HAL open science

Couplage de modèles, algorithmes multi-échelles et calcul hybride

Jean-Matthieu Etancelin

To cite this version:

Jean-Matthieu Etancelin. Couplage de modèles, algorithmes multi-échelles et calcul hybride. Analyse numérique [math.NA]. Université de Grenoble, 2014. Français. NNT : 2014GRENM072 . tel-01094645v3

HAL Id: tel-01094645

https://theses.hal.science/tel-01094645v3

Submitted on 1 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques Appliquées**

Arrêté ministériel : 7 août 2006

Présentée par

Jean-Matthieu Etancelin

Thèse dirigée par **Georges-Henri Cottet**
et codirigée par **Christophe Picard**

préparée au sein du **Laboratoire Jean Kuntzmann**
et de **l'école doctorale MSTII: Mathématiques, Sciences et Technologies de l'Information, Informatique**

Couplage de modèles, algorithmes multi-échelles et calcul hybride

Thèse soutenue publiquement le **4 décembre 2014**,
devant le jury composé de :

M. Stéphane Labbé
Professeur, Université Joseph Fourier, Président
M. Florian De Vuyst
Professeur, École Normale Supérieure de Cachan, Rapporteur
M. Philippe Helluy
Professeur, Université de Strasbourg, Rapporteur
M. Guillaume Balarac
Maître de conférences, Grenoble INP, Examinateur
M. Georges-Henri Cottet
Professeur, Université Joseph Fourier, Directeur de thèse
M. Alexis Herault
Maître de conférences, Conservatoire National des Arts et Métiers, Examinateur
M. Christophe Picard
Maître de conférences, Grenoble INP, Co-Directeur de thèse
M. Christophe Prud'homme
Professeur, Université de Strasbourg, Examinateur
M. Jean-Baptiste Lagaert
Maître de conférences, Université Paris-Sud, Invité

Couplage de modèles, algorithmes multi-échelles et calcul hybride

Thèse présentée et soutenue publiquement par :
Jean-Matthieu Etancelin

le 4 décembre 2014

Composition du jury :

Président :
Stéphane Labbé,
Professeur, Université Joseph Fourier

Rapporteurs :
Florian De Vuyst,
Professeur, École Normale Supérieure de Cachan
Philippe Helluy,
Professeur, Université de Strasbourg

Examinateurs :
Guillaume Balarac,
Maître de conférences, Grenoble INP
Georges-Henri Cottet,
Professeur, Université Joseph Fourier (Directeur de thèse)
Alexis Herault,
Maître de conférences, Conservatoire National des Arts et Métiers
Christophe Picard,
Maître de conférences, Grenoble INP (Co-Directeur de thèse)
Christophe Prud'homme,
Professeur, Université de Strasbourg

Invité :
Jean-Baptiste Lagaert,
Maître de conférences, Université Paris-Sud

Thèse préparée au sein du Laboratoire Jean Kuntzmann
et de l'école doctorale MSTII : Mathémaiques, Sciences
et Technologies de l'Information, Informatique.

Cette thèse a été effectuée dans le cadre du projet ANR HAMM (ANR-10-COSI-0009).

Remerciements

Je voudrais exprimer ma gratitude auprès de toutes les personnes, du Laboratoire Jean Kuntzmann ou d'ailleurs, qui ont contribué à la réussite de cette thèse aussi bien sur le plan scientifique qu'humain.

Je remercie très chaleureusement mon directeur de thèse, Georges-Henri Cottet, et mon co-encadrant, Christophe Picard, pour leur disponibilité et leur soutient tout au long de ces trois années. Votre encadrement a été réel et complémentaire sur le fond et la forme de ces travaux.

Je remercie l'ensemble des membres de mon jury d'avoir pris le temps d'examiner mon travail, en particulier Florian De Vuyst et Philippe Helluy, rapporteurs de cette thèse.

Je remercie également Franck, Chloé, Jean-Baptiste et Éric pour les nombreux questionnements et discussions lors des réunions du groupe de travail. Ces échanges ont été en grande partie à l'origine des avancées majeures des travaux et ont permis de conserver une motivation à chaque instant.

Je voudrais adresser ma reconnaissance aux membres de l'équipe des Moyens Informatiques et Calcul Scientifique du LJK et du mésocentre CIMENT qui ont contribué à la résolution des nombreux problèmes techniques qui se sont posés tout au long de la thèse. Je remercie en particulier Franck (encore) pour sa grande disponibilité, surtout le vendredi après-midi, pour répondre à mes très nombreuses questions techniques. Pour les aspects informatiques, je remercie aussi Frédéric pour ses nombreuses remises en route de la machine de calcul du laboratoire. Je voudrais remercier l'ensemble des gestionnaires administratives du laboratoire qui assurent une présence rassurante pour mener à bien toutes les démarches.

Je remercie l'ensemble des doctorants et post-doctorants sur lesquels on peut compter pour un repas au RU, un café, des mots fléchés, de bons conseils, une soirée jeux, une sortie cinéma, une bière, une randonnée, des crêpes, une sortie ski, un footing, un trail,... Pour tout cela, merci à Chloé, Roland, Vincent, Bertrand, Thomas, Madison, Lukáš, Pierre-Olivier, Matthias, Nelson, Romain, Morgane, Pierre-Jean, Meriem, Kevin, Gilles, Euriell, Amin, Kolé, Louis, Federico Z., Mahamar, Abdoulaye, Federico P., Margaux, Olivier, ...

Merci à Martial pour les très nombreuses « buissonnades » en tout genre dans lesquelles on se laisse facilement embarquer. Je remercie également les copains de l'INSA pour leur promptitude à traverser la France et à proposer des hébergements pour passer du bon temps. Merci à ma famille pour son soutient tout au long de mes études.

Enfin, et non des moindres, merci à Myriam pour tout ces instants que l'on partage au quotidien.

Résumé

Dans cette thèse nous explorons les possibilités offertes par l'implémentation de méthodes hybrides sur des machines de calcul hétérogènes dans le but de réaliser des simulations numériques de problèmes multiéchelles. La méthode hybride consiste à coupler des méthodes de diverses natures pour résoudre les différents aspects physiques et numériques des problèmes considérés. Elle repose sur une méthode particulaire avec remaillage qui combine les avantages des méthodes Lagrangiennes et Eulériennes. Les particules sont déplacées selon le champ de vitesse puis remaillées à chaque itération sur une grille en utilisant des formules de remaillage d'ordre élevés. Cette méthode semi-Lagrangienne bénéficie des avantages du maillage régulier mais n'est pas contrainte par une condition de CFL.

Nous construisons une classe de méthodes d'ordre élevé pour lesquelles les preuves de convergence sont obtenues sous la seule contrainte de stabilité telle que les trajectoires des particules ne se croisent pas.

Dans un contexte de calcul à haute performance, le développement du code de calcul a été axé sur la portabilité afin de supporter l'évolution rapide des architectures et leur nature hétérogène. Une étude des performances numériques de l'implémentation GPU de la méthode pour la résolution d'équations de transport est réalisée puis étendue au cas multi-GPU. La méthode hybride est appliquée à la simulation du transport d'un scalaire passif dans un écoulement turbulent 3D. Les deux sous-problèmes que sont l'écoulement turbulent et le transport du scalaire sont résolus simultanément sur des architectures multi-CPU et multi-GPU.

Mots clés: Méthodes pariculaires ; couplage de modèles ; calcul hybride ; écoulements turbulents.

Abstract

In this work, we investigate the implementation of hybrid methods on heterogeneous computers in order to achieve numerical simulations of multi-scale problems. The hybrid numerical method consists of coupling methods of different natures to solve the physical and numerical characteristics of the problem. It is based on a remeshed particle method that combines the advantages of Lagrangian and Eulerian methods. Particles are pushed by local velocities and remeshed at every time-step on a grid using high order interpolation formulas. This forward semi-lagrangian method takes advantage of the regular mesh on which particles are reinitialized but is not limited by CFL conditions.

We derive a class of high order methods for which we are able to prove convergence results under the sole stability constraint that particle trajectories do not intersect.

In the context of high performance computing, a strong portability constraint is applied to the code development in order to handle the rapid evolution of architectures and their heterogeneous nature. An analysis of the numerical efficiency of the GPU implementation of the method is performed and extended to multi-GPU platforms. The hybrid method is applied to the simulation of the transport of a passive scalar in a 3D turbulent flow. The two sub-problems of the flow and the scalar calculations are solved simultaneously on multi-CPU and multi-GPU architectures.

Keywords: Particle methods; model coupling; hybrid computing; turbulent flows.

Table des matières

Remerciements **iii**

Résumé **iv**

Introduction générale **1**

1. Calcul intensif pour la mécanique des fluides **5**
1.1. Mécanique des fluides numérique 6
1.1.1. Modèle mathématique 6
Équations de Navier-Stokes 6
Fluides incompressibles 8
1.1.2. Principales méthodes de résolution 9
Méthodes Eulériennes 9
Méthodes de Lattice Boltzmann 12
Méthodes Lagrangiennes 13
Qualité des résultats numériques 16
1.1.3. Exemples d'application 16
1.2. Calcul intensif 17
1.2.1. Augmentation des besoin et des ressources de calcul 18
Ressources de calcul pour les simulations numériques 18
Augmentation de la puissance de calcul des machines 18
Difficulté d'adaptation des codes de calcul 20
1.2.2. Mesure de performances 20
Scalabilité 21
Modèle roofline 22
Efficacité énergétique 22
1.2.3. Défi de l'exascale 24
Pourquoi l'exascale ? 24
Principales difficultés 25
Quelques stratégies 26
1.3. Transport de scalaire passif dans un écoulement turbulent 26
1.3.1. Domaines d'application 26
1.3.2. Physique du problème 27
1.3.3. Formulation mathématique 28

2. Méthode particulaire pour l'équation de transport **31**
2.1. Méthode particulaire avec remaillage 32
2.1.1. Deux classes de méthodes semi-Lagrangiennes 32
2.1.2. Principe général de la méthode particulaire avec remaillage 33
2.1.3. Cas monodimensionnel 36
2.1.4. Lien avec la méthode des différences finies 37
2.1.5. Construction des formules de remaillage 39
Construction à partir des moments discrets 39
Construction à partir de B-splines 39
Extrapolation à partir de noyaux réguliers 40
2.1.6. Exemple d'implémentation 42
2.2. Advection semi-Lagrangienne d'ordre élevé 42
2.2.1. Construction de formules de remaillage d'ordre élevé 42
Utilisation des poids de remaillage 45
Précision numérique du schéma d'Horner 46
2.2.2. Consistance de la méthode semi-Lagrangienne 47
Advection par un schéma d'Euler 50
Advection par un schéma de Runge-Kutta du second ordre 50
Cas d'une seule particule par cellule 51
2.2.3. Stabilité de la méthode semi-Lagrangienne 51
Advection à vitesse constante 52
Advection à vitesse non constante 56

3. Méthode hybride pour le transport turbulent **59**
3.1. Idée générale d'une méthode hybride 59
3.1.1. Méthode hybride en méthode numérique 60
3.1.2. Méthode hybride en résolution 60
3.1.3. Méthode hybride en matériel 61
3.2. Application au transport turbulent 61
3.2.1. Méthodes spectrale et différences finies 61
3.2.2. Méthodes spectrale et semi-Lagrangienne 62
3.2.3. Méthodes semi-Lagrangiennes et architecture hybride 62

4. Développement d'un code multiarchitectures **65**
4.1. Développement d'une librairie de calcul scientifique 66
4.1.1. Conception préliminaire 66
Découpage sémantique des concepts mathématiques 66
Différents niveaux d'abstraction 67
Couplage faible et cohésion forte 67
Schéma d'utilisation 68
4.1.2. Conception détaillée 68
4.1.3. Fonctionnement de la librairie 70
Langages de programmation 70
Dépendances externes 71
Fonctionnement global 72

4.2. Calcul générique sur carte graphique 72
4.2.1. Description du matériel 72
Fonctionnement des cartes graphiques 72
Architecture des cartes graphiques 74
4.2.2. Différentes méthodes de programmation 75
Programmation par directives 75
Programmation directe 76
4.2.3. Les modèles de programmation OpenCL 76
4.2.4. Analyse de performances par le modèle roofline 80
4.3. Utilisation de cartes graphiques dans la librairie 82

5. Mise en œuvre sur cartes graphiques 85
5.1. Implémentations GPU de méthodes semi-Lagrangiennes 86
5.1.1. Méthodes semi-Lagrangiennes 86
5.1.2. Deux types d'interpolations 86
5.2. Implémentation et performances 89
5.2.1. Adéquation de la méthode au matériel 89
5.2.2. Préambule à l'étude des performances 91
5.2.3. Initialisation des particules 94
Copie 94
Transposition XY 95
Transposition XZ 97
5.2.4. Advection et remaillage 98
Advection 98
Remaillage 102
Noyau complet 105
5.3. Application simple GPU 107
5.3.1. Transport de scalaire 2D 107
5.3.2. Transport de scalaire 3D 111

6. Implémentation sur architectures hétérogènes 115
6.1. Lien entre la méthode et l'architecture hybride 116
6.1.1. Description d'une machine hybride 116
6.1.2. Différents niveaux de parallélisme 117
6.1.3. Stratégie d'utilisation 118
6.2. Application multi-GPU 119
6.2.1. Mécanisme de communication 119
6.2.2. Performances 121
6.3. Transport turbulent d'un scalaire passif 126
6.3.1. Application hybride 126
6.3.2. Exploitation d'une machine hétérogène 128
6.3.3. Résultats et performances 129

Conclusion générale 135

Bibliographie 139

A. Formules de remaillage de type $\Lambda_{p,r}$ **145**
A.1. Formules de type $\Lambda_{2,r}$ 145
A.2. Formules de type $\Lambda_{4,r}$ 148
A.3. Formules de type $\Lambda_{6,r}$ 150
A.4. Formules de type $\Lambda_{8,r}$ 154

B. Publications **157**
B.1.. High order semi-Lagrangian particle methods for transport equations : numerical analysis and implementation issues 159
B.2.. Multi-scale problems, high performance computing and hybrid numerical methods 190

Introduction générale

La réalisation de simulations numériques est une activité majeure dans les secteurs de la recherche et de l'industrie. Une simulation numérique permet d'obtenir une solution à un problème mathématique qu'il n'est généralement pas possible de résoudre analytiquement. Ces problèmes sont constitués d'équations modélisant un phénomène à étudier. Les premières simulations numériques datent du milieu du XXe siècle lorsque l'informatique commence à être utilisée dans ce but. En 1955, Fermi *et al.* réalisent une des premières simulations numériques d'un système dynamique monodimensionnel constitué de 64 masses reliées par des ressorts. Depuis, l'usage des calculateurs n'a cessé de se développer avec l'essor de l'informatique et des machines de calcul.

D'une part, les objectifs des simulations numériques sont de permettre une validation des modèles physiques et mathématiques utilisés pour l'étude d'un phénomène, par rapport à la théorie et à d'éventuelles données expérimentales. D'autre part, elles servent d'outils de conception et d'optimisation à des fins industrielles et permettent d'éviter le recours à de nombreux et coûteux modèles réduits et prototypes. L'explosion de la puissance de calcul développée par les machines parallèles au cours des dernières décennies permet la réalisation de simulations numériques de plus en plus précises et complexes tout en conservant des temps de calculs comparables. Ainsi, dans le domaine de la mécanique des fluides, l'évolution des simulations numériques contribue fortement à l'amélioration de la compréhension de phénomènes complexes comme la turbulence. En effet, les machines sont capables de traiter des résolutions toujours plus grandes, ce qui permet d'accroître la finesse des résultats, en particulier pour des Simulations Numériques Directes (DNS en anglais). De même, des études de fluides complexes (non Newtoniens, biologiques, alimentaires, ...) ou de problèmes multiphysiques (interaction fluide-structure, combustion, plasma, ...) font appel à des schémas numériques plus complexes et dont la mise en œuvre nécessite d'importantes ressources de calcul. Enfin, dans le cadre d'applications graphiques pour le cinéma ou les jeux vidéos, les capacités des machines actuelles permettent une utilisation de véritables modèles de fluide plutôt que des approximations afin d'augmenter le réalisme des écoulements dans des temps de calcul compatibles avec les contraintes de l'application.

Les simulations numériques reposent à la fois sur une ou plusieurs méthodes numériques et sur un code de calcul qui transcrit ces méthodes sur les machines de calcul. Le choix de la méthode de résolution dépend essentiellement de l'adéquation entre la nature du problème à résoudre, les caractéristiques attendues pour les solutions et les spécificités de la méthode. Le code de calcul met en œuvre les algorithmes issus de la méthode.

