces simulations, la relaxation de la galaxie a donc été faite à basse résolution (niveau AMR $[9, 13]$ pour le 512 et $[10, 13]$ pour les autres), et le changement de processus MPI a été effectué **avant** d'augmenter la résolution, ce qui permet de limiter le coût de l'initialisation en réduisant le nombre de cellules. On notera cependant que l'augmentation de l'écart entre les deux courbes est lié à l'augmentation de l'empreinte mémoire du code de

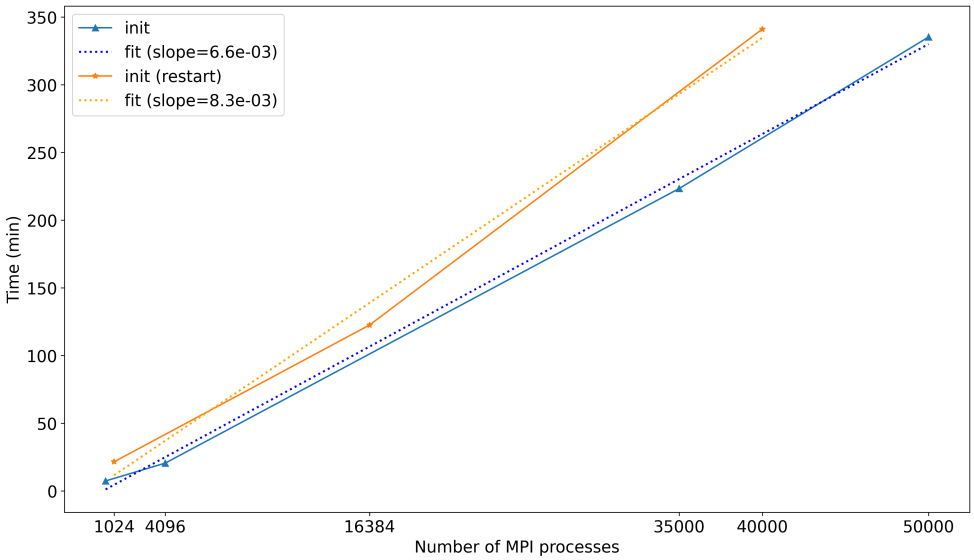

FIGURE 6.2 – Temps d'initialisation classique en bleu et depuis une une base `HDep` en orange pour une simulation de galaxie isolée avec des niveaux `AMR` dans l'intervalle $[9, 13]$ pour le run à 512 `MPI` et $[10, 13]$ pour les run $4096, 40000, 50000$ `MPI`.

manière générale, qui implique de réduire la taille du tampon, et donc à une augmentation du nombre des `E/S`. On notera qu'il est possible de dépeupler afin de réduire les coûts `E/S` aux prix d'une augmentation de la consommation des heures de calculs. Enfin, on notera qu'après l'initialisation, l'enveloppe feuille de la grille `AMR` et ses grandeurs sont identiques.

Pour d'autres types de simulation, comme par exemple les simulations cosmologiques, la problématique du changement de nombre de processus avec `RAMSES` est un point limitant. En effet, au cours de l'évolution de simulation et de l'expansion de l'univers simulé, les besoins en mémoire augmentent et les utilisateurs doivent généralement avoir recours au dépeuplement. L'avantage de ce type de simulation, comme on l'a mentionné précédemment, est d'avoir une répartition plus uniforme des domaines au sein de la boîte. En conséquence, même si la grille `AMR` est plus remplie et contient plus de niveaux, on peut s'attendre à un surcoût modéré par rapport à une initialisation classique. Dans notre cas, les particules n'ont pas fait l'objet d'une attention particulière parce que le nombre de particules est relativement faible (15 millions). Ainsi, tous les domaines lisent les particules, ce qui peut également expliquer l'écart à 40000 processus. Pour les simulations où les particules représentent une volume non négligeable, il faudra mettre en place une stratégie plus complexe. Par exemple, il est possible d'exploiter les intervalles de découpage des clés de Hilbert pour sélectionner les domaines de la simulation d'origine intersectés par le domaine courant (du processus de l'initialisation) pour définir les domaines qui devront être lus.

6.5 Conditions initiales

6.5.1 Le disque de gaz

Le disque de gaz initialisé dans les conditions initiales est défini par un profil radial et vertical exponentiel. Pour cela, on introduit deux grandeurs de coupures : r_c et z_c, respectivement le rayon de coupure et la hauteur de coupure. De plus, on définit ρ_0 comme étant le pic de densité au centre du disque. Dès lors, la densité du gaz dans le disque s'exprime suivant la relation

$$\rho(r, z) = \rho_0 \, f(r) \, h(z) \tag{6.1}$$

Pour un profil exponentiel, on a introduit également les quantités r_0, z_0, permettant d'affiner la vitesse

de décroissance de l'exponentielle afin d'obtenir une disque plus ou moins piqué au centre :

$$\begin{cases} f(r) = \exp(-r/r_0) \\ \\ h(z) = \exp(-z/z_0) \end{cases} \tag{6.2}$$

La constante ρ_0 est choisie en fonction de la masse du disque renseignée par l'utilisateur M_{gaz} et peut s'exprimer analytiquement :

$$\rho_0 = \frac{M_{\text{gaz}}}{2 \int_0^{2\pi} \int_0^{r_c} r f(r) dr d\theta \int_0^{z_c} h(z) dz} \tag{6.3}$$

On a donc :

$$\begin{aligned} \int_0^{r_c} r f(r)\, dr d\theta &= 2\pi \left[-r\, r_0 \exp(\frac{-r}{r_0}) \right]_0^{r_c} + \int_0^{r_c} r_0 \exp(\frac{-r}{r_0}) \\ &= 2\pi\, r_0^2 \left[1 - \exp(\frac{-r_c}{r_0}) \left(1 + \frac{r_c}{r_0} \right) \right] \end{aligned}$$

Et :

$$\int_0^{z_c} \exp(\frac{-z}{z_0}) dz = z_0 \left(1 - \exp(\frac{-z_c}{z_0}) \right)$$

Finalement,

$$\rho_0 = \frac{M_{\text{gaz}}}{4\pi\, r_0^2\, z_0 \left[1 - exp(\frac{-z_c}{z_0}) \right] \left[1 - \exp(\frac{-r_c}{r_0}) \left(1 + \frac{r_c}{r_0} \right) \right]} \tag{6.4}$$

Pour la simulation `ExaMilkyWay` on prend une masse de gaz $M_{gaz} = 5.94 \cdot 10^9\ M_\odot$, un rayon de coupure radial $r_c = 15\ kpc$ et un rayon de décroissance $r_0 = 4.5\ kpc$. La hauteur de coupure du disque est de $z_c = 1.5\ kpc$, avec une hauteur de décroissance à $z_0 = 0.14\ kpc$. On notera que le ratio de densité entre les bords du disque de gaz et le milieu inter-galactique est défini à 10^{-6}. Ainsi, la masse totale de gaz dans la boite de simulation à $t = 0$ est de $\simeq 6.04 \cdot 10^9$ $M_\odot$. Avec ces paramètres, on peut calculer quelques densités clés, par exemple le pic central $\rho_0 = 0.2\ M_\odot \cdot pc^{-3}$, ou au bord du disque dans le plan équatorial $\rho(r = r_c, z = 0.0) = 7.0 \cdot 10^{-3}\ M_\odot \cdot pc^{-3}$. Enfin, le disque de gaz est orienté suivant l'axe z, dans le plan (x, y). Il est également centré au milieu de la boîte de simulation dont les conditions aux bords sont positionnés à "isolés".

6.5.2 Particules : matières noires, vieilles étoiles et trou noir

En plus du disque de gaz, il faut également définir des particules dans les conditions initiales. En effet, on renseigne trois types de particules : matière noires, vieilles étoiles et trou noir central. Les particules de matière noire forment un halo de 50 *kpc* de rayon constitué de $10 \cdot 10^6$ particules, traitées de manière non collisionnelle. La masse totale des particules de matière noire est de $4.53 \cdot 10^{11}\ M_\odot$. Les particules de *vieilles étoiles* sont également traitées comme des particules non collisionnelles et le resteront tout au long de la simulation. On donne sur la figure 6.3 la distribution des particules de vieilles étoiles en densité de surface, vu suivant l'axe z sur l'image de gauche et une vue par la tranche sur l'image de droite. Il y a $5 \cdot 10^6$ de particules de ce type pour une masse totale de $4.597 \cdot 10^{10}\ M_\odot$. Enfin, un trou noir central super-massif de $4.0 \cdot 10^6\ M_\odot$ est rajouté.

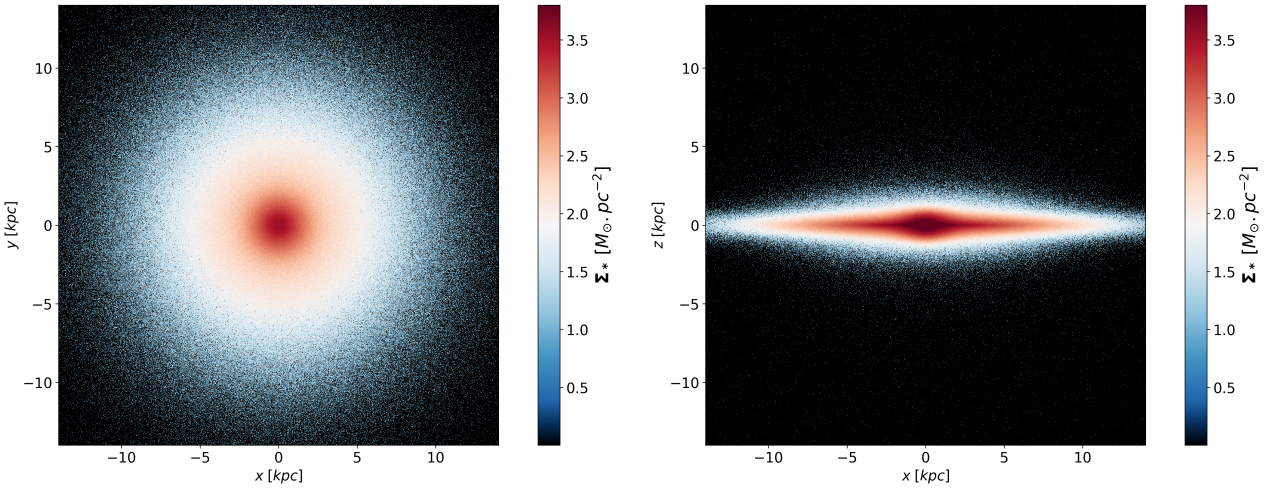

FIGURE 6.3 – Image en vue par le dessus (à gauche) et par la tranche (à droite) des conditions initiales de la densité de surface des particules de vieilles étoiles.

On note dans un premier temps qu'avec cette configuration, le rapport entre la masse totale d'étoiles sur la masse totale du gaz dans les conditions initiales est de l'ordre de $M_g/M_* \simeq 11$ %. De plus, puisque c'est le halo de matière noire qui est le plus étendu spatialement, il va définir la taille de la boite de simulation, que l'on fixera à 120 *kpc*. Avec ce paramètre, le volume du disque de gaz représente moins de 0.15 % du volume total de la boite.

Sur la figure 6.4, on donne les courbes de vitesse circulaire pour les particules et le gaz dans les conditions initiales en fonction du rayon à partir du centre de masse (centre de la galaxie). On fait ici l'approximation d'une distribution sphérique, voir [Binney and Knebe, 2002]. La courbe pour les particules de vieilles étoiles est donnée en bleu. La courbe noire est associée aux particules de matière noire et intègre également le trou noir central. La vitesse associée du disque de gaz, dans les conditions initiales, est donnée par la courbe en vert foncé. Enfin la courbe rouge, représente la somme des contributions de ces différents éléments. On notera la dominance des particules de vieilles étoiles pour un rayon $0.0 < r < 7.5$ *kpc* puis l'impact de la matière noire pour $r > 7.5$ *kpc*. On rappelle que ce type de courbe était la preuve historique de la présence de matière "non visible" impactant la courbe de vitesse de rotation des disques galactiques observés, [Zwicky, 1933].

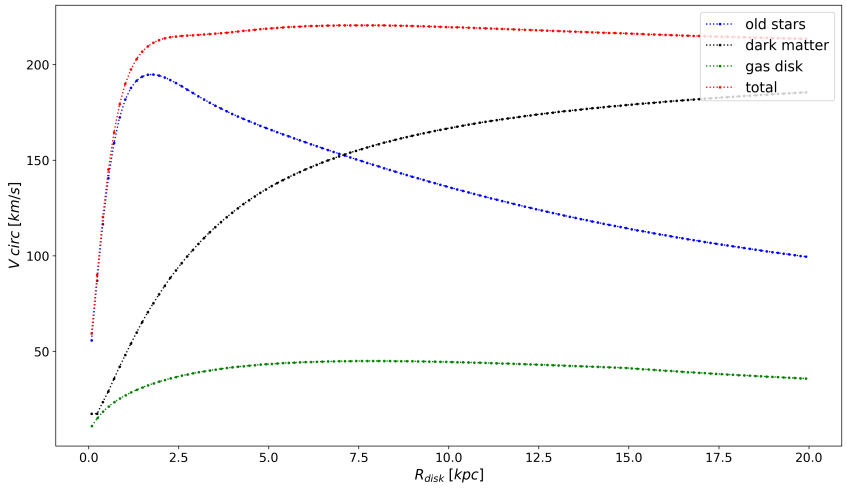

FIGURE 6.4 – Vitesse circulaire dans les conditions initiales de la simulation. En bleu la vitesse circulaire des particules des vieilles étoiles, en noir celle du halo de matière noire et le trou noir central, en vert celle associée au gaz. La courbe rouge représente la vitesse circulaire totale pour ces trois contributions.

6.6 Modèles physiques

Dans cette section, on détaille les différents modèles physiques utilisés ainsi que leur paramétrisation pour la formation stellaire, la rétroaction des particules d'étoiles et la thermodynamique du gaz. On rappelle que ces modèles sont activés lors de la deuxième phase de la simulation (haute résolution). Pour des raisons de capacité mémoire et de temps de calcul, compte tenu de l'envergure de la simulation, il n'est pas possible de rajouter d'autres modèles pour tenir compte de processus physiques qui pourraient avoir un impact sur le taux de formation stellaire ou la turbulence tel que : *le vent stellaire*, [Gatto et al., 2016], la rétroaction des régions `HII`, [Brucy et al., 2020], ou encore le champ magnétique. Il s'agit d'une approximation acceptable puisqu'il a été montré que le champ magnétique à un impact secondaire sur la `SFR` comparativement à la turbulence [Federrath and Klessen, 2012]. Néanmoins, il a été montré que le champ magnétique introduit une composante additionnelle de pression qui peut avoir un impact sur la structure des bras spiraux, [Pakmor and Springel, 2013].

6.6.1 Modèle de formation stellaire

On utilise un modèle de formation stellaire qui se base sur un critère de densité seuil, que l'on notera densité critique ρ_{crit}, au-delà de laquelle une cellule sera considérée comme pouvant potentiellement former une ou des particules d'étoiles. On choisit $\rho_{crit} = 3000\ H \cdot cm^{-3}$ estimé à partir de cartes de densité maximum le long de la ligne de visée, voir section 6.7. L'objectif recherché avec cette valeur seuil est d'activer la formation stellaire aussi bien au sein de la barre centrale que dans les bras spiraux. Il n'est pas possible de résoudre à l'échelle de toute une galaxie la physique d'objet de l'ordre d'une masse solaire, à cause du trop grand nombre de particules qu'il faudrait gérer, de la complication d'échantillonage de l'`IMF` et de la nécessité d'utiliser une approche collisionelle. On définit donc la masse minimale d'une nouvelle particule à $m_* = 50\ M_\odot$. De plus, la création de nouvelles particules dans `RAMSES` est un processus aléatoire similaire à celui proposé par [Katz, 1992]. En effet, à chaque pas de temps d'intégration Δt, on évalue la quantité de gaz, $m_{g,*}$ à former dans une cellule validant le critère de formation stellaire en fonction de la masse de gaz disponible m_{cell} et du temps de chute libre :

$$m_{g,*} = \frac{\Delta t}{t_{ff}} \cdot m_{cell} \cdot \epsilon_* \tag{6.5}$$

Le paramètre ϵ_* permet de contraindre artificiellement la formation stellaire, pour rester dans une zone correspondante aux observations, de l'ordre de $1 - 10\ M_\odot \cdot yr^{-1}$ pour la Voie lactée, [Krumholz and Tan, 2007]. Ce paramètre a été calibré empiriquement au début de l'activation de la formation stellaire et sera finalement positionné à $\epsilon = 0.01$, soit 1%. Il est important de noter que ce paramètre est ajusté en fonction de la physique résolue dans la simulation, puisque certains processus peuvent plus ou moins réguler la `SFR`, comme on l'a mentionné précédemment. La masse d'une nouvelle particule est évaluée selon une loi de Poisson, [Rasera and Teyssier, 2006]. En effet, on tire un entier n à partir de la relation suivante :

$$P(n) = \frac{\lambda}{n!}\ e^{-\lambda} \tag{6.6}$$

dont la moyenne λ est donnée par le rapport de la masse de gaz à former durant le pas de temps Δt et la masse unitaire m_* :

$$\lambda = \frac{m_{g,*}}{m_*} \tag{6.7}$$

Ainsi, la masse de la nouvelle particule est de $M_* = n \cdot m_*$. On peut également le voir comme la formation de n particule de masse égale. En conséquence, on retire $n \cdot m_*$ masse de gaz dans la cellule tout en s'assurant de ne pas dépasser plus de 90% de la masse de gaz disponible, pour éviter de générer de forts gradients de pression. La particule est formée au centre de la cellule parent et se voit attribuée une vitesse égale à la vitesse locale du fluide. Bien évidemment, par souci de conservation, les grandeurs : masse, moment et énergie interne, de la cellule parent sont corrigées.

6.6.2 Modèle de rétroaction stellaire

Comme on l'a vu précédemment, la rétroaction stellaire est l'injection d'énergie, de masse et de moment par les étoiles dans l'environnement qui les entoure. Il existe des modèles numériques de rétroaction pour tenir compte des différents processus physiques : photoionisation, pression radiative, région `HII`, explosion de supernovae, etc, qui influent sur la formation stellaire en perturbant et détruisant les nuages moléculaires, [Keres et al., 2009]. D'un point de vue théorique on notera qu'en l'absence de rétroaction, la matière baryonique refroidirait rapidement et efficacement pour former des étoiles et produirait une masse stellaire, plusieurs ordres de grandeurs au-dessus de ce qui est observé, en plus de consommer une grande partie du gaz disponible dans la galaxie. Il est donc important de tenir compte d'un modèle de rétroaction stellaire. Les supernovae jouent un rôle fondamental dans la rétroaction des étoiles, de part la quantité importante d'énergie et de moments qu'elles permettent de réinjecter dans le milieu interstellaire, [Mac Low and Klessen, 2004]. Dans différentes études, il a été montré que la rétroaction des supernovae peut efficacement entretenir la turbulence aux larges échelles dans le milieu interstellaire, [Dubois and Teyssier, 2008], [Bournaud et al., 2010], [Hennebelle and Iffrig, 2014]. En conséquence, on intègre un modèle de rétroaction des supernovae.

Comme on l'a vu au chapitre 5, il y a plusieurs phases dans l'explosion d'une supernova. Aussi, au-delà d'un certain rayon d'action, l'énergie devient principalement cinétique. Néanmoins, il est désormais clair que modéliser l'injection d'énergie uniquement sous la forme thermique n'est pas suffisant parce que dans les milieux à forte densité, toute l'énergie serait dissipée rapidement, sans avoir le temps d'induire la phase cinétique, [Navarro and White, 1993]. Il y donc une dépendance au milieu environnant et donc, dans le cadre de simulation numérique, une dépendance à la résolution. En conséquence, de nombreux modèles proposent de corriger cela pour tenir compte de la partie cinétique. En effet, si la résolution est suffisamment grande, inférieure à 0.1 pc, alors il est raisonnable d'utiliser un modèle thermique (ou mixte : thermique et cinétique), parce que le rayon d'action sera suffisamment résolu pour permettre la mise en place de la phase cinétique. Dans notre cas, la résolution maximale de la simulation est de l'ordre de ~ 1 pc. Le risque avec un modèle thermique serait donc la dissipation précoce de l'énergie avec la mise en place de la phase cinétique. On va donc plutôt utiliser une approche mettant directement en place la solution d'une onde de choc sphérique. La rétroaction des `SNe` sera donc purement cinétique. Il s'agit d'une modèle déjà implémenté et disponible dans `RAMSES`, [Dubois and Teyssier, 2008].

L'énergie caractéristique d'une supernovae est de l'ordre de 10^{51} erg pour une étoile d'une masse de l'ordre de $10\mathrm{M}_\odot$. Ainsi, après une vie de 10 Myr, une fraction de la masse des particules stellaires formées dans la simulation va exploser sous la forme d'une supernovae et l'énergie réinjectée dans le `MIS` sera de $E \simeq \eta_{SN} \cdot M_* \cdot 10^{51}/10\mathrm{M}_\odot$ erg. Il faut donc manuellement ajuster le paramètre η_{SN} pour que $\eta_{SN} \times M_*/10\mathrm{M}_\odot \sim 10^{51}$ erg. En prenant une M_* de l'ordre de $50\mathrm{M}_\odot$, on choisira donc un $\eta_{SN} = 0.2$. De plus, on supposera que chaque `SN` réinjecte la moitié de son énergie sous forme cinétique. On rajoute aux variables du fluide la solution d'une onde de choc de rayon r_b. La valeur du rayon est un paramètre important qui doit être suffisamment grande comparée à la résolution spatiale de la grille, de l'ordre de quelques cellules, pour minimiser les effets parasites de perte d'énergie. Ce rayon permet de définir l'énergie d'injection des supernovae dans le milieu, [Mac Low and Klessen, 2004]. Avec une résolution spatiale dans les régions de formation stellaires de l'ordre de 1.8 pc, on choisit un rayon $r_b = 10$ pc.

6.6.3 Modèle thermodynamique du gaz : chauffage / refroidissement

L'objectif de la première partie de la simulation est de laisser le disque de gaz se relaxer dans le potentiel gravitationnel afin de former les grandes structures : bras spiraux et barre centrale. Pour ce faire, on maintient le gaz à une température de $\sim 10^4$ K durant toute la première partie de la simulation (~ 980 Myr). Dans la seconde partie de la simulation, on utilise une fonction de chauffage/refroidissement semi-analytique du gaz, [Audit, E. and Hennebelle, P., 2005], plutôt qu'une fonction de pseudo-refroidissement instantané, [Teyssier et al., 2010]. Cette fonction permet de définir une variation de la température d'une cellule qui peut être soit positive, chauffage, soit négative, refroidissement. Ce modèle permet de tenir compte des différents processus qui ont lieu dans le gaz atomique, tels que l'émission de photons par des atomes d'oxygène ou de carbone ionisés, ou encore le chauffage par effet photoélectrique. On notera qu'on introduit également deux seuils en température pour éviter d'avoir des températures au-delà de $6 \cdot 10^6$ K et en-deçà de 20 K.

De plus, afin d'éviter la fragmentation numérique dans une simulation de fluide auto-gravitant, [Truelove

et al., 1997] ont montré que la longueur de Jeans doit être résolue par au moins quelques cellules. Ainsi, le critère de Truelove au niveau l définit par $\lambda_J \geq 4 \cdot \Delta x_l$ où Δx_l est la taille d'une cellule au niveau l est imposé dans la simulation. Au niveau maximum l_{max}, on atteint la taille minimale autorisée d'une cellule Δx_{min}. Pour continuer de valider ce critère, on augmente artificiellement la température du gaz en imposant un seuil en température définit par un polytrope tel que $T \propto C_p \rho$. La constante C_p est donnée par la relation 6.8 dépendante de la résolution Δx_{min}, de l'indice adiabatique γ et de la constante de gravitation G. La mise en place de ce polytrope permet d'éviter efficacement la formation d'amas se fragmentant à cause d'une résolution limitée, [Robertson and Kravtsov, 2008]. On notera que cette méthode peut être interprétée comme un modèle sous-grille pour les processus physiques non résolus intervenant en dessous de la résolution maximale.

$$C_p = \frac{16G}{\pi\gamma} \left(\Delta x_{min}\right)^2 \tag{6.8}$$

6.7 Résultats

6.7.1 Méthode d'analyse

On effectue toutes les analyses avec l'outil d'analyse, détaillé au chapitre 4, qui produit des données réduites brutes : carte 2D de quantité intégrée le long d'une ligne de visée permettant de produire des cartes de densité surfacique, par exemple. Afin d'optimiser le nombre de cellules à traiter et le nombre de domaines à lire, les analyses du disque de gaz entier se basent sur un filtrage géométrique des domaines et des cellules, à l'aide d'une région ellipsoïdale englobant tout le disque. Même les analyses sous forme de vignette ou de région cubique 3D, englobent tout le disque, afin d'éviter tout risque d'oubli ou de trou dans les données. On rappelle également que pour éviter tout problème avec les unités, l'outil d'analyse travaille en unité "code", et donc toutes les conversions d'unités se font à partir de script Python, dans un deuxième temps.

Au chapitre 5, on a introduit la turbulence d'un point de vue énergétique au travers de la théorie de Kolmogorov. On a vu que la cascade d'énergie présente trois zones importantes : injection, transfert et dissipation, et plus précisément que le transfert d'énergie peut se représenter sous la forme d'une loi de puissance dont on a détaillé les pentes attendues en fonction de la dimension. Les spectres de puissance sont un outil particulièrement utilisé dans la littérature afin d'étudier la turbulence dans le MIS. Néanmoins, il faut faire attention aux grandeurs utilisées ainsi qu'à la comparaison des spectres par rapport à l'observation. En effet, les observations sont limitées par les observables, c'est-à-dire des grandeurs moyennées le long de la ligne de visée, alors que les simulations ont naturellement accès aux 3 dimensions. Dans notre cas, bien qu'on pourrait utiliser les données 3D, le volume de données que cela représente pose problème. On calculera donc des spectres de carte 2D produites par la méthode du *slicing*, voir chapitre 4. Les transformées de Fourier sont calculées à l'aide de la bibliothèque *Numpy* sur les cartes brutes produites par l'outil d'analyse présenté au chapitre précédent. A l'avenir, des spectres 3D par échantillonage dans une grille régulière permettront d'enrichir les analyses de la simulation.

On donnera également, pour comparaison, les spectres issus de cartes produites par intégration de long de la ligne de visée, dont on discutera la pertinence. De plus, la théorie K41 (et ses extensions) se base sur le spectre de puissance de l'énergie et donc la vitesse. Or, certaines études basées sur des simulations numériques, telles que [Kim and Ryu, 2005], ont montré que le spectre de puissance de la densité est également une loi de puissance. Dans le cas d'un fluide subsonique, sa pente est identique à celle de la vitesse. En effet, il y a une forte corrélation densité-vitesse, ce qui n'est plus forcément le cas dans un régime supersonique. Il est important de noter que le spectre de cette grandeur permet de mettre en évidence le nombre de structures aux différentes échelles. Ils ont également mis en évidence que dans le régime supersonique, les chocs créent des fluctuations de densité aux petites échelles. [Saury et al., 2014] ont constaté le même comportement dans le cas d'un milieu subsonique non isotherme avec deux phases (WNM et CNM). On calculera donc les spectres de carte de densité en plus des spectres de vitesses.

6.7.2 Résultats

6.7.2.1 Relaxation du disque

On présente sur la figure 6.5, la densité surfacique du disque de gaz, en $M_\odot \cdot pc^{-2}$, après $\sim 980\ Myr$ d'évolution à basse résolution. Les particules du disque stellaire des conditions initiales ont une dispersion de vitesse suffisante pour être stable par rapport aux instabilités axisymmétriques mais instable pour la formation d'une barre. On constate donc la formation des structures à grande échelle tels que les bras spiraux et la barre centrale qui mesure environ 4.5 kpc. On notera que sur les figures, le sens de rotation du disque est le sens trigonométrique. De plus, dans cette première partie on rappelle que la résolution spatiale est comprise entre de 127 pc et 14.7 pc. La figure 6.6 donne les niveaux `AMR` maximum le long de la ligne de visée, en vue par le dessus (ligne de visée suivant l'axe z) et en vue par la tranche (ligne de visée suivant l'axe y). Durant cette première partie, 45% de la masse de gaz est résolue à 29.3 pc (niveau 12) et 24% à 14.7 pc (niveau 13).