Le rôle de l'informatique est primordial pour l'obtention d'un code de calcul efficace. La qualité d'une méthode se mesure à la fois à travers les résultats qu'elle produit, mais aussi à son implémentation. L'efficacité numérique d'un code se mesure, entre autres, par le temps de calcul nécessaire à l'obtention de la solution en fonction de la qualité désirée. Ainsi, l'implémentation d'une méthode nécessite une attention particulière dans le but d'exploiter au mieux les ressources informatiques disponibles. Le développement d'un code efficace dans un contexte de calcul à hautes performances est un véritable défi sur des machines hétérogènes et massivement parallèles. Par conséquent, les méthodes numériques doivent non seulement être adaptées aux caractéristiques des problèmes à résoudre, mais également aux architectures des machines sur lesquelles elles seront implémentées. En particulier, l'aspect hétérogène des composants (CPU, GPU, coprocesseurs, ...) doit être pris en compte dès la conception de l'algorithme de résolution et de l'élaboration de la méthode numérique.

Dans cette thèse, nous considérons un problème de transport de scalaire passif dans un écoulement turbulent comme cadre applicatif. De nombreuses applications sont concernées par ce type de problème notamment dans les domaines de l'environnement, de l'industrie ou de la biologie (Shraiman *et al.*, 2000). Le scalaire transporté peut représenter une quantité réelle comme une concentration en espèce chimique ou en bactéries, ou bien abstraite comme une interface entre deux fluides. Dans ce travail nous nous limitons à l'étude du transport de scalaires passifs qui n'influent pas sur l'écoulement en retour. En général, la physique de ce type de problème se caractérise par la présence de plusieurs phénomènes à différentes échelles (Batchelor, 1958). L'approche envisagée ici est celle d'une résolution par une méthode hybride. Cette notion nécessite une précision quant au sens du mot hybride tel que nous l'entendons. Une méthode hybride consiste à résoudre les différents aspects physiques à l'aide d'une ou plusieurs méthodes numériques adaptées en fonction de leurs caractéristiques respectives. La méthodologie employée dans cette thèse est de chercher à exploiter les spécificités des aspects physiques des problèmes par la mise en place d'une méthode hybride (Gotoh *et al.*, 2012 ; Lagaert *et al.*, 2014).

Une des problématiques traitées dans ce manuscrit concerne l'exploitation de l'analogie entre l'aspect hybride de la méthode numérique et la nature hétérogène des machines de calcul. Nous nous intéressons à exécuter les différents éléments de résolution sur différents composants des machines. Nous en exposerons les raisons dans le chapitre 6. La mise en œuvre d'une stratégie hybride sera explorée en combinant des méthodes numériques de diverses natures, des résolutions différentes et une implémentation sur architecture hétérogène. L'approche suivie repose sur une implémentation sur cartes graphiques d'une méthode particulaire avec remaillage. Cette dernière permet, entre autres, de résoudre naturellement des équations de conservations sans imposer de condition de type Courant-Friedrichs-Lewy mais une condition de stabilité moins restrictive permettant l'usage de plus grands pas de temps. L'utilisation de cette méthode conduit, à travers la présence d'une grille sous-jacente, à des algorithmes et des structures de données régulières, ce qui est parfaitement adapté à l'emploi de cartes graphiques. Enfin, nous utiliserons la puissance de calcul offerte par les GPU pour proposer une extension à un ordre élevé de la méthode particulaire avec remaillage. L'analyse des schémas numériques se base sur une approche similaire à celle employée pour l'étude de schémas aux différences finies (Cottet *et al.*, 2006). Un des aspects informatiques considéré est lié à la nécessité du développement d'un code de calcul portable et non spécifique à une architecture afin de supporter l'évolution rapide des matériels et des librairies.

L'enchaînement des six chapitres de ce manuscrit reprend une démarche classique du calcul scientifique. En partant de la description des modèles puis des méthodes numériques et de leurs spécificités, on aboutit à l'implémentation générique multicœur GPU et hybride sur des machines hétérogènes combinant des processeurs multicœurs et des cartes graphiques. La progression et la répartition des différents chapitres sont schématisés par la figure 1.

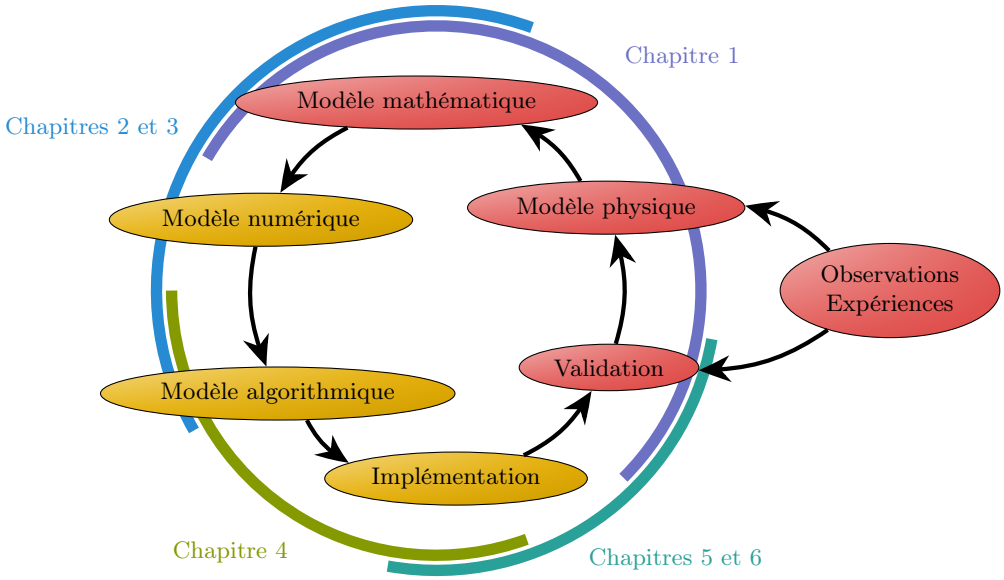


Figure 1. – Cycle de résolution d'un problème en mathématiques appliquées

Dans un premier chapitre, les modèles mathématiques pour la modélisation de fluides Newtoniens seront exposés. Nous présenterons également quelques méthodes numériques usuelles. Elles font appel à des ressources informatiques dont l'évolution, en termes de capacité et d'architecture, joue un rôle important dans le développement des codes de calcul. Une description des applications envisagées et notamment le transport de scalaire passif dans un écoulement turbulent terminera ce chapitre. Le second chapitre sera consacré à la description de la méthode particulaire avec remaillage. À travers une revue bibliographique de son évolution et de ses utilisations, nous exposerons l'analogie entre cette méthode et celle des différences finies. Cette revue sera complétée par une extension à un ordre élevé faisant intervenir la construction de formules de remaillage appropriées. Une analyse des schémas numériques en termes de consistance et de stabilité sera également développée. Dans un troisième chapitre, nous introduirons, à partir de travaux existants, la stratégie hybride envisagée pour la résolution de problèmes de transport de scalaire. Son application à un contexte de machine de calcul hétérogène sera abordé. Le quatrième chapitre sera consacré aux détails du développement d'une librairie de calcul scientifique portable et multiarchitecture. En particulier, nous verrons comment une conception souple simplifie l'utilisation de composants comme les cartes graphiques. Le chapitre cinq est dédié aux détails techniques de l'implémentation sur cartes graphiques de la méthode particulaire avec remaillage. Les choix d'implémentation seront éclairés par l'analyse de travaux similaires existants. Ce chapitre sera illustré par des exemples de transport de scalaire dans des cas où la vitesse est analytique afin de valider la méthode et d'en exposer les performances. Le

sixième chapitre sera consacré à l'extension multi-GPU de la méthode. Une analyse de la stratégie de couplage entre la méthode hybride et l'architecture hétérogène des machines de calcul sera donnée à travers un exemple de transport turbulent de scalaire passif. Enfin, nous terminerons ce manuscrit par un chapitre de conclusion donnant lieu à une critique des travaux effectués ainsi qu'à un énoncé des perspectives dégagées par cette étude.

1. Calcul intensif pour la mécanique des fluides

Le contexte des travaux de cette thèse est celui des mathématiques appliquées, au point de rencontre d'une application, d'une méthode numérique et d'un code de calcul. À partir d'observations de phénomènes naturels ou artificiels, des modèles physiques sont élaborés puis mis en équations sous la forme de modèles mathématiques. La spécialisation de ces modèles par rapport à une situation concrète se fait par la spécification de divers paramètres, de conditions initiales et de conditions aux limites des modèles. Ils expriment ainsi les conditions d'observations des phénomènes et le contexte expérimental des reproductions en laboratoire. Dans le cadre de simulations numériques, les modèles mathématiques sont discrétisés selon une ou plusieurs méthodes numériques conduisant à l'expression d'algorithmes qui sont implémentés dans des codes de calcul. L'exécution de ces codes sur des machines de calcul permet d'une part de valider les méthodes et les modèles par rapport aux phénomènes physiques mais aussi d'explorer les configurations qui ne sont pas aisément observables ou mesurables.

Ce premier chapitre est dédié à la présentation du contexte mathématique de la mécanique des fluides à travers la construction du modèle des équations de Navier-Stokes. Nous donnerons ensuite quelques méthodes de résolution classiques. Dans un second temps, nous verrons que la mise en pratique des méthodes numériques est étroitement liée à l'exploitation des ressources de calcul. Une forte contrainte de portabilité et d'adaptivité pèse sur les algorithmes et leurs implémentations du fait de la rapidité à laquelle les technologies des machines évoluent. Afin de garantir une certaine pérennité pour les codes de calcul, il est nécessaire de prendre en compte, dès la conception, non seulement les types ressources et les technologies actuelles mais aussi à venir, en particulier dans la perspective de l'exascale. Dans ce cadre, une grande importance est attachée à l'évaluation des performances des implémentations et à des comparaisons d'algorithmes dont nous donnerons quelques métriques couramment utilisées. Enfin, nous détaillerons les modèles physiques des applications envisagées pour ce travail.

1.1. Mécanique des fluides numérique

L'objectif de la mécanique des fluides est d'analyser et de comprendre les comportements de fluides, qu'ils soient liquides ou gazeux, lorsqu'ils sont en mouvement et éventuellement en présence d'obstacles ou de structures avec lesquels ils interagissent. La majeure partie des problèmes de mécanique des fluides sont issus de l'ingénierie avec notamment l'industrie aéronautique et plus généralement les domaines des transports, de l'énergie, du génie civil et des sciences de l'environnement. Des applications portant sur l'étude de fluides moins conventionnels sont également considérées en biologie et par l'industrie agro-alimentaire. La modélisation mathématique de ces problèmes conduit à des systèmes d'équations trop complexes pour être résolus formellement. Dans de nombreux cas, la démonstration formelle de l'existence de solutions à ces système reste encore un problème ouvert. Une modélisation numérique de ces problèmes permet d'approcher une solution par l'utilisation de méthodes numériques.

Les équations de Navier-Stokes permettent de décrire de manière générale le comportement d'un fluide en se basant sur des principes physiques de conservation de masse, de quantité de mouvement et d'énergie.

1.1.1. Modèle mathématique

Équations de Navier-Stokes

Conservation de la masse : Ce principe fondamental de la physique stipule que la masse totale contenue dans un système fermé reste constante au cours du temps. La matière, ou l'énergie, du système considéré peut éventuellement changer de forme ou se réarranger dans l'espace. La variation temporelle de la masse totale contenue dans un volume Ω est égale au bilan de masse traversant la frontière du domaine, $\partial\Omega$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\int_\Omega \rho \mathrm{d}x\right) = -\oint_{\partial\Omega} \rho \boldsymbol{u}\cdot\boldsymbol{n}\mathrm{d}s,$$

où ρ est la masse volumique, $\boldsymbol{u}$ la vitesse et $\boldsymbol{n}$ la normale extérieure unitaire à la surface du volume de fluide Ω. L'équation se réécrit de manière équivalente sous sa forme locale :

$$\frac{\partial\rho}{\partial t} + \mathrm{div}(\rho\boldsymbol{u}) = 0. \tag{1.1}$$

Conservation de la quantité de mouvement : De même, la variation temporelle de la quantité de mouvement d'un système, notée $\rho\boldsymbol{u}$, est égale à son bilan sur la frontière auquel s'ajoutent d'éventuelles forces $\boldsymbol{F}$. On note $\boldsymbol{u}:\boldsymbol{u}$ le produit tensoriel de $\boldsymbol{u}$ avec lui-même. Cela se traduit par la relation :

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\int_\Omega \rho\boldsymbol{u}\mathrm{d}x\right) = -\oint_{\partial\Omega} \rho\boldsymbol{u}:\boldsymbol{u}\cdot\boldsymbol{n}\mathrm{d}s + \int_\Omega \boldsymbol{F}\mathrm{d}x,$$

ou encore sous forme locale :

$$\frac{\partial\rho\boldsymbol{u}}{\partial t} + \mathrm{div}(\rho\boldsymbol{u}:\boldsymbol{u}) = \boldsymbol{F}.$$

Le développement de cette équation se simplifie avec l'équation de conservation de la masse (1.1) pour donner :

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t}+(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}\right)=\boldsymbol{F}.$$

Les forces $\boldsymbol{F}$ s'exerçant sur un volume de fluide se décomposent en deux termes selon leur nature :

$$\boldsymbol{F}=\operatorname{div}\bar{\bar{\boldsymbol{\sigma}}}+\boldsymbol{f}_e.$$

Les forces extérieures qui agissent sur l'ensemble du système sont notées $\boldsymbol{f}_e$, et les forces générées par les contraintes internes du fluide dépendent du tenseur des contraintes totales $\bar{\bar{\boldsymbol{\sigma}}}$ qui se décompose également en deux termes :

$$\bar{\bar{\boldsymbol{\sigma}}}=-P\bar{\bar{\boldsymbol{I}}}+\bar{\bar{\boldsymbol{\tau}}}, \tag{1.2}$$

où P est la pression, $\bar{\bar{\boldsymbol{I}}}$ le tenseur identité et $\bar{\bar{\boldsymbol{\tau}}}$ le tenseur des contraintes. Ainsi on obtient la formulation suivante :

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t}+(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}\right)=-\nabla P+\operatorname{div}\bar{\bar{\boldsymbol{\tau}}}+\boldsymbol{f}_e. \tag{1.3}$$

Loi de comportement : Le tenseur des contraintes $\bar{\bar{\boldsymbol{\tau}}}$ issu de la décomposition du tenseur des contraintes totales (1.2) permet de modéliser le comportement d'un fluide en particulier. Dans le cadre de fluides Newtoniens, le tenseur des contraintes est proportionnel au taux de déformations :

$$\bar{\bar{\boldsymbol{\tau}}}_{i,j}=\nu\left(\frac{\partial \boldsymbol{u}_i}{\partial x_j}+\frac{\partial \boldsymbol{u}_j}{\partial x_i}\right),$$

avec ν la viscosité dynamique du fluide. Lorsque la viscosité ne dépend pas de l'espace, on a :

$$\operatorname{div}\bar{\bar{\boldsymbol{\tau}}}=\nu\left(\Delta\boldsymbol{u}-\nabla\operatorname{div}\boldsymbol{u}\right). \tag{1.4}$$

Équations de Navier-Stokes : Dans le cadre général, les équations de Navier-Stokes constituent un système d'équations aux dérivées partielles obtenues à partir des équations de conservation de masse, (1.1), de quantité de mouvement (1.3) et de la loi de comportement du fluide (1.4).

$$\begin{cases}\dfrac{\partial\rho}{\partial t}+\operatorname{div}(\rho\boldsymbol{u})=0,\\ \rho\left(\dfrac{\partial \boldsymbol{u}}{\partial t}+(\boldsymbol{u}\cdot\nabla)\,\boldsymbol{u}\right)=-\nabla P+\nu\left(\Delta\boldsymbol{u}-\nabla\operatorname{div}\boldsymbol{u}\right)+\rho\boldsymbol{f}_e.\end{cases} \tag{1.5}$$

Les inconnues du problème sont la vitesse $\boldsymbol{u}$, la masse volumique ρ, et la pression P. Une troisième équation est alors nécessaire pour obtenir un modèle complet. Une équation de conservation de l'énergie est généralement utilisée. Les forces extérieures sont à préciser selon le cas d'étude.