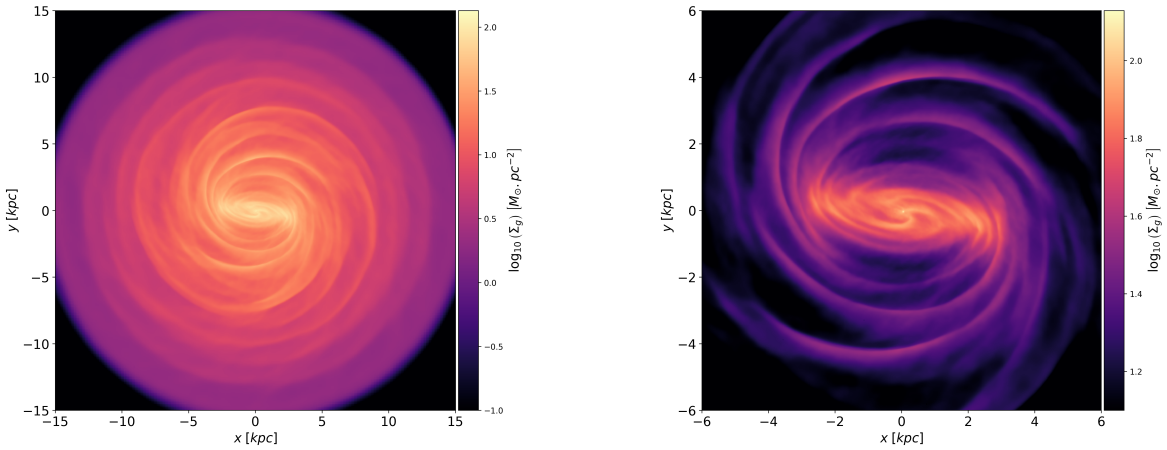

FIGURE 6.5 – Densité de surface du disque de gaz à la fin de l'étape de relaxation à $t = 980\ Myr$ en vue par le dessus. L'image de gauche englobe la totalité du disque de gaz, et l'image de droite est un zoom sur la partie centrale de la carte de gauche (l'échelle de couleur est ajustée à l'intervalle de valeur sur chaque image). La résolution spatiale est de 14.7 $pc/pixel$.

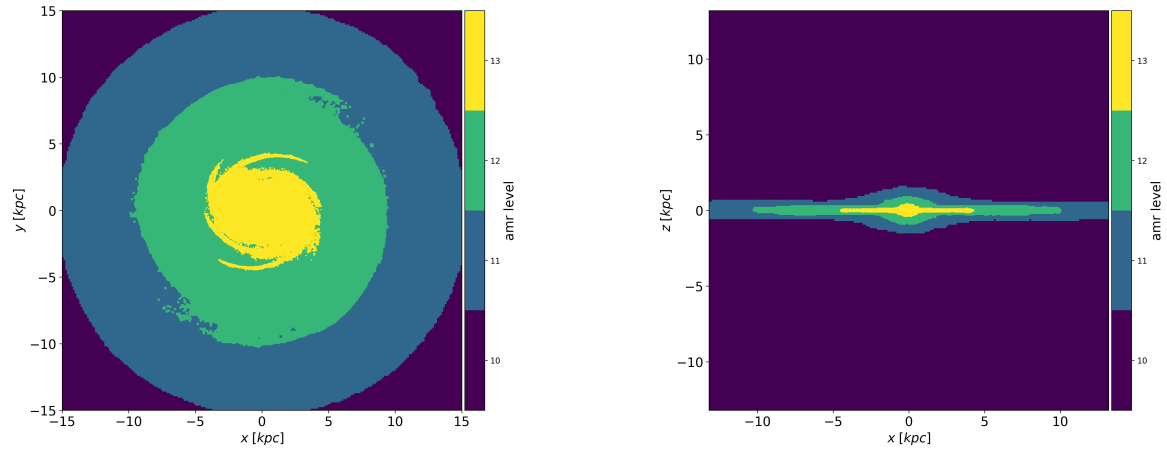

FIGURE 6.6 – Niveau `AMR` maximum suivant la ligne de visée. A gauche, en vue par le dessus et à droite, en vue par la tranche, à $t = 980\ Myr$. La résolution spatiale est de 14.7 pc par pixel.

Les `PDF` de densité du gaz sont un outil statistique intéressant pour avoir un aspect global de la turbulence compressible. En effet, elles permettent d'avoir une signature de la thermodynamique du gaz [Passot and Vá zquez-

Semadeni, 1998], des instabilités gravitationnelles, [Kritsuk et al., 2010], et de la présence de chocs. Typiquement, dans un fluide isotherme avec une turbulence supersonique, la PDF présente une forme qui peut être approximée par une courbe *log-normal*, [Vazquez-Semadeni, 1994], [Kritsuk et al., 2007]. L'origine de cette approximation est liée à la turbulence dans le gaz, qui va générer des chocs qui vont compresser le gaz. Une explication intuitive est proposée par [Kevlahan and Pudritz, 2009]. En partant de l'hypothèse que les chocs sont des évènements aléatoires, si on considère un élément de fluide de densité ρ_0, celui-ci verra sa densité multipliée par le nombre de Mach associé aux chocs successifs. Au bout de i chocs, $\log(\rho) = \log(\rho_0) + \sum_j \log(\mathcal{M}_j)$. On retrouve la forme d'une *log-normale*, grâce à l'application de la fonction logarithme et du théorème central limite. Ainsi, la forme de la PDF du gaz, normalisée, à $t = 980\ Myr$, donnée sur figure 6.7, est un résultat attendu.

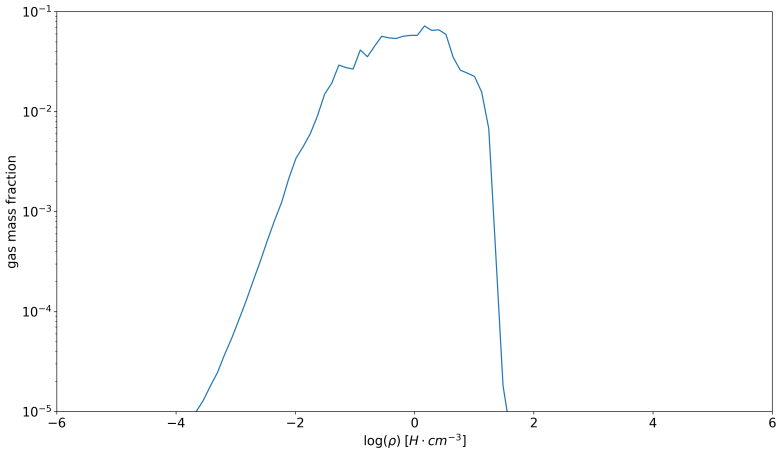

FIGURE 6.7 – Fonction de distribution de la masse de gaz (PDF) normalisée à $t = 980\ Myr$.

6.7.2.2 Haute résolution

Dans cette seconde partie de la simulation, le temps est "réinitialisé" suite au changement du nombre de processus. Ainsi $t = 0\ Myr$ doit être interprété comme le début de la seconde partie de la simulation, à partir des données du disque relaxé sur 4096 processus MPI. On active le module de chauffage/refroidissement du gaz et on augmente progressivement les niveaux AMR à l'aide d'un critère de raffinement sur la masse. On notera que le critère est ajusté de manière à atteindre une résolution inférieure ou égale à 3.4 pc dans les régions les moins denses du disque, entre les bras. La densité de surface du gaz augmente avec le refroidissement et les structures s'affinent, voir figure 6.8. On voit apparaître dans les régions de faible densité, entre les bras spiraux et au niveau de la barre centrale, des traînées de gaz. On verra par la suite que ces traînées vont former des filaments denses, voir figure 6.16.

Sur le graphique de gauche de la figure 6.10, on donne l'évolution de la PDF du gaz durant la montée en résolution jusqu'à 9.4 Myr, et à droite, le diagramme densité - température à 9.4 Myr. Sur les PDF, on retrouve la bosse caractéristique du WIM à $10^4\ K$ autour de $1\ H \cdot cm^{-3}$. On voit apparaître un pic de masse de gaz à partir de $t = 2.1\ Myr$ à environ $10^3\ H \cdot cm^{-3}$ qui va augmenter progressivement au cours du temps. En dessous de $\sim 1\ H \cdot cm^{-3}$, la PDF du gaz n'évolue pas. Pour les densités au-delà de $10^2\ H \cdot cm^{-3}$, le polytrope chauffe le gaz, apportant ainsi un support de pression thermique qui va contre-balancer l'effondrement gravitationnel du gaz, voir graphique de droite sur la figure 6.10. Le point majeur dans ces courbes, est l'accumulation du gaz dans le pic de densité à $10^3\ H \cdot cm^{-3}$. Il est important d'activer la formation stellaire pour consommer ce gaz avant d'en accumuler trop et de former des structures non physiques, [Bournaud et al., 2010]. Au bout de 10 Myr, d'évolution on déclenche donc le module de formation stellaire.

L'épaisseur du disque de gaz étant particulière faible par rapport à son rayon, on se place en coordonnées cylindriques pour calculer les composantes du champ de vitesse v_r, v_θ, v_z. On notera qu'on prend le centre du disque aux coordonnées $(60, 60, 60)$ comme centre du référentiel. On néglige donc les très faibles variations de la position du centre de masse. Les cartes des composantes v_r et v_θ projetées en vue de face et pondérées par la masse sont données sur la figure 6.11. On estime à partir de la densité de surface, la position de la barre centrale sous la forme

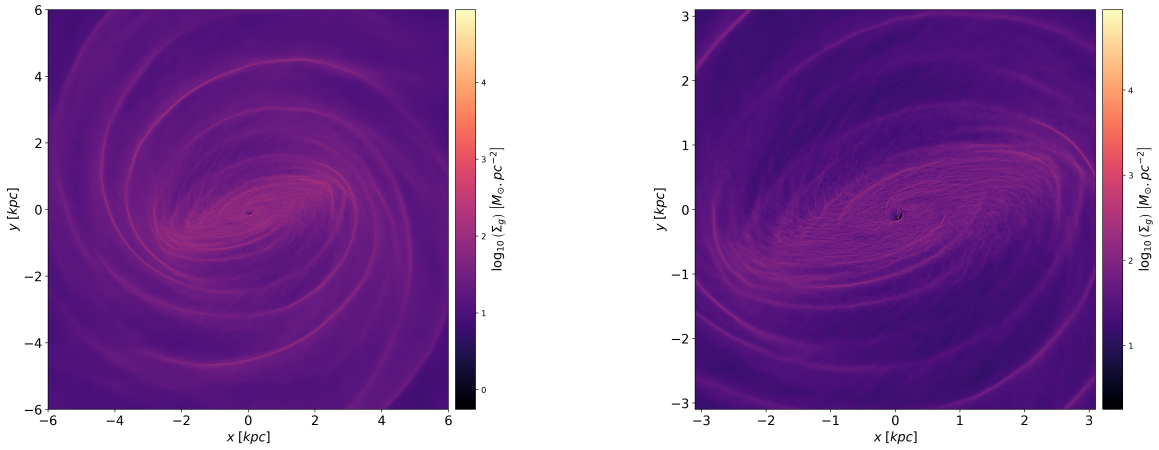

FIGURE 6.8 – Densité de surface du disque de gaz à $t = 9.4\ Myr$ en vue de face avec une résolution spatiale de 2.9 $pc/pixel$. La carte de gauche englobe la majorité du disque de gaz, et celle de droite est un zoom sur la partie centrale (l'échelle de couleur est ajusté à l'intervalle des valeurs sur la carte).

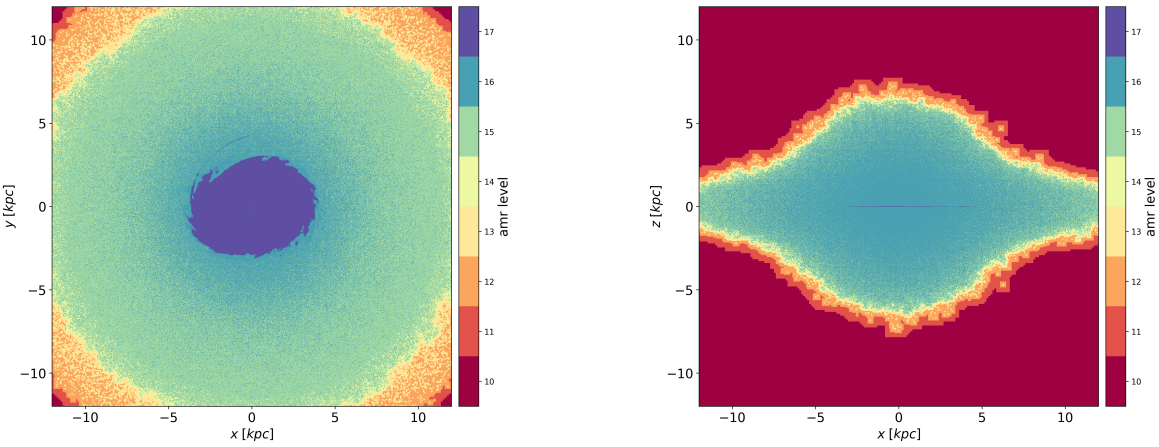

FIGURE 6.9 – Niveau `AMR` à $t = 9.4\ Myr$ maximum le long de la ligne de visée, à gauche, en vue de face, et à droite, en vue par la tranche. La résolution spatiale est de 2.9 $pc/pixel$.

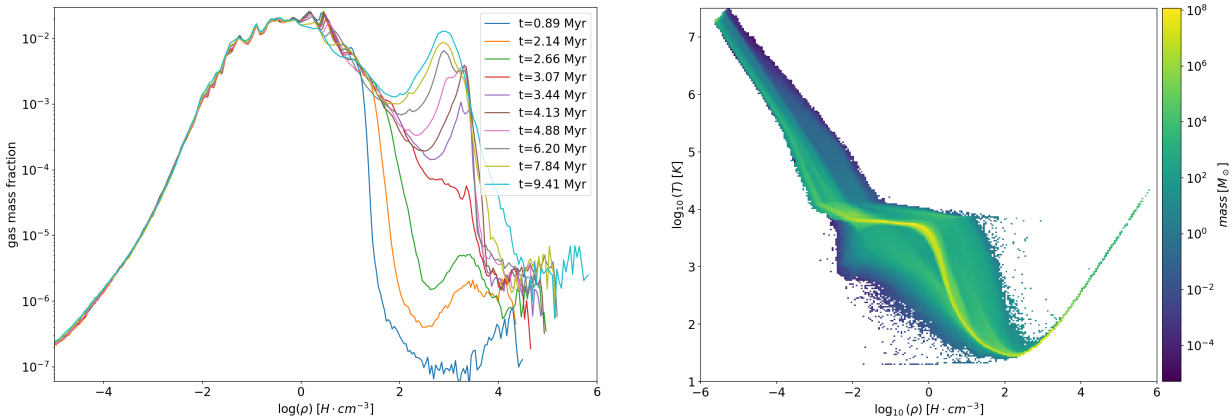

FIGURE 6.10 – Évolution de la `PDF` du gaz lors de la montée en résolution jusqu'à $t = 9.4\ Myr$ à gauche. A droite, le diagramme (ρ, T).

d'une ellipse, que l'on trace en pointillé sur les cartes de vitesse. Les caractéristiques de l'ellipse sont : grand axe 2.3 kpc (4.6 kpc de long) et de petit axe ~ 1 kpc de large.

Ces deux cartes mettent en avant le cisaillement important qui intervient au niveau des bras spiraux, aussi bien pour la composante radiale qu'angulaire. On voit également que la vitesse radiale est négative à l'avant de la barre dans le sens de rotation et positive à l'arrière de la barre. En effet, le gaz à l'avant est ralenti, attiré par la gravité de la barre, et chute vers le centre. Sur la carte de vitesse azimutale, la région de la barre a une vitesse moins élevée, de l'ordre de 100 $km \cdot s^{-1}$ par rapport au reste du disque. On notera la légère augmentation de la vitesse v_θ au niveau du petit axe de l'ellipse.

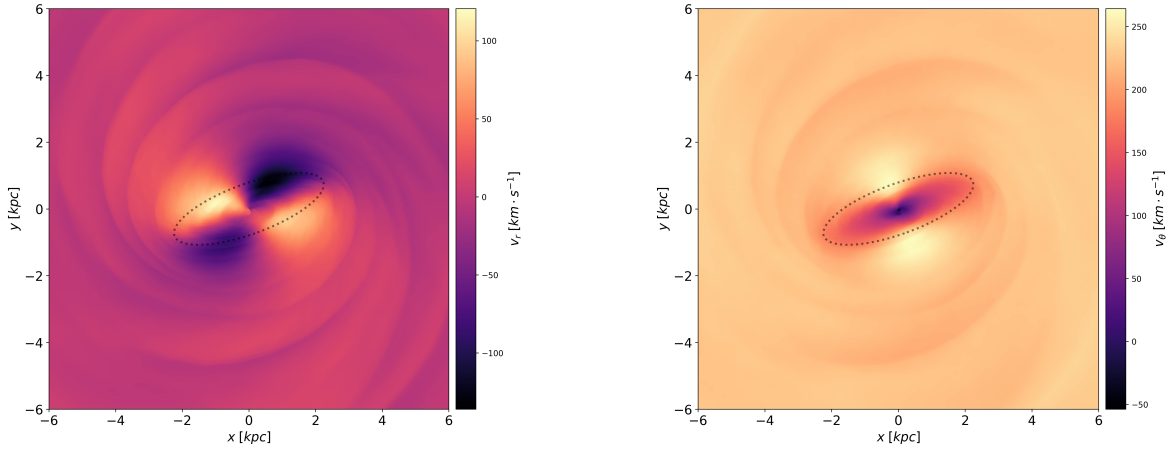

FIGURE 6.11 – Carte 2D de vitesse des composantes r et θ, pondérées par la masse, en vue de face à $t = 9.4\ Myr$, exprimées en $km \cdot s^{-1}$. A gauche, la composante radiale et à droite la composante angulaire, (système de coordonnée cylindrique centré en $(0, 0, 0)$).

6.7.2.3 Fragmentation et formation stellaire

Dans cette section, on s'intéressera à la fragmentation du gaz et à la formation des particules d'étoiles. Sur le graphique 6.12, on donne les grandes étapes de la simulation, à savoir : l'activation de la formation stellaire à $t \simeq 10\ Myr$ et les premières SNe à $t \simeq 18\ Myr$, ainsi que l'évolution de la masse résolue au cours du temps en fonction des différents niveaux AMR (donc de la résolution spatiale). Les courbes sont mutuellement exclusives, c'est-à-dire qu'à $t = 26\ Myr$ par exemple, il y a $\sim 45\%$ de la masse qui est résolue à $0.9\ pc$, $\sim 10\%$ à $1.8\ pc$, $\sim 30\%$ à $3.4\ pc$ et 10% à $7.3\ pc$.

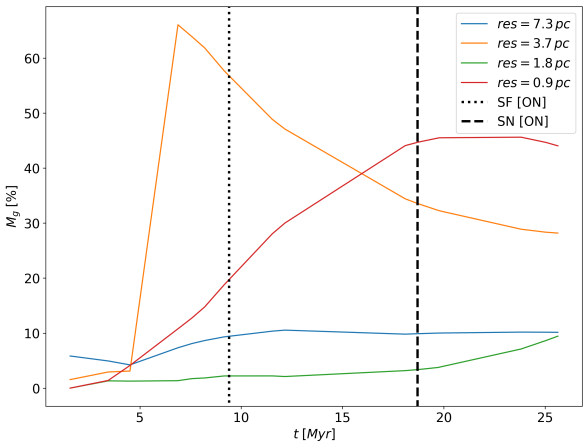

FIGURE 6.12 – Évolution de la masse de gaz résolue aux différents niveaux AMR au cours du temps, depuis le début de la deuxième partie de la simulation. On rappelle que les résolutions $[7.3, 3.7, 1.8, 0.9]\ pc$ correspondent respectivement aux niveaux AMR $[14, 15, 16, 17]$.

Bien qu'on utilise un modèle simple de déclenchement de la formation stellaire au sein d'une cellule (critère de densité seuil), sa paramétrisation reste délicate. En effet, comme on l'a mentionné précédemment, on utilise une efficacité numérique de formation que l'on notera SFE_n. On donne la courbe d'évolution du taux de formation stellaire sur la figure 6.13, calculée à $t = 26\ Myr$. Le paramètre de régulation, initialement à $SFE_n = 2\%$, a du être ajusté à partir du temps $t = 18\ Myr$. En effet, l'augmentation significative de la `SFR`, bien au-delà de $10\ M_\odot.yr^1$, a obligé, après un retour en arrière de la simulation, à réduire le paramètre à $SFE_n = 1\%$, expliquant ainsi la chute de `SFR` à $t = 18\ Myr$, qui est donc artificielle. De $t = 20\ Myr$ à $t = 24\ Myr$, il a y une augmentation quasi-constante de la formation de particules, malgré la (faible) rétroaction des supernovae issues des particules crées entre $t = 10\ Myr$ et $t = 14\ Myr$. Néanmoins, les premières particules sont formées principalement dans la barre centrale, qui abrite le plus de régions denses. En plus d'être faible, la rétroaction est donc localisée uniquement dans la barre. La diminution de la `SFR` à partir de $t = 24\ Myr$, coïncide avec l'augmentation de la `SFR` $10\ Myr$ auparavant, à $t = 14\ Myr$. Malgré tout, on ne peut pas conclure à un lien directe avec la rétroaction des `SNe` pour l'instant.

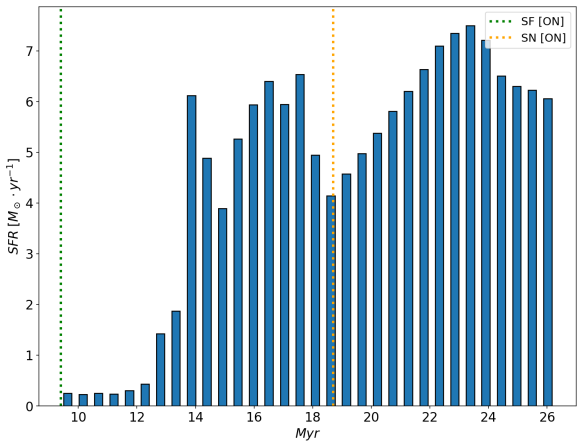

FIGURE 6.13 – Évolution du taux de formation stellaire entre $t = 10\ Myr$ (activation de la `SF`) et $t = 26\ Myr$. Les traits verticaux vert et orange représentent, respectivement, l'activation de la formation stellaire dans la simulation, et l'explosion des 1ères supernovae ($t =\sim 19\ Myr$). L'efficacité numérique SFE_n initialement de 2% a été réduite à 1%, à $t = 18\ Myr$.

Il est intéressant de regarder les régions de formation stellaire. Les cartes `2D` de la figure 6.14 montrent la densité de surface des particules d'étoiles formées entre $t = 10\ Myr$ et $t = 26\ Myr$, vues d'au dessus. Il y a environ $1.1\ 10^6$ particules formées pour une masse totale de $7.8 \cdot 10^7\ M_\odot$. On constate que les particules sont principalement concentrées en amas dans la barre centrale et dans les bras spiraux jusqu'à une distance supérieure à 5 kpc du centre du disque. De plus, ces amas sont répartis à intervalles plus ou moins réguliers et sont plus denses sur l'avant de la barre (dans le sens de rotation). De plus amples analyses, notamment pour extraire ces amas et les nuages denses associés (voir au paragraphe suivant), sont nécessaires (non présentées dans ce document). Enfin, on note la présence de régions vides de formation stellaire entre les bras spiraux. A titre d'exemple, on montre une carte qui superpose la densité de surface et la position des particules crées, dans une région de 4 kpc x 4 kpc, figure 6.15.

Afin de limiter le nombre de carte `2D`, on donne deux cartes, haute résolution, produites par lancer de rayon en vue de face du disque de gaz, auxquelles on fera référence par la suite. La carte 6.16 représente la densité de surface du gaz, en échelle logarithmique, en $M_\odot \cdot pc^{-2}$, avec une résolution spatiale de 1.8 pc (niveau 16). La deuxième représente la température, pondérée par la masse avec la même résolution, en Kelvin (logarithmique également), 6.17.

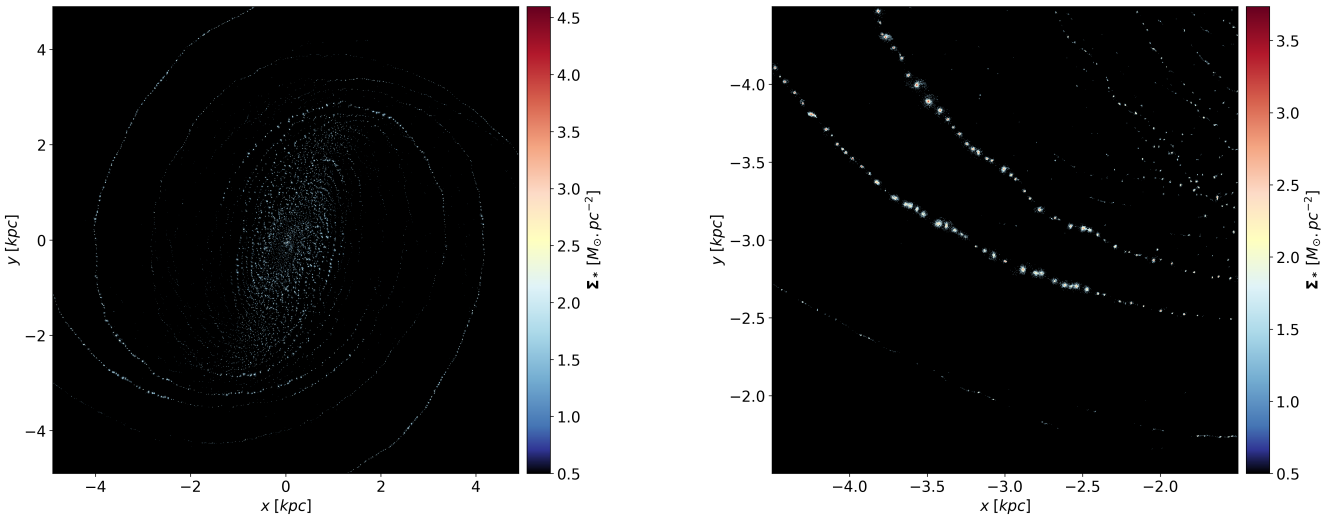

FIGURE 6.14 – Densité de surface, vue du dessus, des particules d'étoiles formées entre $t = 10\ Myr$ (activation de la formation stellaire) et $t = 26\ Myr$. A droite, un zoom d'une région de 3 kpc x 3 kpc centré aux coordonnées (-3,-3) (relative au centre du disque), mettant en avant la structure sous forme d'amas.

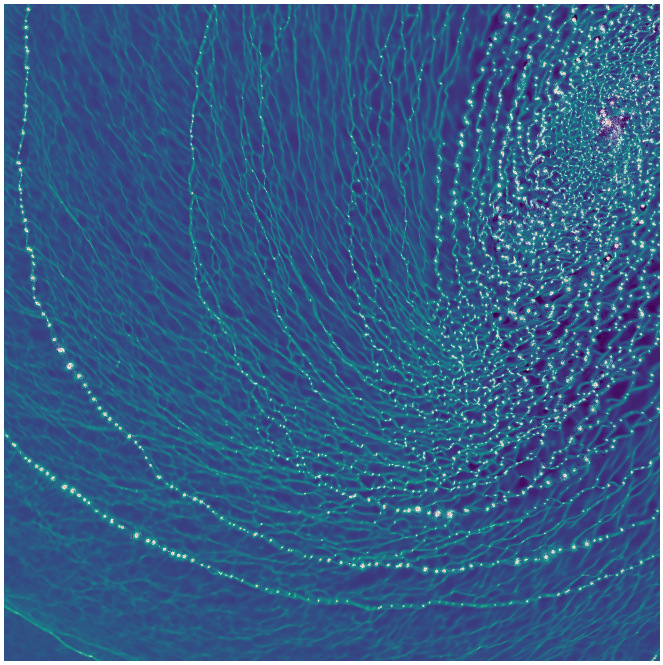

FIGURE 6.15 – Superposition de la densité de surface du gaz, vue de face, et de la position des particules d'étoiles formées entre $t = 10\ Myr$ et $t = 26\ Myr$.

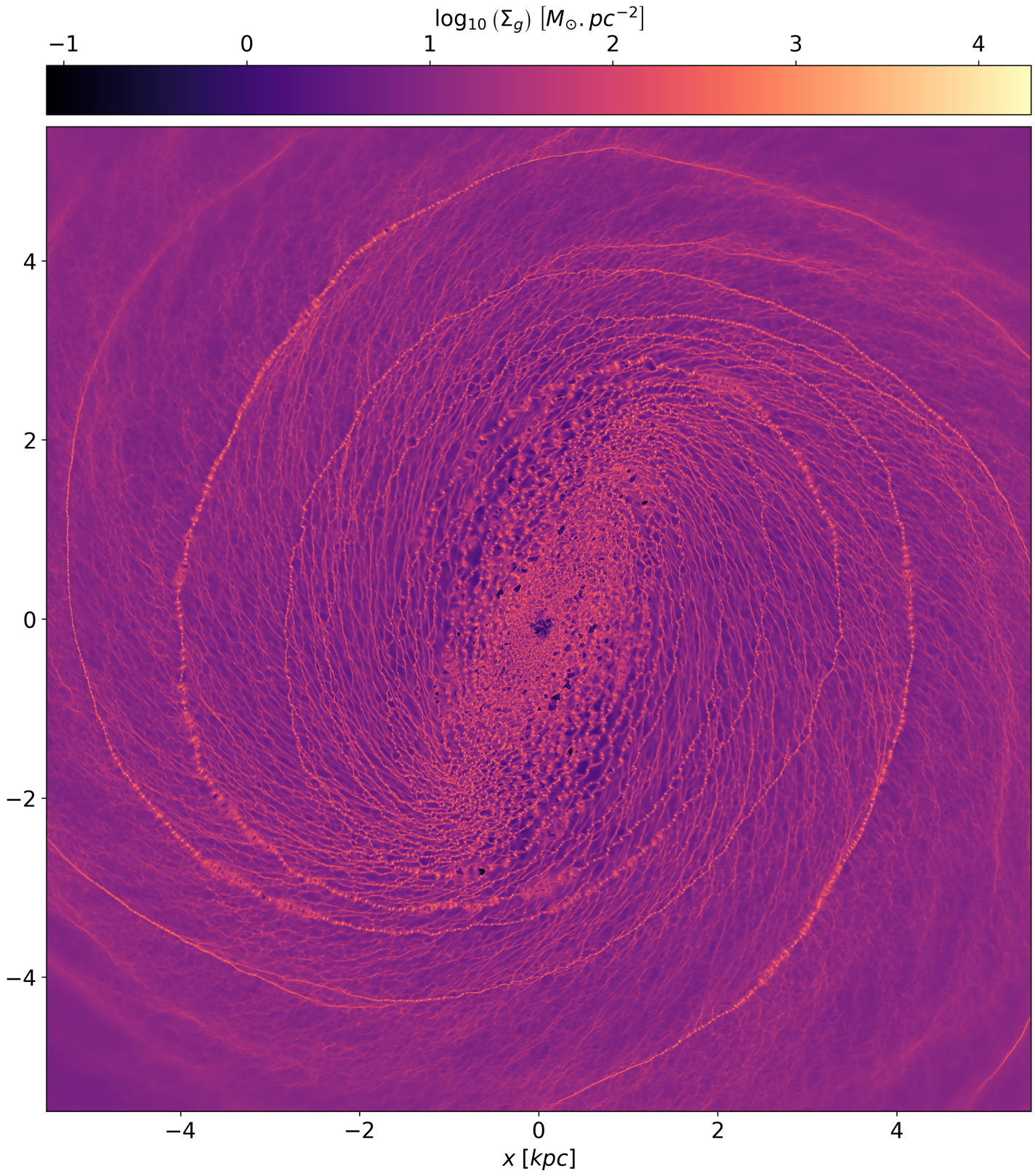

FIGURE 6.16 – Densité de surface du gaz à $t = 26\ Myr$, projetée en vue de face avec une résolution spatiale de 1.8 pc.

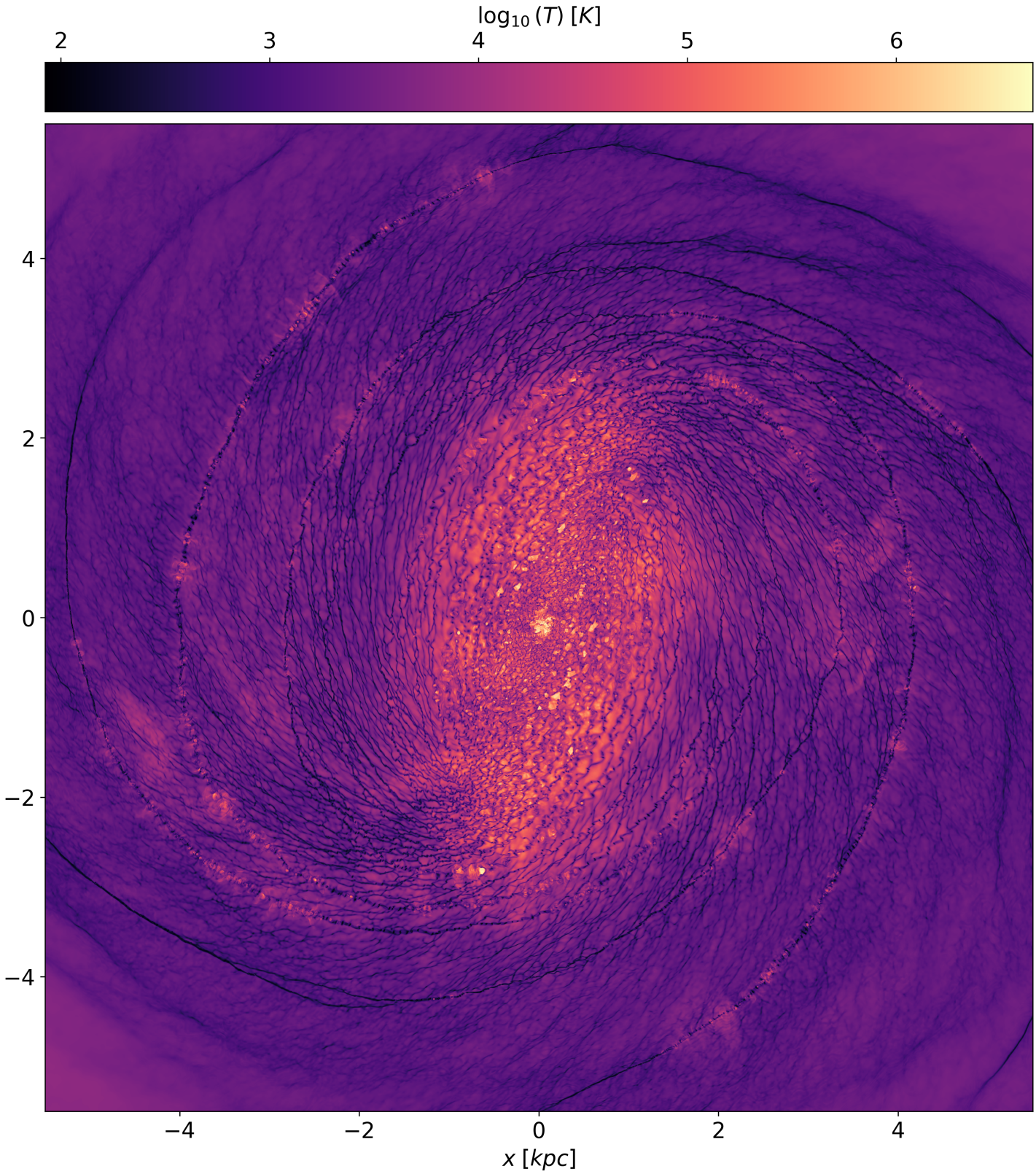

FIGURE 6.17 – Carte de température pondérée par la masse à $t = 26\ Myr$, projetée en vue de face avec une résolution spatiale de 1.8 pc.

Les amas de particules, mentionnés précédemment, se forment à l'intérieur de régions de sur-densité sphérique ou ovale, que l'on retrouve dans les bras spiraux et dans la barre centrale (comme on peut le voir sur la carte de densité de surface, sur la figure 6.16). A une distance faible du centre galactique, < 4 kpc, ces structures sont présentes tout le long des bras spiraux. Bien que l'intervalle entre ces nuages denses soit variable, on retrouve des séquences alignées à intervalle régulier le long des bras, [Renaud et al., 2013]. La taille des structures est relativement régulière, estimé à ~ 30 pc (estimation faite à partir de la carte dont la résolution spatiale est de 1.8 pc). Avant de faire une analyse plus précise, on estime approximativement qu'il y a au moins 2 ordres de grandeurs, en terme de densité, entre les structures denses et le reste du bras. On retrouve ces nuages sur la carte de température pondérée par la masse, figure 6.17. L'effondrement gravitationnel de ces structures est contrebalancé par le support de pression apporté par le polytrope de Jeans du modèle thermodynamique, assimilable au coeur chaud observé dans les nuages moléculaires. On notera qu'on voit également, sur ces deux cartes, les zones de souffles des `SNe`. En effet, on voit sur la carte de densité surfacique, ces nuages denses se faire détruire par le souffle cinétique de l'explosion et la carte en température met en évidence le support thermique apporté par les `SNe`, beaucoup plus important aux zones de fortes activités de formation stellaire. On rappelle que le modèle utilisé est un modèle purement cinétique avec un rayon d'injection des chocs sphériques lors de l'explosion d'une `SN` égale à 10 pc.

6.7.2.4 Spectres de puissance

Par le calcul de spectres de puissance, on cherche à caractériser la turbulence, et ses variations de régimes, à différentes échelles caractéristiques. On cherche dans un premier temps à estimer l'épaisseur du disque de gaz. Une première approximation peut être faite à partir d'une tranche verticale dans le plan XZ, de la densité, et permet d'estimer l'épaisseur du disque à environ ~ 180 pc au-delà de 3 kpc du centre, sur cette tranche. Néanmoins, dans la région centrale $< 3kpc$, à cause des éjectas, il est n'est pas possible avec cette approche d'estimer la hauteur du disque. On calcule alors le contraste en densité sur les différentes tranches dans des plans XY espacés verticalement de 6 pc. On définit le contraste comme étant le rapport de la dispersion de la densité sur sa valeur moyenne au sein de la tranche : $c(\rho) = \sigma(\rho)/ < \rho >$. On trace sur le graphique de droite, de la figure 6.18, l'évolution du contraste en fonction de la position en z. On estime alors l'épaisseur à environ ~ 100 pc.

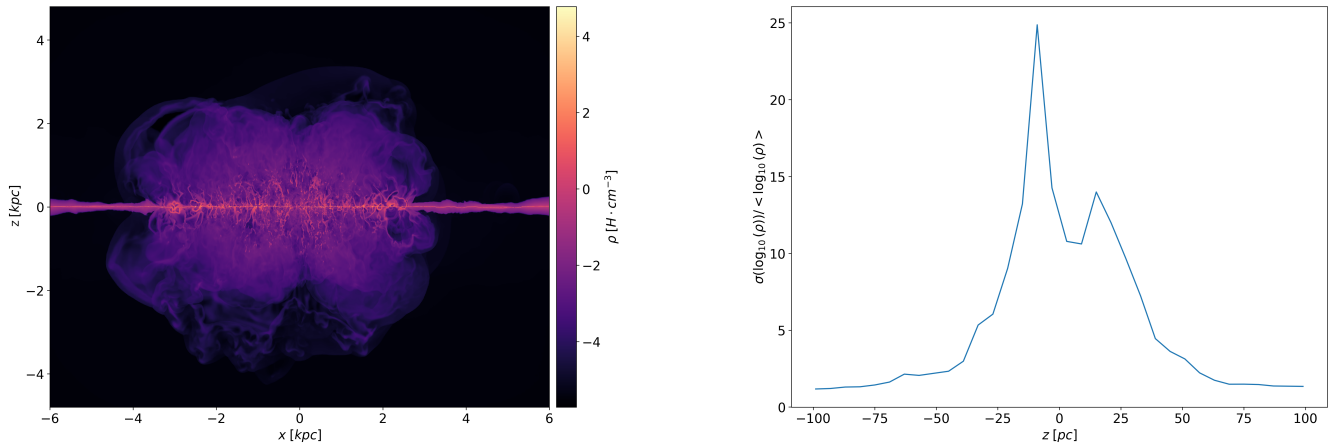

FIGURE 6.18 – A gauche, une tranche dans le plan $XZ, y = 0$, de la densité maximale, en échelle logarithmique. A droite, l'évolution du contraste de la densité calculé sur des plans XY espacées régulièrement de 6 pc, suivant l'axe z.

Afin de caractériser la turbulence dans le `MIS`, on trace des spectres de puissance. A partir de données issus de simulation numérique, il y a plusieurs façons de calculer ces spectres. En effet, contrairement à l'observation, on a accès à des données `3D` de densité et de vitesse. On a également la possibilité de faire des cartes, intégrées ou non, suivant n'importe quelle ligne de visée. Néanmoins, comme on l'a mentionné précédemment, le calcul de spectre `3D` peut rapidement devenir très coûteux en temps de calculs et consommation mémoire. En conséquence, tous les spectres sont calculés à partir de carte `2D` en vue de face, i.e. ligne de visée $+z$. Toutes les cartes, intégrées ou issues de tranches, ont une résolution spatiale 1.8 $pc \cdot pixel^{-1}$. Par souci de facilité d'interprétation, on convertira l'axe des abscisses des spectres en pc^{-1}. De plus, une régression linéaire par moindres carrés sera utilisée pour déterminer les

pentes des lois de puissance mises en évidence sur les spectres.

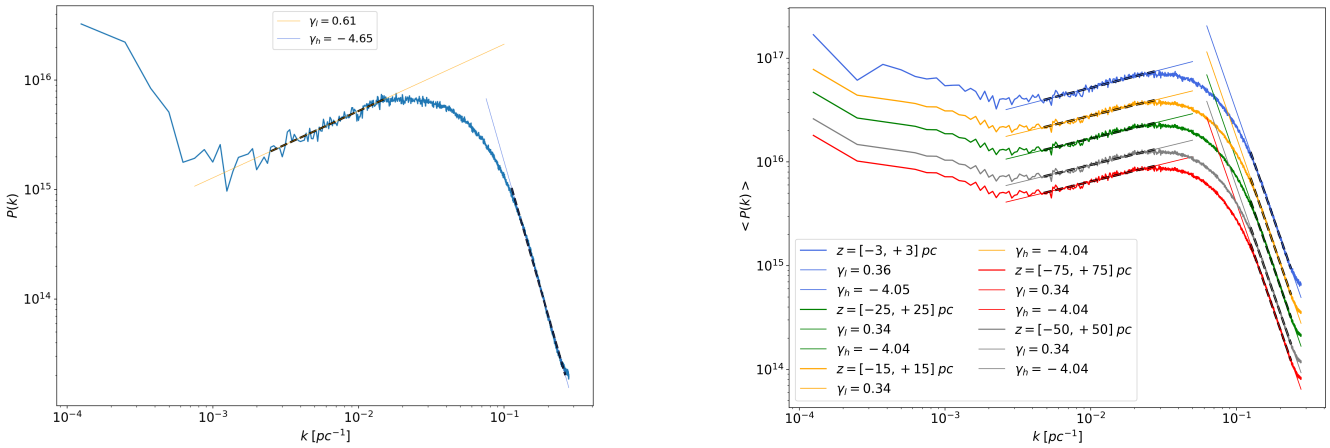

FIGURE 6.19 – Spectre de puissance de carte de densité à $t = 26\ Myr$. A gauche, le spectre d'une carte de densité intégrée, avec une pente de $\lambda_l = 0.61$ pour les échelles $> 90\ pc$ et $\lambda_h = -4.65$ pour les petites échelles $< 10\ pc$. A droite, les spectres moyennées sur des épaisseurs entre 6 pc (2 tranches en $z = -3, z = +3\ pc$) et 150 pc (34 tranches). Deux lois de puissances sont trouvées dans les intervalles similaires, une pente de l'ordre de 0.3 pour grandes les échelles $> 80\ pc$, et de -4 pour les petites échelles $< 10\ pc$.

On donne sur la figure 6.19, les spectres de puissance de la densité de surface à $t = 26\ Myr$. A gauche, il est calculé depuis une carte intégrée le long de la ligne de visée $+z$. Il est important de rappeler que l'algorithme de lancer de rayon ne tient pas compte de l'opacité ou de l'absorption. Les rayons traversent donc l'ensemble du milieu comme s'il était parfaitement transparent. On cherche a montré ici, que la structuration du milieu est la même sur différente épaisseur du disque. Le spectre de gauche, montre une première pente de $\lambda_l = 0.61$ pour les échelles caractéristiques comprises entre $[400, 70]\ pc$. Et $\lambda_h = -4.65$ aux petites échelles $< 10\ pc$. Sur la figure de droite, on donne plusieurs spectres, moyennés sur l'ensemble des tranches, prises tous les 6 pc, centrées autour de $z = 0$. On prend donc des tranches aux coordonnées $z = [.., -15, -9, -3, 3, 9, 15, ..]\ pc$. On constate que les amplitudes des spectres diminuent avec l'épaisseur, environ un facteur 10 entre $z \pm 75$ et $z \pm 3$. On notera que les deux pentes, aux petites et grandes échelles, restent inchangées et sont ici de $\lambda_l = 0.34$ et $\lambda_h = -4.04$. La structuration du `MIS`, dans la simulation, ne change donc pas avec l'épaisseur du disque. Par rapport au spectre intégré, l'échelle caractéristique associées à la pente λ_l est légèrement décalée, plutôt dans l'intervalle $200 - 50\ pc$, alors que celle de λ_h est inchangée. Enfin, le spectre intégré montre une différence sur les pentes de ~ 0.30 aux moyennes échelles $> 50\ pc$ et de 0.60 aux petites échelles, bien que la tendance reste la même.

Les cartes 6.16 et 6.17 montrent beaucoup de filaments, dans le `MIS` entre les bras spiraux, composés de gaz froid dense dans le milieu chaud. A partir de la carte de température et de tranches en densité, on peut estimer, dans un premier temps, que les filaments ont des densités, relativement variables, de l'ordre $\sim 100\ H \cdot cm^{-3}$, ainsi qu'une température de l'ordre de 500 K. De plus, la longueur des filaments est variable et ils peuvent mesurer jusqu'à plusieurs centaines de parsecs, voir figures 6.21. Il s'agit d'estimations pour donner des ordres de grandeur, qui doivent être précisées par des analyses plus poussées. Trois causes principales sont proposées pour expliquer l'origine des filaments interstellaires : la gravité, la turbulence supersonique et le champ magnétique, [André, 2017]. Dans cette simulation, le champ magnétique n'a pas été considéré. Afin de caractériser et de mieux comprendre ces zones et ces structures, on extrait 3 régions carrées de 800 pc à différentes coordonnées de carte de densité et de norme de vitesse, voir figure 6.20. Par la suite, on note ces régions $F1, F2, F3$.

Qualitativement, on constate que les filaments sont plus longs et plus denses lorsqu'on se rapproche du centre du disque, voir figure 6.21. On notera que les cartes de la norme de la vitesse, dans les 3 régions, figure 6.22, mettent en avant le cisaillement avec une variation de l'ordre de 5 à 10% de la vitesse.

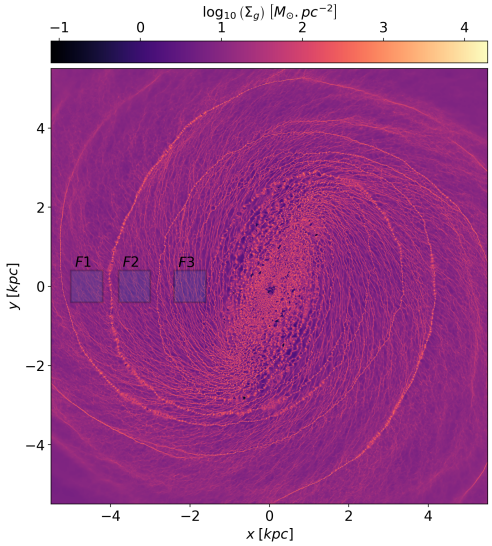

FIGURE 6.20 – Position globale dans le disque des zones $F1, F2, F3$ sur la carte de densité de surface à $t = 26\ Myr$.

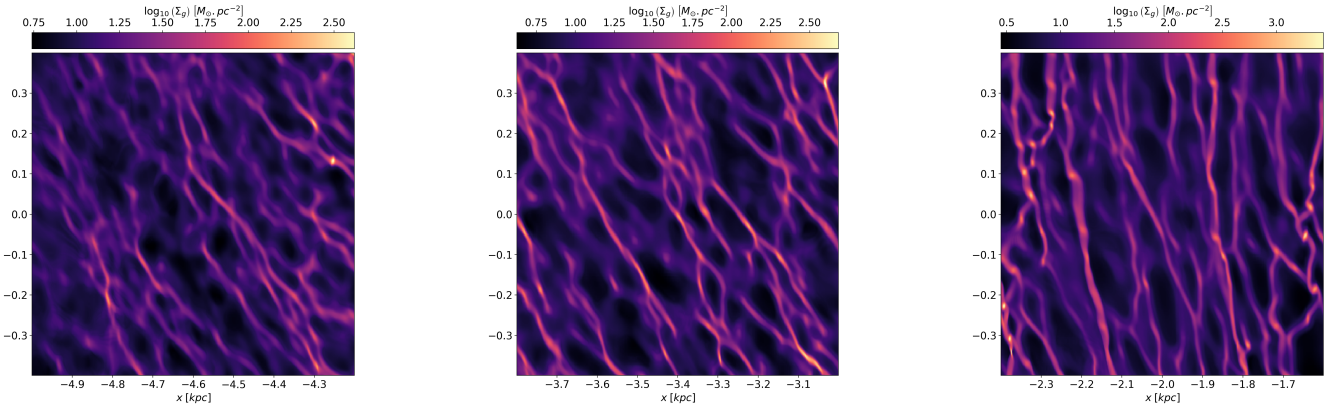

FIGURE 6.21 – Carte 2D de densité intégrée de 3 régions filamentaires de 800 pc par 800 pc, centrées respectivement à $(-4.6, 0.0)$, $(-3.4, 0.0)$, $(-2.0, 0.0)$. Soit respectivement à 4.4, 3.4 et 2.0 kpc du centre. On notera ces régions, respectivement $F1$, $F2$, $F3$.

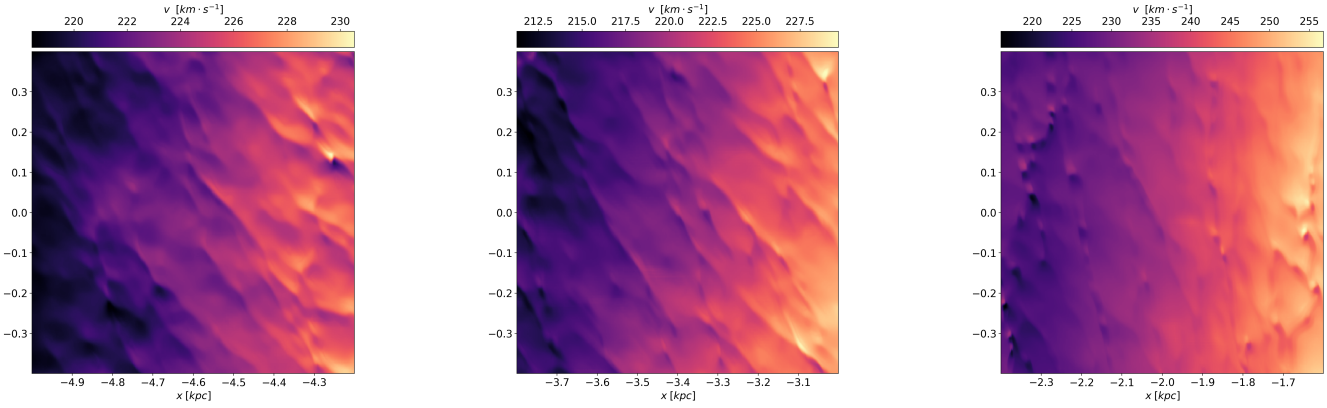

FIGURE 6.22 – Carte 2D de norme de vitesse intégrée et pondérée par la masse de 3 régions filamentaires de 800 pc par 800 pc, centrées respectivement à $(-4.6, 0.0)$, $(-3.4, 0.0)$, $(-2.0, 0.0)$. Soit respectivement à 4.4, 3.4 et 2.0 kpc du centre.

Pour chacune des régions $F1, F2, F3$ on calcule le spectre de puissance de la densité et de la vitesse (pondérée par la masse), figure 6.23. On constate une variation de l'ordre de 0.4 de la pente entre les zones $F1$ et

$F2$, mais une très forte variation dans la zone $F3$ avec une pente de l'ordre de -3. Avec le spectre de la densité de surface, on cherche ici à montrer une structuration différente du milieu en fonction de la région. Ainsi, ils mettent en avant une augmentation des structures lorsqu'on se rapproche du centre, on constate d'ailleurs que des grumeaux denses apparaissent sur la région $F3$. Les spectres de la vitesse montrent une faible variation de la pente entre les différentes régions de l'ordre de 10%, ainsi qu'une décroissance de la pente lorsqu'on se rapproche du centre du disque.

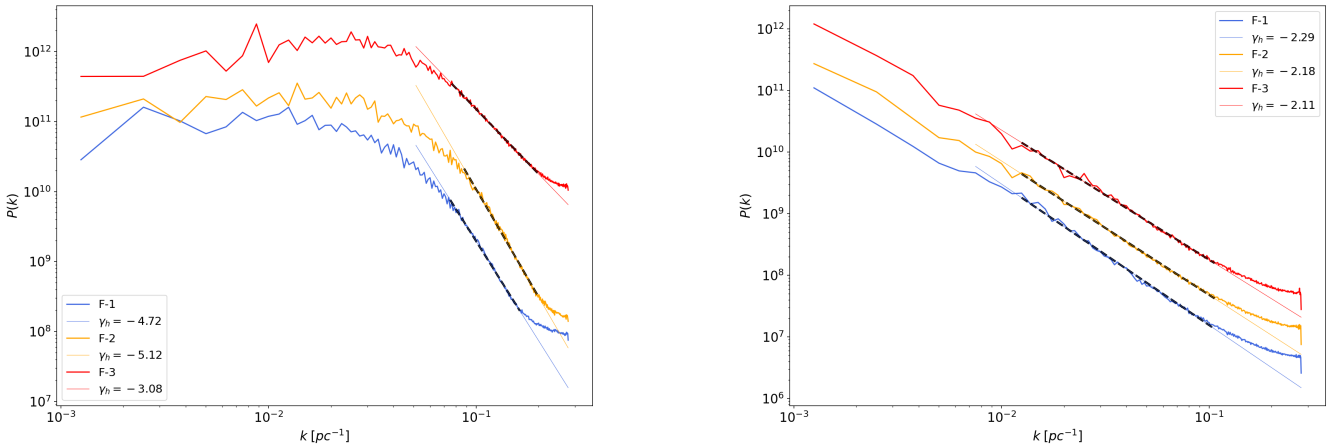

FIGURE 6.23 – Spectres de puissance des différentes régions filamentaires $F1, F2, F3$. A gauche, le spectre de la densité intégrée et à droite de la norme de la vitesse pondérée par la masse.

Enfin, il est intéressant de montrer les spectres pour la composante radiale et angulaire (toutes deux pondérées par la masse), en coordonnées cylindriques, des différentes régions, figure 6.24. En effet, on constate que pour les zones $F1, F2$ les deux spectres ont des pentes différentes qui engendrent un croisement à partir d'une échelle de ~ 30 pc où la composante angulaire domine. La zone $F3$ montre une différence significative entre les deux composantes. Le spectre de la vitesse azimutale est largement supérieur, tout en montrant des pentes relativement similaires (écart de 0.1).

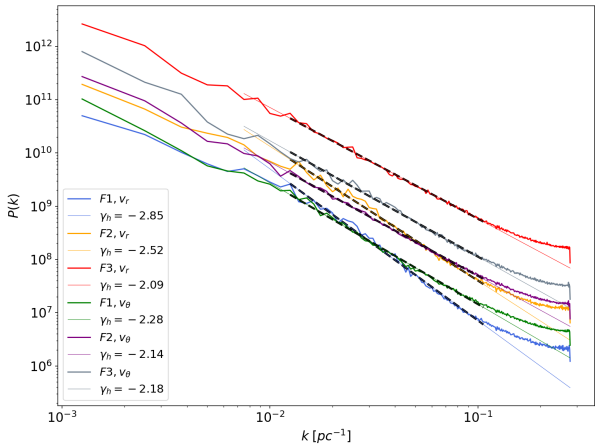

FIGURE 6.24 – Spectres de puissance des composantes r, θ de la vitesse (pondérées par la masse) pour les différentes régions filamentaires $F1, F2, F3$.

6.7.2.5 Dispersion de vitesse

Afin d'évaluer la dispersion de vitesse, on définit une grille, dans le plan du disque, composée de 200 x 200 régions cubiques. On appellera chaque cube un secteur, qui aura pour taille 50 pc de côté, et sera espacé,

suivant les 3 directions de l'espace, de 3.4 pc de ses voisins (une cellule de niveau 15, pour éviter les effets de double comptabilisation des cellules). Ainsi, on couvre une zone d'environ 5 kpc de rayon autour du centre du disque. Pour chaque secteur, on calcule la vitesse moyenne et la dispersion de vitesse pour les composantes r, θ, z (en coordonnée cylindrique), ainsi que la masse de gaz dans le secteur. On donne sur la figure 6.25, une vue de la dispersion de vitesse dans les secteurs. On notera qu'on définit $\sigma_{disk} = \sqrt{\sigma_r^2 + \sigma_\theta^2}/\sqrt{2}$, pour mettre en avant la dispersion de vitesse dans le plan du disque, voir graphique de gauche sur la figure 6.25. Le graphique de droite donne la dispersion de la composante z. On remarque principalement les effets des `SNe` qui agitent le `MIS`, en augmentant localement la dispersion de vitesse suivant les deux composantes représentées. On notera également que les régions entre les bras ont une dispersion de vitesse σ_{disk} moins homogène que la dispersion suivant σ_z. En effet, dans la zone correspondant à la région $F3$ décrite précédemment, les filaments ressortent avec une dispersion de vitesse plus élevée dans le plan du disque que suivant l'axe z, de l'ordre de quelques $km \cdot s^{-1}$.