Formulation vitesse-vorticité : Dans bon nombre d'écoulements, les zones d'intérêts, où se forment et évoluent des tourbillons, sont généralement de petite taille relativement au domaine d'étude et se localisent, entre autres, dans les couches limites et les sillages. Il est naturel d'employer une formulation adaptée. La vorticité de l'écoulement $\boldsymbol{\omega}$ est un champ vectoriel défini comme le rotationnel du champ de vitesse.

$$\boldsymbol{\omega} = \operatorname{rot} \boldsymbol{u} \tag{1.6}$$

On obtient la formulation vitesse-vorticité en appliquant l'opérateur rotationnel à la seconde équation du système (1.5) :

$$\rho \left(\frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla) \boldsymbol{\omega} \right) = \rho(\omega \cdot \nabla)\boldsymbol{u} + \nu \Delta \boldsymbol{\omega} + \rho \operatorname{rot} \boldsymbol{f}_e. \tag{1.7}$$

Fluides incompressibles

Hypothèse d'incompressibilité : Dans de nombreux cas, les fluides considérés ont une très faible compressibilité. Un fluide est dit incompressible lorsque que la variation de volume sous l'effet d'une pression extérieure est négligée. Cette hypothèse n'est valide que lorsque la vitesse du fluide est inférieure à la vitesse du son dans ce fluide. Dans ce cas, la densité ρ est approchée par une constante, l'équation (1.1) se réduit à :

$$\operatorname{div} \boldsymbol{u} = 0, \tag{1.8}$$

et la loi de comportement (1.4) se simplifie en :

$$\operatorname{div} \bar{\bar{\boldsymbol{\tau}}} = \nu \Delta \boldsymbol{u}. \tag{1.9}$$

L'obtention d'un champ de vitesse à divergence nulle, vérifiant (1.8), est généralement effectuée par l'utilisation d'une méthode de projection. Cette méthode a été introduite par Chorin (1968) et Temam (1968) et se base sur une décomposition du champ de vitesse selon une composante à divergence nulle et une composante à rotationnel nul :

$$\boldsymbol{u} = \boldsymbol{u}_\nabla + \nabla \phi, \qquad \text{avec } \operatorname{div} \boldsymbol{u}_\nabla = 0 \text{ et } \operatorname{rot} \nabla \phi = 0.$$

En pratique, un champ de vitesse intermédiaire $\boldsymbol{u}^*$ à divergence non nulle est obtenu par la méthode de résolution puis la fonction scalaire ϕ est obtenue par résolution d'une équation de Poisson :

$$\operatorname{div} \boldsymbol{u}^* = \Delta \phi,$$

et enfin le champ de vitesse incompressible est calculé simplement par : $\boldsymbol{u} = \boldsymbol{u}^* - \nabla \phi$. Dans le cas de la formulation vitesse-pression, la pression peut jouer le rôle de la fonction scalaire ϕ.

Équations de Navier-Stokes pour les fluides Newtoniens incompressibles : À l'aide de l'hypothèse d'incompressibilité (1.8) et pour une loi de comportement correspondant à un fluide Newtonien (1.9), le système générique (1.5) se réécrit :

$$\begin{cases} \operatorname{div} \boldsymbol{u} = 0, \\ \rho \left(\dfrac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla) \boldsymbol{u} \right) = -\nabla P + \nu \Delta \boldsymbol{u} + \rho \boldsymbol{f}_e. \end{cases} \tag{1.10}$$

La formulation vitesse-vorticité conduit au système suivant :

$$\begin{cases} \operatorname{div} \boldsymbol{u} = 0, \\ \boldsymbol{\omega} = \operatorname{rot} \boldsymbol{u}, \\ \rho \left(\dfrac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla) \boldsymbol{\omega} \right) = \rho(\omega \cdot \nabla)\boldsymbol{u} + \nu \Delta \boldsymbol{\omega} + \rho \operatorname{rot} \boldsymbol{f}_e. \end{cases} \tag{1.11}$$

La vorticité et la vitesse sont couplées par la relation $\boldsymbol{\omega} = \operatorname{rot} \boldsymbol{u}$, ou bien, de façon équivalente, par une équation de Poisson, en utilisant $\nabla \cdot \boldsymbol{u} = 0$:

$$\begin{cases} \Delta \boldsymbol{\psi} = -\boldsymbol{\omega}, \\ \boldsymbol{u} = \operatorname{rot} \psi. \end{cases} \tag{1.12}$$

1.1.2. Principales méthodes de résolution

Dans cette section nous donnons quelques unes des méthodes de résolution classiques des équations précédentes. La plupart de ces méthodes ne se restreignent pas à la mécanique des fluides et sont utilisées pour résoudre des problèmes aux dérivées partielles plus généraux.

Méthodes Eulériennes

Dans le cas des méthodes Eulériennes, la discrétisation des équations se fait par un découpage du domaine de calcul en un maillage, structuré ou non. Un maillage structuré se caractérise par une connectivité régulière des éléments, ce qui permet un accès efficace aux données des éléments voisins. Dans le cas non structuré, une table de connectivité définissant les relations de voisinage est nécessaire et permet alors une meilleure discrétisation de géométries complexes.

Différences finies La plus ancienne et la plus simple de ces méthodes est celle des différences finies. Elle a été formalisée par Euler en 1768. La méthode est construite sur la définition de la dérivée :

$$U'(x) = \lim_{h \to 0} \frac{U(x+h) - U(x)}{h},$$

qui se généralise sous la forme d'un développement limité :

$$U(x+h) = U(x) + hU'(x) + \frac{h^2}{2} U''(x) + \cdots \tag{1.13}$$

Lorsque la quantité h représente une distance entre deux points d'une grille et que la quantité U est approchée en chaque point x_i par $U(x_i) = U_i$, le développement précédent, tronqué au premier ordre, donne le schéma aux différences finies du premier ordre décentré en amont suivant :

$$U_i' = \frac{U_{i+1} - U_i}{h} + \mathcal{O}(h). \tag{1.14}$$

Il est possible de combiner différents développements d'ordres plus élevés pour construire des schémas d'ordre plus élevés ou approcher des dérivées d'ordres supérieurs. Ces schémas

peuvent être étendus pour l'approximation de variables multidimensionnelles, puis à la discrétisation de dérivées partielles. Le cas des grilles non uniformes se traite en prenant un paramètre h dépendant de la grille. Au point d'indice i, on a $h_i = x_{i+1} - x_i$.

Ainsi la méthode conduit à un calcul relativement simple à mettre en œuvre, dans les cas explicites, avec un schéma identique sur chaque point du domaine. Dans les cas implicites, il est nécessaire de résoudre un système linéaire. L'inconvénient majeur de cette méthode est la difficulté de prise en compte de géométries complexes.

Volumes finis La méthode des volumes finis, est aujourd'hui très largement utilisée, notamment par les industriels dans les domaines de l'aéronautique et de l'hydrodynamique. Cette popularité vient, entre autres, du fait que le caractère conservatif des équations est traduit directement dans la discrétisation. De plus cette méthode s'adapte parfaitement à de nombreux problèmes qu'ils soient en domaines fermés ou ouvert et sur des discrétisation spatiales structurées ou non.

Cette méthode permet de résoudre une équation de conservation, sous forme intégrale :

$$\frac{\partial}{\partial t}\int_{\Omega} \boldsymbol{U}\mathrm{d}\Omega + \oint_{S} \boldsymbol{F}\cdot\mathrm{d}\boldsymbol{S} = \int_{\Omega} \boldsymbol{Q}\mathrm{d}\Omega, \tag{1.15}$$

où $\boldsymbol{U}$ représente le vecteur des variables conservatives, $\boldsymbol{F}$ est un flux et $\boldsymbol{Q}$ un terme source. La discrétisation de ces équations repose sur le découpage du domaine de calcul en petits volumes de contrôle Ω_J autour d'un point de maillage J. L'équation (1.15) se discrétise donc sur chaque volume élémentaire par la relation suivante :

$$\frac{\partial}{\partial t}\left(\boldsymbol{U}_J\Omega_J\right) + \sum_{\Omega_k \in \text{ voisins de } \Omega_j} \boldsymbol{F}_{k,j}\cdot\boldsymbol{\Gamma}_{k,j} = \boldsymbol{Q}_J\Omega_J, \tag{1.16}$$

puis est intégrée entre deux instants consécutifs t_n et t_{n+1}. Les quantités $\boldsymbol{U}_J$ et $\boldsymbol{Q}_J$ sont les valeurs moyennées sur Ω_J à l'instant t_n. Ainsi, il est possible d'utiliser un schéma d'intégration en temps implicite ou explicite. L'application de ce schéma nécessite une méthode de calcul des flux $\boldsymbol{F}_{k,j}$ traversant la face $\Gamma_{k,j}$ commune aux éléments de maillage Ω_j et Ω_k.

En plus du calcul des flux en chaque face du maillage, la méthode nécessite un calcul direct, dans le cas explicite en temps, ou la résolution d'un système, pour le cas implicite. Le principal inconvénient de cette méthode est la difficulté de la montée en ordre.

Éléments finis La méthode des éléments finis est issue du domaine de la mécanique des structures. Elle repose sur des principes bien plus abstraits que ceux des deux méthodes précédentes. L'espace est discrétisé en un ensemble de cellules appelés éléments et se base sur une formulation intégrale du problème.

Le point clé de la méthode consiste à exprimer le problème sous forme variationnelle, à partir de sa formulation intégrale. Cette formulation variationnelle du problème fait intervenir des espaces fonctionnels qu'il est nécessaire de décrire avec précision ainsi que les normes qui leur sont associées afin de pouvoir déterminer certaines propriétés de convergence. Les espaces fonctionnels utilisés définissent des classes d'éléments finis sur les éléments de maillage. À partir de ce maillage, le plus souvent non structuré, il est nécessaire de

définir des fonctions d'interpolation des variables du problème sur les éléments du maillage. La discrétisation de la formulation variationnelle sur ces éléments conduit à l'obtention d'un système linéaire creux dont la taille dépend de la finesse du maillage. L'approximation d'une quantité u sur un élément se fait par combinaison linéaire des fonctions de base v_I pour chaque nœud I de l'élément.

$$\tilde{u}(x) = \sum_I u_I v_I(x). \tag{1.17}$$

Selon l'ordre d'approximation, les nœuds sont définis aux sommets, sur les arrêtes ou encore à l'intérieur des éléments du maillage.

Une fois la discrétisation effectuée, la solution est calculée par l'inversion d'un système linéaire dont la taille dépend du nombre d'éléments et du nombre de degrés de libertés associé. Le cadre mathématique de cette méthode permet de réaliser des études des propriétés numériques des schémas, une montée en ordre assez aisée ainsi que la prise en compte de géométries complexes. Toutefois, le système linéaire obtenu, peut parfois être difficile à résoudre. Pour cela des méthodes itératives avec préconditionnement sont généralement employées. D'autre part, l'inversion d'un système de grande taille creux n'est pas parallélisable aisément sans utiliser de méthodes additionnelles, de décomposition de domaine par exemple. Dans le cas des équations de Navier-Stokes, les variables ne sont pas exprimées dans les mêmes espaces fonctionnels. L'introduction de contraintes supplémentaires entre les classes d'éléments finis utilisés pour discrétiser la vitesse et la pression permettent d'assurer la convergence.

Cette méthode n'est pas la plus adaptée pour la résolution de problèmes dont la physique est dominée par l'advection. Dans un contexte de mécanique des fluides, elle est utilisée essentiellement pour des simulations de fluides complexes.

Méthodes spectrales Le principe général des méthodes spectrales est de transformer l'ensemble du problème dans un espace spectral. Pour ce faire, on exprime les variables du problème dans cet espace, par exemple en séries de Fourier tronquées :

$$u(\boldsymbol{x}, t) = \sum_{\boldsymbol{k}=-N/2}^{N/2} \hat{u}_{\boldsymbol{k}}(t) e^{i\boldsymbol{k}\cdot\boldsymbol{x}},$$

avec $\hat{u}_{\boldsymbol{k}}(t)$ les coefficients de Fourier tels que :

$$\hat{u}_{\boldsymbol{k}}(t) = \left(\frac{1}{2\pi}\right)^3 \int_0^{2\pi} u(\boldsymbol{x}, t) e^{-i\boldsymbol{k}\cdot\boldsymbol{x}} \mathrm{d}\boldsymbol{x}.$$

Avec ces représentations, le système d'équations à résoudre se réécrit pour les $\hat{u}_{\boldsymbol{k}}$. On aboutit généralement à un système d'équations différentielles ordinaires complexes ne dépendant que du temps. Les équations dans l'espace spectral sont généralement plus simples à résoudre que dans l'espace réel et permettent une précision numérique très importante même avec un petit nombre de coefficients. Néanmoins, il est assez difficile de traiter, avec cette méthodes, des domaines non périodiques ainsi que des géométries complexes. De plus, les transformations dans l'espace spectral impliquent un calcul global sur l'ensemble du domaine.

L'utilisation de méthodes spectrales nécessite quelques précisions quant à leur mise en œuvre sur des équations non linéaires. En effet, un produit de fonctions ou des termes non linéaires dans l'espace réel conduisent à des opérations de type convolution en espace complexe. Dans le cas discret, une erreur d'aliasing apparaît lors du calcul de ces termes de manière directe. Il existe différentes techniques de suppression de cette erreur. La règle des 3/2 est couramment utilisée et son principe est de réaliser les transformées de Fourier discrètes sur M points au lieu de N, avec $M \geqslant 3N/2$. Les coefficients de Fourier des termes du produit de convolution pour des nombres d'ondes supérieurs à $N/2$ sont simplement pris égaux à zéro. De même, le résultat est tronqué aux N premiers coefficients permettant, par transformée inverse, d'obtenir les valeurs réelles. En 3D, des erreurs d'aliasing doubles et triples sont corrigées par des des troncatures adaptées (Canuto *et al.*, 1987).

Méthodes de Lattice Boltzmann

Cette classe de méthode n'est pas basée sur la résolution des équations de Navier-Stokes mais sur une équation de Boltzmann qui modélise l'évolution cinétique de particules de fluide (Chen *et al.*, 1998). La méthode de *Lattice Boltzmann* dérive d'un modèle discret de dynamique particulaire sur une grille qui peut être vu comme un automate cellulaire *(Lattice gaz method)*. Le principe de base de cette méthode est de modéliser l'évolution d'une fonction de distribution de la vitesse des particules f en fonction d'un opérateur de collision Ω pour les interactions entre particules :

$$f_i(\boldsymbol{x} + \boldsymbol{e}_i, t + \Delta t) = f_i(\boldsymbol{x}, t) + \Omega_i(f(\boldsymbol{x}, t)). \tag{1.18}$$

Cette équation est donnée pour chaque direction i de la grille utilisée. La figure 1.1 représente les vecteurs $\boldsymbol{e}_i$ utilisés dans des grilles courantes. La notation DxQy est employée pour référencer ces grilles où x est la dimension de l'espace et y le nombre de vecteurs $\boldsymbol{e}_i$ considérés.

(a) Grille D2Q9 **(b)** Grille D3Q15

Figure 1.1. – Exemple de grilles pour la méthode Lattice Boltzman

Les grandeurs macroscopiques de l'écoulement sont retrouvées à partir des moments de f :

$$\rho = \sum_i f_i \quad \text{et} \quad \rho \boldsymbol{u} = \sum_i f_i \boldsymbol{e}_i.$$

Dans le cas le plus simple, un modèle de collision de Bhatnagar-Gross-Krook (BGK) permet d'exprimer Ω_i comme une relaxation vers une distribution à l'équilibre f_i^{eq} en un temps caractéristique τ :

$$\Omega_i = \frac{f_i^{eq} - f_i}{\tau},$$

avec :

$$f_i^{eq} = \rho w_i \left(1 + 3\boldsymbol{e}_i \cdot \boldsymbol{u} + \frac{9}{2}(\boldsymbol{e}_i \cdot \boldsymbol{u})^2 - \frac{3}{2}\boldsymbol{u}^2\right), \tag{1.19}$$

où les w_i sont des poids associés à chaque direction $\boldsymbol{e}_i$.

Le schéma numérique obtenu est local et la mise en œuvre est généralement réalisée en deux étapes. Dans un premier temps, le membre de droite de l'équation (1.18) est évalué avec le modèle de collision (1.19). Dans un second temps, l'équation (1.18) est résolue dans une étape de propagation des fonctions de distribution.