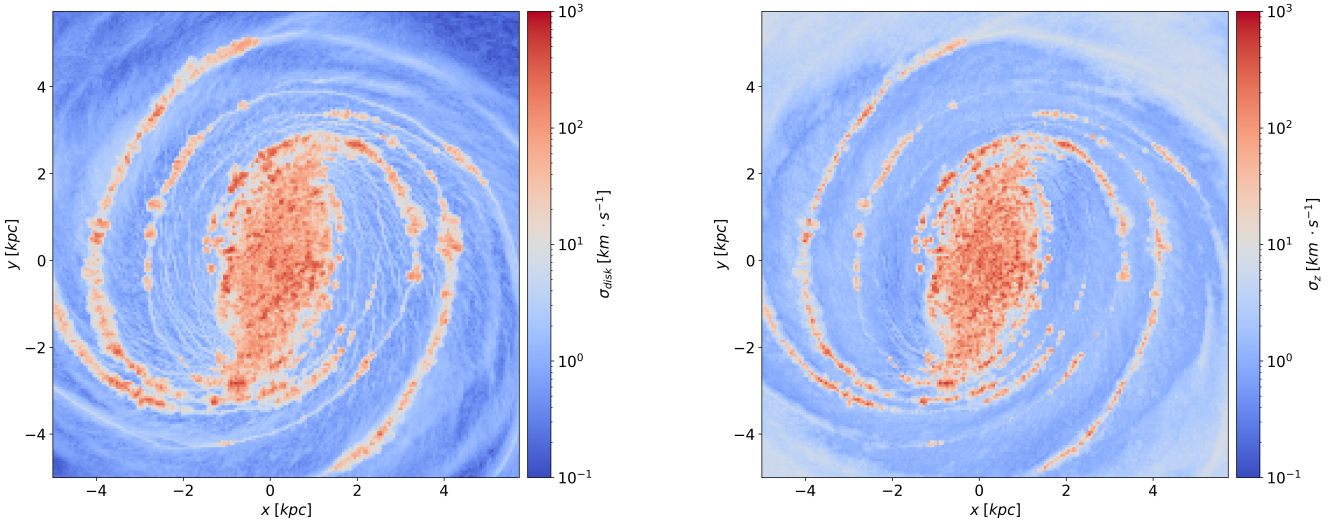

FIGURE 6.25 – Carte des secteurs avec la dispersion de vitesse dans le disque à gauche et la dispersion de vitesse suivant l'axe z à droite, à $t = 26\ Myr$. Les secteurs sont des régions cubiques de 50 pc de côté dans le plan du disque.

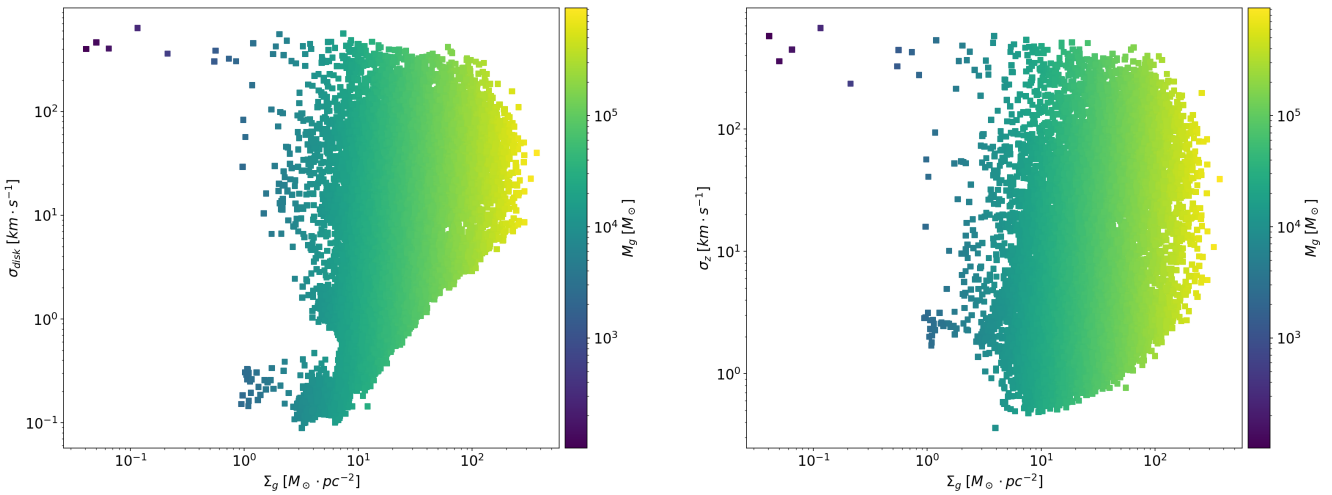

FIGURE 6.26 – Dispersion de vitesse dans le plan du disque, σ_{disk}, à gauche, et suivant l'axe z à droite, en fonction de la densité de surface des secteurs, à $t = 26\ Myr$. Les secteurs sont des régions cubiques de 50 pc de côté dans le plan du disque.

A partir de la masse de gaz contenue dans les secteurs qui varient de $10^2\ M_\odot$ à $10^6\ M_\odot$, on peut calculer la densité de surface du gaz vue de face et ainsi tracer la dispersion de vitesse en fonction de la densité de surface

du gaz, voir figure 6.26. Les ordres de grandeurs de σ_{disk} et σ_z sont similaires, on ne retrouve pas dans le plan du disque, sur une épaisseur de 50 pc (taille d'un secteur) de différence entre les deux variables. On notera que les dispersions de vitesses élevées de l'ordre de 10^2 $km \cdot s^{-1}$ sont liées à la rétroaction cinétique des SNe, dont l'injection de l'onde sphérique initiale est à une échelle de 10 pc. Il est intéressant de noter une limite presque linéaire sur la dispersion σ_{disk}.

6.7.2.6 Rétroaction des supernovae

Comme on l'a mentionné précédemment, les effets des SNe sont visibles sur les cartes de densité surfacique 6.17 et de température 6.16. Afin de mettre en relief leur apport cinétique, les cartes 6.27 représentent la valeur maximale des composantes le long de la ligne de visée. Le milieu dans lequel se propage le souffle de l'explosion a un impact sur l'injection du moment cinétique des SNe dans le MIS, [Iffrig and Hennebelle, 2015]. C'est une des raisons de l'importance de la rétroaction "avant explosion", qui permet notamment de produire des régions ionisées à plus faible densité autour des étoiles massives. Il s'agit d'une limite de notre modèle de rétroaction qui ne tient pas compte des régions HII.

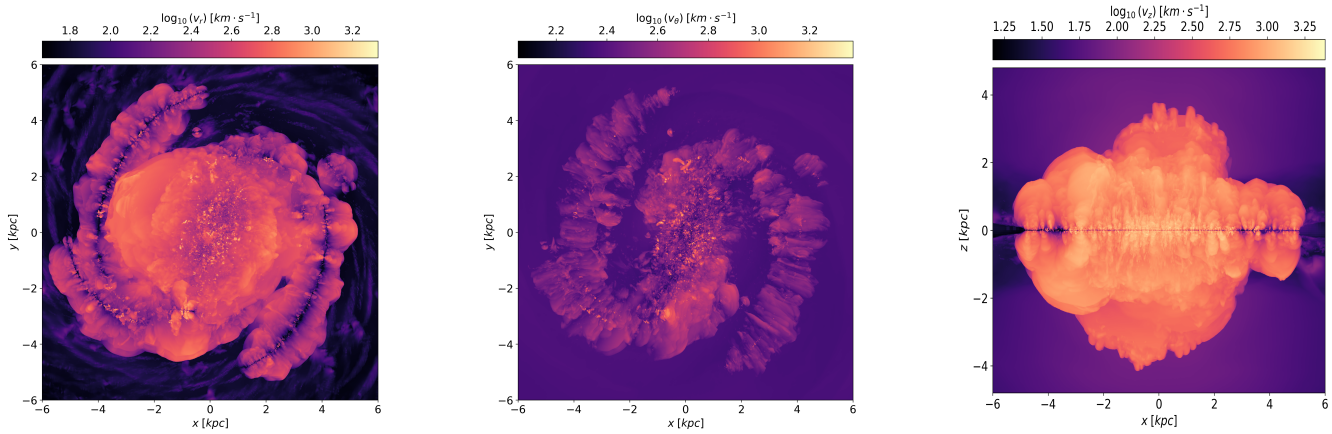

FIGURE 6.27 – Carte de vitesse maximale à $t = 26$ Myr vue de face (échelle *logarithmique*). De gauche à droite, la composante radiale v_r, la composante angulaire v_θ et la composante verticale v_z.

Sur ces cartes, en vitesse radiale, on constate que les ondes de chocs se propagent plus loin que sur la carte de v_θ. En effet, entre les bras, la densité est plus faible, favorisant le développement du souffle. On notera que les filaments se sont développés avant la rétroaction des SNe. Suivant l'axe θ, la densité du bras et des nuages adjacents impactent le développement de la partie cinétique de la rétroaction. De même, la carte suivant l'axe vertical met en évidence des vitesses suivant v_z supérieures à 1700 $km \cdot s^{-1}$, expulsant une faible quantité de gaz hors du disque galactique, jusqu'à 4 kpc du plan du disque.

Dans un milieu interstellaire turbulent avec deux phases et une dispersion de vitesse faible, on s'attend à observer deux "bosses" dans la PDF, correspondant aux deux phases stables du gaz, à savoir le WNM et le CNM, [Audit and Hennebelle, 2010], [Hennebelle and Falgarone, 2012]. On retrouve à $\rho = 0.01$ $H \cdot cm^{-3}$ et $\rho \sim 1$ $H \cdot cm^{-3}$, un pic de masse associé au WNM qui correspond à une température de l'ordre de 10^4 K. On notera que l'aspect *log-normal* peut se retrouver sur ce premier pic. Le deuxième pic à 10^2 $H \cdot cm^{-3}$ est associé au gaz froid neutre. Sa forme s'éloigne d'une *log-normale* comme montré par [Federrath et al., 2008]. En effet, lorsque le milieu devient compressible, sa forme change, en particulier à haute densité car le nombre de Mach augmente, donc la largeur de la PDF augmente, car les chocs génèrent de large contrastes en densité.

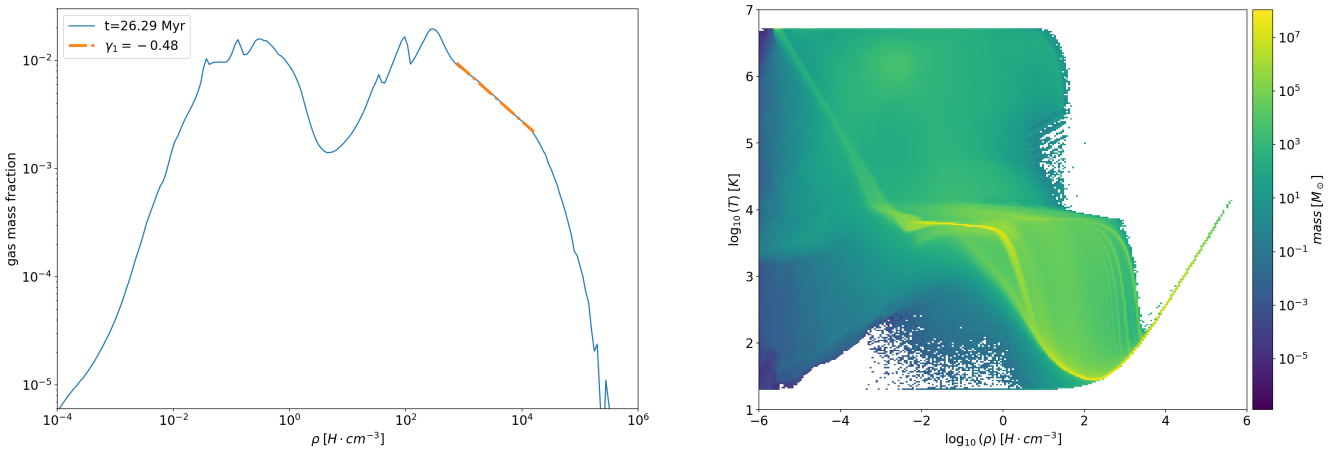

FIGURE 6.28 – PDF de gaz à $t = 26\ Myr$ en échelle *log-log* à gauche. Une régression linéaire de pente $\gamma_1 = -0.48$ est faite entre $7\,10^2\ H \cdot cm^{-3}$ et $1.8\,10^4\ H \cdot cm^{-3}$. A droite, le diagramme (ρ, T).

Les spectres de puissance de la densité de surface et de la vitesse (pondérée par la masse) à différents temps physiques entre $t = 18\ Myr$ à $t = 26\ Myr$ sont donnés sur la figure 6.29. A $t = 18\ Myr$ il n'y a pas encore eu les premières explosions de SNe. Leur nombre augmente ensuite progressivement de $t = 20\ Myr$ à $t = 26\ Myr$. On constate une augmentation du nombre de structure à partir de $k \simeq 2.8\,10^{-3}$, soit une échelle caractéristique de l'ordre de $\sim 350\ pc$. Cette échelle correspond à l'ordre de grandeur estimé précédemment pour les structures filamentaires qui continuent effectivement de se développer. On notera toutefois qu'à partir de $t = 26\ Myr$, il semble y avoir une diminution du nombre de structure aux échelles inférieure à $60\ pc$.

Sur le graphique de droite, on peut extraire pour chaque temps au moins deux lois de puissance. Aux échelles entre $\sim 500\ pc$ et $\sim 80\ pc$, la pente est donné par le paramètre λ_l et aux petites échelles avec une pente λ_h. Pour $t = 26\ Myr$, on met en évidence une troisième loi de puissance aux échelles intermédiaires, voir paragraphe suivant. La pente λ_l diminue significativement au cours du temps de -1.55 à $t = 18\ Myr$ pour devenir presque constante à $t = 26\ Myr$, avec $\lambda_l \simeq -0.4$. La pente λ_h diminue d'abord rapidement puis remonte légèrement, traduisant ainsi une injection d'énergie importante aux plus petites échelles, puis une relative stabilisation.

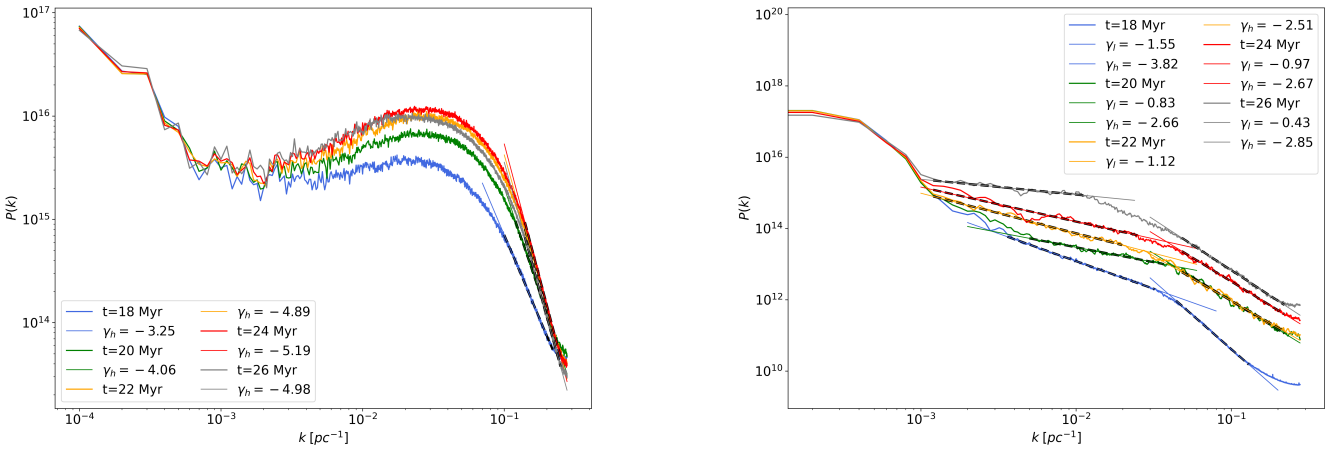

FIGURE 6.29 – Spectres de la densité intégrée à gauche et de la norme de la vitesse pondérée par la masse pour différents temps physiques. Les premières explosions de SNe débutent après $t = 18\ Myr$. Les régressions linéaires sont faites par la méthode des moindres carrés.

Le spectre d'énergie, figure 6.30, montre une forte injection d'énergie à une longueur caractéristique de $\sim 4\ kpc$ ($k \sim 2.4e-4\ pc^{-1}$), qui correspond à la barre centrale très développée dans notre simulation. Il est attendu que la présence de la barre centrale permette une injection d'énergie aux grandes échelles dans le système. En effet, elle permet un transfert du moment angulaire vers l'extérieur, entraînant un transfert de masse vers le centre du

disque. En conséquence, la baisse de l'énergie gravitationnelle qui est compensée par l'augmentation de l'énergie cinétique (turbulente) dans le MIS. La dissipation d'énergie dans l'intervalle $[700 - 90]$ pc se fait avec une loi de puissance de pente -0.4. Cette échelle peut correspondre aussi bien aux bras spiraux qu'aux filaments. On notera qu'on détecte une transition à l'échelle de hauteur du disque qui fait basculer la pente de -0.39 à -1.90, sur les échelles de 90 pc à 20 pc. Cette transition a déjà été montrée par [Bournaud et al., 2010] pour une simulation du Nuage de Magellan, ou par [Combes et al., 2012] pour la galaxie M33, mais aussi observationnellement, [Block et al., 2010]. Et enfin, on trouve une deuxième transition pour le passage aux petites échelles < 20 pc avec une pente de -2.9 caractéristique d'une turbulence 3D à l'échelle des nuage denses.

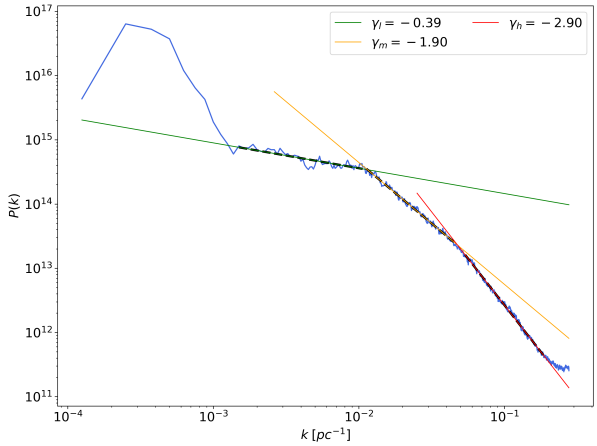

FIGURE 6.30 – Spectres de la norme de la vitesse pondérée par la masse pour $t = 26$ Myr sur lequel on peut mettre en évidence trois lois de puissance à différentes échelles : > 80 pc, $80 > l > 30$ pc, et < 30 pc. Dont les pentes sont estimés par une régression linéaire (moindres carrés), et sont données respectivement par λ_l, λ_m, λ_h.

On notera également que le spectre se relève aux petites échelles au bout de la droite de pente λ_h à cause de la limite de résolution qui ne permet pas de résoudre complètement la cascade turbulente manquant ainsi les petits tourbillons.

6.8 Résumé et discussion

On commence d'abord par faire une résumé rapide des résultats préliminaires de la simulation ExaMilkyWay présenté aux sections précédentes.

1. On a montré la formation des grandes structures dans le disque : bras spiraux et barre centrale au bout de ~ 1 Gyr de relaxation par rapport aux conditions initiales, avec une résolution spatiale entre 127 pc et 14.7 pc, figure 6.5. La barre centrale est estimée à environ ~ 4.5 kpc de long. La PDF du gaz, figure 6.7, à ce stade, est celle d'un gaz isotherme avec une forme du type *log-normale*, [Vazquez-Semadeni, 1994].
2. La montée en résolution s'est faite pendant ~ 10 Myr avec l'activation du module de chauffage/refroidissement du gaz. Les PDF de gaz montre la formation d'un pic de densité croissant à $\sim 10^3$ $H{\cdot}cm^{-3}$, associé à la formation de zone dense, figure 6.8. Des cartes de densité projetées ont permis de mettre en évidence des instabilités naissantes entre les bras spiraux, figure 6.8. De plus, des cartes des composantes r, θ de la vitesse, pondérées par la masse, ont mis en évidence les zones de cisaillements au niveau des bras, mais également la dynamique de la barre centrale, figure 6.11. Le gaz à l'avant de la barre, dans le sens de rotation, a une vitesse radiale négative très marquée, alors que le gaz à l'arrière de la barre à une vitesse radiale positive. De même, la vitesse azimutale de la barre centrale est plus faible que celle du reste du milieu. Enfin, la barre peut être approximée par une ellipse de grand axe 2.3 kpc (4.6 kpc de long) et de petit axe 500 pc, ~ 1 kpc de large, à partir de carte de densité de surface.
3. Bien que le pourcentage de masse résolue aux différentes échelles varie au cours du temps, le maillage se stabilise vers $t =\sim 20$ Myr. Finalement, on résout $\sim 85\%$ de la masse du gaz à une résolution de 3.4 pc dont $\sim 45\%$ à 0.9 pc, figure 6.9.

4. L'activation de la formation stellaire au bout de $t \simeq 10\ Myr$, avec un critère de densité seuil à 3000 $H \cdot cm^{-3}$, a montré une SFR comprise entre $1 - 8\ M_{\odot} \cdot yr^{-1}$, et semble montrer un début de stabilisation à partir de $t = 24\ Myr$, figure 6.13. Les régions de formation stellaire, correspondant à des nuages denses d'une taille d'environ $\sim 50\ pc$, sont présentes tout le long des bras spiraux et dans la barre centrale, figures 6.13, 6.14. Ces amas de particules, espacés relativement régulièrement, et particulièrement nombreux au sein de la barre, sont présents jusqu'à plus de 5 kpc du centre du disque. En dehors des bras et de la barre, on ne retrouve quasiment pas de particules d'étoiles.
5. Les cartes intégrées, en vue de face, de la densité et de la température pondérée par la masse, figures 6.16 et 6.17 ont permis de montrer la formation de structures filamentaires froides dans les régions peu denses, entre les bras spiraux où la résolution spatiale est de l'ordre du parsec. La taille de ces filaments varie avec la distance au centre du disque et peut être estimée à quelques centaines parsecs pour une température de l'ordre de 500 K. Ces cartes permettent également de faire ressortir les régions associées au SNe. En effet, on voit notamment des nuages denses soufflés par la rétroaction cinétique et des zones de gaz chaud. La barre centrale ressort particulièrement sur la carte de température de part l'activité importante des SNe.
6. Les cartes de vitesses maximales, pour les trois composantes r, θ, z, (en vue de face et de côté) ont mis en évidence une rétroaction cinétique plus forte des SNe suivant la direction radiale (dans un référentiel cylindrique au centre du disque) que suivant la direction θ. La différence de densité du milieu est à l'origine de cette asymétrie. De même verticalement, les bulles se propagent facilement dans le milieu peu dense jusqu'à 4 kpc, avec une asymétrie entre $+z$ et $-z$, et des vitesses supérieures à 1700 $km \cdot s^{-1}$.
7. La PDF de gaz à partir des premières SNe, à $t = 26\ Myr$, a deux pics de densité à $10^{-1}\ H \cdot cm^{-3}$ et $10^{2}\ H \cdot cm^{-3}$, caractéristique d'un milieu à deux phases WMN et CNM. On peut extraire une loi de puissance de pente -0.48, pour les densités entre $\sim 700\ H \cdot cm^{-3}$ et $\sim 2 \cdot 10^{4}\ H \cdot cm^{-3}$, figure 6.28.
8. L'étude de 3 régions filamentaires carrés de 800 pc de côté, extraites des cartes 2D, montre une élongation des filaments lorsqu'on se rapproche du centre du disque. Les spectres de densité dans ces régions ont montré une différence de comportement aux petites échelles $< 10\ pc$ ainsi que du nombre de structures. La région $F3$, plus proche du disque, présente un plus grand nombre de structures aux petites échelles. Les spectres d'énergie font ressortir un comportement similaire entre les 3 régions, avec une faible variation de la loi de puissance aux échelles comprises entre $[100, 10]\ pc$. Le spectre des composantes radiale et angulaire de la vitesse montre une domination de la composante radiale lorsqu'on se rapproche du centre du disque.
9. Les spectres de puissance des cartes 2D de densité de surface montrent une pente croissante révélant une augmentation importante de structures entre les échelles $[400, 90]\ pc$ entre les temps $t = 18\ Myr$ et $t = 26\ Myr$. Les spectres d'énergie issus de cartes de la norme de la vitesse (pondérée par la masse) ont montré un aplatissement significatif du spectre aux échelles comprises entre 500 pc et 80 pc, passant de -1.5 à -0.4. De plus, on constate une augmentation, puis une diminution de la pente aux petites échelles $< 20\ pc$. On a vu également qu'à $t = 26\ Myr$, une troisième loi de puissance pouvait être extraite du spectre, mettant ainsi en évidence 2 points d'inflexion du spectre aux échelles 80 pc et 20 pc, avec des pentes qui diminuant progressivement : -0.39 pour $l > 80\ pc$, -1.9 entre $80 < l < 30\ pc$ et -2.9 pour $l < 10\ pc$.

Pour la distribution du gaz dans les PDF, [Passot and Vá zquez-Semadeni, 1998] propose de relier la pente de la loi de puissance à haute densité à un indice effectif du gaz, λ_{eff} qui permet de faire le lien avec la capacité du gaz à se refroidir et sa compressibilité. Ils calculent l'indice, compris entre -0.33 et -0.66 pour du gaz HI thermiquement instable. On obtient une valeur de -0.48 en accord avec leur estimation. Néanmoins, à ces densités, dans notre simulation, on se situe dans la zone du polytrope qui apporte un support en pression contre l'effondrement gravitationnel du gaz. La pente peut donc être influencée par ce dernier. De plus amples analyses sont donc nécessaires afin d'évaluer cette possible loi de puissance.

Les structures filamentaires, avec une densité de l'ordre de 100 $H \cdot cm^{-3}$, sont des structures froides interconnectées dans du gaz chaud, stockant un partie du gaz disponible pour la formation stellaire entre les bras. Il est attendu que la dispersion de vitesse augmente avec la longueur des filaments comme suggère la relation de Larson $\sigma \propto L^{0.5}$, [Larson, 1981]. Les cartes de dispersion de vitesses σ_{disk} de l'analyse en secteurs vont dans ce sens, bien que de plus amples analyses soient nécessaires. [Colman et al., 2022] ont montré que l'injection d'énergie turbulente aux grandes échelles pouvait mener à une augmentation de grandes structures. En partant de ces résultats, on pourrait supposer que les filaments observés seraient issus d'instabilités générées par le passage des bras spiraux. Il semble également que la composante radiale du champ de vitesse joue un rôle important dans le développement et/ou longueur de ces filaments, et donc le cisaillement.

De plus, d'après les estimations, la densité des filaments correspondraient à la zone de stabilité isotherme du CNM sur les PDF. [Kim and Ryu, 2005] ont montré que dans le cas de la turbulence transsonique, dans le cas d'un fluide isotherme, la pente est proche de celle de Kolmogorov. Néanmoins, lorsque le nombre de Mach augmente, la pente s'aplatit, ce qui peut refléter le développement de structures filamentaires, et le passage à un régime supersonique. Les spectres de la vitesse de la région $F1$ sont plus pentus que ceux de la région $F2$ et $F3$, traduisant peut-être ce changement de régime. Il faudrait regarder l'évolution temporelle de spectre dans ces régions pour mieux les caractériser et mettre en évidence le(s) régime(s) turbulent(s) et leur évolution dans le temps.

Les spectres de puissances de la densité de surface au cours du temps, figure 6.29, montrent un changement de structuration du MIS. Bien que l'augmentation franche du spectre aux échelles caractéristiques associées aux filaments et aux nuages denses corresponde à ce qui est observé entre $t = 18\ Myr$ et $t = 26\ Myr$, sur les cartes de densité de surface, l'impact de la rétroaction des SNe est moins claire. En effet, bien qu'on ait envie d'associer la diminution de la pente aux petites échelles à partir de $t = 26\ Myr$, à la rétroaction cinétique des SNe soufflant les nuages denses, comme observé sur la carte 6.16, la variation du spectre reste faible. Il n'est pas clair si cette variation reflète une réelle diminution du nombre de structure ou une simple fluctuation du spectre. D'autre part, il serait intéressant d'utiliser un algorithme de type *Friend-Of-Friend* pour valider la correspondance entre la taille typique de structures détectées par l'algorithme et l'échelle de cassure des spectres aux petites échelles.

Le taux de formation stellaire commence à se stabiliser alors que la forte rétroaction attendue des particules formées à $t > 16\ Myr$ n'a pas encore eu lieu. Une partie de cette régulation peut être attribuée aux réservoirs de gaz stockés entre les bras spiraux, sous la forme de structures filamentaires. En effet, on ne retrouve pas de zone suffisamment dense au sein des filaments pour abriter de la formation stellaire. Ainsi, la turbulence donne une pression contre la fragmentation gravitationnelle dans ces régions peu denses. Néanmoins, poursuivre la simulation permettrait de voir l'impact sur le SFR du pic de formation stellaire à $t \simeq 16\ Myr$.

Enfin, il faut être conscient des limites des modèles qui sont utilisés dans cette simulation. Par exemple, on ne tient pas compte du chauffage du gaz par les étoiles massives pour former des régions HII, qui pourraient faciliter la propagation du choc des SNe. En effet, comme on l'a vu, le choc se propage plus facilement verticalement et radialement, c'est-à-dire dans les directions ou le gaz est le moins dense. De plus, la fonction de chauffage/refroidissement tient compte d'un champ ultraviolet global. Il serait intéressant de pouvoir changer ces paramètres pour tenir compte d'un champ qui serait dépend de la SFR local par exemple. Pousser la résolution maximale permettrait de mieux résoudre aux petites échelles la cascade de turbulence, mais également d'utiliser un modèle de rétroaction des supernovae plus riche que le modèle purement cinétique. Par exemple, [Kretschmer and Teyssier, 2019] proposent de moduler l'explosion des supernovae en fonction de la résolution spatiale et de la densité au point d'explosion. Ils définissent ainsi le rayon de refroidissement comme le rayon au moment de la transition entre la conservation de l'énergie et la conservation du moment. Ce rayon est inversement proportionnel à la densité et métallicité locale, et l'injection d'énergie thermique et/ou cinétique se fait alors en fonction de la résolution spatiale de ce rayon.

6.9 Conclusion et perspectives

Dans ce chapitre, on a présenté la simulation ExaMilkyWay, qui s'est déroulée au TGCC (Très Grand Centre de Calcul du CEA), en deux étapes clés : relaxation, évolution à haute résolution. Après avoir discuté du plan de gestion de données et du dimensionnement du calcul, on a détaillé deux méthodes clés pour réduire indirectement le nombre d'heures de calcul consommées. En effet, on a montré qu'il était possible, grâce à l'utilisation de structures de communications légères, de réduire de 25%, sur 16384 processus MPI, l'empreinte mémoire du code et ainsi d'éviter le dépeuplement des coeurs pendant la simulation. De plus, on a montré qu'il est possible de tirer profit du modèle de données lightAMR, pour changer le nombre de processus d'une simulation. Cette approche nous a permis de faire la relaxation des conditions initiales à basse résolution sur 4096 processus MPI, et de monter à 16384 processus pour la phase haute résolution.

Ensuite, on a présenté les modèles physiques utilisés ainsi que les conditions initiales de la simulation. On a vu que la première étape se base sur la relaxation d'un disque de gaz, aux profils exponentiels, dans un potentiel gravitationnel produit par des particules de matière noire, un trou noir centrale et de particules de vieilles étoiles. Dans la première partie, ces conditions initiales, sans activation de la formation stellaire, ont permis de développer des grandes structures aux grandes échelles dans le disque : bras spiraux et barre centrale.

Dans un deuxième temps, on a présenté les premiers résultats de la simulation dans sa partie haute résolution avec l'activation de la fonction de chauffage/refroidissement, de la formation stellaire et la rétroaction cinétique de supernovae. Dans cette deuxième partie de `ExaMilkyWay`, on résout 85% la masse du disque à 3.4 pc et 45% à 0.9 pc. On a pu mettre en évidence la formation de structures filamentaires entre les bras spiraux, dans les régions peu denses, sans formation stellaire, où la résolution spatiale est $<= 1.8$ pc. On a vu également, grâce au spectre d'énergie à $t = 26$ Myr qu'on pouvait mettre en évidence trois pentes différentes mettant en relief des régimes turbulents différents.

Il est important de noter que grâce au modèle `lightAMR` et à ses algorithmes d'élagage et de compression, on a pu stocker plus de 150 sorties d'analyses soit environ ~ 25 To. Les dernières sorties d'analyses occupent un espace d'environ 200 Go par sortie. Il reste donc encore de nombreuses analyses à effectuer, notamment temporelle, afin d'éclaircir la formation des structures filamentaires ou l'impact des `SNe` par exemple. De plus, on a présenté jusqu'ici uniquement des analyses `2D`. L'apport d'analyses et/ou de visualisations en `3D` peut être un atout non négligeable qui reste encore à exploiter.

Chapitre 7

Conclusion et perspectives

La turbulence du gaz dans le milieu interstellaire est un phénomène complexe qu'il n'est pas possible d'étudier en laboratoire. Les simulations numériques sont donc un outil puissant et indispensable pour son étude. La nature multi-physique et multi-échelles de la turbulence et de ses origines, impliquent l'utilisation d'un logiciel massivement parallèle, ainsi que de méthodes de maillage adaptatif. Dans le but de pouvoir résoudre finement le développement de la turbulence à grande échelle, et de suivre sa cascade jusqu'aux petites échelles, on a réalisé une simulation de galaxie isolée ainsi qu'un plan de gestion des données. Il était alors clair que la faisabilité de cette simulation, avec le code `RAMSES`, reposait sur la résolution de problèmes techniques, rencontrés par des simulations équivalentes dans la littérature, centrés autour des `E/S` et de la gestion de grands volumes de données.

Dans un premier temps, on a apporté une solution aux problèmes d'`E/S` du code `RAMSES` avec l'intégration d'une bibliothèque d'`E/S` parallèle : `Hercule`, voir chapitre 2. On a montré comment la séparation des flux de données permet d'optimiser à la fois les écritures/lectures, mais aussi les modèles de données. Ces développements ont permis d'améliorer la scalabilité du code dont les écritures/lectures ont été testées, avec succès, sur 40000 et 50000 processus `MPI` durant les premiers tests avec la simulation `ExaMilkyWay`. Avec ces développements, le nombre de fichiers produits peut être facilement réduit d'un facteur 64, voire plus. Alors que les temps d'écriture/lecture d'une protection/reprise d'une simulation telle que `Extreme-Horizon` (12800 processus `MPI`, $\sim 4\ To$) pouvaient dépasser 1h, ceux de la simulation `ExaMilkyWay` (16384 processus, $\sim 4.5\ To$) sont tous inférieurs à 5 minutes. Et de l'ordre de quelques dizaines de secondes pour les données destinées au post-traitement avec le modèle `lightAMR` et les méthodes de compressions associées.

Dans un deuxième temps, au chapitre 3, on a montré que la mise en place, pour le flux de données destinées au post-traitement, d'un modèle de données compact et auto-portant, spécifique pour les données `AMR` en arbre, permet un gain significatif en terme d'espace de stockage de l'ordre d'un facteur 40. On a également mis en avant l'apport de la compression de données par le biais du développement d'un algorithme de compression de données flottantes, dédié au modèle `lightAMR`, capable d'atteindre des performances supérieures, en terme de ratio et de vitesse, sur des données `RAMSES` converties au modèle `lightAMR`, aux bibliothèques à l'état de l'art.

Afin de pouvoir analyser les données décrites avec le modèle `lightAMR`, un code d'analyse au parallélisme hybride `MPI-OpenMP`, compatible avec les architectures multi-coeurs, a été développé et est détaillé au chapitre 4. Différentes méthodes permettent d'analyser efficacement les grands volumes de données, tout en minimisant les ressources nécessaires aussi bien en terme de mémoire et de calculs, que de lecture de données. On a également détaillé différents algorithmes de production de cartes `2D` et notamment un algorithme de lancer de rayon efficace, exploitant le pavage cubique minimal comme structure d'accélération naturellement accessible avec des données `AMR` en arbre. Dans le but de mettre en avant le partage des données et la communication scientifique de résultats de simulation, une contribution a été faite à l'écosystème de la base de données en ligne `Galactica`. Enfin, on a décrit brièvement la nouvelle structure `VTK`, les `HyperTreeGrid`, ainsi que leur construction à partir des données `lightAMR`. Cette nouvelle structure très prometteuse pour la visualisation des données `AMR` à grande volumétrie, plusieurs centaines de millions de cellules, sur un ordinateur portable, est nativement compatible avec des outils `HPC` tel que `Paraview`.

Après une introduction au milieu interstellaire, à la formation d'étoile et à leur rétroaction au chapitre 5, on met en place au chapitre 6, la simulation `ExaMilkyWay`. Lors du déploiement de la simulation, de nouveaux

développements ont été nécessaires afin de résoudre des points bloquants. En effet, la mise en place de structures de communication légères a permis un gain de 25% de la mémoire consommé par le code. Ainsi, il n'a pas été nécessaire, durant toute la simulation, de dépeupler les coeurs de calculs. De plus, le développement de routines de modification du nombre de processus `MPI` a permis d'effectuer une partie de la simulation, à basse résolution sur 4096 processus et de porter ensuite la simulation sur 16384 processus pour la phase haute résolution. Ces développements ont permis de consommer efficacement les heures de calcul du projet.

Fort de tout ces développements, la simulation `ExaMilkyWay` d'une galaxie isolée comparable à la Voie Lactée, a atteint une résolution spatiale de ~ 3 *pc* dans plus de 85% de la masse du disque de gaz dont 50% à ~ 1 *pc*, ce qui n'avait jamais été fait jusque là. Grâce à cette résolution élevée dans les régions peu denses du milieu interstellaire par rapport aux simulations de la littérature, on a pu mettre en évidence, lors de la présentation des résultats préliminaires, la formation de structures filamentaires entre les bras spiraux. Ces régions stockent une partie du gaz disponible tout en étant dépourvues de formation stellaire et donc de rétroaction. Au travers des résultats préliminaires présentés, la signature des supernovae dans les spectres de puissance n'est pas claire et nécessite de continuer l'évolution temporelle de la simulation. De même, de plus amples analyses sont nécessaires pour caractériser les différents régimes turbulents mis en évidence.

Les tests d'`E/S` effectués jusqu'à 50000 processus `MPI`, montrent que la stratégie d'intégration de `Hercule` permet d'améliorer la scalabilité du code. Grâce à la frugalité et à la modularité de la nouvelle approche des `E/S`, il est possible d'affiner l'analyse temporelle des simulations. En effet, la séparation des flux et la mise en place du modèle `lightAMR`, qui permet de stocker dans seulement 25 *To* plus de 150 sorties d'analyses, montrent que notre approche permet un gain en terme de scalabilité, mais aussi du point de vue des données et de l'espace de stockage utilisé. Enfin, l'outil d'analyse et la méthode de traitement par domaine permettent d'analyser rapidement, avec une bonne scalabilité également, les données produites malgré le grand nombre de cellules.

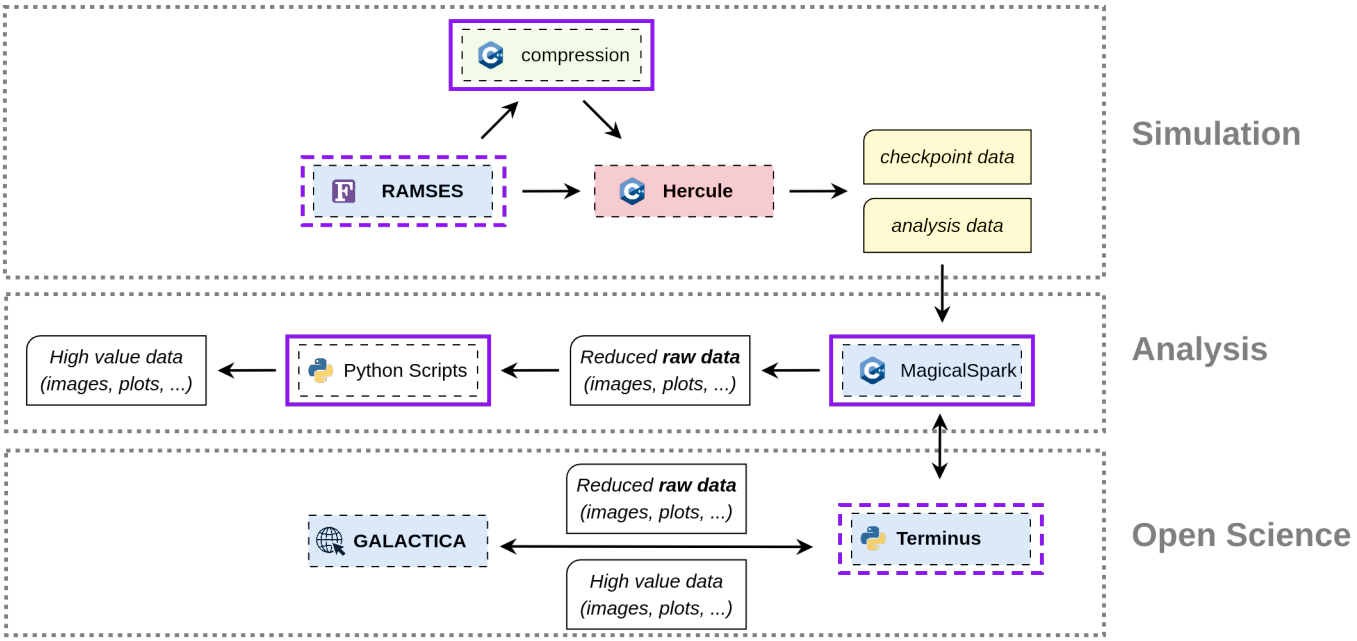

FIGURE 7.1 – Pipeline de données et contributions. Les contributions à un code existant sont encadrés en pointillé mauve et les contributions sous la forme d'un développement "*from scratch*" sont encadrés en ligne pleine mauve.

Pour conclure, au cours de ces travaux de thèse, j'ai développé le *pipeline* de données dont on donne un schéma sur la figure 7.1. Il reste néanmoins de nombreux points à finaliser afin de faire bénéficier la communauté des utilisateurs des développements effectués durant ma thèse. En effet, l'outil de conversion des données `RAMSES` vers des données au modèles `lightAMR` avec `Hercule` (ou `HDF5`) doit être rendu public, de même que l'outil d'analyse des données au modèle `lightAMR`. Il est donc nécessaire de finaliser les développements et la documen-

tation. De plus, l'interfaçage avec `Galatica` de l'outil d'analyse, doit être enrichi afin de rendre disponible aux utilisateurs : la conversion de données au modèle `lightAMR`, l'extraction de zones d'intérêts, la conversion vers le modèle `vtkHyperTreeGrid` pour de la visualisation `3D`. Cette interaction avec `Galactica` est un point important qui favoriserait la réutilisation des données de la simulation `ExaMilkyWay` dans une démarche de science ouverte.

Bibliographie

[Adams et al., 2000] Adams, M., Colella, P., Graves, D. T., Johnson, J., Keen, N., Ligocki, T. J., Martin, D. F., McCorquodale, P., Modiano, D., Schwartz, P., Sternberg, T., and Straalen, B. V. (2000). Chombo software package for amr applications. *Report LBNL-6616E, Lawrence Berkeley National Laboratory Technical*.

[Adler and Al., 2022] Adler, M. and Al. (2022). Portable Network Graphics (PNG) Specification (Third Edition). https ://www.w3.org/TR/png/.

[Ahrens et al., 2005] Ahrens, J., Geveci, B., Law, C., Hansen, C., and Johnson, C. (2005). paraview : An end-user tool for large-data visualization. *The visualization handbook*, 717 :50038–1.

[Almeida et al., 2014] Almeida, S., Oliveira, V., Pina, A., and Melle-Franco, M. (2014). Two high-performance alternatives to zlib scientific-data compression. In Murgante, B., Misra, S., Rocha, A. M. A. C., Torre, C., Rocha, J. G., Falcão, M. I., Taniar, D., Apduhan, B. O., and Gervasi, O., editors, *Computational Science and Its Applications – ICCSA 2014*, pages 623–638, Cham. Springer International Publishing.

[André, 2017] André, P. (2017). Interstellar filaments and star formation. *Comptes Rendus. Géoscience*, 349(5) :187–197.

[Athanassoula and Misiriotis, 2002] Athanassoula, E. and Misiriotis, A. (2002). Morphology, photometry and kinematics of N -body bars - I. Three models with different halo central concentrations. *MNRAS*, 330(1) :35–52.

[Audit and Hennebelle, 2010] Audit, E. and Hennebelle, P. (2010). On the structure of the turbulent interstellar clouds . Influence of the equation of state on the dynamics of 3D compressible flows. *A&A*, 511 :A76.

[Audit, E. and Hennebelle, P., 2005] Audit, E. and Hennebelle, P. (2005). Thermal condensation in a turbulent atomic hydrogen flow*. *A&A*, 433(1) :1–13.

[Baker et al., 2016] Baker, A. H., Hammerling, D. M., Mickelson, S. A., Xu, H., Stolpe, M. B., Naveau, P., Sanderson, B., Ebert-Uphoff, I., Samarasinghe, S., De Simone, F., Carbone, F., Gencarelli, C. N., Dennis, J. M., Kay, J. E., and Lindstrom, P. (2016). Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development*, 9(12) :4381–4403.

[Ballester-Ripoll et al., 2018] Ballester-Ripoll, R., Lindstrom, P., and Pajarola, R. (2018). TTHRESH : tensor compression for multidimensional visual data. *CoRR*, abs/1806.05952.

[Bennett et al., 2012] Bennett, J. C., Abbasi, H., Bremer, P.-T., Grout, R., Gyulassy, A., Jin, T., Klasky, S., Kolla, H., Parashar, M., Pascucci, V., Pebay, P., Thompson, D., Yu, H., Zhang, F., and Chen, J. (2012). Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, Washington, DC, USA. IEEE Computer Society Press.

[Berger and Colella, 1989] Berger, M. and Colella, P. (1989). Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1) :64–84.

[Berger and Oliger, 1984] Berger, M. J. and Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3) :484–512.

[Berthoud et al., 2020] Berthoud, F., Bzeznik, B., Gibelin, N., Laurens, M., Bonamy, C., Morel, M., and Schwindenhammer, X. (2020). Estimation de l'empreinte carbone d'une heure.coeur de calcul. Research report, UGA - Université Grenoble Alpes ; CNRS ; INP Grenoble ; INRIA.

[Bigiel et al., 2011] Bigiel, F., Leroy, A., Walter, F., Brinks, E., de Blok, W., Kramer, C., Rix, H., Schruba, A., Schuster, K.-F., Usero, A., et al. (2011). A constant molecular gas depletion time in nearby disk galaxies. *The Astrophysical Journal Letters*, 730(2) :L13.

[Bigiel et al., 2008] Bigiel, F., Leroy, A., Walter, F., Brinks, E., de Blok, W. J. G., Madore, B., and Thornley, M. D. (2008). The star formation law in nearby galaxies on sub-kpc scales. *The Astronomical Journal*, 136(6) :2846.

[Binney and Knebe, 2002] Binney, J. and Knebe, A. (2002). Two-body relaxation in cosmological simulations. *MNRAS*, 333(2) :378–382.

[Block et al., 2010] Block, D. L., Puerari, I., Elmegreen, B. G., and Bournaud, F. (2010). A Two-component Power Law Covering Nearly Four Orders of Magnitude in the Power Spectrum of Spitzer Far-infrared Emission from the Large Magellanic Cloud. *The Astrophysical Journal Letters*, 718(1) :L1–L6.

[Bohanec and Bratko, 1994] Bohanec, M. and Bratko, I. (1994). Trading accuracy for simplicity in decision trees. *Machine Learning*, 15 :223–250.

[Boix and Cantó, 2010] Boix, M. and Cantó, B. (2010). Wavelet transform application to the compression of images. *Mathematical and Computer Modelling*, 52(7) :1265–1270.

[Bournaud et al., 2010] Bournaud, F., Elmegreen, B. G., Teyssier, R., Block, D. L., and Puerari, I. (2010). ISM properties in hydrodynamic galaxy simulations : turbulence cascades, cloud formation, role of gravity and feedback. *MNRAS*, 409(3) :1088–1099.

[Bressand et al., 2012] Bressand, O., Colombet, L., Fontaine, A., Harel, G., and Lekien, J.-B. (2012). Hercule : A library of scientific data management for numerical simulation. *Chocs*, (41) :29–37.

[Brucy et al., 2020] Brucy, N., Hennebelle, P., Bournaud, F., and Colling, C. (2020). Large-scale turbulent driving regulates star formation in high-redshift gas-rich galaxies. *The Astrophysical Journal*, 896(2) :L34.

[Burtscher and Ratanaworabhan, 2007] Burtscher, M. and Ratanaworabhan, P. (2007). High throughput compression of double-precision floating-point data. In *2007 Data Compression Conference (DCC'07)*, pages 293–302.

[Byna et al., 2012] Byna, S., Chou, J., Rubel, O., Prabhat, Karimabadi, H., Daughter, W. S., Roytershteyn, V., Bethel, E. W., Howison, M., Hsu, K.-J., Lin, K.-W., Shoshani, A., Uselton, A., and Wu, K. (2012). Parallel i/o, analysis, and visualization of a trillion particle simulation. In *SC '12 : Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12.

[Cappello et al., 2019] Cappello, F., Di, S., Li, S., Liang, X., Gok, A. M., Tao, D., Yoon, C. H., Wu, X.-C., Alexeev, Y., and Chong, F. T. (2019). Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications*, 33(6) :1201–1220.

[Capul et al., 2018] Capul, J., Morais, S., and Lekien, J.-B. (2018). PaDaWAn : a Python Infrastructure for Loosely Coupled In Situ Workflows. In *ISAV '18 : In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 7–12, Dallas, United States. ACM Press.

[Chabanier et al., 2020] Chabanier, S., Bournaud, F., Dubois, Y., Codis, S., Chapon, D., Elbaz, D., Pichon, C., Bressand, O., Devriendt, J., Gavazzi, R., and et al. (2020). Formation of compact galaxies in the extreme-horizon simulation. *Astronomy and Astrophysics*, 643 :L8.

[Chapon, 2011] Chapon, D. (2011). *Simulations of binary galaxy mergers : the effect of gas physics on small scales*. PhD thesis, "University of Paris 7". 2011PA077210.

[Chou et al., 2011a] Chou, J., Howison, M., Austin, B., Wu, K., Qiang, J., Bethel, E. W., Shoshani, A., Rübel, O., Prabhat, and Ryne, R. D. (2011a). Parallel index and query for large scale data analysis. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, New York, NY, USA. Association for Computing Machinery.

[Chou et al., 2011b] Chou, J., Wu, K., and Prabhat (2011b). Fastquery : A general indexing and querying system for scientific data. In Bayard Cushing, J., French, J., and Bowers, S., editors, *Scientific and Statistical Database Management*, pages 573–574, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Chou et al., 2011c] Chou, J., Wu, K., and Prabhat (2011c). Fastquery : A parallel indexing system for scientific data. In *2011 IEEE International Conference on Cluster Computing*, pages 455–464.

[Colling et al., 2018] Colling, C., Hennebelle, P., Geen, S., Iffrig, O., and Bournaud, F. (2018). Impact of galactic shear and stellar feedback on star formation. *Astronomy And Astrophysics*, 620 :A21.

[Colman et al., 2022] Colman, T., Robitaille, J.-F., Hennebelle, P., Miville-Deschênes, M.-A., Brucy, N., Klessen, R. S., Glover, S. C. O., Soler, J. D., Elia, D., Traficante, A., Molinari, S., and Testi, L. (2022). The signature of large-scale turbulence driving on the structure of the interstellar medium. *Monthly Notices of the Royal Astronomical Society*, 514(3) :3670–3684.

[Combes et al., 2012] Combes, F., Boquien, M., Kramer, C., Xilouris, E. M., Bertoldi, F., Braine, J., Buchbender, C., Calzetti, D., Gratier, P., Israel, F., Koribalski, B., Lord, S., Quintana-Lacaci, G., Relañ o, M., Röllig, M., Stacey, G., Tabatabaei, F. S., Tilanus, R. P. J., van der Tak, F., van der Werf, P., and Verley, S. (2012). Dust and gas power spectrum in m 33 (HERM33es). *Astronomy And Astrophysics*, 539 :A67.

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

[Di and Cappello, 2016] Di, S. and Cappello, F. (2016). Fast error-bounded lossy hpc data compression with sz. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 730–739.

[Draine, 2011] Draine, B. T. (2011). *Physics of the interstellar and intergalactic medium*, volume 19. Princeton University Press.

[Dubois and Teyssier, 2008] Dubois, Y. and Teyssier, R. (2008). On the onset of galactic winds in quiescent star forming galaxies. *A&A*, 477(1) :79–94.

[Durocher and Delorme, 2024] Durocher, A. and Delorme, M. (2024). Dyablo : a new hardware agnostic amr code. *In prep.*

[Elmegreen, 1985] Elmegreen, B. G. (1985). Molecular clouds and star formation : an overview. In Black, D. C. and Matthews, M. S., editors, *Protostars and Planets II*, pages 33–58.

[Elmegreen, 2002] Elmegreen, B. G. (2002). Star Formation from Galaxies to Globules. *The Astrophysical Journal*, 577(1) :206–220.

[Elmegreen and Falgarone, 1996] Elmegreen, B. G. and Falgarone, E. (1996). A fractal origin for the mass spectrum of interstellar clouds. *The Astrophysical Journal*, 471(2) :816.

[Emsellem et al., 1994] Emsellem, E., Monnet, G., and Bacon, R. (1994). The multi-gaussian expansion method : a tool for building realistic photometric and kinematical models of stellar systems I. The formalism. *A&A*, 285 :723–738.

[Falgarone and Phillips, 1991] Falgarone, E. and Phillips, T. G. (1991). Signatures of turbulence in the dense interstellar medium. In Falgarone, E., Boulanger, F., and Duvert, G., editors, *Fragmentation of Molecular Clouds and Star Formation*, pages 119–136, Dordrecht. Springer Netherlands.

[Federrath and Klessen, 2012] Federrath, C. and Klessen, R. S. (2012). The Star Formation Rate of Turbulent Magnetized Clouds : Comparing Theory, Simulations, and Observations. *ApJ*, 761(2) :156.

[Federrath et al., 2008] Federrath, C., Klessen, R. S., and Schmidt, W. (2008). The density probability distribution in compressible isothermal turbulence : Solenoidal versus compressive forcing. *The Astrophysical Journal*, 688(2) :L79–L82.

[Fensch, Jérémy et al., 2023] Fensch, Jérémy, Bournaud, Frédéric, Brucy, Noé, Dubois, Yohan, Hennebelle, Patrick, and Rosdahl, Joakim (2023). Universal gravity-driven isothermal turbulence cascade in disk galaxies. *A&A*, 672 :A193.

[Ferguson et al., 1996] Ferguson, A. M. N., Wyse, R. F. G., Gallagher, J. S., I., and Hunter, D. A. (1996). Diffuse Ionized Gas in Spiral Galaxies : Probing Lyman Continuum Photon Leakage From H II Regions ? *The Astrophysical Journal*, 111 :2265.

[Fout and Ma, 2012] Fout, N. and Ma, K.-L. (2012). An adaptive prediction-based approach to lossless compression of floating-point volume data. *Visualization and Computer Graphics, IEEE Transactions on*, 18 :2295–2304.

[Friesen et al., 2016] Friesen, B., Almgren, A. S., Lukic, Z., Weber, G. H., Morozov, D., Beckner, V. E., and Day, M. S. (2016). In situ and in-transit analysis of cosmological simulations. *Computational Astrophysics and Cosmology*, 3.

[Fryxell et al., 2000] Fryxell, B., Olson, K., Ricker, P., Timmes, F. X., Zingale, M., Lamb, D. Q., MacNeice, P., Rosner, R., Truran, J. W., and Tufo, H. (2000). Flash : An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131(1) :273.

[Fu et al., 2017] Fu, H., He, C., Chen, B., Yin, Z., Zhang, Z., Zhang, W., Zhang, T., Xue, W., Liu, W., Yin, W., Yang, G., and Chen, X. (2017). 18.9-pflops nonlinear earthquake simulation on sunway taihulight : Enabling depiction of 18-hz and 8-meter scenarios. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA. Association for Computing Machinery.

[Gabor and Bournaud, 2013] Gabor, J. M. and Bournaud, F. (2013). Delayed star formation in high-redshift stream-fed galaxies. *Monthly Notices of the Royal Astronomical Society : Letters*, 437(1) :L56–L60.

[Gatto et al., 2016] Gatto, A., Walch, S., Naab, T., Girichidis, P., Wünsch, R., Glover, S. C. O., Klessen, R. S., Clark, P. C., Peters, T., Derigs, D., Baczynski, C., and Puls, J. (2016). The SILCC project – III. Regulation of star formation and outflows by stellar winds and supernovae. *Monthly Notices of the Royal Astronomical Society*, 466(2) :1903–1924.

[Girichidis et al., 2020] Girichidis, P., Offner, S. S. R., Kritsuk, A. G., Klessen, R. S., Hennebelle, P., Kruijssen, J. M. D., Krause, M. G. H., Glover, S. C. O., and Padovani, M. (2020). Physical processes in star formation. *Space Science Reviews*, 216(4).