Méthodes Lagrangiennes

De manière générale, les méthodes Lagrangiennes sont adaptées à la résolution d'équations de conservation de la forme :

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega \boldsymbol{U}(x,t)\mathrm{d}\boldsymbol{x} = \int_\Omega \boldsymbol{F}(x,t,\boldsymbol{U},\nabla\boldsymbol{U},\ldots)\mathrm{d}\boldsymbol{x}. \tag{1.20}$$

Elles se distinguent des méthodes Eulériennes par le fait que la discrétisation spatiale du domaine de calcul par un ensemble de particules suit la dynamique du système contrairement au cadre Eulérien où le maillage reste fixe. On définit la dérivée Lagrangienne d'une fonction f à partir de la trajectoire d'un point de l'écoulement.

$$\begin{aligned}\frac{\mathrm{D}f}{\mathrm{D}t}(\boldsymbol{x}(t),t) &= \frac{\partial f}{\partial t}(\boldsymbol{x},t) + \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}\cdot\nabla f(\boldsymbol{x},t)\\ &= \frac{\partial f}{\partial t}(\boldsymbol{x},t) + (\boldsymbol{u}\cdot\nabla)f(\boldsymbol{x},t)\end{aligned} \tag{1.21}$$

Dans un cadre Lagrangien, les variables sont discrétisées sur un ensemble de particules qui correspondent à un volume élémentaire du domaine. Le principe général est que les particules portent des quantités correspondant aux variables du problème. Une quantité U est ainsi approchée par l'ensemble des particules selon la formule :

$$U(\boldsymbol{x}) = \int_\Omega U(\boldsymbol{y})\delta(\boldsymbol{x}-\boldsymbol{y})\mathrm{d}\boldsymbol{y}, \tag{1.22}$$

ou sa forme discrète régularisée :

$$U(\boldsymbol{x}) \simeq \sum_p U_p W_\varepsilon(\boldsymbol{x}-\boldsymbol{x}_p). \tag{1.23}$$

La quantité portée par la particule p, est donnée par sa valeur moyenne :

$$v_p U_p = \int_{V_p} U(\boldsymbol{x})\mathrm{d}\boldsymbol{x},$$

sur le volume V_p. La fonction W_ε, qui apparaît dans l'équation (1.23) est une fonction de régularisation dont le choix et les propriétés sont propres à chaque méthode. Dans tous les cas, elle est construite de telle sorte que sa limite quand ε tend vers 0 soit la mesure de Dirac δ.

Au cours de la résolution, les particules sont déplacées relativement au champ de vitesse puis interagissent pour donner une nouvelle répartition des quantités transportées qui conduisent à un nouveau champ de vitesse. La position de chaque particule, $\boldsymbol{x}_p$, est solution d'une équation de mouvement :

$$\frac{\mathrm{d}\boldsymbol{x}_p}{\mathrm{d}t} = \boldsymbol{u}(\boldsymbol{x}_p, t). \tag{1.24}$$

D'autre part, la discrétisation de l'équation (1.20) donne le système d'équations différentielles :

$$\frac{\mathrm{d}U_p}{\mathrm{d}t} = v_p F_p, \tag{1.25}$$

où F_p représente le terme source local à la particule p. La difficulté de ces méthodes réside dans le calcul du second membre de cette équation, en particulier le terme F_p, à partir des particules. Différentes approches sont possible et conduisent à des méthodes comme SPH et Vortex, détaillées plus loin. Enfin, le volume des particules varie selon l'équation :

$$\frac{\mathrm{d}v_p}{\mathrm{d}t} = v_p \operatorname{div} \boldsymbol{u}(\boldsymbol{x}_p). \tag{1.26}$$

Une propriété remarquable des méthodes particulaires est que la stabilité du schéma numérique ne dépend pas explicitement du maillage ou de la distance entre deux particules contrairement aux méthodes Eulériennes dont le pas de temps est contraint par une condition de Courant Friedrichs Lewy, notée CFL, dont la constante C est souvent inférieure à 1.

$$CFL = \frac{\Delta t ||\boldsymbol{u}||_\infty}{\Delta x} \leqslant C \tag{1.27}$$

Dans le cas des méthodes particulaires, la contrainte sur le pas de temps dépend du gradient du champ de vitesse et consiste en l'inégalité suivante :

$$LCFL = \Delta t ||\nabla \boldsymbol{u}||_\infty \leqslant C \tag{1.28}$$

En pratique, la constante C, appelée quelquefois nombre LCFL (pour *Lagrangian CFL*), conduit à une condition moins restrictive que la condition de CFL (1.27). Cela permet généralement d'utiliser de plus grands pas de temps que dans le cas d'une méthode Eulérienne, le nombre de CFL obtenu peut alors atteindre des valeurs de plusieurs dizaines.

Méthode SPH Dans la méthode *Smoothed Particle Hydrodynamics* (SPH), les dérivées spatiales des variables sont calculées sur les particules à partir d'une régularisation de la mesure de Dirac, à l'aide d'un noyau W_ε. Pour tendre vers la mesure de Dirac lorsque ε tend vers 0, la fonction W_ε est construite à partir d'une fonction d'intégrale égale à 1, à symétrie radiale et à support compact de rayon supérieur à ε (Liu *et al.*, 2010). Enfin le noyau W_ε doit être suffisamment régulier pour approcher les dérivées spatiales des variables intervenant dans le calcul des forces.

Le principe suppose qu'en plus de l'équation d'approximation (1.23), on dispose d'approximations des dérivées obtenues à partir des dérivées de W_ε :

$$\nabla U(\boldsymbol{x}_i) \simeq \sum_p U(\boldsymbol{x}_p) \nabla W_\varepsilon(\boldsymbol{x}_i - \boldsymbol{y}_p) v_p. \tag{1.29}$$

Ainsi, avec un schéma d'intégration en temps explicite et une discrétisation du membre de droite de l'équation (1.20), on obtient le schéma de résolution globale. Les sommes dans les équations (1.23) et (1.29) impliquent des interactions entre toutes les paires de particules. Cependant, la compacité du support de la fonction W_ε permet de ne considérer que les particules les plus proches du point de calcul courant à l'aide d'un tri des particules. Les particules dont le support du noyau d'interpolation intersecte le bord du domaine nécessitent un traitement particulier en fonction des conditions limites. L'inconvénient majeur de cette méthode est la difficile montée en ordre du schéma de résolution.

Méthode vortex La méthode vortex est une méthode Lagrangienne adaptée à la résolution des équations de Navier-Stokes en formulation vitesse et vorticité (1.11). Les particules portent alors la vorticité de l'écoulement, selon l'approximation (1.23) :

$$\boldsymbol{\omega}(\boldsymbol{x}) = \sum_p \boldsymbol{\omega}_p W_\varepsilon(\boldsymbol{x} - \boldsymbol{x}_p). \tag{1.30}$$

La méthode développée par Chorin (1973) et Leonard (1980) conduit à résoudre le système d'équations différentielles suivant :

$$\begin{cases} \dfrac{\mathrm{d}\boldsymbol{x}_p}{\mathrm{d}t} = \boldsymbol{u}(\boldsymbol{x}_p), & (1.31\text{a}) \\ \dfrac{\mathrm{d}\boldsymbol{\omega}_p}{\mathrm{d}t} = \boldsymbol{\omega}_p \nabla \boldsymbol{u}(\boldsymbol{x}_p) + \nu \Delta \boldsymbol{\omega}(\boldsymbol{x}_p). & (1.31\text{b}) \end{cases}$$

Le champ de vitesse est alors obtenu en résolvant l'équation de Poisson (1.12) en fonction de la vorticité portée par les particules :

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}_\infty + \int_\Omega K_\varepsilon(\boldsymbol{x} - \boldsymbol{y})\omega(\boldsymbol{y})\mathrm{d}\boldsymbol{y}. \tag{1.32}$$

Le noyau K_ε est une régularisation d'un noyau obtenu par le rotationnel de la solution fondamentale de l'équation de poisson. Ce noyau s'exprime comme la convolution de la fonction de Green pour l'opérateur laplacien et du champ de vorticité. D'autre part, le gradient du champ de vitesse est obtenu par :

$$\nabla \boldsymbol{u}(\boldsymbol{x}) = \int_\Omega \nabla K_\varepsilon(\boldsymbol{x} - \boldsymbol{y})\omega(\boldsymbol{y})\mathrm{d}\boldsymbol{y}.$$

Différentes techniques peuvent être employées pour discrétiser le terme de diffusion des équations (1.31). Nous pouvons citer par exemple la méthode de marche aléatoire, *Random-walk*, introduite par Chorin (1973) ou encore la méthode PSE, *Particle Strength Exchange* par Degond *et al.* (1989) et Cottet *et al.* (1990).

La discrétisation du terme d'étirement au second membre de l'équation (1.31b) et de l'équation de Poisson par une équation de Biot-Savart (1.32) nécessitent l'évaluation d'interactions entre toutes les paires de particules, ce qui engendre une complexité en $\mathcal{O}(N^2)$ pour N particules. Une complexité en $\mathcal{O}(N)$ est possible en utilisation une méthode FMM *(Fast Multipole Method)*. Cette dernière consiste en un algorithme hiérarchique, basé sur un arbre, permettant de traiter des problèmes à N corps par une approximation des interactions combinant des développement locaux et multipôles du noyau K_ε (Cheng *et al.*, 1999).

Méthode particulaire avec remaillage Cette méthode faisant l'objet du chapitre 2, nous donnons ici uniquement l'idée générale de la méthode. Le remaillage est employé dans les méthodes particulaire, en particulier les méthodes Vortex, et consiste en une redistribution des particules sur une grille (Koumoutsakos *et al.*, 1995). Il permet d'assurer le recouvrement nécessaire au maintient des propriétés de convergence de la méthode. La méthode obtenue lorsque le remaillage est effectué systématiquement à chaque itération s'apparente à une méthode semi-Lagrangienne explicite. Ainsi, après une advection des particules de manière Lagrangienne, une étape de remaillage permet de redistribuer les quantités portées par les particules sur une grille sous-jacente. Enfin les termes d'étirement et de diffusion ainsi que l'équation de Poisson peuvent être résolus sur cette grille.

Qualité des résultats numériques

Toutes ces méthodes de résolution reposent sur une approximation numérique des variables et des équations. Le principe commun est de discrétiser le domaine de calcul en divers éléments : grille, maillage ou particules. Dans tous les cas, la solution est donnée sur ces éléments discrets. Ainsi, l'augmentation du nombre d'éléments conduit à un meilleur niveau de détail de la solution. Une discrétisation temporelle plus fine permet également une meilleure description de la solution. Cependant, les pas de temps sont généralement choisis les plus grands possibles tout en veillant à préserver la stabilité du schéma.

D'autre part, la plupart des méthodes permettent une estimation de l'erreur d'approximation des équations par le schéma numérique. Cette erreur s'exprime généralement en fonction des résolutions spatiales et temporelles de la discrétisation. En contrepartie de l'amélioration de la précision et du niveau de détail des solutions, le coût de calcul augmente avec la résolution. Ce coût provient du nombre d'opérations arithmétiques à réaliser mais aussi du stockage et de l'accès aux données. En pratique, l'obtention de résultats pour une résolution plus fine permet généralement une plus grande précision dans la prise en compte des phénomènes physiques, ce qui contribue à une meilleure compréhension des mécanismes mis en jeu.

La conséquence de ce phénomène est que les ressources informatiques nécessaires à un raffinement des solutions numériques deviennent importantes et que les stratégies mises en œuvre pour l'implémentation des méthodes ont un impact direct sur cette augmentation de qualité numérique.

1.1.3. Exemples d'application

Dans ce paragraphe nous donnons quelques exemples d'implémentation des méthodes décrites précédemment dans un contexte de calcul à haute performance. L'objectif est simplement d'illustrer comment sont employées les différentes méthodes numériques à travers les quantités de ressources de calcul nécessaires.

Récemment, une méthode de volumes finis a été implémentée par Rossinelli *et al.* (2013) pour la simulation de la coalescence de 15 000 bulles de vapeur. Les auteurs ont réalisé des simulations allant jusqu'à 13.10^{12} points de calcul en utilisant 1,6 millions de cœurs CPU. Les performances obtenues sont, en partie, dues à l'utilisation de schémas numériques

d'ordre élevés en espace ce qui conduit à un nombre d'opérations par itérations élevé. Les auteurs emploient également des techniques permettant d'augmenter la localité des données ainsi que des opérations vectorisées spécifiques à la machine utilisée.

L'écoulement autour d'un modèle complet de voiture a été étudié par Jansson *et al.* (2011) en utilisant une méthode de résolution par éléments finis. Le domaine de calcul est discrétisé par 4,5 millions de points et 24 millions d'éléments et les solutions sont obtenues en utilisant 3 000 cœurs CPU. Cette étude montre que les résultats obtenus par une méthode par éléments finis associée à simulation aux larges échelles adaptative et à un raffinement automatique de maillage sont comparables, voire meilleurs, que certains modèles de turbulence.

Enfin, une comparaison de méthodes spectrales et vortex est réalisée par Yokota *et al.* (2013) sur 4 000 cartes graphique pour des résolutions de 69 milliards de particules ou points de calcul dans le cadre de simulations de turbulence homogène. Dans cet article, les auteurs montrent les limites des méthodes spectrales lors de l'utilisation d'un très grand nombre de cartes graphiques. En particulier, les opérations non locales des transformées de Fourier engendrent un coût de communication très important, ce qui n'est pas le cas en utilisant une méthode multipôle rapide *(Fast Multipole Method)*. Ainsi, l'étude montre que la méthode FMM permet d'atteindre une efficacité parallèle bien meilleure que les méthodes spectrales sur plusieurs milliers de cartes graphiques.

Ces études montrent l'importance d'une bonne exploitation des ressources informatiques. En général, les méthodes numériques ne sont pas directement adaptées à de grandes machines de calcul car les coûts de communication deviennent prohibitifs. Ainsi, il est nécessaire de prendre en compte ces architectures au niveau de la méthode numérique développée.

L'implémentation de méthodes numériques, nécessaire à la résolution de nombreux problèmes physiques, est indissociable des ressources informatiques disponibles. En particulier lorsque les applications considérées nécessitent un haut niveau de détail des solutions, ce qui implique un coût de calcul élevé.

1.2. Calcul intensif

Le calcul intensif est un domaine de l'informatique qui consiste à utiliser des supercalculateurs pour des applications relevant généralement du calcul scientifique. L'objectif est d'exploiter au maximum de leurs performances l'ensemble des ressources disponibles pour obtenir la solution d'un problème numérique.

On distingue deux contextes d'utilisation intensive des ressources de calcul. Le premier, que nous considérerons par la suite, consiste à résoudre un seul problème sur un ensemble de ressources d'une machine de calcul. Dans ce cas, la méthode numérique employée est distribuée sur les éléments de calculs de manière collaborative, ce qui implique des communications entre les éléments. Une machine de calcul est constituée de différents niveaux hiérarchiques de ressources. Elle est composée de nœuds de calcul interconnectés sur lesquels est installé un système d'exploitation. Un nœud comprend généralement de la mémoire vive, un disque local, des processeurs et éventuellement des accélérateurs, coprocesseurs ou cartes graphiques. Ces ressources sont partagées par les différents processeurs multicœurs. Les uti-

lisateurs se partagent la machine et peuvent réserver des ressources selon leurs besoins par l'intermédiaire d'un gestionnaire de ressources.

D'un autre côté, pour des calculs d'optimisation, des études de sensibilité par rapport à des paramètres ou des calculs de quantification d'incertitudes, le problème initial est de taille relativement modeste mais il doit être résolu un très grand nombre de fois à partir de données initiales différentes . Pour cela, des grilles de calcul sont utilisées, comme par exemple dans le projet `Folding@home` de l'Université de Standford. Dans ce cadre, un réseau composé d'un très grand nombre de machines grand public est constitué à travers Internet par les utilisateurs volontaires et est en perpétuelle évolution au gré des connexions et déconnexions des utilisateurs. Les différentes exécutions sont indépendantes et ne nécessitent pas de communications entre elles. Une des difficultés techniques rencontrées dans ce contexte est que les machines présentent de très nombreuses caractéristiques différentes notamment en termes de matériels et de systèmes d'exploitation.

1.2.1. Augmentation des besoin et des ressources de calcul

Ressources de calcul pour les simulations numériques

Comme nous l'avons vu en section 1.1.2, l'obtention de nouveaux résultats physiques permettant une meilleure compréhension des phénomènes se fait par une amélioration de la précision et du niveau de détail des résultats numériques. Pour la plupart des méthodes cela implique une complexité plus grande aussi bien en terme de calcul qu'en terme de quantité de données. Le temps de calcul est directement lié à cette complexité et augmente rapidement avec la résolution. C'est pourquoi il est nécessaire, pour rendre réalisable cette augmentation, que le temps de calcul soit le plus faible possible.