[Gnat and Sternberg, 2007] Gnat, O. and Sternberg, A. (2007). Time-dependent Ionization in Radiatively Cooling Gas. *ApJS*, 168(2) :213–230.

[Gray et al., 2005] Gray, J., Liu, D. T., Nieto-Santisteban, M., Szalay, A., DeWitt, D. J., and Heber, G. (2005). Scientific data management in the coming decade. *SIGMOD Rec.*, 34(4) :34–41.

[Guillet et al., 2013] Guillet, T., Chapon, D., and Labadens, M. (2013). PyMSES : Python modules for RAMSES. Astrophysics Source Code Library, record ascl :1310.002.

[Guillet and Teyssier, 2011] Guillet, T. and Teyssier, R. (2011). A simple multigrid scheme for solving the Poisson equation with arbitrary domain boundaries. *Journal of Computational Physics*, 230(12) :4756–4771.

[Habets and Heintze, 1981] Habets, G. and Heintze, J. (1981). Empirical bolometric corrections for the main-sequence. *Astronomy and Astrophysics Supplement Series, vol. 46, Nov. 1981, p. 193-237.*, 46 :193–237.

[Harel et al., 2017a] Harel, G., Lekien, J.-B., and Pébaÿ, P. P. (2017a). Two New Contributions to the Visualization of AMR Grids : I. Interactive Rendering of Extreme-Scale 2-Dimensional Grids II. Novel Selection Filters in Arbitrary Dimension. *arXiv e-prints*, page arXiv :1703.00212.

[Harel et al., 2017b] Harel, G., Lekien, J.-B., and Pébaÿ, P. P. (2017b). Visualization and Analysis of Large-Scale, Tree-Based, Adaptive Mesh Refinement Simulations with Arbitrary Rectilinear Geometry. *arXiv e-prints*, page arXiv :1702.04852.

[Harrington, 2018] Harrington, P. (2018). Diagnosing Parallel I/O Bottlenecks in HPC Applications.

[Heiles and Troland, 2003] Heiles, C. and Troland, T. H. (2003). The millennium arecibo 21 centimeter absorption-line survey. i. techniques and gaussian fits. *The Astrophysical Journal Supplement Series*, 145(2) :329.

[Hennebelle, 2018] Hennebelle, P. (2018). The frigg project : From intermediate galactic scales to self-gravitating cores. *Astronomy and Astrophysics*, 611 :A24.

[Hennebelle and Audit, 2007] Hennebelle, P. and Audit, E. (2007). On the structure of the turbulent interstellar atomic hydrogen. I. Physical characteristics. Influence and nature of turbulence in a thermally bistable flow. *A&A*, 465(2) :431–443.

[Hennebelle and Falgarone, 2012] Hennebelle, P. and Falgarone, E. (2012). Turbulent molecular clouds. *The Astronomy and Astrophysics Review*, 20(1).

[Hennebelle and Iffrig, 2014] Hennebelle, P. and Iffrig, O. (2014). Simulations of magnetized multiphase galactic disc regulated by supernovae explosions. *A&A*, 570 :A81.

[Hockney and Eastwood, 1988] Hockney, R. W. and Eastwood, J. W. (1988). *Computer simulation using particle*. Bristol : Hilger, 1988.

[Hollenbach et al., 1971] Hollenbach, D. J., Werner, M. W., and Salpeter, E. E. (1971). Molecular hydrogen in h i regions. *The Astrophysical Journal*, 163 :165.

[Hopkins et al., 2014] Hopkins, P. F., Keres, D., Onorbe, J., Faucher-Giguère, C.-A., Quataert, E., Murray, N., and Bullock, J. S. (2014). Galaxies on FIRE (Feedback In Realistic Environments) : stellar feedback explains cosmologically inefficient star formation. *Monthly Notices of the Royal Astronomical Society*, 445(1) :581–603.

[Hornung and Kohn, 2002] Hornung, R. D. and Kohn, S. R. (2002). Managing application complexity in the samrai object-oriented framework. *Concurrency and Computation : Practice and Experience*, 14(5) :347–368.

[Huffman, 1952] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9) :1098–1101.

[Ibarria et al., 2003] Ibarria, L., Lindstrom, P., Rossignac, J., and Szymczak, A. (2003). Out-of-core compression and decompression of large n-dimensional scalar fields. *Comput. Graph. Forum*, 22 :343–348.

[Iffrig and Hennebelle, 2015] Iffrig, O. and Hennebelle, P. (2015). Mutual influence of supernovae and molecular clouds. *A&A*, 576 :A95.

[Isenburg et al., 2005] Isenburg, M., Lindstrom, P., and Snoeyink, J. (2005). Lossless compression of predicted floating-point geometry. *Computer-Aided Design*, 37(8) :869–877. CAD '04 Special Issue : Modelling and Geometry Representations for CAD.

[Janka, 2012] Janka, H.-T. (2012). Explosion mechanisms of core-collapse supernovae. *Annual Review of Nuclear and Particle Science*, 62(1) :407–451.

[Jin et al., 2020] Jin, S., Grosset, P., Biwer, C. M., Pulido, J., Tian, J., Tao, D., and Ahrens, J. (2020). Understanding gpu-based lossy compression for extreme-scale cosmological simulations. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 105–115.

[Katz, 1992] Katz, N. (1992). Dissipational Galaxy Formation. II. Effects of Star Formation. *ApJ*, 391 :502.

[Kennicutt, 1989] Kennicutt, Robert C., J. (1989). The Star Formation Law in Galactic Disks. *ApJ*, 344 :685.

[Kennicutt and Evans, 2012] Kennicutt, R. C. and Evans, N. J. (2012). Star Formation in the Milky Way and Nearby Galaxies. *Annual Review of Astronomy and Astrophysics*, 50 :531–608.

[Kennicutt Jr, 1998] Kennicutt Jr, R. C. (1998). The global schmidt law in star-forming galaxies. *The astrophysical journal*, 498(2) :541.

[Keres et al., 2009] Keres, D., Katz, N., Davé, R., Fardal, M., and Weinberg, D. H. (2009). Galaxies in a simulated CDM universe – II. Observable properties and constraints on feedback. *Monthly Notices of the Royal Astronomical Society*, 396(4) :2332–2344.

[Kevlahan and Pudritz, 2009] Kevlahan, N. and Pudritz, R. E. (2009). Shock-generated vorticity in the interstellar medium and the origin of the stellar initial mass function. *The Astrophysical Journal*, 702(1) :39.

[Khokhlov, 1998] Khokhlov, A. (1998). Fully Threaded Tree Algorithms for Adaptive Refinement Fluid Dynamics Simulations. *Journal of Computational Physics*, 143(2) :519–543.

[Kim and Ryu, 2005] Kim, J. and Ryu, D. (2005). Density power spectrum of compressible hydrodynamic turbulent flows. *The Astrophysical Journal*, 630(1) :L45.

[Kitware, 2023] Kitware (2023). Paraview. https ://www.paraview.org/.

[Kolmogorov, 1968] Kolmogorov, A. N. (1968). Local structure of turbulence in an incompressible viscous fluid at very high reynolds numbers. *Soviet Physics Uspekhi*, 10(6) :734.

[Kraljic et al., 2014] Kraljic, K., Renaud, F., Bournaud, F., Combes, F., Elmegreen, B., Emsellem, E., and Teyssier, R. (2014). The Role of Turbulence in Star Formation Laws and Thresholds. *The Astrophysical Journal*, 784(2) :112.

[Kravtsov et al., 1997] Kravtsov, A. V., Klypin, A. A., and Khokhlov, A. M. (1997). Adaptive Refinement Tree : A New High-Resolution N-Body Code for Cosmological Simulations. *ApJS*, 111(1) :73–94.

[Kretschmer and Teyssier, 2019] Kretschmer, M. and Teyssier, R. (2019). Forming early-type galaxies without AGN feedback : a combination of merger-driven outflows and inefficient star formation. *Monthly Notices of the Royal Astronomical Society*, 492(1) :1385–1398.

[Kritsuk et al., 2007] Kritsuk, A. G., Norman, M. L., Padoan, P., and Wagner, R. (2007). The statistics of supersonic isothermal turbulence. *The Astrophysical Journal*, 665(1) :416–431.

[Kritsuk et al., 2010] Kritsuk, A. G., Norman, M. L., and Wagner, R. (2010). On the density distribution in star forming intertellar clouds. *The Astrophysical Journal*, 727(1) :L20.

[Kroupa, 2001] Kroupa, P. (2001). On the variation of the initial mass function. *MNRAS*, 322(2) :231–246.

[Krumholz and Tan, 2007] Krumholz, M. R. and Tan, J. C. (2007). Slow star formation in dense gas : Evidence and implications. *The Astrophysical Journal*, 654(1) :304–315.

[Kulkarni and Heiles, 1988] Kulkarni, S. R. and Heiles, C. (1988). Neutral hydrogen and the diffuse interstellar medium. In Kellermann, K. I. and Verschuur, G. L., editors, *Galactic and Extragalactic Radio Astronomy*, pages 95–153. Springer.

[Kumar et al., 2014] Kumar, S., Edwards, J., Bremer, P.-T., Knoll, A., Christensen, C., Vishwanath, V., Carns, P., Schmidt, J. A., and Pascucci, V. (2014). Efficient i/o and storage of adaptive-resolution data. In *SC '14 : Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 413–423.

[Kumar et al., 2012] Kumar, S., Vishwanath, V., Carns, P., Levine, J. A., Latham, R., Scorzelli, G., Kolla, H., Grout, R., Ross, R., Papka, M. E., Chen, J., and Pascucci, V. (2012). Efficient data restructuring and aggregation for i/o acceleration in pidx. In *SC '12 : Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11.

[Labadens et al., 2012] Labadens, M., Pomarède, D., Chapon, D., Teyssier, R., Bournaud, F., Renaud, F., and Grandjouan, N. (2012). Volume rendering of amr simulations.

[Lada and Lada, 2003] Lada, C. J. and Lada, E. A. (2003). Embedded Clusters in Molecular Clouds. *Annual Review of Astronomy and Astrophysics*, 41 :57–115.

[Lakshminarasimhan et al., 2011] Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., and Samatova, N. F. (2011). Compressing the incompressible with isabela : In-situ reduction of spatio-temporal data. In Jeannot, E., Namyst, R., and Roman, J., editors, *Euro-Par 2011 Parallel Processing*, pages 366–379, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Lakshminarasimhan et al., 2013] Lakshminarasimhan, S., Shah, N., Ethier, S., Ku, S., Chang, C.-S., Klasky, S., Latham, R., Ross, R. B., and Samatova, N. F. (2013). Isabela for effective in situ compression of scientific data. *Concurrency and Computation : Practice and Experience*, 25.

[Larson, 1981] Larson, R. B. (1981). Turbulence and star formation in molecular clouds. *MNRAS*, 194 :809–826.

[Larson, 2003] Larson, R. B. (2003). The physics of star formation. *Reports on Progress in Physics*, 66(10) :1651–1697.

[Latham et al., 2012] Latham, R., Daley, C., Liao, W.-k., Gao, K., Ross, R., Dubey, A., and Choudhary, A. (2012). A case study for scientific I/O : improving the FLASH astrophysics code. *Computational Science and Discovery*, 5(1) :015001.

[Lebreuilly et al., 2020] Lebreuilly, U., Commerçon, B., and Laibe, G. (2020). Protostellar collapse : the conditions to form dust-rich protoplanetary disks. *A&A*, 641 :A112.

[Lewis et al., 2022] Lewis, J. S. W., Ocvirk, P., Sorce, J. G., Dubois, Y., Aubert, D., Conaboy, L., Shapiro, P. R., Dawoodbhoy, T., Teyssier, R., Yepes, G., Gottlöber, S., Rasera, Y., Ahn, K., Iliev, I. T., Park, H., and Thelie, E. (2022). The short ionizing photon mean free path at z = 6 in Cosmic Dawn III, a new fully coupled radiation-hydrodynamical simulation of the Epoch of Reionization. *Monthly Notices of the Royal Astronomical Society*, 516(3) :3389–3397.

[Liang et al., 2018] Liang, X., Di, S., Tao, D., Li, S., Li, S., Guo, H., Chen, Z., and Cappello, F. (2018). Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 438–447.

[Liang et al., 2023] Liang, X., Zhao, K., Di, S., Li, S., Underwood, R., Gok, A. M., Tian, J., Deng, J., Calhoun, J. C., Tao, D., Chen, Z., and Cappello, F. (2023). Sz3 : A modular framework for composing prediction-based error-bounded lossy compressors. *IEEE Transactions on Big Data*, 9(2) :485–498.

[Lindstrom, 2014] Lindstrom, P. (2014). Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20.

[Lindstrom and Isenburg, 2006] Lindstrom, P. and Isenburg, M. (2006). Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5) :1245–1250.

[Liu et al., 2023] Liu, T., Wang, J., Liu, Q., Alibhai, S., Lu, T., and He, X. (2023). High-ratio lossy compression : Exploring the autoencoder to compress scientific data. *IEEE Transactions on Big Data*, 9(1) :22–36.

[Mac Low and Klessen, 2004] Mac Low, M.-M. and Klessen, R. S. (2004). Control of star formation by supersonic turbulence. *Reviews of Modern Physics*, 76(1) :125–194.

[Maffeis, 1993] Maffeis, S. (1993). Cache management algorithms for flexible filesystems. *SIGMETRICS Perform. Eval. Rev.*, 21(2) :16–25.

[McGuire, 2018] McGuire, B. A. (2018). 2018 census of interstellar, circumstellar, extragalactic, protoplanetary disk, and exoplanetary molecules. *The Astrophysical Journal Supplement Series*, 239(2) :17.

[McKee and Ostriker, 2007] McKee, C. F. and Ostriker, E. C. (2007). Theory of Star Formation. *Annual Review of Astronomy and Astrophysics*, 45(1) :565–687.

[McKee and Ostriker, 1977] McKee, C. F. and Ostriker, J. P. (1977). A theory of the interstellar medium : three components regulated by supernova explosions in an inhomogeneous substrate. *ApJ*, 218 :148–169.

[Mihalas and Binney, 1981] Mihalas, D. and Binney, J. (1981). *Galactic astronomy. Structure and kinematics.*

[Monnet et al., 1992] Monnet, G., Bacon, R., and Emsellem, E. (1992). Modelling the stellar intensity and radial velocity fields in triaxial galaxies by sums of Gaussian functions. *A&A*, 253 :366–373.

[Murray et al., 2011] Murray, N., Ménard, B., and Thompson, T. A. (2011). Radiation Pressure from Massive Star Clusters as a Launching Mechanism for Super-galactic Winds. *ApJ*, 735(1) :66.

[NASA, 2023] NASA (2002-2023). Panoply netCDF, HDF and GRIB Data Viewer. https ://www.giss.nasa.gov/tools/panoply/.

[Navarro and White, 1993] Navarro, J. F. and White, S. D. M. (1993). Simulations of Dissipative Galaxy Formation in Hierarchically Clustering Universes - Part One - Tests of the Code. *MNRAS*, 265 :271.

[Ntormousi and Hennebelle, 2019] Ntormousi, E. and Hennebelle, P. (2019). Core and stellar mass functions in massive collapsing filaments. *Astronomy and Astrophysics*, 625 :A82.

[Ohno and Kageyama, 2021] Ohno, N. and Kageyama, A. (2021). In-situ visualization library for Yin-Yang grid simulations. *Earth, Planets and Space*, 73(1) :158.

[Pakmor and Springel, 2013] Pakmor, R. and Springel, V. (2013). Simulations of magnetic fields in isolated disc galaxies. *MNRAS*, 432(1) :176–193.

[Pascucci and Frank, 2001] Pascucci, V. and Frank, R. J. (2001). Global static indexing for real-time exploration of very large regular grids. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, SC '01, page 2, New York, NY, USA. Association for Computing Machinery.

[Passot and Vá zquez-Semadeni, 1998] Passot, T. and Vá zquez-Semadeni, E. (1998). Density probability distribution in one-dimensional polytropic gas dynamics. *Physical Review E*, 58(4) :4501–4510.

[Peano, 1890] Peano, G. (1890). Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36 :157–160.

[Pontzen et al., 2013] Pontzen, A., Roškar, R., Stinson, G., and Woods, R. (2013). pynbody : N-Body/SPH analysis for python. Astrophysics Source Code Library, record ascl :1305.002.

[Rasera and Teyssier, 2006] Rasera, Y. and Teyssier, R. (2006). The history of the baryon budget. Cosmic logistics in a hierarchical universe. *A&A*, 445(1) :1–27.

[Ratanaworabhan et al., 2006] Ratanaworabhan, P., Ke, J., and Burtscher, M. (2006). Fast lossless compression of scientific floating-point data. In *Data Compression Conference (DCC'06)*, pages 133–142.

[Renaud et al., 2014] Renaud, F., Bournaud, F., and Duc, P.-A. (2014). A parsec-resolution simulation of the antennae galaxies : formation of star clusters during the merger. *Monthly Notices of the Royal Astronomical Society*, 446(2) :2038–2054.

[Renaud et al., 2013] Renaud, F., Bournaud, F., Emsellem, E., Elmegreen, B., Teyssier, R., Alves, J., Chapon, D., Combes, F., Dekel, A., Gabor, J., Hennebelle, P., and Kraljic, K. (2013). A sub-parsec resolution simulation of the Milky Way : global structure of the interstellar medium and properties of molecular clouds. *MNRAS*, 436(2) :1836–1851.

[Renaud et al., 2012] Renaud, F., Kraljic, K., and Bournaud, F. (2012). Star Formation Laws and Thresholds from Interstellar Medium Structure and Turbulence. *The Astrophysical Journal Letters*, 760(1) :L16.

[Reynolds, 1895] Reynolds, O. (1895). On the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Proceedings : Mathematical and Physical Sciences*, 451(1941) :5–47.

[Robertson and Kravtsov, 2008] Robertson, B. E. and Kravtsov, A. V. (2008). Molecular Hydrogen and Global Star Formation Relations in Galaxies. *ApJ*, 680(2) :1083–1111.

[Robinson and Cherry, 1967] Robinson, A. and Cherry, C. (1967). Results of a prototype television bandwidth compression scheme. *Proceedings of the IEEE*, 55(3) :356–364.

[Roche and Dubois, 2020] Roche, A. and Dubois, J. (2020). Evaluation of Mesh Compression and GPU Ray Casting for Tree Based AMR data in VTK. In Byška, J. and Jänicke, S., editors, *EuroVis 2020 - Posters*. The Eurographics Association.

[Ross et al., 2008] Ross, R. B., Peterka, T., Shen, H.-W., Hong, Y., Ma, K.-L., Yu, H., and Moreland, K. (2008). Visualization and parallel i/o at extreme scale. *Journal of Physics : Conference Series*, 125(1) :012099.

[Roussel et al., 2017] Roussel, C., Keller, K., Gaalich, M., Bautista Gomez, L., and Bigot, J. (2017). Pdi, an approach to decouple i/o concerns from high-performance simulation codes. working paper or preprint.

[Roy et al., 2013] Roy, N., Kanekar, N., and Chengalur, J. N. (2013). The temperature of the diffuse h i in the milky way - II. gaussian decomposition of the h i-21 cm absorption spectra. *Monthly Notices of the Royal Astronomical Society*, 436(3) :2366–2385.

[Saintonge and Catinella, 2022] Saintonge, A. and Catinella, B. (2022). The cold interstellar medium of galaxies in the local universe. *Annual Review of Astronomy and Astrophysics*, 60(1) :319–361.

[Salomon and Motta, 2009] Salomon, D. and Motta, G. (2009). *Handbook of Data Compression*. Springer Publishing Company, Incorporated, 5th edition.

[Salpeter, 1955] Salpeter, E. E. (1955). The Luminosity Function and Stellar Evolution. *ApJ*, 121 :161.

[Saury et al., 2014] Saury, E., Miville-Deschê nes, M.-A., Hennebelle, P., Audit, E., and Schmidt, W. (2014). The structure of the thermally bistable and turbulent atomic gas in the local interstellar medium. *Astronomy And Astrophysics*, 567 :A16.

[Sedov, 1959] Sedov, L. (1959). Foreword to fourth edition. In Sedov, L., editor, *Similarity and Dimensional Methods in Mechanics*, pages xiii–xiv. Academic Press.

[Sewell et al., 2015] Sewell, C., Heitmann, K., Finkel, H., Zagaris, G., Parete-Koon, S. T., Fasel, P. K., Pope, A., Frontiere, N., Lo, L.-t., Messer, B., Habib, S., and Ahrens, J. (2015). Large-scale compute-intensive analysis via a combined in-situ and co-scheduling workflow approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, New York, NY, USA. Association for Computing Machinery.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3) :379–423.

[Solomon et al., 1997] Solomon, P., Downes, D., Radford, S., and Barrett, J. (1997). The molecular interstellar medium in ultraluminous infrared galaxies. *The Astrophysical Journal*, 478(1) :144.

[Stone et al., 2020] Stone, J. M., Tomida, K., White, C. J., and Felker, K. G. (2020). The Athena++ Adaptive Mesh Refinement Framework : Design and Magnetohydrodynamic Solvers. *ApJS*, 249(1) :4.

[Strafella and Chapon, 2020] Strafella, L. and Chapon, D. (2020). Boosting I/O and visualization for exascale era using Hercule : test case on RAMSES. In *Journal of Physics Conference Series*, volume 1623 of *Journal of Physics Conference Series*, page 012019.

[Strafella and Chapon, 2022] Strafella, L. and Chapon, D. (2022). Lightamr format standard and lossless compression algorithms for adaptive mesh refinement grids : Ramses use case. *Journal of Computational Physics*, 470 :111577.

[Sun et al., 2023] Sun, J., Leroy, A. K., Ostriker, E. C., Meidt, S., Rosolowsky, E., Schinnerer, E., Wilson, C. D., Utomo, D., Belfiore, F., Blanc, G. A., et al. (2023). Star formation laws and efficiencies across 80 nearby galaxies. *The Astrophysical Journal Letters*, 945(2) :L19.

[Tacconi et al., 2013] Tacconi, L., Neri, R., Genzel, R., Combes, F., Bolatto, A., Cooper, M. C., Wuyts, S., Bournaud, F., Burkert, A., Comerford, J., et al. (2013). Phibss : molecular gas content and scaling relations in z 1–3 massive, main-sequence star-forming galaxies. *The Astrophysical Journal*, 768(1) :74.

[Tao et al., 2017] Tao, D., Di, S., Chen, Z., and Cappello, F. (2017). Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE.

[Tao et al., 2019] Tao, D., Di, S., Liang, X., Chen, Z., and Cappello, F. (2019). Optimizing lossy compression rate-distortion from automatic online selection between sz and zfp. *IEEE Transactions on Parallel and Distributed Systems*, 30(8) :1857–1871.

[Teyssier, 2002] Teyssier, R. (2002). Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES. *Astronomy and Astrophysics*, 385 :337–364.

[Teyssier et al., 2010] Teyssier, R., Chapon, D., and Bournaud, F. (2010). The driving mechanism of starbursts in galaxy mergers. *The Astrophysical Journal Letters*, 720(2) :L149.

[The HDF Group, 2023] The HDF Group (1997-2023). Hierarchical Data Format, version 5. https ://www.hdfgroup.org/HDF5/.

[Toomre, 1964] Toomre, A. (1964). On the gravitational stability of a disk of stars. *ApJ*, 139 :1217–1238.

[Trott et al., 2022] Trott, C. R., Lebrun-Grandié, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., Gayatri, R., Harvey, E., Hollman, D. S., Ibanez, D., Liber, N., Madsen, J., Miles, J., Poliakoff, D., Powell, A., Rajamanickam, S., Simberg, M., Sunderland, D., Turcksin, B., and Wilke, J. (2022). Kokkos 3 : Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4) :805–817.

[Truelove et al., 1997] Truelove, J. K., Klein, R. I., McKee, C. F., Holliman, John H., I., Howell, L. H., and Greenough, J. A. (1997). The Jeans Condition : A New Constraint on Spatial Resolution in Simulations of Isothermal Self-gravitational Hydrodynamics. *The Astrophysical Journal Letters*, 489(2) :L179–L183.

[Tumblin et al., 2015] Tumblin, R., Ahrens, P., Hartse, S., and Robey, R. W. (2015). Parallel compact hash algorithms for computational meshes. *SIAM J. Sci. Comput.*, 37(1) :C31–C53.

[Turk et al., 2011] Turk, M. J., Smith, B. D., Oishi, J. S., Skory, S., Skillman, S. W., Abel, T., and Norman, M. L. (2011). yt : A Multi-code Analysis Toolkit for Astrophysical Simulation Data. *ApJS*, 192(1) :9.

[Utomo et al., 2019] Utomo, D., Blitz, L., and Falgarone, E. (2019). The origin of interstellar turbulence in m33. *The Astrophysical Journal*, 871(1) :17.

[van der Holst and Keppens, 2007] van der Holst, B. and Keppens, R. (2007). Hybrid block-amr in cartesian and curvilinear coordinates : Mhd applications. *Journal of Computational Physics*, 226(1) :925–946.

[Vazquez-Semadeni, 1994] Vazquez-Semadeni, E. (1994). Hierarchical Structure in Nearly Pressureless Flows as a Consequence of Self-similar Statistics. *ApJ*, 423 :681.

[Vitter, 2001] Vitter, J. S. (2001). External memory algorithms and data structures : Dealing with massive data. *ACM Comput. Surv.*, 33(2) :209–271.

[Wald et al., 2021] Wald, I., Zellmann, S., Usher, W., Morrical, N., Lang, U., and Pascucci, V. (2021). Ray tracing structured amr data using exabricks. *IEEE Transactions on Visualization and Computer Graphics*, 27(02) :625–634.

[Wang et al., 2020] Wang, F., Marshak, N., Usher, W., Burstedde, C., Knoll, A., Heister, T., and Johson, C. R. (2020). CPU Ray Tracing of Tree-Based Adaptive Mesh Refinement Data. *Computer Graphics Forum*.

[Wautelet and Kestener, 2011] Wautelet, P. and Kestener, P. (2011). Parallel io performance and scalability study on the prace curie supercomputer. In *Proceedings of Partnership for Advanced Computing in Europe (PRACE)*.

[Wells et al., 1981] Wells, D. C., Greisen, E. W., and Harten, R. H. (1981). FITS - a Flexible Image Transport System. *Astronomy and Astrophysics Supplement*, 44 :363.

[Wilkinson et al., 2016] Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., et al. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3.

[Williams et al., 2005] Williams, A., Barrus, S., Morley, R. K., and Shirley, P. (2005). An efficient and robust ray-box intersection algorithm. *ACM SIGGRAPH 2005 Courses*.

[Williams et al., 2000] Williams, J. P., Blitz, L., and McKee, C. F. (2000). The Structure and Evolution of Molecular Clouds : from Clumps to Cores to the IMF. In Mannings, V., Boss, A. P., and Russell, S. S., editors, *Protostars and Planets IV*, page 97.

[Ziv and Lempel, 1977] Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3) :337–343.

[Zwicky, 1933] Zwicky, F. (1933). Die rotverschiebung von extragalaktischen nebeln. *Helvetica Physica Acta, Vol. 6, p. 110-127*, 6 :110–127.

Annexe A

Conversion *LightAMR* vers *vtkHyperTreeGrid*

```
# Written by L. Strafella, sept. 2023
#
# Demonstration of conversion from LightAMR data model [Strafella And Chapon, 2022]
# to vtkHyperTreeGrid [Harel et al., 2017a], [Harel et al., 2017b]
#
# Conversion of lightAMR example of chapter 3 of my phd manuscrit.
#
# Dependencies: vtk, numpy
# Validated version: vtk [9.2.6], numpy [1.24.4]

import time
import numpy
import vtk

def computeFirstChildIndex( info, lightAMR_obj ):
  """ Build indirection array for child index from coarse cell  """

  nchildren = info["fd"]**info["dimension"]
  ncplsum = numpy.zeros( lightAMR_obj["nbElementsPerLevel"].size )
  ncplsum[0] = lightAMR_obj["nbElementsPerLevel"][0]
  for ilvl in range( 1, lightAMR_obj["nbElementsPerLevel"].size ):
    ncplsum[ilvl] = lightAMR_obj["nbElementsPerLevel"][ilvl] + ncplsum[ ilvl - 1 ]

  firstChildIndex = numpy.zeros( lightAMR_obj["tree"].size )

  offset = 0
  for ilvl in range(0, ncplsum.size):
    bufferIndirectArrayFirstChild = 0
    for icell in range( 0, lightAMR_obj["nbElementsPerLevel"][ ilvl ]):
      if lightAMR_obj["tree"][ offset ] == 1 :
        idFirstChild = bufferIndirectArrayFirstChild * nchildren
        idFirstChild += ncplsum[ ilvl ]
        bufferIndirectArrayFirstChild += 1

        firstChildIndex[ offset ] = idFirstChild

      offset += 1

  return firstChildIndex

def recursiveIteration(cursor, lightAMR_Obj, FirstChildIndex, idAMR, field ):
  """
    process one cell of the lightAMR "tree" based on its coarsiness & mask """

  isGhost = False
  if lightAMR_Obj["mask"][idAMR] == 1: # skip masked cells
    return

  # add quantity to cells that belong to current domain
  field.InsertTuple1(cursor.GetGlobalNodeIndex(), lightAMR_Obj["domainID"])

  if lightAMR_Obj["tree"][idAMR] == 1:              # check if cell is coarse
    cursor.SetMask(False)
    firstChildId = FirstChildIndex[ idAMR ]         # get first child index in lightAMR "tree" array
```

```
    if cursor.IsLeaf():                              # subdivide hyperTree node
      cursor.SubdivideLeaf()
      for ison in range(cursor.GetNumberOfChildren()):
        cursor.ToChild(ison)
        cursor.SetMask(True)
        cursor.ToParent()

    for ison in range(cursor.GetNumberOfChildren()): # recursively process leaves of node
      cursor.ToChild(ison)
      recursiveIteration(cursor, lightAMR_Obj, FirstChildIndex, int(firstChildId + ison), field )
      cursor.ToParent()

  else:
    cursor.SetMask( cursor.IsMasked() and isGhost )

def createHyperTreeGrid(info, lightAMR_objs):
  """ Create an vtkHyperTreeGrid from LightAMR object """

  tic = time.perf_counter()

  htg = vtk.vtkHyperTreeGrid()
  htg.Initialize()

  # remember that one root tree is [2,2,2]
  if info["dimension"] == 2:
      htg.SetDimensions([2, 2, 1])
  else:
      htg.SetDimensions([2, 2, 2])

  htg.SetBranchFactor( info["fd"] )
  htg.SetIndexingModeToIJK()

  xValues = vtk.vtkDoubleArray()  # x
  xValues.SetNumberOfValues(2)
  xValues.SetValue(0, 0.0)
  xValues.SetValue(1, 1.0)
  htg.SetXCoordinates(xValues)

  yValues = vtk.vtkDoubleArray()  # y
  yValues.SetNumberOfValues(2)
  yValues.SetValue(0, 0.0)
  yValues.SetValue(1, 1.0)
  htg.SetYCoordinates(yValues)

  mask = vtk.vtkBitArray()
  htg.SetMask( mask )

  if info["dimension"] > 2:
    zValues = vtk.vtkDoubleArray()  # y
    zValues.SetNumberOfValues(2)
    zValues.SetValue(0, 0.0)
    zValues.SetValue(1, 1.0)
    htg.SetZCoordinates(zValues)

  cellDomainArray = vtk.vtkUnsignedCharArray()
  cellDomainArray.SetName("domain")
  cellDomainArray.SetNumberOfTuples(0)

  crtIndex = 0
  domain_done = 0
  cursor = vtk.vtkHyperTreeGridNonOrientedCursor()
  create_cursor = True
  for domID in lightAMR_objs.keys():

    print("\t> Building htg for domain: {}".format(domID))

    firstChildIndexes = computeFirstChildIndex( info, lightAMR_objs[domID] )
    if domain_done > 0:
      create_cursor = False

    htg.InitializeNonOrientedCursor(cursor, crtIndex, create_cursor)
```

```
      cursor.SetGlobalIndexStart( crtIndex )

      recursiveIteration( cursor, lightAMR_objs[domID], firstChildIndexes, 0, cellDomainArray )

      domain_done += 1

    tac = time.perf_counter()

    print("\t> Build HTG, %d root, time spent : %10.6f " % (htg.GetMaxNumberOfTrees(), tac - tic))

    htg.GetCellData().AddArray( cellDomainArray )

    return htg

if __name__ == "__main__":

    # LightAMR data
    info = { "dimension":2, "ndomains":3, "fd":2, "levelmax":5 }
    meshes = {0:{"nlevel":4, "ncells":25, "nbElementsPerLevel":numpy.array( [1,4,8,12] ), "domainID":0,
                 "tree":numpy.array( (1,1,0,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0), dtype=numpy.int8),
                 "mask":numpy.array( (0,0,1,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0), dtype=numpy.int8),
               },
             1:{"nlevel":4, "ncells":25, "nbElementsPerLevel":numpy.array( [1,4,8,12] ), "domainID":1,
                 "tree":numpy.array( (1,0,0,1,1,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0), dtype=numpy.int8),
                 "mask":numpy.array( (0,1,1,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0), dtype=numpy.int8),
               },
             2:{"nlevel":4, "ncells":17, "nbElementsPerLevel":numpy.array( [1,4,8,4] ), "domainID":2,
                 "tree":numpy.array( (1,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,), dtype=numpy.int8),
                 "mask":numpy.array( (0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0), dtype=numpy.int8),
               },
            }
    # build vtkHyperTreeGrid
    htg = createHyperTreeGrid( info, meshes )

    # export to file
    htg_file = "example_2D.htg"
    print( "\t> Export vtkHyperTreeGrid to .htf file: {}".format(htg_file) )
    writer = vtk.vtkXMLHyperTreeGridWriter()
    writer.SetDataSetMajorVersion(2)
    writer.SetFileName( htg_file )
    writer.SetInputDataObject( htg )
    writer.SetCompressorTypeToNone()
    writer.Write()
```

Annexe B

Published *LightAMR* data model

LightAMR format standard and lossless compression algorithms for adaptive mesh refinement grids: RAMSES use case

L.Strafella[a,b], D.Chapon[b]

[a]AIM, CEA, CNRS, Université Paris-Saclay, 91191 Gif-sur-Yvette, France
[b]IRFU, CEA, Université Paris-Saclay, 91191 Gif-sur-Yvette, France

Abstract

The evolution of parallel I/O library as well as new concepts such as 'in transit' and 'in situ' visualization and analysis have been identified as key technologies to circumvent I/O bottleneck in pre-exascale applications. Nevertheless, data structure and data format can also be improved for both reducing I/O volume and improving data interoperability between data producer and data consumer. In this paper, we propose a very lightweight and purpose-specific post-processing data model for AMR meshes, called *lightAMR*. Based on this data model, we introduce a tree pruning algorithm that removes data redundancy from a fully threaded AMR octree. In addition, we present two lossless compression algorithms, one for the AMR grid structure description and one for AMR double/single precision physical quantity scalar fields. Then we present performance benchmarks on RAMSES simulation datasets of this new lightAMR data model and the pruning and compression algorithms. We show that our pruning algorithm can reduce the total number of cells from RAMSES AMR datasets by 10-40% without loss of information. Finally, we show that the RAMSES AMR grid structure can be compacted by ~ 3 orders of magnitude and the float scalar fields can be compressed by a factor ~ 1.2 for double precision and $\sim 1.3 - 1.5$ in single precision with a compression speed of ~ 1 GB/s.

Keywords: computational astrophysics, Adaptive Mesh Refinement, lossless compression, data model

1. Introduction

RAMSES is a massively parallel hydrodynamical code for self-gravitating magnetized flows widely used in the computational astrophysics community [1], to solve the evolution of dark matter, stellar populations, and gas via gravity, hydrodynamics, radiative transfer, and non-equilibrium radiative cooling/heating. For hydrodynamics, it uses the HLLC Riemann solver [2] and the MinMod slope limiter to construct gas variables at cell interfaces from their cell-centered values. The dynamics of collision-less dark matter and star particles are evolved with a multi-grid particle-mesh solver and cloud-in-cell interpolation [3].

It implements an adaptative mesh refinement (AMR) technique to optimize the memory/precision ratio in numerical simulations, [4], [5]. Its data structure is a cell-based fully threaded octree [6], which means that each subdomain describes a full octree of the entire simulation domain up to its topmost root grid which has the size of the whole simulation domain. In this distributed AMR tree approach each subdomain defines its own local grid refinement. In most use cases, the domain decomposition is based on a 1D Hilbert curve to split the calculation between MPI processes. It was first designed for cosmological and galaxy numerical simulation, and was used for large simulation [7], [8], [9]. However, RAMSES faces I/O scalability bottlenecks when used over a few thousand of MPI processes, partly due to its "multiple file per process" I/O strategy. A typical cosmological simulation using 10,000 MPI processes will produce at least 40,000 files (and filesystem inodes) per output with small file size from dozens of megabytes to a few hundred megabytes, which is far from optimal on modern Lustre filesystems and generally rapidly reach the user's limits on supercomputer. What is more, since RAMSES only implements one output data format, for both checkpointing and post-processing purposes, it is not optimized for both cases. Therefore, a simulation like Extreme-Horizon [9] produced 3.6 terabytes of data with more than 50,000 files per snapshot (AMR files, hydrodynamical quantity files, particles files, etc). Considering that several snapshots are required for scientific analysis that kind of simulation becomes rapidly challenging in terms of I/O, data management and post-processing.

In a previous work, the integration of the the Hercule library [10] for parallel I/O and data management in RAMSES was the first step to improve the I/O performance, scalability and data management [11]. In terms of data management, the integration of a parallel I/O library was also the occasion to add a post-processing specific data-flow to RAMSES. This was an important step which permitted to optimize significantly the checkpoint dump since the checkpoint's raw data will no longer be used by post-processing tools for simulation data analysis. With the original version of RAMSES, sometimes the size of the simulation is calculated not based on CPU performance, memory consumption, or pure scalability of the code but simply based on the user's disk storage available capacity. We expect

Email addresses: loic.strafella@cea.fr (L.Strafella), damien.chapon@cea.fr (D.Chapon)

this problem to occur more and more often in the coming years. The usually adopted solution is either to reduce the size of the simulation, the precision of the simulation by tuning AMR parameters or reducing the number of produced snapshots. The question was then is there a standard and compact data model that can be used to describe the AMR data from RAMSES new post-processing data-flow ?

Hercule library's post-processing database strategy is to propose an XML dictionary containing the description of standardized data model for different kind of meshing strategies and objects. Therefore, once a standardized data model is described in the dictionary, every tool using Hercule will be able to "understand" Hercule post-processing data. As a reminder, we show in table 1 a small recap of I/O data-flows now available in RAMSES.

I/O library	Posix	Hercule	Hercule
Purpose	C/R+ P-P	C/R	P-P
Data Structure	Specific	Free	Standardized
Data Volume	Large	Large	Light
Compression	N/A	N/A	Efficient

Table 1: Difference between various RAMSES I/O formats : the legacy binary posix and the new Hercule database HProt, HDep. C/R: Checkpoint/Restart, P-P: Post-processing.

Until a few years ago, AMR data did lack of a standardized format for post-processing and visualization purposes partly due to the different flavor of AMR strategies: block-based / patched-based [12], [13], [14], cell-based (RAMSES), etc. Thus AMR data still required some data transformation step (e. g. down-casting to unstructured grids, resampling to fixed resolution cartesian grids) to be manipulated by general purpose post-processing or visualization tools (e. g. Paraview [1], VisIt [2]). An AMR mesh described using unstructured grids can easily lead to memory issues and produce high volume data files, thus each AMR code uses a custom data model to output the data. In the case of RAMSES, the AMR internal 32 bits integer linked-lists are dumped level by level. Code-specific data-processing libraries were then implemented as solutions to read the RAMSES datafile format and handle its AMR grid structures (e. g. PyMSES[3], Osiris [4]) but these libraries needed to be updated to follow the RAMSES data format evolutions [15]. A standardized and self-consistent data model aims at reducing the cost of post-processing tools maintenance and encourage sharing development efforts.

In this work, we introduce a new lightweight AMR data model that we applied on RAMSES AMR data. By using this data model, we show that data volume of RAMSES outputs can be significantly reduced, without loss of information . In addition, we present two lossless data compression algorithms that can be built upon this new data model standard to further reduce both the size of the AMR grid refinement description and the size of associated physical quantity scalar field data (float 32/64 bits). In the results section, we present lightAMR data volume reduction as well as tree pruning and compression benchmarks on published RAMSES simulation datasets.

[1] Paraview.org
[2] VisIt home page
[3] http://irfu.cea.fr/Projets/PYMSES
[4] https://www.nbi.dk/ nvaytet/osiris/osiris.html

2. Scientific analysis-specific AMR light tree data model

2.1. Motivation

The RAMSES checkpoint/restart data format was designed, as in many other simulation codes used in astrophysics, to hold all the information necessary for the code to restart a simulation run and resume the dynamical evolution of the numerical model. Since our goal is to design a new output data format for the specific purpose of scientific analysis and post-processing, we are free to select from all the information available in memory at runtime what is strictly necessary to answer one's post-processing need.

To fit one's post-processing needs, a user can choose to export a subset of physical quantities (e. g. store only the density and thermal pressure scalar fields defined on the AMR grid, and discard the gravitational potential/acceleration AMR fields, the velocity AMR vector field and the Lagrangian particle information), hence reducing the overall dataset volume on disk. In RAMSES, this user-defined data field subset selection can be configured at runtime using the input parameter namelist, without the need to recompile the code. What is more, for some data post-processing use cases, down-casting these quantities from double to single precision could even prove sufficient, reducing further the data volume by a factor of 2. Finally, since this new data format is not meant to be read by RAMSES itself upon restart but only by data post-processing tools, we can decide to adopt a new standardized AMR grid data structure and a custom encoding to be able to answer any type of post-processing software requirement in an optimized and standardized way.

2.2. Implementation

The Hercule I/O library HDep standardized data model for AMR trees aims to offer a light and self-consistent way to describe an AMR grid. In this work, we adopt a similar approach as the one proposed in the HDep format, the details of our approach is described here. We further reference this standardized data model for AMR grids as *lightAMR*. Since that kind of mesh can be expressed in the form of a tree, it uses a boolean description of the AMR tree from top to bottom, level by level, as shown on figure 1. With this breadth first traversal description of an AMR tree, two main arrays are used: one for the grid refinement description and one for the cell ownership mask. The first array describes the refinement state of a cell: 1 for a refined cell and 0 for a leaf (unrefined) cell. All child cells from a refined one must be described, as shown in 1. This implies that the number of cells at a level l must be equal to the number of refined cells at level $l-1$ times the refinement factor to the power of the dimension. In our 2D example, the refinement factor being 2 by dimension, the level 2 must contain 3×2^2

boolean values since there are 3 refined cells at level 1. The value in the ownership mask array describes for each cell if it does not belong to the current domain (according to the domain decomposition policy): 0 if it belongs to the current domain, otherwise 1.

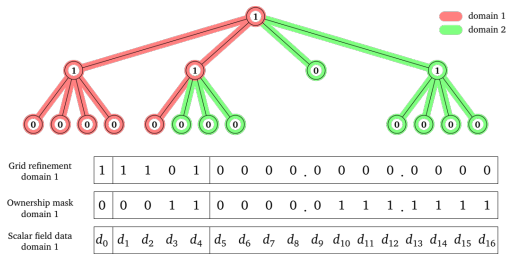

Figure 1: Description of a 2D AMR grid (with a refinement factor of 2) dispatched on two MPI processes according to the lightAMR standardized data model (above). Grid refinement, ownership mask and scalar data field arrays for the domain #1 (below).

In order to be self-consistent, a few other (lightweight) meta-information such as the size of the simulation box, the physical quantity units, physical model details or used numerical schemes need to be stored, using an explicit semantic (string key-value pairs). In addition, some useful meta-information like the number of cells per level, the total number of grids, the number of levels, if somewhat redundant with the AMR grid description, can be conveniently stored to enhance the post-processing tools performance. AMR-bound physical quantities are described in scalar field data arrays that follow the same ordering as the grid refinement array, hence providing a straightforward mapping (which facilitates the LOD approach). Since RAMSES describes the AMR tree from the root node which is the simulation box itself on each domain and stores scalar field values even for refined (coarse) cells, the construction of these arrays is straightforward when converting in the lightAMR format. Of course, this lightAMR tree data format can be easily extended to be compatible with other numerical codes that do not share RAMSES fully-threaded tree data structure, refined cell (coarse) scalar field values only requiring to be computed on-the-fly upon data export.

Collision-less particles are treated in a Lagrangian way within RAMSES and are linked to the AMR grid. The particles and their associated properties are simply added to our new data output format in the form of one dimensional arrays. Particle positions for example are in the form $x_1, y_1, z_1, ..., x_n, y_n, z_n$. We chose not to associate particles to their linked AMR cells and consequently not overload the data format. The particle-cell binding is left to the post-processing tools to be performed on-the-fly if needed.

For simulation codes that do not use the fully-threaded approach and naturally describe collections of trees at a defined minimum level l_0 on simulation domains, the LightAMR data model can be trivially extended. One just needs to provide an additional AMR root description array to detail the I-J-K logical position indices of the root coarse cells of those trees in a structured grid at level l_0. While providing an array containing the logical position in $(i_1, j_1, k_1, i_2, j_2, k_2, ...)$ format would be the natural approach, one may choose to provide an array of Morton indices of logical position at the level l_0. Indeed, using the Morton curve, 3D/2D logical coordinates can be mapped on 1D Morton curve by bits interleaving, hence reducing the size of the array by a factor 3 (resp. 2) in 3D (resp. 2D) but with a limitation on the maximum value of l_0 32 bits Morton indices: $l_{0,\max} = 10$ in 3D ($l_{0,\max} = 16$ in 2D). Within this extension, compared to their fully-threaded tree version, the grid refinement, ownership mask and scalar field data arrays should be trimmed up to level l_0 as a consequence.

3. Tree pruning to remove redundancy

Due to the fully threaded octree approach in RAMSES, some AMR cells may be described several times in the trees of different domains, leading to data redundancy. Even if those multiple AMR cell descriptions are required by a numerical solver like the Poisson equation multi-grid solver [3], describing those cells only once is sufficient for post-processing and visualization purposes. Therefore, a tree pruning algorithm can be implemented to reduce the memory footprint of our file format.

Once the tree is built according to lightAMR format described above, most of the data redundancy in the AMR dataset can be removed. We developed a tree pruning algorithm, propagating from the bottom of the tree to its top, in order to remove the redundant cells. The unnecessary cells are removed by changing their refinement value (C.R.V on figure 2) and removing their described children cells on the next level. The C.R.V. is done only for a refined cell if all its children belong to another domain (mask = 1), as shown on figure 2.

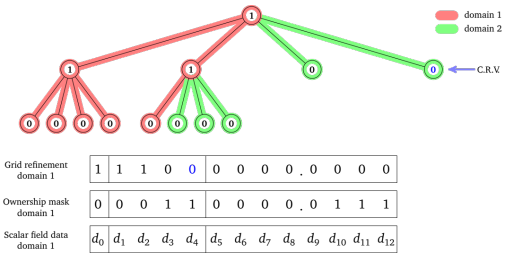

Figure 2: Applying a tree pruning algorithm on the AMR tree of figure 1 for domain #1 (red), and Change of Refinement Value (C.R.V) for an unnecessary refined cell for the domain 1 . Notice the change of value in the grid refinement array in blue, as well as the propagation of this change on the ownership mask and scalar field data arrays (truncation of the arrays after d_{12}).

Then those changes are propagated to the associated arrays: the ownership mask and scalar field data arrays. Therefore, removing those refined cells and its children is the first step to reduce the total volume of data before applying any compression

algorithm. Depending on the simulation, this step can prune a significant part of any RAMSES simulation dataset, see results on astrophysical simulation datasets in section 5.1. Prior to the tree pruning step, a temporary mask array needs to be adapted from the original ownership mask array to properly compute the C.R.V. This slight adaptation insures that any refined cell is not flagged as masked if at least one of its children is not masked. This temporary mask is only used during the tree pruning step .

At the end of this step, the AMR grid is properly described with one or several trees and most of the data redundancy was removed. However, for a fully-threaded octree the redundancy cannot be entirely removed unless the tree is split at the coarse level of refinement (a.k.a. *levelmin* in RAMSES) and domain boundaries set between sibling cells at that coarse level (and not deeper in the tree). The next stage in the lightAMR formatting is to apply data compression on both the grid refinement/ownership mask arrays and the simple/double precision associated scalar field data arrays.

4. AMR data compression

To overcome the I/O bottleneck, in addition to the integration of parallel I/O [11] and a data reduction strategy, data compression can reduce the volume of data to transfer or store on disk/object store and further improve the overall I/O performance. Combining those different strategies can have a significant impact on the I/O times, data volume storage and data transfer rates, whether for in-transit or in-situ data post-processing. In this section, we present two lossless compression schemes specific to octree AMR meshes that we developed for the lightAMR data model; one for the grid description boolean valued arrays and one for the scalar field float (32/64 bits) data arrays.

4.1. LightAMR grid boolean arrays lossless compression

Once the lightAMR structure has been described with boolean valued arrays, we developed a deterministic, lossless and performant algorithm to compress the information contained in the two main arrays (grid refinement and ownership mask arrays) based on a custom run-length encoding which is well known compression technique [16] . The first step of the algorithm is to consider the boolean array as an ordered list of packs of identical bit values, either successive zeroes or successive ones, and count the lengths of these continuous packs. The second step is to digitize these ordered pack sizes with a special encoding and store the result in a single byte integer array. Out of the 256 integer values available, our encoding only need values between 1 and 62. To encode pack sizes, we use a base-52 decomposition and we also record the number of digits (only if greater than 1) in that decomposition. The uint8 values ranging from 2 to 7 are used to store the number of bytes used for the encoding of a each pack size, and the 52 uint8 values between 11 and 62 (values 9 and 10 are used for specific purposes, see next paragraph) to store the digits of the base-52 decomposition (with an additional shift of 11). For example, to encode a pack size of 2904 which decompose in $1 \cdot 52^2 + 3 \cdot 52^1 + 44 \cdot 52^0$, the number of digit (3) is stored, followed by the digits $[1, 3, 44]$, encoded as $[12, 14, 55]$ ($[1 + 11, 3 + 11, 44 + 11]$), resulting in the following 4 bytes $[3, 12, 14, 55]$ instead of the 262 bytes required to store 2904 zeroes or ones.

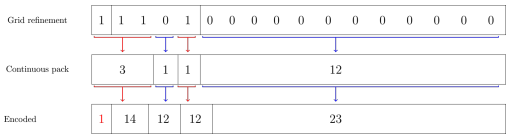

Figure 3: AMR grid refinement boolean array encoding in base 52 after counting the sizes of continuous packs of successive 0 and 1. The bit value of the first pack is encoded in red.

With that encoding, the greatest pack size that can be encoded is $52^7 - 1 \sim 10^{12}$ continuous 0 or 1, which is more than enough, the maximum number of cells per domain in a typical RAMSES simulation run never exceeding a 10^7 due to memory limitations. Since the encoded pack sizes describe identical bit values, alternatively contiguous zeroes and contiguous ones, only the first value of the array needs to define the bit value of the first pack (see the red value in figure 3), and all subsequent bit values of the following packs can be determined by the decompression algorithm.

To give the possibility to data-processing tools to improve their memory footprint, we included special values in this encoding to mark the boundaries of AMR levels so that data-processing tools can perform on-the-fly decompression on a *level-by-level* basis for level-of-details approach. We used the 9 and 10 uint8 values as level boundary markers, two different values being required to mark level boundaries in case a bit flip occur (or not) at the end of an AMR level (see orange values in figure 4). In addition, we developed a special algorithm which is able to directly process the compressed stream to get the value at a given offset in the uncompressed stream without the need of uncompressing the entire AMR compressed array.

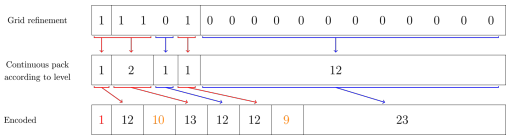

Figure 4: Encoding of AMR description array by counting the size of continuous pack of 0 and 1 according to the number of cells by level, and adding markers 10 or 9 according to a potential bit value flip.

We further reference this LightAMR grid boolean arrays lossless compression as base 52 continuous pack size encoding (CPS52). While this CPS52 encoding may seem peculiar compared to the majority of other RLE schemes from the literature, it yields very good performance. In the process of LightAMR data format standardisation though, this custom RLE would require to be updated to match more closely standard encoding schemes, prior to the LightAMR format standard release. Once the AMR grid description arrays are compressed, the next step

in order to reduce the data volume is to compress the float (simple/double precision) scalar field data arrays.

4.2. LightAMR scalar field float data lossless compression

The development of our custom compression algorithm was motivated by the poor compression rates obtained with the deflate algorithm from the zlib [5], about 3 to 5% with a compression speed about 50 MB/s. Even if higher compression rates could be achieved using the LZMA algorithm, it would require far more memory for a lower compression speed, such memory-hungry approach would not be viable in memory-bound massively parallel simulations as are many RAMSES runs. In addition, compression of float valued arrays using entropic encoding without prior knowledge of the data generally leads to poor performance due to its intrinsic homogeneous byte distribution unlike for example compressing a text where the distribution of byte (letters) is not homogeneous and therefore entropic compression is highly efficient.

In this section we present an adaptation of delta compression algorithm for lossless float data [17], [18] that we specifically designed for lightAMR octree structures. Delta compression is a commonly used compression method based on prediction functions. Indeed, this method is based on the choice of a mathematical predictor function and encode directly the prediction error. The compression process itself is performed by removing the leading zeroes in the encoding of the prediction error [17].

With this kind of approach, high compression ratios are reached when the predictor function is quite accurate, which is not easy to achieve when dealing with an AMR octree structure. Thus, we extended this delta compression algorithm to AMR grids by taking advantage of the fully threaded approach of the AMR strategy in RAMSES.

The only information our compression algorithm needs is the grid refinement array and a physical quantity field to compress (simple or double precision float). We propose to use an AMR-specific mathematical function to predict intensive variable scalar field values called the *Parent-Child predictor* (PCP) function. Indeed, the value of the parent cell is used as the predicted value for its child cells and so on from the top of the AMR tree down to the bottom, as represented on figure 5. The compression is then made octant by octant (if 3 dimensions are used). There are two mains steps in the data compression process: first, the delta compression itself with the leading zeroes removal and second, the encoding of the compressed scalar field on a bit-field .

As for the delta compression stage, the single (or double) precision parent float value of the physical field is mapped on unsigned integer of the same size (8 bytes unsigned integer for a double precision float and 4 bytes for a single precision) as well as the values of the children cells. Then we use an XOR

[5] https://zlib.net/

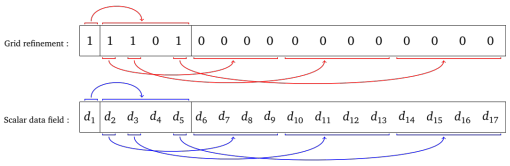

Figure 5: Parent-Child predictor : the parent cell value is used to predict the child cell value in order to achieve delta compression.

operator to compute the differences between the mapped parent value and all the children ones, leading to a pack of prediction errors (s_1, s_2, s_3, s_4 in the 2D example shown on figure 6). The next step is to compute an OR operation between all the s_i values in order to determine how many leading zeroes could be removed on each s_i value at most, which corresponds to the stage $s_0+s_1+s_2+s_3$ in figure 6. In the example, we can remove a maximum of 5 leading zeroes on each s_i value without losing information.

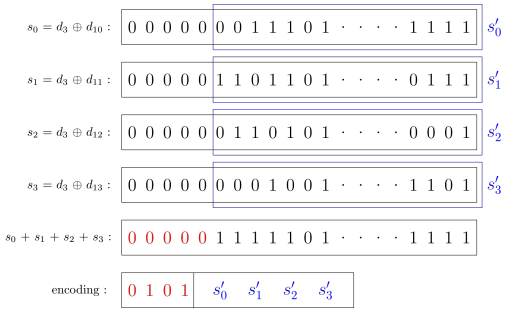

Figure 6: XOR operation (sign $\oplus$) between the parent cell value $d3$ and its children values $d_{10}, d_{11}, d_{12}, d_{13}$, and the OR (sign +) operation between s_0, s_1, s_2, s_3 in order to compute the number of removable leading zeroes, in red. Residues of s_i values are noted s'_i.

At the encoding stage, we need to encode the residues of the prediction errors of all children cell values as well as the number of removed leading zeroes. On figure 7 we show the probability mass function (PMF) of the number of leading zeroes on cell basis and on pack basis on the double precision density scalar field of the Extreme-Horizon simulation (12800 density datasets). Since residue bits are close to being uniformly random, we cannot compress them well. If we compute the compression ratio in this example, without adding the encoding of the number of removed leading zeroes (nLZ), we obtain 1.303 on cell-by-cell basis and 1.237 on pack basis which is expected since more bits can be remove in the first scenario. Nevertheless, the number of removed leading zeroes for each cell or each pack needs to be encoded for the decompression stage. The shannon's entropy of the nLZ field on a cell-by-cell basis is 3.5 and 3.2 on a pack basis, this smaller value being also an expected result since we smooth the nLZ value in each pack. Nevertheless, the total number of nLZ values to encode in

the cell-by-cell case is, in 3D, 8 times the total number of nLZ values to encode in the pack case. Therefore, even by adding another stage of compression of the nLZ values, we cannot hope to achieve a better compression ratio even if more bits can be removed from the data. If we take into account the encoding of both the nLZ values and the residues, the compression ratio drops to 1.217 for the the cell-by-cell case, while it only drops to 1.225 for the pack case. What is more, the cell-by-cell approach would require 8 times more computations of leading zeroes and an additional compression step to barely achieve a similar compression ratio. Thus, we chose to compute and encode the nLZ value once per pack of children as shown at the encoding stage on figure 6.