Par exemple, lorsque l'on souhaite simplement doubler la résolution dans toutes les directions spatiales d'un problème tridimensionnel, on obtient un facteur 8 sur le nombre total d'éléments. Ce qui implique un facteur au moins égal à 8 sur l'empreinte mémoire du code. De même, la complexité algorithmique du code est multipliée par un facteur 8, 16 ou 64 selon que la complexité s'exprime en $\mathcal{O}(N)$, $\mathcal{O}(N\ln(N))$ ou $\mathcal{O}(N^2)$. Cela implique que le coût total de calcul devient rapidement assez élevé et justifie le recours à des techniques de parallélisation mises en places dans le cadre du calcul intensif.

Deux points de vue sont envisageables. D'une part, pour un problème donné, l'augmentation des ressources allouées à sa résolution permettent de réduire le temps nécessaire à l'obtention de la solution. D'autre part, à temps de calcul constant, une solution plus précise est obtenue en augmentant conjointement les ressources et la taille du problème.

Augmentation de la puissance de calcul des machines

Loi de Moore Une caractéristique marquante de l'évolution de l'informatique est la loi de Moore, énoncée en 1965 par un des cofondateur d'Intel, G. Moore. En considérant la complexité de fabrication des circuits intégrés ainsi que leur coût en fonction du nombre de transistors, Moore postule que le nombre de transistors par puce double chaque année. Il observe par la suite, en 2005, que cette croissance est légèrement plus faible mais tout de

même exponentielle. Nous illustrons cette loi en figure 1.2 où le nombre de transistors par processeur en fonction des dates de mise sur le marché suit une tendance exponentielle. Il

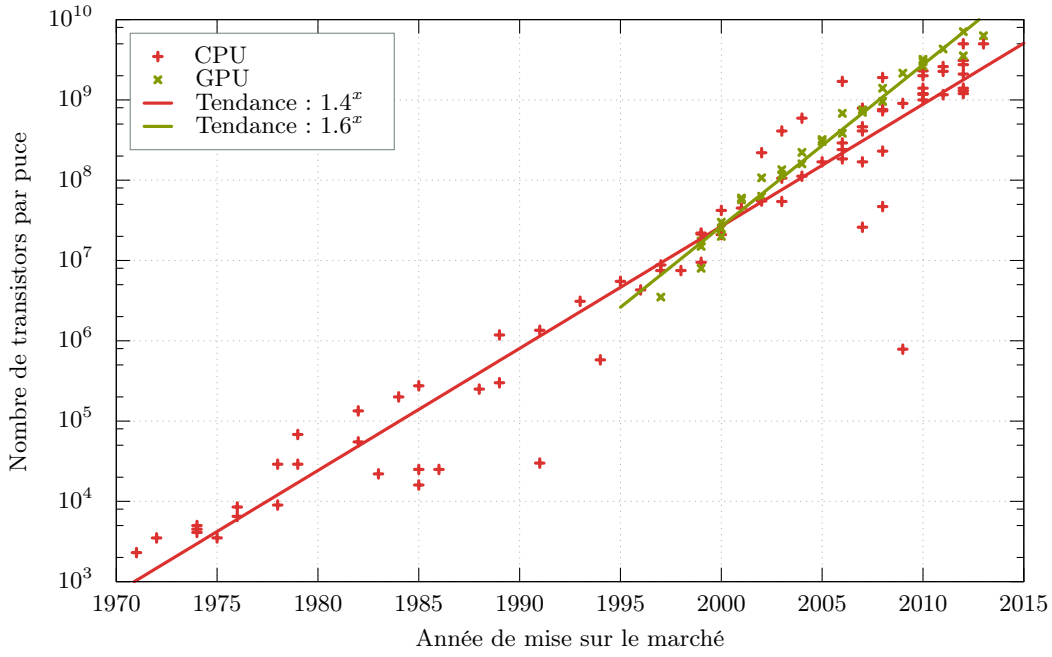


Figure 1.2. – Évolution du nombre de transistors par processeurs.

est à noter que les processeurs graphiques suivent également une tendance similaire.

Top500 Depuis 1993, le projet Top500 établit un classement mondial des 500 machines les plus performantes. La figure 1.3 montre l'évolution de la puissance de calcul théorique développée par ces machines. Sur cette figure sont représentés les machines de rang 1 et 500 ainsi que quelques composants introduits sur le marché en 2012 et 2013. Ces courbes conduisent à deux remarques : en 8 ans, une machine donnée passe de la première à la dernière place du classement ; sur la même durée, la puissance totale de la dernière machine du classement est disponible en un seul composant sur le marché grand public. Ainsi, une machine de bureau standard récente dispose de la puissance de calcul de la première machine du classement il y a 20 ans.

La tendance de ces courbes laisse suggérer que le régime de l'exascale sera vraisemblablement atteint d'ici 2020 (Shalf *et al.*, 2010 ; Dongarra *et al.*, 2011 et 2014). Il correspond à la construction et à l'exploitation de systèmes capables d'exécuter 10^9 GFLOPS. C'est un défi majeur dans la communauté du calcul haute performance. Dans les années 90, l'augmentation de la puissance des machines était principalement due à l'augmentation de la fréquence des processeurs et à la diminution de leurs taille. Ce mécanisme a atteint ses limites principalement à cause d'une trop forte densité d'énergie dans les composants. Par la suite, seule l'augmentation du nombre de processeurs par machine a permis de poursuivre cette évolution. Désormais, la tendance serait plutôt à l'augmentation du nombre de cœurs par processeurs comme c'est le cas, entre autres, avec les cartes graphiques et la dernière génération de processeurs Intel Xeon Phi.

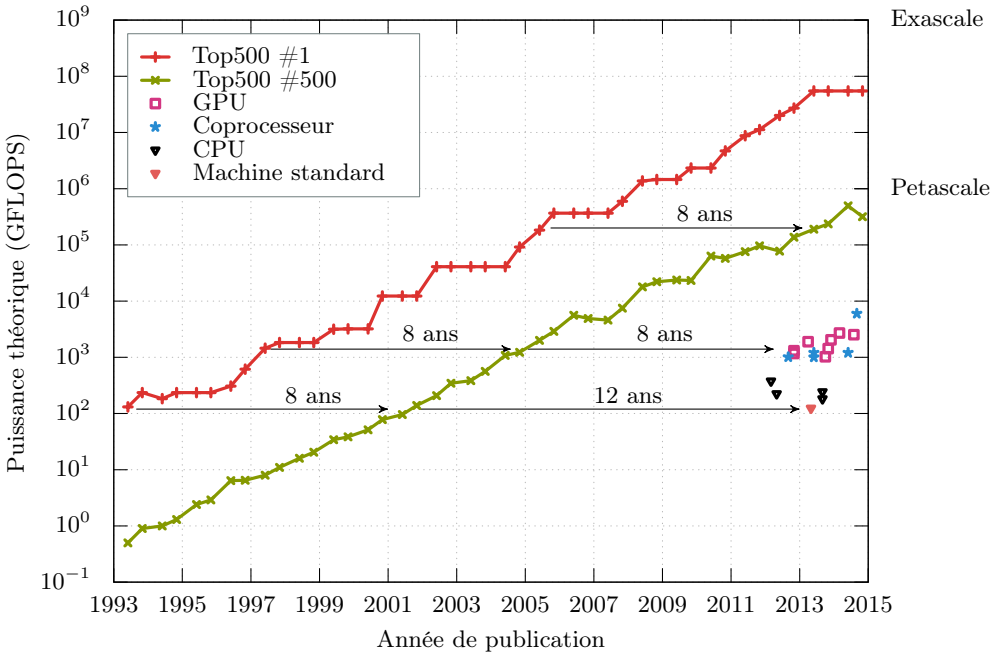


Figure 1.3. – Évolution du classement du TOP500

Difficulté d'adaptation des codes de calcul

Le détail des caractéristiques des machines du Top500 montre une hétérogénéité de composants depuis ces dernières années. En effet, bon nombre de machines de ce classement ne contiennent pas seulement des processeurs mais aussi des accélérateurs, que ce soit des cartes graphiques ou des coprocesseur. La difficulté majeure de l'exploitation de ces éléments est que le développement et l'optimisation des codes de calculs ne fait pas appel aux mêmes techniques que sur une machine homogène. De plus, les techniques de programmation sont parfois différentes entre les processeurs et les accélérateurs. Ce qui implique une réécriture du code. D'autre part, la réutilisation d'un code sur une architecture plus récente conduit parfois à des temps de calculs plus grands dans les cas où aucune adaptation n'est réalisée. Ce travail est assez lourd et ne peut pas être réalisé pour chaque nouvelle architecture. Toutefois, cette difficulté est atténuée par l'utilisation de langages portables pour le développement de codes multiarchitectures.

1.2.2. Mesure de performances

Dans le cadre du calcul intensif, il est important de pouvoir comparer les différentes implémentations et codes de calculs et d'en mesurer les performances. Pour cela des métriques spécifiques sont utilisées.

La première métrique utilisée est la puissance brute développée par un algorithme. Cette grandeur est exprimée par le nombre d'opérations sur des nombres réels à virgules flottante réalisées par secondes d'exécution. On utilisera la notation FLOPS pour désigner cette unité, également notée $\mathrm{FLOP\,s^{-1}}$ dans la littérature. Cette mesure est notamment utilisée pour établir le classement du Top500, En pratique, le nombre d'opérations effectuées lors d'une

exécution peut être obtenu par un outil de profilage et dépend fortement du compilateur, des options de compilations employées ainsi que de l'architecture utilisée.

Scalabilité

Scalabilité forte Lors de la parallélisation d'un code, la performance maximale atteignable est bornée par la loi d'Amdahl donnée par Rodgers (1985). L'accélération maximale dépend de la proportion de code parallélisable, notée $p \in [0;1]$. Le temps de calcul total, pour un problème de taille fixe, sur n processeurs T_n est donné en fonction du temps de calcul séquentiel T_1 par : $T_n = T_1(1-p+p/n)$. Le facteur d'accélération, S_n, égal au rapport entre le temps de calcul sur n processeurs et le temps de calcul sur un seul processeur est alors donné par l'expression :

$$S_n = \frac{T_1}{T_n} = \frac{1}{1-p+p/n} < n. \tag{1.33}$$

On remarque que lorsque le nombre de processeurs devient grand, le rapport tend vers $(1-p)^{-1}$. Cela signifie que l'accélération maximale, lorsque la partie parallélisable devient négligeable, est bornée par la partie non parallélisable du code. De manière optimale, pour $p = 1$, on attend que le facteur d'accélération du code soit égal au nombre de processeurs.

La performance d'un code peut être mesurée en réalisant une étude de scalabilité forte. Il s'agit d'étudier l'évolution du facteur d'accélération du code pour un problème de taille fixe en faisant augmenter les ressources employées à la résolution. Généralement, ces résultats sont présentés sur un graphe donnant le rapport T_1/T_n en fonction de n. Une scalabilité idéale conduit à l'obtention d'une droite de pente égale à 1.

Scalabilité faible On peut également considérer l'évolution du facteur d'accélération lorsque l'augmentation des ressources est réalisée simultanément avec une augmentation de la taille des données du problème. Dans ce cas, la loi de Gustafson-Barsis (Gustafson, 1988) conduit à une décomposition du temps de calcul sur n processeurs en une partie séquentielle t_s et une partie parallèle t_p, $T_n = t_s + t_p$. Ainsi, le temps de calcul de ce problème sur un seul processeur est égal à $T_1^{(n)} = t_s + nt_p$ et le facteur d'accélération est donné par :

$$S_n = \frac{T_1^{(n)}}{T_n} = n - \frac{t_s}{t_s + t_p}(n-1) \tag{1.34}$$

De même que précédemment, on représente S_n en fonction de n dans un graphe. Dans le cadre d'une étude de scalabilité faible, à chaque S_n correspond une taille de problème contrairement à la scalabilité forte où l'étude complète est réalisée pour une même taille de problème. Dans le cas idéal, le temps de calcul est entièrement dédié à la partie parallèle, $t_s = 0$, on obtient une droite de pente 1.

Le temps de calcul séquentiel du problème associé à n processeur, $T_1^{(n)}$, peut parfois devenir prohibitif et même impossible à obtenir lorsque n est grand. Pour cela, on pose l'hypothèse que ce temps est proportionnel au temps de calcul séquentiel du problème unitaire $T_1^{(n)} = nT_1^{(1)}$. On représente alors le facteur d'accélération par :

$$\tilde{S}_n = \frac{T_1^{(1)}}{T_n} = \frac{T_1^{(n)}}{nT_n} = 1 - \frac{t_s(n-1)}{n(t_s+t_p)}. \tag{1.35}$$

Modèle roofline

Les architectures multicœurs se caractérisent par le partage d'une mémoire entre les différents cœurs dans lesquelles la bande passante est parfois un facteur limitant les performances. Le modèle roofline, introduit par Williams *et al.* en 2009, est un modèle de visualisation de performances basé sur l'intensité opérationnelle d'une implémentation. Cette grandeur est définie comme le nombre d'opérations en virgule flottante par Byte de donnée transférée entre les unités de calcul et la mémoire globale. La quantité de données échangées entre les unités de calculs et la mémoire du système est constituée des accès en lecture et écriture. Ce modèle permet de représenter graphiquement les performances de calcul en fonction de l'intensité opérationnelle. Le graphe, en échelle logarithmique, présente la performance atteignable en fonction de l'intensité opérationnelle selon la formule :

$$P^A = \min(P, B \times I), \tag{1.36}$$

avec P^A la puissance atteignable en GFLOPS, P la puissance crête théorique de la machine, également en GFLOPS, B la bande passante théorique en GByte/s et I l'intensité opérationnelle du code en FLOP/Byte. L'intensité opérationnelle dépend de la machine et du code et peut être calculée à partir des compteurs présents dans le matériel et éventuellement accessibles via un outil de profilage. Dans certains cas, il est possible d'approcher cette intensité par un comptage des opérations du code. Le modèle conduit à la représentation donnée en exemple en figure 1.4.

On détermine si une implémentation est bornée par la bande passante, comme pour l'intensité I_A, ou par la puissance de calcul théorique, comme pour I_B. Cela donne une borne de performance maximale atteignable pour une implémentation donnée. Une première stratégie d'optimisation pour un algorithme donné consiste à se déplacer vers une intensité opérationnelle plus grande, flèches horizontales sur la figure 1.4. Elle consiste en une optimisation de l'implémentation pour limiter les accès à la mémoire globale afin de maximiser la puissance atteignable. La seconde stratégie consiste à optimiser le code pour diminuer le temps de calcul. En effet, la performance atteinte lors d'une exécution correspond à un point sur le graphe. Visuellement, les optimisations visent à déplacer ce point vers la borne théorique le long d'une demi-droite correspondant à l'intensité opérationnelle de l'implémentation évaluée. Cette seconde stratégie est représentée par les flèches verticales sur la figure 1.4.

Efficacité énergétique

Dans le cadre de l'étude de l'impact environnemental des machines, la puissance de calcul est mise en regard de la consommation électrique. En particulier, depuis fin 2007, le classement du Green500 ordonne les éléments du Top500 en fonction de leur performance énergétique qui est exprimée en MFLOPS par watt consommé au cours d'un calcul. La figure 1.5 donne l'évolution des machines de rang 1 et 500 du classement Green500 ainsi que, à titre de comparaison, du Top500.

La figure 1.6 représente les 150 premières machines des classements du Top500 et Green500 par groupes de couleurs selon le type d'accélérateurs ou selon l'architecture. Comme le montre Subramaniam *et al.* (2013), dans leur analyse de la consommation énergétique des super-calculateurs l'utilisation d'accélérateurs graphiques, GPU, ou de coprocesseurs semble contribuer fortement à l'amélioration de l'efficacité énergétique. À titre de

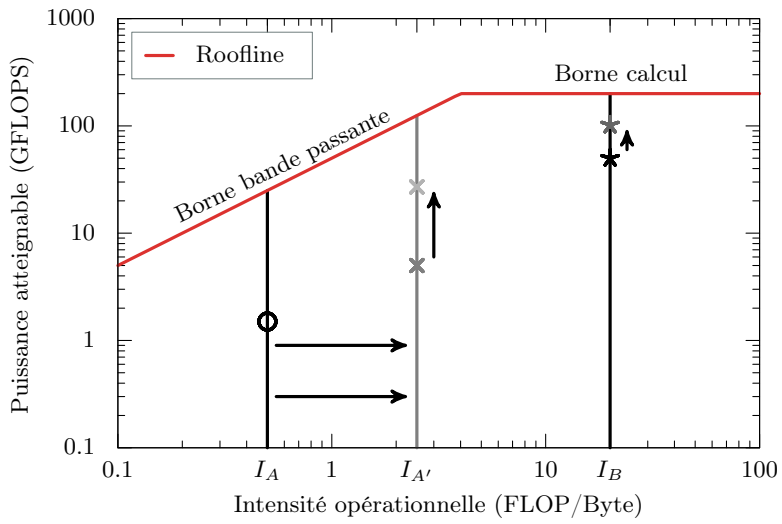


Figure 1.4. – Exemple de modèle roofline.

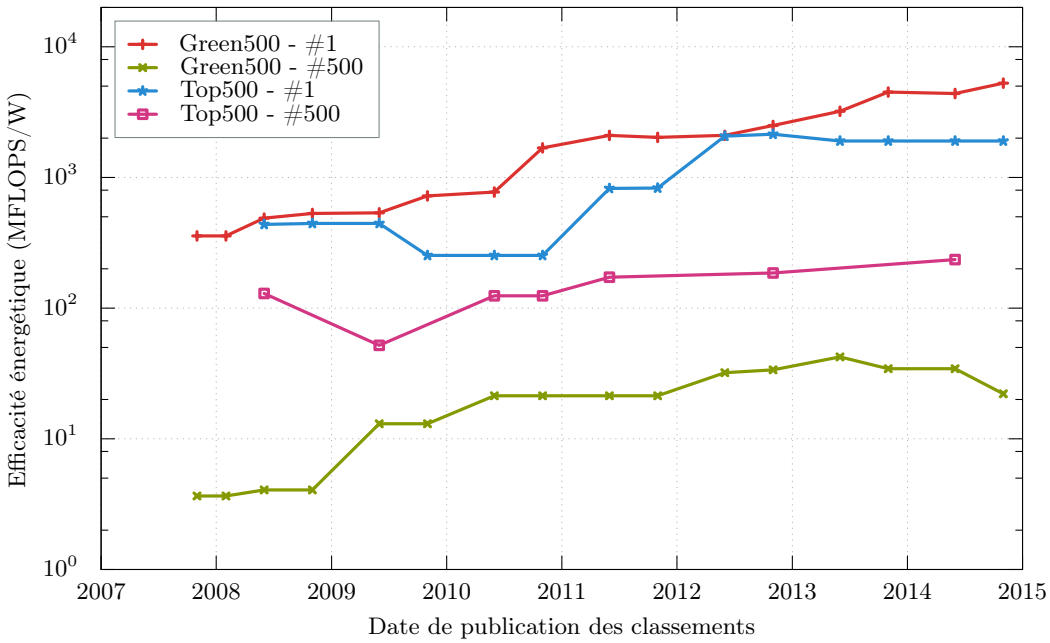


Figure 1.5. – Évolution de l'efficacité énergétique des machines.

comparaison, les ordres de grandeurs de puissance consommée sont illustrés par des matériels usuels.

Le code de couleurs utilisé sur la figure 1.6 permet de distinguer quatre types d'architecture. L'architecture classique correspond à une machine constituée de processeurs multicœurs traditionnels et ne contenant aucun accélérateur. L'architecture BlueGene se distingue de la précédente par un nombre bien plus grand de processeurs, par une faible fréquence des processeurs et une faible quantité de mémoire des nœuds. Ces deux dernières caractéristiques permettent à ce type de machine d'être extrêmement efficace en termes de performance énergétique. Les machines dotées de coprocesseurs représentées sur la figure 1.6 sont constituées de processeurs multicœurs Xéon Phi comportant jusqu'à 61 cœurs de calcul et qui ont été introduits récemment par Intel. La machine à coprocesseurs la plus efficace est constituée de processeur PEZY-SC contenant 1 024 cœurs qui sont développés par la société japonaise PEZY Computing. Enfin, le second type d'accélérateur est représenté par les GPU dont les caractéristiques seront détaillées dans le chapitre 4.

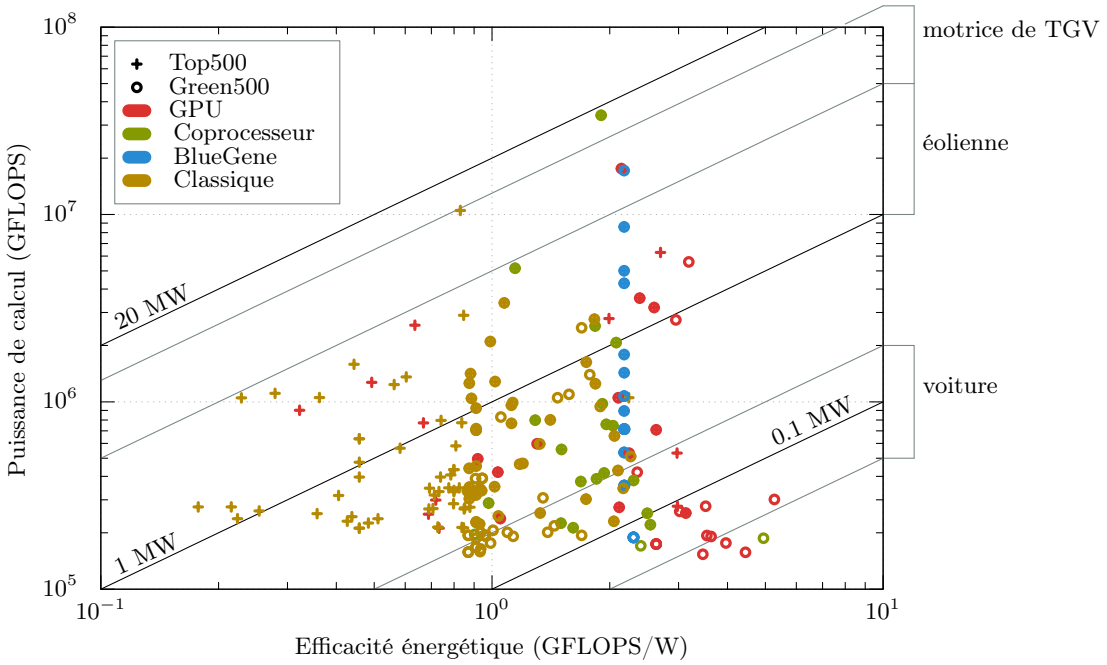


Figure 1.6. – Comparaison des performances énergétiques des 150 premières machines du Top500 et Green500 selon le matériel (classements de novembre 2014).

1.2.3. Défi de l'exascale

Pourquoi l'exascale ?

Comme nous l'avons vu précédemment, l'amélioration de la qualité des simulations numériques nécessite des résolutions plus importantes, afin d'obtenir de nouveaux résultats pour améliorer la compréhension et la prise en compte de phénomènes physiques. Cela permet ensuite d'envisager une complexification des modèles physiques puis des modèles

mathématiques et numériques dans le processus itératif schématisé en figure 1. De ce point de vue, ces simulations ont besoin de ressources de calcul toujours plus importantes. D'autre part, les classements du Top500 et Green500 nous montrent que la puissance des supercalculateurs est en augmentation, apportant ainsi une réponse à ces besoins en ressources de calcul. Cette réponse permet d'envisager la simulation de problèmes physiques encore plus complexes.

Toutefois, les applications sont encore loin d'exploiter toute la puissance offerte par ces calculateurs. La littérature ne fait état que de très peu d'applications capables d'atteindre le PFLOP. Des simulations d'interactions gravitationnelles en astrophysique entre 10^{12} corps atteignant 4.45 PFLOPS ont été obtenues par Ishiyama *et al.* (2012), soit 42% de la puissance maximale de la machine pour leurs configuration. Plus récemment, une performance de 11 PFLOPS a été atteinte par Rossinelli *et al.* (2013) pour la simulation de la coalescence de 15 000 bulles de vapeur sur 13.10^{12} points de calcul en utilisant 1.6 millions de cœurs CPU. Cette performance correspond à 55% de la puissance de la machine. Enfin, 4096 cartes graphiques ont permis à Yokota *et al.* (2013) d'atteindre 1.08 PFLOPS en simple précision, soit 25% de la puissance maximale théorique.

Principales difficultés

Le développement d'applications capables d'exploiter une telle puissance se confronte à de nouvelles problématiques dont nous reprenons quelques éléments, d'après Dongarra *et al.* (2011).

Concurrence L'architecture des machines de l'exascale, basée sur des composants multicœurs ayant un très grand nombre de cœurs, soulève de nombreuses difficultés quant à l'utilisation des mécanismes et des algorithmes actuels notamment en terme de synchronisation des différents processus ou encore d'accès aux données. En effet, les architectures multicœurs ont la particularité de partager une même mémoire entre l'ensemble des cœurs ce qui rend complexe l'accès aux données, à travers une hiérarchie de caches, qui ont éventuellement été modifiées dans le même temps. Leur originalité est qu'elles disposent d'une quantité de mémoire disponible par cœur de calcul plus faible que dans les architectures classiques. L'augmentation du nombre de cœurs par processeurs conduit à une diminution de la mémoire disponible par cœurs.

Les accès à des données distantes, non présentes sur la mémoire associée à un processus, deviennent rapidement très coûteux pour un très grand nombre de cœurs. Ce qui pose la question de l'élaboration d'algorithmes de communications et de synchronisations distants adaptés à ces architectures.

Énergie La consommation énergétique de ces supercalculateurs devient un problème non seulement au niveau de l'alimentation électrique globale des machines mais aussi du refroidissement des composants. Une limite de faisabilité d'une consommation de 20 MW a été proposée dès 2008 par Bergman *et al.* et détaillée plus récemment par Subramaniam *et al.* (2013). Les performances énergétiques des algorithmes commencent à être considérées, en complément des mesures traditionnelles, notamment à travers le classement Green500 et contribuent à une amélioration des performances énergétiques des machines.

Robustesse Les applications déployées sur des machines exascale devront être robustes aux pannes. En effet, statistiquement, la probabilité qu'un cœur tombe en panne au cours de l'exécution d'un programme est non négligeable. Des solutions doivent être développées afin de s'affranchir de la technique traditionnelle de la sauvegarde de l'état complet d'un programme à un instant donné en vue d'un redémarrage qui devient impossible à mettre en place dans un tel environnement. De nombreuses techniques sont développées (Dongarra *et al.*, 2014) afin de pouvoir, d'une part, détecter les pannes et les erreurs introduites dans les résultats et, d'autre part, effectuer un redémarrage local des processus en échec en utilisant les données locales pour reconstruire une solution au moment de la panne. Dans ce contexte l'exécution devient non reproductible.

Quelques stratégies

Les contraintes de ce passage à l'échelle permettent, par nécessité, de dégager quelques stratégies au niveaux des algorithmes et des méthodes numériques employées. Ces stratégies sont détaillées de manière exhaustive par Dongarra *et al.* (2014). Les auteurs soulignent notamment que l'utilisation de modèles couplés, de méthodes d'ordre élevé ou de parallélisation en temps permettent d'aboutir à des implémentations susceptibles de s'adapter à l'architecture des machines. L'intérêt est de pouvoir introduire des considérations matérielles dès la spécification des méthodes numériques et même au niveau de la modélisation mathématique du problème. D'autre part, au niveau des algorithmes, l'accent est mis sur la scalabilité des solveurs, l'utilisation d'algorithmes multiniveaux et de librairies de calcul existantes pour lesquelles les optimisations par rapport aux architectures auront été réalisées de manière très pointue.

Le cadre applicatif de la mécanique des fluides s'adapte assez bien aux caractéristiques du calcul intensif. En effet, à travers une application, nous allons pouvoir employer une méthode de résolution ayant de bonnes propriétés de parallélisation dans l'optique de l'exploitation de serveurs de calculs hétérogènes. Le développement et l'implémentation de cette méthode numérique est réalisé dans le cadre défini par l'application que nous détaillons ci-après.

1.3. Transport de scalaire passif dans un écoulement turbulent

1.3.1. Domaines d'application

De très nombreux phénomènes physiques sont basés, au moins en partie, sur le transport d'un ou plusieurs scalaires par un écoulement turbulent. Les scalaires correspondent à des quantités de diverses natures (Shraiman *et al.*, 2000). Dans le domaine de l'environnement, le scalaire peut représenter un polluant transporté par l'air ou l'eau (Michioka *et al.*, 2008). En biologie il peut s'agir de l'évolution des concentrations de certaines molécules chimiques, comme les phéromones, que de nombreuses espèces utilisent pour communiquer ou se repérer (Murtis *et al.*, 1992) ou encore la concentration de bactéries et de leur nourriture (Taylor *et al.*, 2012). Le domaine de l'ingénierie est concerné par de nombreux problèmes

de ce type, notamment en combustion. Dans ces problèmes, plusieurs scalaires peuvent être transportés simultanément comme les différentes concentrations en réactifs chimiques ou encore la fraction de mélange entre le combustible et l'oxydant (Pitsch *et al.*, 2008). La température est un autre exemple de quantité scalaire, importante dans l'étude de systèmes de refroidissements (Kuczaj *et al.*, 2010).

Un transport de scalaire peut également être utilisé comme outil de modélisation. Par exemple, le transport d'une fonction levelset permet de modéliser les interfaces dans un fluide multiphasique (Tryggvason *et al.*, 2011) ou un objet en mouvement (Sethian *et al.*, 2003).

1.3.2. Physique du problème

Généralement, le scalaire a une influence sur l'écoulement dans lequel il est transporté. Par exemple dans les cas où on considère la densité d'un fluide multiphasique, ou encore lors du transport de concentrations en réactifs chimiques dans une réaction de combustion. Dans certaines applications, les effets du scalaire sur l'écoulement deviennent négligeables, ce qui est le cas dans le transport de petits éléments comme des polluants, molécules ou colorants. Le scalaire est alors passif dans l'écoulement et n'a pas d'effet rétroactif sur le fluide.

Dans un écoulement turbulent et indépendemment du transport de scalaire, des structures de différentes tailles se forment. Des structures de grandes tailles sont créées et entretenues par l'écoulement. Les fluctuations de ces structures génèrent des structures plus petites qui elles-mêmes génèrent, en cascade, d'autre structures encore plus fines. Un tel écoulement se caractérise par une large palette de taille de structures s'étalant sur plusieurs ordres de grandeur. Cependant, à partir d'une certaine échelle, dite de Kolmogorov, les effets visqueux du fluide deviennent prédominants et engendrent une dissipation de l'énergie cinétique de l'écoulement (Lesieur, 2008). La longueur de cette échelle ne dépend que des caractéristiques du fluide dont notamment la viscosité ν. Ainsi, un écoulement turbulent ne présente pas de structures de tailles inférieures à la longueur de l'échelle de Kolmogorov, notée η_ν. Lors d'une simulation directe, *DNS* en anglais pour *Direct Numerical Simulation*, l'écoulement est résolu à toutes les échelles jusqu'à l'échelle de Kolmogorov. Le maillage doit avoir une longueur de maille suffisamment petite pour capter les plus petites structures de l'écoulement. Dans les cas où le maillage ne résout pas toutes les échelles, il est nécessaire de faire appel à un modèle de turbulence permettant de simuler cette dissipation d'énergie par les petites échelles non résolues (Lesieur *et al.*, 2005 ; Sagaut, 2006).

La dynamique du scalaire se caractérise par deux phénomènes : une dispersion par convection du scalaire puis sa diffusion. Le premier effet, induit par le fluide turbulent, a pour conséquence d'étirer les structures du scalaire, de générer des fluctuations à des échelles de plus en plus petites et d'amplifier ses concentrations locales. De manière similaire au fluide, l'échelle de Batchelor, notée η_κ, décrit l'échelle de concentration d'un scalaire à partir de laquelle la diffusion moléculaire devient prédominante (Batchelor, 1958). Là encore, cette échelle ne dépend que des caractéristiques du fluide, dont la viscosité et du coefficient de diffusion, noté κ. Cette diffusion permet une dissipation des variations, localement fortes, du scalaire.

Le nombre de Schmidt, sans dimension, est défini par le rapport entre la viscosité du fluide et le coefficient de diffusion du scalaire, $Sc = \nu/\kappa$. Dans la plupart des fluides, le nombre de Schmidt est grand, au moins égal à 1. Ce nombre permet de relier les échelles de dissipation du fluide et du scalaire à travers la relation :

$$\eta_\kappa \sqrt{Sc} = \eta_\nu.$$

En pratique, l'échelle de Batchelor est donc (beaucoup) plus petite que l'échelle de Kolmogorov. Cela implique que le transport du scalaire développe des structures plus petites que les plus petites structures développées par le fluide. Pour de grands nombre de Schmidt, la dynamique du scalaire est essentiellement donnée par l'advection du scalaire sous l'effet du fluide. Un cadre théorique décrit l'influence du nombre de Schmidt sur des quantités statistiques du scalaire. En particulier, la décroissance du spectre de variance du scalaire, en fonction du nombre d'onde k, est modélisé par une loi de puissance en $k^{-5/3}$ qui est associée au transfert d'énergie en cascade vers les petites échelles. Cette décroissance est suivie d'une loi en k^{-1} puis de la dissipation par les plus petites échelles. Cette lente décroissance renforce la nécessité d'une résolution des petites échelles de l'écoulement.