After computing the PMF of scalar field data, we notice that setting the number of removed leading zeroes on 4 bits by default is a good balance between compression speed and compression ratio. In the example of the Extreme-Horizon density field, increasing to 5 bits will only increase the compression ratio by 0.23 % but reduce the compression speed by almost 8%. Indeed, the nLZ computation is a key point of the algorithm performance and it requires more if-branches to compute a higher value. Finally, after the 4 bits used to encode the nLZ value, all residues $s_i^{'}$ are encoded, see figure 6.

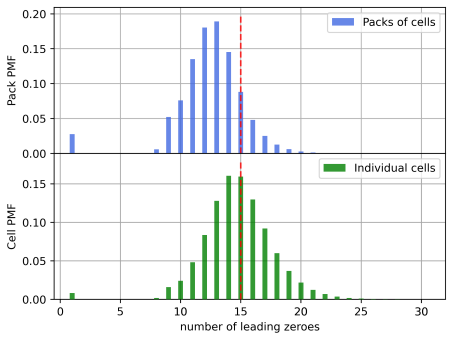

Figure 7: Histograms of number of removable leading zeroes (nLZ) on pack basis (top) and cell basis (bottom) on the double precision density field of Extreme-Horizon. In red dashed line is the threshold of 4 bits for the PCP4 encoding.

According to the number of bits used for the number of removed leading zeroes, the theoretical maximum compression ratio in dimension d is given by (E_{enc} is the number of bits, 64 for double precision and 32 for single precision):

$$C_{max}^{d} = \frac{E_{enc} \cdot 2^{d}}{(E_{enc} - 15) \cdot 2^{d} + 4} \tag{1}$$

Therefore, the maximum compression ratio achievable for single precision data is 1.829 and 1.293 for double precision with the default encoding. Nevertheless, we could expect this method to yield poor performance in the case of a difference of sign between a parent cell value and one of its child cells. Indeed, the very first bit of a negative value will be set to one thus no leading zeroes will be removed. This could be the case for example with a small velocity field component varying around zero. Finally, our delta compression algorithm works for intensive variables but could be extended to extensive variables if the predicted values for the child cells were taken as the value of the parent cell divided by the numbers of child cells. If the number of child cells was a multiple of 2, the mantissa of the prediction error would not be modified (only the exponent part would be bitwise right-shifted), which could still lead to a lossless compression algorithm. The RAMSES code using only intensive variable AMR scalar fields, this extension of the Parent-Child predictor function is beyond the scope of this work.

For the float data decompression stage, the AMR grid refinement array is used level-by-level to guide the delta decompression algorithm through the grid from top to bottom since the scalar fields values at level l are necessary as predictor values to decompress scalar field values at level $l + 1$.

5. Results and discussion

In this section we present the results of the lightAMR format conversion as well as tree pruning and compression results obtained on different real simulation snapshots : FRIG[6], snapshot at 10.05 Myr and 1675 coarse steps, [19] composed of 4096 domains, ORION[7] snapshot at 1.26 Myr and 2125 coarse steps, [20] composed of 512 domains and Extreme-Horizon[8], snapshot at $Z \sim 1$ with 11866 coarse steps, [9], composed of 12800 domains. Since, post-processing tools handle RAMSES data on a per-domain basis and that lightAMR is a self-consistent data model also on a per-domain basis, we consider a dataset as the data of one domain . For the tree pruning algorithm we show the minimum, maximum and average value for different metrics that are computed on a per-domain basis.

5.1. Tree pruning results

The table 2 gives the results of the tree pruning applied on the different datasets. The algorithm is 2.3 to 3.4 times more efficient on FRIG and ORION data than on Extreme-Horizon. FRIG and ORION runs are zoom simulations (nested grids with increasing spatial resolution) where most of the fine resolution grids can be found in the center or mid-plane of the simulation box, whereas Extreme-Horizon is a cosmological simulation where finest AMR levels are more homogeneously distributed in the box. These results are consistent with the fact that RAMSES is more memory efficient in cosmological runs than in zoom simulations. Nevertheless, the tree pruning gain is applied on the whole simulation data therefore, 11.42% on Extreme-Horizon means that the size is reduced by about 240 Gb. Moreover, tree pruning is done in a memory friendly way

[6] http://www.galactica-simulations.eu/db/STAR_FORM/FRIG
[7] http://www.galactica-simulations.eu/db/STAR_FORM/ORION
[8] https://www.galactica-simulations.eu/db/COSMOLOGY/EXTREME_HORIZONS

within a negligible time (a few microseconds on the data of an MPI process) . Therefore this step has a significant impact on data volume at almost no cost. Actually, the time spent doing tree pruning is correlated with the number of coarse cells and the position of those coarse cells in the array.

The efficiency of the tree pruning algorithm is far from intuitive in most cases (at least with RAMSES AMR grids). It depends on the maximum level of refinement, on the number of domains, on the geometrical convexity of the subdomains, on their surface to volume ratio, and is greatly impacted by the 2:1 balance constraint in RAMSES required by its internal (hydro) numerical schemes. The tree pruning rates can span a wide range within a factor 3.25, even for different domains of the same simulation snapshot (see table 2).

Simulation	Min	Max	Global
FRIG	20.05 %	65.42 %	38.65 %
ORION	17.23 %	47.32 %	26.35 %
ExH	7.00 %	23.24 %	11.42 %

Table 2: Fraction of removed cells on average by the tree pruning algorithm on Frig, Orion and Extreme-Horizon collections of datasets . Minimum and maximum values are on domain (dataset) basis and global value is for the entire simulation snapshot.

5.2. LightAMR grid boolean arrays compression results

We compared our compression speeds and ratios to commonly used and open source libraries : Zlib [9], LZ4 [10] and Snappy[11]. The compression ratio is simply the ratio between the uncompressed and the compressed buffer sizes. We run this compression benchmark on the Extreme-Horizon datasets and compare the compression speed and ratio of the cumulated grid refinement and ownership mask arrays after tree pruning, see table 3. Domain datasets are processed sequentially, the uncompressed and compressed sizes are accumulated as well as the times to compress them. The compression speeds and ratios are then computed and reported in table 3. While our single threaded algorithm is slower than the LZ4 package, we are able to achieve a higher compression ratio. The zlib at level 9 (best compression ratio) achieves a much higher compression ratio with the grid refinement array but at the cost of a significantly reduced speed. The compression speed of all those algorithms are linked to the layout of the uncompressed array, i.e. consecutive pack of zeroes or ones. With the RAMSES current parallelism (one MPI process per core) we barely have to compress arrays greater than a few millions of cells, even in the Extreme-Horizon use case. Therefore, a compression speed of 374 Mb/s is enough to compress the entire lightAMR arrays of the most populated MPI process of the Extreme-Horizon dataset in less than 8 ms (domain 427 with 3059513 cells after tree pruning).

[9] Zlib website
[10] LZ4 website
[11] Snappy

	grid refinement		Ownership mask	
library	Ratio	Speed (MB/s)	Ratio	Speed (MB/s)
CPS52	23.95	373.58	**11546.10**	835.47
LZ4	17.94	**1255.15**	249.49	**6864.82**
Zlib 1	26.00	233.93	225.47	289.31
Zlib 9	**48.13**	7.46	973.34	155.22
Snappy	12.00	918.88	21.29	1350.23

Table 3: Benchmark of compression of the lightAMR grid refinement and ownership mask arrays of the entire Extreme-Horizon snapshot using different libraries CPS52 (ours), the zlib version 1.2, the LZ4 package version 1.9.3 and Snappy 1.1.7. Benchmark run on a machine with a Intel Xeon Gold 5118 CPU @ 2.30 GHz.

	grid refinement		Ownership mask	
Simulation	Ratio	Speed (MB/s)	Ratio	Speed (MB/s)
FRIG	39.36	1080.88	1627.19	1447.18
ORION	48.06	1216.33	4397.45	1750.51
ExH	23.95	375.58	11546.10	835.47

Table 4: Compression ratios and speeds of lightAMR boolean arrays for FRIG, ORION and Extreme-Horizon collections of datasets using CPS52 algorithm. Benchmark run on a machine with a Intel Xeon Gold 5118 CPU @ 2.30 GHz.

The compression ratios and speeds are much higher with the ownership mask array because it contains very large packs of successive zeroes (cells that belong to the domain). This is due to the fact that boundaries generally happen on the border of a domain but also because the tree pruning algorithm removes ghost nodes. In addition, our algorithm performs much better at compressing such an array than the other tested libraries because of encoding large packs in base 52 produce a very small pattern.

One can notice that our encoding values are stored on 1 byte even if there are only ranging from 1 to 63, and therefore would require only 6 bits thus a potential gain of 25% in compressed size. In addition to that, computing the average Shannon entropy on 1 byte 'word' of all the AMR description array on an entire simulation like Extreme-Horizon leads to a value of 4. Which means that on average only 4 bits could be used to encode the information contained in one 'word'. The reason is simple, this is because all the number from 1 to 63 are not necessaryly used for the encoding. Actually, by computing exactly the number of bits that would be required to strictly encode bit to bit the information leads to a potential gain on average of 45% on Extreme-Horizon and thus to reduce the size on disk of the AMR data by 45% to less than 420 MB. We did not implemented such an optimization because it adds a step that will increase the time for compression for an AMR structure that is no longer significant compared to the size of the physical fields. And it will increase the complexity to rapidly retrieve AMR level seeds but also the direct access of cells states in the compressed data. Finally, in the case of Extreme-Horizon datasets, the AMR part of the data is now only representing 0.05% of the total simulation snapshot volume. Table 5 compare the com-

pressed lightAMR format to the legacy RAMSES AMR format.

One important remark is that most data arrays stored in a RAMSES checkpoint are no longer stored in this new lightAMR format, without any loss of information on the AMR grid structure itself. The large compression ratios presented in table 5 are mainly due to the switch to a boolean array description and the omission of this redundant meta-information, (e. g. neighbour cell indices, cell centers, parent cell indices) which can be easily and efficiently reconstructed on-the-fly by post-processing tools. The tree pruning step and the CPS52 encoding further increase this AMR file size compression ratio although in a less significant way.

Simulation	Ratio	File size (MB)
FRIG	1485.0	20
ORION	1500.0	6
ExH	583.3	840

Table 5: Chained ratio on RAMSES collections of datasets of tree pruning and CPS52 (left) and resulting lightAMR file size for the AMR grid description (right).

5.3. LightAMR scalar field float data compression results

Most of the data volume of an AMR simulation dataset is due to scalar fields represented as floating point data. The lossless compression algorithm takes the grid refinement array and an associated scalar field float data array as input, both after being processed by the tree pruning algorithm. We used a maximum number of removed leading zeroes of 15 (4 bits encoding) and the compression is single threaded. First, we compare in table 6 the compression performances of the different libraries on the whole density field (double precison, 12800 datasets) from the Extreme-Horizon datasets . We also integrate in the benchmark the Zfp library [21], which is more specific to floating point data compression. We show that our algorithm achieve a higher compression ratio on this dataset due to the use of the topology information from the AMR mesh. It was expected that dictionary compression approach used in LZ4, Snappy and Zlib cannot efficiently compress floating point data. Nevertheless, since the Zlib also use an entropy encoded, it is able to achieve a few percent of compression but with very slow speed compared to our algorithm.

On Extreme-Horizon, a single scalar quantity over one snapshot represents about $\simeq$ 160 Gb (after tree pruning) thus a compression of 1.23 on the density field means a gain of about 30 Gb on disk. In table 7 we show the result of the PCP4 algorithm on the density, x-axis velocity, and thermal pressure for the different RAMSES datasets. We achieve a good compression ratio on the three tested scalar fields of about 1.19 in the worst case scenario and 1.23 on the best case. The compression speed is quite regular and above 1 GB/s, we noticed that this compression speed is linked with the CPU frequency thus with a higher CPU frequency we generally achieve a higher compression speed. For example, on an AMD Ryzen 7 3700X @ 3.6 Ghz we were able to reach compression speed above 2.5 GB/s up to 3 GB/s on FRIG and ORION datasets. We expected one current limitation of our algorithm for scalar fields with an average value of zero (e. g. velocity component). Indeed, if a parent value is positive and one of the children value is negative, then the very first bit of the integer representation will be set to 1 (sign bit) and therefore no delta compression could be achieved on that octant/quadrant. This limitation can be easily circumvented by shifting all values with a constant parameter. However, the results on table 7 show that the compression ratios for the velocity component are always at least as good as those of the density and pressure fields, highlighting the fact that the ocurrence of opposite sign between predicted value (parent cell value) and child value (child cell value) is rather uncommon in typical astrophysical use cases.

library	Ratio	Speed (MB/s)
PCP4	**1.230**	1085.58
LZ4	0.996	**2935.65**
Zlib 1	1.060	25.24
Zlib 9	1.070	21.17
Snappy	1.000	990.65
Zfp	1.097	72.03
Zstandard	1.061	585.85

Table 6: Benchmark of compression of the entire density field (12800 datasets) expressed according to the lightAMR format of Extreme-Horizon snapshot using different libraries: PCP4 (ours), the zlib version 1.2, the LZ4 package version 1.9.3, Snappy 1.1.7., zfp 0.5.5 and Zstandard 1.5.1. [22] . Benchmark run on a machine with a Intel Xeon Gold 5118 CPU @ 2.30 GHz.

The main reason that explains our high compression speed is that we do not need to evaluate a mathematical function to predict each value but we merely use the parent value which is already available in memory. Moreover, this parent value is used as a predictor for all the child's cells (4 to 8 cells depending on the number of dimensions). Furthermore, based on the formula of the compression ratio, removing the bit of sign plus the entire exponent on the 64 bits integer representation of a double precision float for a pack of cells leads to a theoretical ratio of 1.219. We generally achieve a ratio higher than this threshold according to the results in table 7, which means that almost every child cell value lies within a factor two of the parent cell value, which demonstrates the interest of using our predictor function. Since the basic idea of AMR is to refine a cell to avoid high variation of the physical field between adjacent cells, it was expected that in most cases we achieve at least this ratio. Nevertheless in the density field of FRIG, this is not the case mainly because a lot of shocks (turbulent interstellar medium modeling) are present in the simulation, a large number of steep density discontinuities can lead to lower compression ratio.

Since the float compression data algorithm process the AMR tree from the top to the bottom, from coarse cells to leaves, we can not access directly the value of a cell at a level l without first decompressing the tree from level 0 up to level l. There-

Simulation	Quantity	Ratio	Speed (MB/s)
FRIG	ρ	1.191	1274.94
FRIG	v_x	1.223	1079.23
FRIG	P_t	1.212	1100.95
ORION	ρ	1.226	1236.57
ORION	v_x	1.219	1101.36
ORION	P_t	1.224	1240.4
Ex-H	ρ	1.225	1059.78
Ex-H	v_x	1.258	1025.72
Ex-H	P_t	1.169	960.54

Table 7: Compression ratios and speeds for **double precision** fields: density, velocity x-axis component, thermal pressure, of FRIG, ORION and Extreme-Horizon using the Parent-Child predictor algorithm using 4 bits encoding (15 removed zeros at maximum), maximum theoretical ratio is 1.293. Single-threaded on a Intel Xeon Gold 5118 CPU @ 2.30 GHz.

fore, we give in table 8, the decompression speed for double precision fields. The decompression speed we obtain is on a par with compression speeds while the decompression remains memory friendly.

Simulation	Quantity	Speed (MB/s)
FRIG	ρ	911.58
FRIG	v_x	931.49
FRIG	P_t	911.60
ORION	ρ	927.62
ORION	v_x	921.84
ORION	P_t	928.15
Ex-H	ρ	760.18
Ex-H	v_x	720.01
Ex-H	P_t	709.69

Table 8: Decompression speeds for double precision fields: density, velocity x-component, thermal pressure, of FRIG, ORION and Extreme-Horizon (full level decompression). Single-threaded on a Intel Xeon Gold 5118 CPU @ 2.30 GHz.

Since in RAMSES one can downcast some physical quantities to single precision in order to reduce data volume by a factor 2, we give in table 9 the compression speeds and ratios for single precision data. If the entire exponent part (8 bits) of a single precision float (mapped on a 32 bits unsigned integer) is removed during the delta compression step, the theoretical compression ratio is 1.362, while it was only 1.219 for a double precision float. As shown previously with double precision float compression, we are still able to remove in most of the datasets the entire sign plus exponent part of the bit representation in the single precision float compression. Since the sign and exponent represent a bigger part of the 32 bits representation, it naturally leads to higher compression ratios. Therefore, down-casting double precision physical scalar fields within RAMSES can not only yield a reduction factor of 2 (due to down-casting) but also an even more efficient compression using our PCP algorithm. Consequently, down-casting from double to single precision can be of great interest in order to significantly reduce data volumes if double precision is not mandatory for the post-processing workflow.

Simulation	Quantity	Ratio	Speed (MB/s)
FRIG	ρ	1.301	1259.76
FRIG	v_x	1.393	1286.26
FRIG	P_t	1.349	1299.11
ORION	ρ	1.401	1292.74
ORION	v_x	1.380	1306.45
ORION	P_t	1.384	1331.21
Ex-H	ρ	1.394	1209.06
Ex-H	v_x	1.537	1217.75
Ex-H	P_t	1.345	1232.63

Table 9: Compression ratios and speeds for down-casted fields to **single precision**: density, velocity x-component, thermal pressure, of FRIG, ORION and Extreme-Horizon using the PCP4 algorithm (15 removed zeros at maximum), maximum theoretical ratio is 1.829. Single-threaded on a Intel Xeon Gold 5118 CPU @ 2.30 GHz.

The PCP4 algorithm is based on parent value as predictor value, this algorithm cannot be used if coarse node value are not available. Nevertheless, the basic idea of the lightAMR data model is to be a post-processing dedicated data model. AMR post-processing tools heavily use LOD approach which requires node value in order to be efficient. Therefore, if no node value is provided, then the whole dataset must be loaded to compute node values through up-sampling, such an operation will introduce very large overhead even for simple task such as low resolution images of the simulation box. The typical compression ratios that we obtain more than compensate the 12.5 % overhead required to store node values (in 3D cell-based AMR octree).

6. Conclusion

Adaptive Mesh Refinement suffers from a lack of standardized format to describe the grid structure and therefore AMR codes generally use their own format or eventually the unstructured grid to describe the mesh. Unfortunately, using the unstructured grid description is highly inefficient with AMR meshes and produce very large file even for relatively small meshes. In this paper, we presented an extremely compact format designed for AMR meshes, called *lightAMR*, for which we developed a set of dedicated data reduction and compression algorithms. To minimize data redundancy in RAMSES AMR grids, we developed a tree pruning algorithm able to remove between 11% and 38 % of cells in various astrophysical datasets. The large discrepancy of the obtained values is mainly due to the diversity of adaptive refinement layouts in our selected dataset simulation boxes. On top of that, we implemented a lossless and memory friendly compression algorithm specific to the lightAMR format that compresses the grid refinement/ownership mask arrays using base 52 continuous pack size, called CPS52. By switching to a boolean array description of the AMR grid in the lightAMR format, by omitting unnecessary meta-information and by chaining the tree pruning and the CPS52 algorithms on the tested RAMSES AMR datasets, we obtain AMR mesh

file size compression ratios ranging from 583.3 to 1500.0, thus reducing the AMR grid description to a negligible part of the overall data volume.

In addition, a lossless, memory friendly and efficient floating point data compression algorithm, called PCP, designed for lightAMR physical scalar fields achieve a high compression ratio and speed compared to commonly used and open source libraries by taking advantage of the topology of the AMR structure. For double precision, we achieve a compression speed ranging from 960 MB/s to 1275 MB/s and a ratio between 1.17 to 1.26, on RAMSES datasets, with a sequential version of the algorithm and a Intel Xeon Gold 5118 @ 2.30 GHz. While down-casting physical scalar fields to single precision naturally reduces the size by a factor 2, it also increases the compression ratios that range from 1.30 to 1.54 with similar compression speeds.

Finally, for the tested astrophysical datasets, the use of the lightAMR format leads to overall data reduction percentages of 62.26 % on FRIG, 49.64 % on ORION and 37% on Extreme-Horizon with no loss of information. If a RAMSES user is ready to sacrifice the precision of the scalar quantity fields and lets the lightAMR format downcast float data to single precision, then he can expect even more significant data volume reduction percentages (83.2% on FRIG, 81.3 % on ORION, 75.8 % on Extreme-Horizon). In the context of the three astrophysical simulation projects tested in this work, all the while using the same disk storage capacity, far more ambitious data output policies could have been considered, with data output done at higher frequencies (e.g. x5 on ORION, x6 on FRIG, x4 on Extreme-Horizon). The lightAMR format, as well as compression algorithms, are compatible by design with *level-of-details* (LOD) approach which is a very important technique used by post-processing tools to improve performance for both visualization and data analysis.

	FRIG	ORION	Ex-H
RAMSES Ncells (10^6)	1288.15	445.28	23660.29
lightAMR Ncells (10^6)	790.28	327.93	20958.90
RAMSES AMR (GB)	29.7	8.70	490
lightAMR (GB)	0.020	0.006	0.84
RAMSES HYDRO (GB)	113.4	46.9	2337.42
lightAMR HYDRO (GB)	54.0	28.0	1781
RAMSES HYDRO SP (GB)	56.7	23.45	1168.71
lightAMR HYDRO SP (GB)	24.0	10.42	683.22

Table 10: Recap of data volume reduction and compression on RAMSES simulation datasets. LightAMR data takes into account the tree pruning algorithm. The first two lines gives the total number of cells before and after tree pruning. The AMR grid data volume using both the tree pruning and the CPS52 compression algorithms and the HYDRO data volume is reduced using the tree pruning and the PCP4 compression algorithms. The last two lines show the data volume reduction when float data are down-casted to single precision (SP).

Nevertheless, we must insist on the fact that the lighAMR data structure is designed to significantly reduce the data storage impact of RAMSES (or any other fully threaded octree code) simulations as presented in the paper and must be seen as a "storage data structure". For analysis purposes, a "computational-friendly" data strucuture, [23], should be constructed from the lightAMR data in order to achieve good performance. In our analysis workflow, for example, we use vectors or hashmaps of leave cells only, allowing us to achieve high performance data analysis with an hybrid multiprocessing / multithreading parallelism but this discussion is beyond the scope of this article.

7. Future work

In the context of the standardisation process of the data format, the custom CPS52 run-length encoder will be updated in the near future to match more closely other mainstream run-length encoding schemes found in the literature, prior to the first release of LightAMR format. While the detailed PCP4 compression is lossless, we will explore in future work lossy compression as a compromise to down-casting physical quantities to single precision in order to further reduce I/O volume. In order to further reduce the data volume for post-processing and since the lightAMR data model is self-consistent and self-describing even over one domain, we will explore the concatenation of multiple ligthAMR data from adjacent domains. Finally, we plan to use this updated version of RAMSES with the integrated lightAMR format and parallel I/O with Hercule library for a large and ambitious astrophysical simulation run on a pre-exascale architecture.

Acknowledgements

The authors are thankful to P. Hennebelle and F. Bournaud for their useful comments on this manuscript. The authors acknowledge financial support from the European Research Council (ERC) via the ERC Synergy Grant *ECOGAL* (grant 855130).

References

[1] R. Teyssier, Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES, Astronomy and Astrophysics 385 (2002) 337–364. arXiv:astro-ph/0111367, doi:10.1051/0004-6361:20011817.

[2] E. F. Toro, M. Spruce, W. Speares, Restoration of the contact surface in the hll-riemann solver, Shock Waves 4 (1994) 25–34. doi:10.1007/BF01414629.

[3] T. Guillet, R. Teyssier, A simple multigrid scheme for solving the poisson equation with arbitrary domain boundaries, Journal of Computational Physics 230 (12) (2011) 4756–4771. doi:10.1016/j.jcp.2011.02.044.
URL http://dx.doi.org/10.1016/j.jcp.2011.02.044

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

[5] M. J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, Journal of Computational Physics 53 (3) (1984) 484–512. doi:10.1016/0021-9991(84)90073-1.

[6] A. Khokhlov, Fully Threaded Tree Algorithms for Adaptive Refinement Fluid Dynamics Simulations, Journal of Computational Physics 143 (2) (1998) 519–543. arXiv:astro-ph/9701194, doi:10.1006/jcph.1998.9998.

[7] R. Teyssier, S. Pires, S. Prunet, D. Aubert, C. Pichon, A. Amara, K. Benabed, S. Colombi, A. Refregier, J. L. Starck, Full-sky weak-lensing simulation with 70 billion particles, Astronomy and Astrophysics 497 (2) (2009) 335–341. arXiv:0807.3651, doi:10.1051/0004-6361/200810657.

[8] P. Ocvirk, D. Aubert, J. G. Sorce, P. R. Shapiro, N. Deparis, T. Dawoodbhoy, J. Lewis, R. Teyssier, G. Yepes, S. Gottlöber, K. Ahn, I. T. Iliev, Y. Hoffman, Cosmic Dawn II (CoDa II): a new radiation-hydrodynamics simulation of the self-consistent coupling of galaxy formation and reionization, Monthly Notices of the Royal Astronomical Society 496 (4) (2020) 4087–4107. doi:10.1093/mnras/staa1266.

[9] S. Chabanier, F. Bournaud, Y. Dubois, S. Codis, D. Chapon, D. Elbaz, C. Pichon, O. Bressand, J. Devriendt, R. Gavazzi, et al., Formation of compact galaxies in the extreme-horizon simulation, Astronomy and Astrophysics 643 (2020) L8. doi:10.1051/0004-6361/202038614.

[10] O. Bressand, L. Colombet, A. Fontaine, G. Harel, J.-B. Lekien, in CHOCS, 41, 29 (2012).

[11] L. Strafella, D. Chapon, Boosting I/O and visualization for exascale era using Hercule: test case on RAMSES, in: Journal of Physics Conference Series, Vol. 1623 of Journal of Physics Conference Series, 2020, p. 012019. arXiv:2006.02759, doi:10.1088/1742-6596/1623/1/012019.

[12] H.-Y. Schive, J. A. ZuHone, N. J. Goldbaum, M. J. Turk, M. Gaspari, C.-Y. Cheng, gamer-2: a GPU-accelerated adaptive mesh refinement code – accuracy, performance, and scalability, Monthly Notices of the Royal Astronomical Society 481 (4) (2018) 4815–4840. doi:10.1093/mnras/sty2586.

[13] G. L. Bryan, M. L. Norman, B. W. O. Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman, B. Smith, R. P. Harkness, J. Bordner, J. hoon Kim, M. Kuhlen, H. Xu, N. Goldbaum, C. Hummels, A. G. Kritsuk, E. Tasker, S. Skory, C. M. Simpson, O. Hahn, J. S. Oishi, G. C. So, F. Zhao, R. Cen, Y. L. and, Enzo: an adaptive mesh refinement code for astrophysics, The Astrophysical Journal Supplement Series 211 (2) (2014) 19. doi:10.1088/0067-0049/211/2/19.

[14] B. Fryxell, K. Olson, P. Ricker, F. Timmes, M. Zingale, D. Lamb, P. MacNeice, R. Rosner, J. Truran, H. Tufo, Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes, The Astrophysical Journal Supplement 131 (1) (2000) 273–334. doi:10.1086/317361.

[15] T. Guillet, D. Chapon, M. Labadens, PyMSES: Python modules for RAMSES (Oct. 2013). arXiv:1310.002.

[16] A. Robinson, C. Cherry, Results of a prototype television bandwidth compression scheme, Proceedings of the IEEE 55 (3) (1967) 356–364. doi:10.1109/PROC.1967.5493.

[17] P. Lindstrom, M. Isenburg, Fast and efficient compression of floating-point data, IEEE Transactions on visualization and computer graphics 12 (2006).

[18] M. Burtscher, P. Ratanaworabhan, IEEE Transactions on Computers 58 (1) (2009) 18–31. doi:10.1109/TC.2008.131.

[19] P. Hennebelle, The frigg project: From intermediate galactic scales to self-gravitating cores, Astronomy and Astrophysics 611 (2018) A24. doi:10.1051/0004-6361/201731071.

[20] E. Ntormousi, P. Hennebelle, Core and stellar mass functions in massive collapsing filaments, Astronomy and Astrophysics 625 (2019) A82. doi:10.1051/0004-6361/201834094.

[21] P. Lindstrom, Fixed-rate compressed floating-point arrays, IEEE Transactions on Visualization and Computer Graphics 20 (12) (2014) 2674–2683. doi:10.1109/TVCG.2014.2346458.

[22] Facebook, Zstandard (2017).
URL http://facebook.github.io/zstd/

[23] W. Bangerth, C. Burstedde, T. Heister, M. Kronbichler, Algorithms and data structures for massively parallel generic adaptive finite element codes, ACM Trans. Math. Softw. 38 (2) (jan 2012). doi:10.1145/2049673.2049678.