1.3.3. Formulation mathématique

Un problème de transport d'un scalaire dans un fluide incompressible turbulent est modélisé par les équations de Navier-Stokes (1.10) pour la partie fluide et une équation de convection et diffusion pour le transport du scalaire. Ces deux parties sont liées par le champ de vitesse. Le problème consiste en le système d'équations suivant, en considérant une densité $\rho = 1$ et en l'absence de forces extérieures $\boldsymbol{f}_e = 0$:

$$\begin{cases} \operatorname{div} \boldsymbol{u} = 0, \\ \dfrac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} = \left(\dfrac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} \right) = -\nabla P + \nu \Delta \boldsymbol{u}, \\ \dfrac{\mathrm{D}\theta}{\mathrm{D}t} = \dfrac{\partial \theta}{\partial t} + (\boldsymbol{u} \cdot \nabla)\theta = \kappa \Delta \theta \end{cases} \tag{1.37}$$

Les trois inconnues du problème sont le champ de vitesse $\boldsymbol{u}$, la pression P et le scalaire passif θ.

Dans le cadre d'un scalaire actif, l'effet sur l'écoulement est généralement pris en compte à travers une force $\boldsymbol{f}_\theta$ présente au second membre de la seconde équation du système. Ici, à travers le couplage faible entre les deux parties, nous obtenons une décomposition du problème global en deux sous-problèmes quasi indépendants. En effet, le champ de vitesse peut être vu simplement comme une donnée d'entrée du problème de transport de scalaire.

Conclusion

À partir d'un même problème mathématique, différentes méthodes numériques peuvent être envisagées. Le choix de l'une ou l'autre dépend des objectifs visés, des applications considérées et des architectures de calcul disponibles. Dans ce chapitre, nous avons exposé

les grandes lignes des méthodes classiques relativement à la résolution d'un problème de mécanique des fluides. L'utilisation d'une méthode numérique permet de discrétiser un problème continu pour en approcher une solution. Leur point commun est que la précision de la solution dépend de la finesse de la discrétisation et de l'ordre d'approximation du schéma numérique. Ces deux paramètres ont une forte incidence sur le temps de calcul. Ce dernier peut rapidement devenir prohibitif en l'absence d'utilisation de techniques de parallélisation et d'optimisation du code. L'exploitation des machines de calcul, dont l'architecture peut être fortement hétérogène, dans le cadre du calcul intensif, implique une prise en compte du matériel et des technologies employées dès la conception des algorithmes. L'importance de cette prise en compte est encore plus grande lors de l'utilisation d'une grande quantité de ressources, notamment dans la perspective de l'évolution des machines vers l'exascale. Le transport d'un scalaire passif par un écoulement turbulent nous conduit à résoudre un problème, composé de deux parties assez indépendantes, présentant plusieurs échelles. Dans la suite, nous mettons en place une résolution basée sur différentes méthodes ainsi que des algorithmes multiéchelles.

2. Méthode particulaire pour l'équation de transport

Comme nous l'avons exposé dans le chapitre 1, la modélisation du transport d'un scalaire passif dans un écoulement turbulent s'effectue à l'aide un système d'équations continues constitué des équations de Navier-Stokes et d'une équation d'advection-diffusion du scalaire. Ces deux équations présentent une équation de transport, une pour la vorticité et l'autre pour le scalaire. Parmi les différentes techniques de résolution possibles, ce travail se base sur le développement d'une méthode semi-Lagrangienne. En effet, elle permet de combiner les avantages des méthodes Lagrangiennes tout en se basant sur une structure de donnée régulière. Ainsi, cette méthode nous permet de choisir de grands pas de temps pour la réalisation de simulations numériques tout en bénéficiant de la régularité des données liée à la présence de la grille sous-jacente.

Dans une première partie de ce chapitre, nous présenterons un état de l'art des méthodes particulaires semi-Lagrangiennes. Elles consistent en une méthode Lagrangienne avec une étape de remaillage systématique à chaque itération permettant de remédier aux éventuels problèmes de distorsions du champ de particules. Elles se distinguent des méthodes semi-Lagrangiennes classiques *(backward)* par une descente des caractéristiques. Cette méthode particulaire semi-Lagrangienne *(forward)* explicite peut être formalisée comme un schéma aux différences finies à travers le remaillage. Une analyse des schémas ainsi obtenus est possible en utilisant une approche propre à la méthode des différences finies. Nous présenterons également les différentes techniques de construction des formules de remaillage ainsi qu'un exemple d'implémentation.

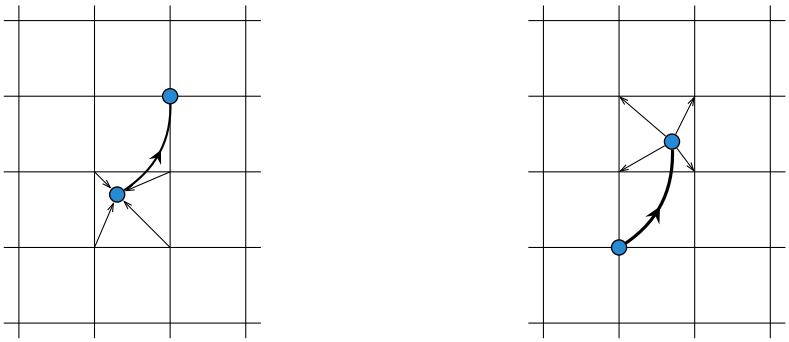
Dans une seconde partie, nous proposerons une méthode systématique pour obtenir des méthodes particulaires semi-Lagrangiennes d'ordres élevés. L'augmentation de l'ordre est obtenu par l'utilisation de formules de remaillage ayant une forte régularité et conservant un grand nombre de moments. Une nouvelle méthode de construction de ces formules permet de produire des formules d'ordre arbitraire. Des analyses de stabilité et consistance seront réalisées pour la méthode semi-Lagrangienne utilisant cette classe de formules de remaillage.

2.1. Méthode particulaire avec remaillage

2.1.1. Deux classes de méthodes semi-Lagrangiennes

À l'origine, les méthodes semi-Lagrangiennes ont été développées pour la résolution d'écoulements dominés par l'advection, essentiellement dans le domaine de la météorologie (Staniforth *et al.*, 1991). Leur principe général est basé sur l'exploitation des caractéristiques en chaque point d'une grille et de combiner les avantages des méthodes Eulériennes et Lagrangiennes. En effet, elles reposent sur le fait que la quantité transportée est invariante le long des caractéristiques. Dans une méthode semi-Lagrangienne rétrograde classique (*backward semi-Lagrangian method* en anglais) la résolution consiste à remonter la trajectoire depuis un point de grille et interpoler la quantité transportée au point de départ. Le résultat de l'interpolation donne directement la valeur de la quantité au point de grille considéré.

Une seconde classe dont principe est similaire repose sur une descente des caractéristiques est appelée *forward semi-Lagrangian method*. Après intégration de l'équation de mouvement, l'étape de distribution de la quantité transportée sur les points de grille est réalisée de manière implicite et nécessite la résolution d'un système dont la complexité dépend de la nature de l'interpolation employée. La seule différence se situe dans le traitement des contributions sur les points de grille. Une comparaison avec la méthode classique a été proposée par Crouseilles *et al.* (2009) sur différents modèles d'équations de Vlasov pour la simulation de plasmas. Nous illustrons les deux classes de méthodes par les figures 2.1.



(a) *Backward* **(b)** *Forward*

Figure 2.1. – Méthodes semi-Lagrangiennes

La méthode particulaire avec remaillage que nous détaillons dans la suite de ce chapitre est très similaire à la méthode *forward* mais en utilisant un schéma explicite. Notre méthode permet facilement de monter en ordre et est conservative.

2.1.2. Principe général de la méthode particulaire avec remaillage

Une méthode particulaire permet de résoudre une équation de transport :

$$\frac{\partial u}{\partial t} + \operatorname{div}(\boldsymbol{a}u) = 0, \tag{2.1}$$

où la quantité u est transportée à vitesse $\boldsymbol{a}$. Comme nous l'avons évoqué en section 1.1.2, la quantité u est discrétisée sur un ensemble de particules et est approchée en tout point de l'espace par l'expression :

$$u(\boldsymbol{x}) = \sum_p u_p W_\varepsilon(\boldsymbol{x} - \boldsymbol{x}_p),$$

avec u_p la quantité portée par la particule p, W_ε une fonction de régularisation et $\boldsymbol{x}_p$ la position d'une particule. L'évolution des particules est soumise au champ de vitesse de l'écoulement et leurs trajectoires sont données par les équations :

$$\frac{\mathrm{d}\boldsymbol{x}_p}{\mathrm{d}t} = \boldsymbol{a}(\boldsymbol{x}_p, t).$$

La présence d'un éventuel second membre dans l'équation de transport (2.1) se traduit par une équation d'évolution de la quantité transportée par les particules. Par exemple, pour une équation de transport de vorticité $\boldsymbol{\omega}$ à vitesse $\boldsymbol{u}$, issue des équations de Navier-Stokes :

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{\omega} = \boldsymbol{\omega} \cdot \nabla \boldsymbol{u} + \nu \Delta \boldsymbol{\omega},$$

le second membre conduit aux équations suivantes :

$$\frac{\mathrm{d}\boldsymbol{\omega}_p}{\mathrm{d}t} = \boldsymbol{\omega}_p \nabla \boldsymbol{u}(\boldsymbol{x}_p) + \nu \Delta \boldsymbol{\omega}(\boldsymbol{x}_p).$$

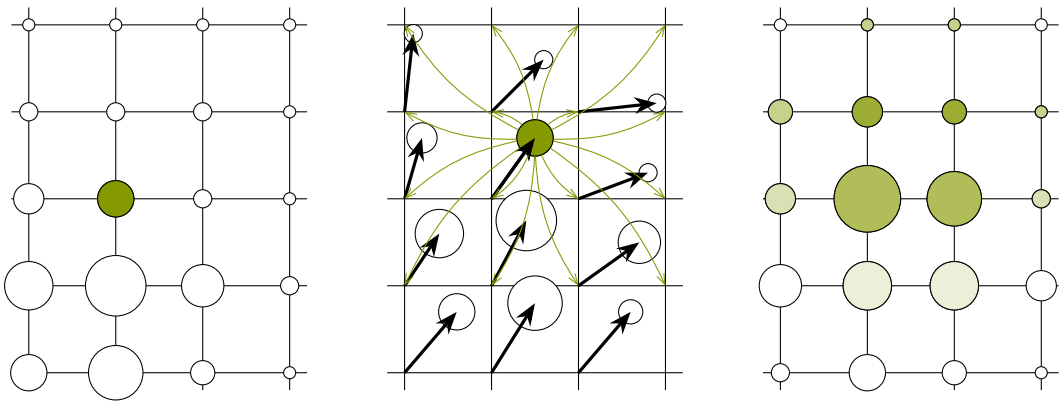
Un inconvénient bien connu des méthodes particulaires est qu'au cours de l'évolution de la simulation, les particules ont des trajectoires qui suivent le champ de vitesse. Par conséquent, il peut arriver que ces particules se retrouvent trop éloignées les unes des autres ou inversement regroupées dans une petite région du domaine. Ainsi, selon les zones de l'écoulement, on assiste à une dégradation de la représentation des champs par manque ou excès de particules. Un contrôle du recouvrement des particules est alors nécessaire pour assurer de la convergence de la méthode. Dans de nombreuses implémentations de cette méthode, une redistribution des particules sur une grille est appliquée régulièrement au cours de la simulation afin de garantir un recouvrement suffisant et les propriétés de consistance de la méthode (Koumoutsakos *et al.*, 1995). Une méthode semi-Lagrangienne est obtenue lorsque le remaillage est effectué systématiquement après chaque étape d'advection. Cela permet également une représentation des variables sur la grille et d'exploiter des méthodes de décomposition de domaine (Cottet, 1991 ; Ould Salihi *et al.*, 2000 ; Cottet *et al.*, 2000 ; Cocle *et al.*, 2008), d'adaptation automatique de maillage (Bergdorf *et al.*, 2005) ou de multirésolution basée sur des ondelettes (Bergdorf *et al.*, 2006). Ainsi, les termes au second membre des équations de transport peuvent être résolus sur la grille et non sur les particules.

Par conséquent, la quantité transportée par les particules est conservée lors de l'advection car la trajectoire des particules suit les caractéristiques et on a :

$$\frac{\mathrm{d}u_p}{\mathrm{d}t} = 0.$$

Ainsi, dans ce type de méthode, les particules sont créées au début de chaque étape d'advection et sont détruites par le remaillage. En pratique, on ne crée des particules qu'aux points de grille où la quantité transportée par les particules est supérieure à une certaine valeur seuil fixée à l'avance. Cela permet d'éviter le traitement, par advection puis remaillage, de particules dont la quantité transportée est nulle ou négligeable.

Le remaillage consiste en une interpolation des particules sur la grille à l'aide d'un noyau de remaillage, également appelé formule de remaillage. Le cas multidimensionnel est généralement traité par produit tensoriel de noyaux monodimensionnels. Cependant, comme nous le verrons par la suite, ces formules ont généralement un coût arithmétique élevé en 3D si des noyaux avec un large support sont utilisés. La figure 2.2 illustre une étape de résolution pour un cas 2D où la taille des disques représente les valeurs portées par les particules. Après un déplacement des particules selon le champ de vitesse représenté par les flèches en trait épais sur la figure 2.2b, les particules sont remaillées sur les S points de grille voisins. Les contributions d'une particule sur les points de grille sont représentées par des flèches en trait fin. La particule colorée sur la figure 2.2a est répartie dans les valeurs des points de grille colorés selon la contribution sur la figure 2.2c.

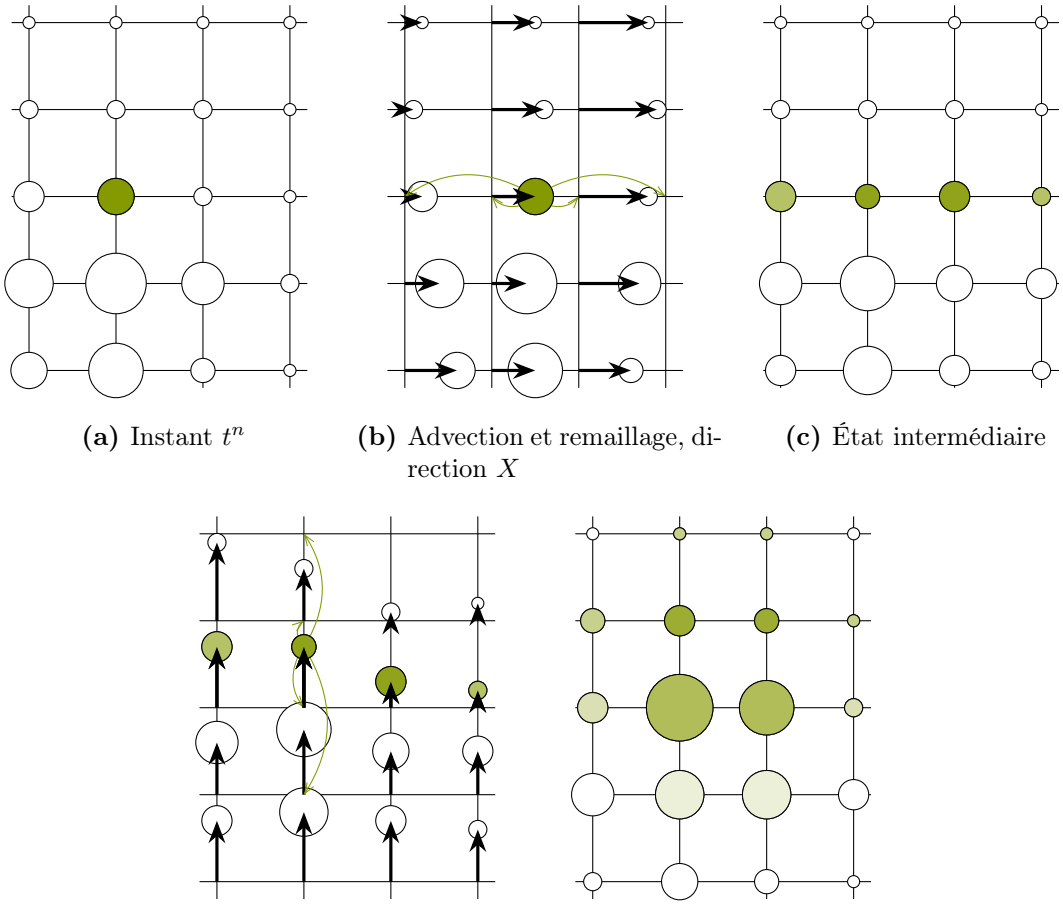


(a) Instant t^n **(b)** Advection et remaillage **(c)** Instant t^{n+1}

Figure 2.2. – Remaillage par produit tensoriel

Magni *et al.* (2012) proposent une alternative à la résolution multidimensionnelle basée sur un splitting dimensionnel. Cette méthode permet de découpler les directions spatiales à travers la résolution de problèmes monodirectionnels en alternant les directions. Le splitting de Strang, d'ordre deux en temps, conduit à résoudre, pour un problème 2D, une première direction sur l'intervalle $[t^n; t^{n+1/2}[$, la seconde direction sur un pas de temps complet et revenir à la première direction pour terminer l'itération sur $[t^{n+1/2}; t^{n+1}[$. En trois dimensions, l'algorithme est identique, toutes les étapes sont réalisées sur des demi pas de temps excepté la troisième direction. Nous illustrons ce splitting dimensionnel en figure 2.3 en utilisant les

mêmes conventions que pour l'illustration précédente. Dans un souci de clarté, l'exemple ne concerne qu'un splitting du premier ordre qui, selon le même principe, consiste en une alternance des directions pour des pas de temps complets. À chaque étape, une particule est remaillée seulement sur les S points voisins dans la direction concernée, soit respectivement $3S$ et $5S$ contributions par particule pour une itération complète de la formule de Strang en 2D et en 3D, au lieu de S^2 et S^3 pour un remaillage tensoriel.



(a) Instant t^n

(b) Advection et remaillage, direction X

(c) État intermédiaire

(d) Advection et remaillage, direction Y

(e) Instant t^{n+1}

Figure 2.3. – Remaillage par splitting dimensionnel du premier ordre

À travers l'utilisation d'un splitting dimensionnel de Strang, la résolution d'une équation de transport multidimensionnelle se réduit à une succession de problèmes 1D. Cela permet d'obtenir un coût de calcul proportionnel au nombre de particules dont la constante multiplicative dépend uniquement de la formule de remaillage. Dans un remaillage tensoriel, chaque particule contribue une fois par itération à S^3 (S^2) points de grilles voisins en 3D (2D). En revanche, dans notre cas, une itération est constituée de plusieurs étapes dans lesquelles une particule contribue simplement aux S points de grille. D'autres avantages en termes de complexité algorithmique seront donnés ultérieurement.

2.1.3. Cas monodimensionnel

La méthode semi-Lagrangienne d'advection avec remaillage est donnée ici pour une équation de transport monodimensionnelle :

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(au) = 0. \tag{2.2}$$

On note la vitesse, a et la quantité transportée par les particules, u. À chaque instant, ces deux variables sont connues en chaque point de la grille.

À chaque itération temporelle de la méthode, les particules sont initialisées à un instant t^n, en chaque point de grille $\tilde{x}_i^n = x_i$ avec $\tilde{u}_i^n = u_i^n$ pour la quantité transportée. Puis l'équation de mouvement est résolue pour chaque particule à l'aide d'un schéma d'intégration en temps. La nouvelle position de la particule i, $\tilde{x}_i^{n+1}$ est la solution du problème suivant :

$$\begin{cases} \dfrac{\mathrm{d}\tilde{x}_i}{\mathrm{d}t} = a(\tilde{x}_i) \quad t \in [t^n, t^{n+1}[, \\ \tilde{x}_i^n = x_i. \end{cases} \tag{2.3}$$

En pratique, pour chaque particule, le problème (2.3) est résolu par un schéma d'intégration explicite. La vitesse de la particule i, $a(\tilde{x}_i)$, est supposée constante tout au long de l'intervalle de temps $[t^n; t^{n+1}[$ et son approximation est notée $\tilde{a}^n(\tilde{x}_i)$. Dans un cadre multiéchelle, la vitesse est connue sur une grille plus grossière que celle du scalaire, on utilise alors une interpolation linéaire du champ de vitesse pour obtenir l'approximation $\tilde{a}^n$ en un point de la grille fine. Dans le cas d'une même résolution pour les deux grilles, l'approximation du champ de vitesse sur un point de grille est directement donnée par la valeur de la vitesse en ce point, sans interpolation, $\tilde{a}^n(x_i) = a_i^n$. Pour une intégration du premier ordre, avec un schéma d'Euler, la position de la particule après advection s'exprime par :

$$\tilde{x}_i^{n+1} = x_i + a_i^n \Delta t. \tag{2.4}$$

Comme nous l'avons exposé précédemment, un splitting de Strang du second ordre est utilisé pour les cas multidimensionnels. Cela implique l'utilisation d'un schéma d'intégration d'ordre plus élevé afin de conserver le second ordre en temps. En utilisant un schéma de Runge-Kutta, du second ordre, la position de la particule est donnée par l'expression :

$$\tilde{x}_i^{n+1} = x_i + \tilde{a}^n \left(x_i + a_i^n \frac{\Delta t}{2} \right) \Delta t. \tag{2.5}$$

Ici encore, la valeur du champ de vitesse a_i^n n'est utilisée directement que dans les cas d'une même résolution pour la vitesse et le scalaire, sinon une interpolation linéaire supplémentaire est nécessaire.

Après déplacement des particules, l'étape de remaillage permet de redistribuer les quantités portées par les particules sur la grille. Cette étape est réalisée à l'aide du noyau de remaillage W :

$$u_i^{n+1} = \sum_p u_p^n W \left(\frac{\tilde{x}_p^{n+1} - x_i}{\Delta x} \right) \tag{2.6}$$

2.1.4. Lien avec la méthode des différences finies

Comme nous l'avons vu précédemment, une particule initialement au point de grille x_p se retrouve après advection, par la résolution de (2.3), en position $\tilde{x}_p$, transportant la quantité u_p. Ensuite la particule est remaillée sur les $2S$ points de grille voisins de $\tilde{x}_p$ par un noyau de remaillage $\Lambda_{p,r}$. Si on note x_i le point de la grille le plus proche à gauche de $\tilde{x}_p$, la quantité u_p est distribuée sur les points de grille d'indices $[i - S + 1; i + S]$, comme l'illustre la figure 2.4.

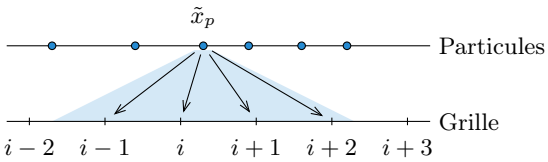


Figure 2.4. – Remaillage d'une particule pour un noyau de support $[-2; 2]$

De même, un point de la grille se voit collecter toutes les contributions de particules situées à sa proximité indépendemment du nombre de particules. En effet, le nombre de particules par cellules peut changer (c.-à-d. être différent de un) selon les variations locales du champ de vitesse. Toutefois, les particules ne peuvent se croiser au cours de l'advection lorsque le pas de temps est contraint par une condition CFL Lagrangienne :

$$\Delta t ||a'||_\infty \leqslant C \tag{2.7}$$

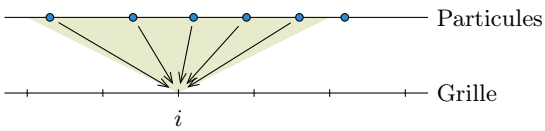


Figure 2.5. – Remaillage sur un point de grille pour un noyau de support $[-2; 2]$

La méthode d'advection suivie d'un remaillage montre une similarité avec la méthode des différences finies. Nous l'illustrons sur un exemple linéaire simple, lorsque le nombre CFL des particules, $\lambda_i = a_i \Delta t / \Delta x$, est inférieur à 1 et que la vitesse est positive. En utilisant la formule de remaillage à deux points (2.10), les poids sont donnés par :

$$\alpha_i = 1 - \lambda_i, \qquad \beta_i = \lambda_i.$$

Cette configuration est schématisée sur la figure 2.6.

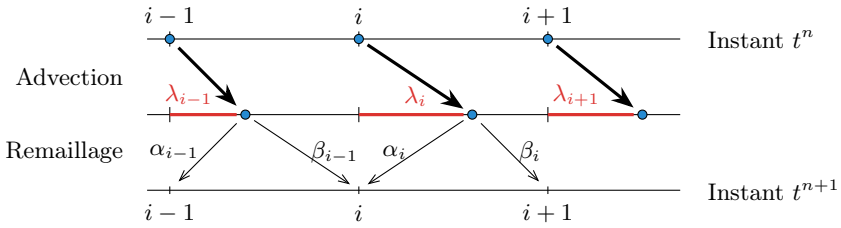


Figure 2.6. – Analogie entre le remaillage et les différences finies

À l'issue du remaillage, le point de grille d'indice i collecte des contributions des particules $i-1$ et i :

$$\begin{aligned} u_i^{n+1} &= \beta_{i-1} u_{i-1}^n + \alpha_i u_i^n \\ &= \lambda_{i-1} u_{i-1}^n + (1-\lambda_i) u_i^n \\ &= \frac{a_{i-1}^n u_{i-1}^n \Delta t}{\Delta x} + u_i^n - \frac{a_i^n u_i^n \Delta t}{\Delta x} \end{aligned}$$

On reconnaît ici aisément un schéma aux différences finies du premier ordre consistant avec l'équation de transport :

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{(au)_{i-1}^n - (au)_i^n}{\Delta x}$$

Cette approche permet à Cottet *et al.* (2006) de démontrer le parallèle entre cette méthode semi-Lagrangienne et le schéma de Lax-Wendroff pour l'équation de Burgers. Les auteurs distinguent les cas linéaires et non linéaires. Pour une équation à vitesse constante, l'utilisation d'une formule de remaillage de support $[-1.5; 1.5]$ conservant les trois premiers moments conduit, lorsque la CFL est inférieure à $1/2$, à un schéma aux différences fines de Lax-Wendroff. Pour une CFL dans $[1/2; 1]$, le schéma obtenu est décentré. Une formule de remaillage de support $[-2; 2]$ conservant quatre moments est également analysée et conduit à un schéma d'ordre 3. Le cas non linéaire est bien plus complexe et n'est étudié que pour la formule Λ_2 et une CFL inférieure à $1/2$. Le schéma aux différences finies obtenu est d'ordre deux si les particules sont advectées avec un schéma également d'ordre deux. Un point intéressant de cette étude est que l'ordre des schémas obtenus ne dépend pas explicitement de la formule de remaillage mais simplement du nombre de moments qu'elle conserve.

L'exploitation des outils de la méthodologie des différences finies permet à Cottet *et al.* (2009b) de développer des limiteurs pour les formules de remaillage Λ_2. Les auteurs se restreignent à des CFL inférieures à 1 dans les cas linéaires et non linéaires pour l'équation de Burgers. En pratique des CFL plus grandes sont traitées en deux étapes. Les particules sont d'abord advectées sans remaillage sur la partie entière de la CFL puis advectées et remaillées avec une CFL inférieure à 1.

Plus récemment, Magni *et al.* (2012) étendent la construction de formules de remaillage avec limiteurs d'ordre 2 et 4 corrigées pour pouvoir traiter le cas de grands pas de temps. En effet, lors d'une advection avec une CFL supérieure à 1, les nombres CFL peuvent varier d'une particule à l'autre ce qui entraîne une perte de consistance avec l'équation de transport. L'idée développée consiste en un regroupement des particules en blocs selon leur nombre CFL local. Les particules d'un même blocs sont remaillées soit avec une formule de remaillage centrée soit décentrée à gauche. Ainsi, au sein des blocs, la consistance est assurée directement. Par contre, entre deux blocs de types différents, la consistance de la méthode nécessite une correction des formules de remaillage selon les nombres CFL locaux des particules à l'interface entre les blocs. Les corrections consistent en une modification du nombre de points de grille sur lesquels ces particules sont remaillées. La consistance pour les grands pas de temps conduit à des formules de remaillage de plus grande complexité algorithmique du fait de l'étape de construction des blocs et de choix de la correction à appliquer.

Enfin, Weynans *et al.* (2013) proposent une extension des formules de remaillage avec limiteurs dans le cas non linéaire pour une vitesse de signe quelconque. Les auteurs proposent également une preuve de convergence de la méthode en utilisant leurs schémas. Cette

démonstration est réalisée dans le cadre de la résolution d'une équation de Burgers. Il s'agit, à notre connaissance, de la seule preuve de convergence des méthodes particulaires pour l'équation de Burgers.

2.1.5. Construction des formules de remaillage

Du fait de leur nature, les méthodes particulaires sont adaptées à la résolution d'équations de conservation. Il est donc impératif que le remaillage, par l'équation (2.6), possède également ces propriétés de conservation des moments des quantités transportées. Mathématiquement le moment discret d'ordre p d'une quantité u est défini par :

$$\sum_i x_i^p u_i. \tag{2.8}$$

Par conséquent, un noyau de remaillage doit satisfaire les égalités de conservation des moments discrets d'ordre α suivantes :

$$\sum_i x_i^\alpha u_i^{n+1} = \sum_i x_i^\alpha u_i^n, \quad 0 \leqslant \alpha \leqslant p,$$

ou encore, en utilisant la relation (2.6) :

$$\sum_{k\in\mathbb{Z}} k^\alpha W(x-k) = x^\alpha, \quad 0 \leqslant \alpha \leqslant p, x \in \mathbb{R}. \tag{2.9}$$

Construction à partir des moments discrets

La première catégorie de noyaux est obtenue directement à partir des relations (2.9) et les noyaux sont notés Λ_p, pour un noyau conservant les moments d'ordre 0 à p. Le support est fixé à une largeur égale à $p+1$ et le noyau est défini par une fonction polynomiale par morceaux de $p+1$ polynômes de degrés p. Les p^2+p coefficients sont alors obtenus de manière à vérifier les relations (2.9). Cette méthode permet notamment d'obtenir le noyau d'interpolation linéaire :

$$\Lambda_1(x) = \begin{cases} 1-|x| & \text{si } |x| \leqslant 1 \\ 0 & \text{sinon} \end{cases} \tag{2.10}$$

Les quatre premiers noyaux de cette classe sont représentés en figure 2.7. Comme le montre la figure 2.7, certains de ces noyaux ne sont pas continus. Les pertes de précisions numérique dues au manque de régularité de ces noyaux ont été étudiées par Cottet *et al.* (2009b). L'ordre du remaillage peut être conservé à l'aide de corrections locales.

Construction à partir de B-splines

Une seconde classe de noyaux est obtenue à partir de B-splines régulières (Schoenberg, 1946) :

$$M_p(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\left(\frac{\sin\xi/2}{\xi/2}\right)^p e^{i\xi x}\mathrm{d}\xi. \tag{2.11}$$

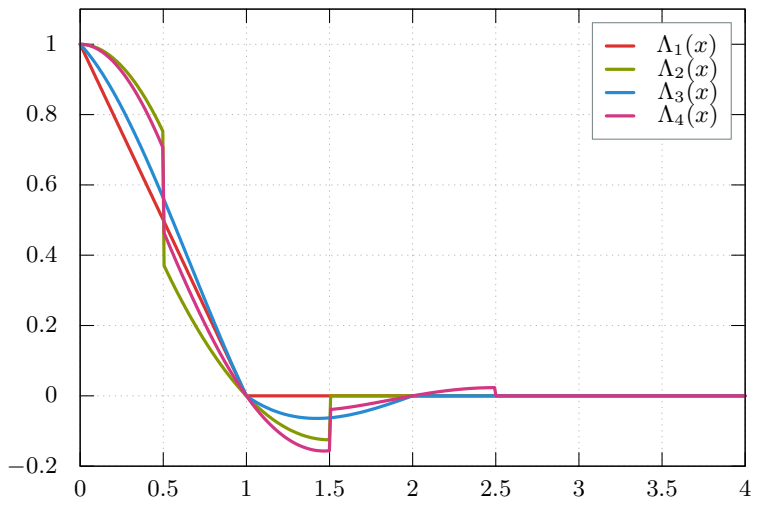


Figure 2.7. – Noyaux de remaillage de type Λ_p

Ainsi un noyau M_p sera de classe C^{p-2} avec un support de largeur $2p-3$. La figure 2.8 représente quelques noyaux de type M_p. On remarque que M_1 est la fonction chapeau de support $[-1/2; 1/2]$ et que $M_2 = \Lambda_1$. Cependant, ces noyaux ne permettent qu'une conservation des deux premiers moments uniquement.

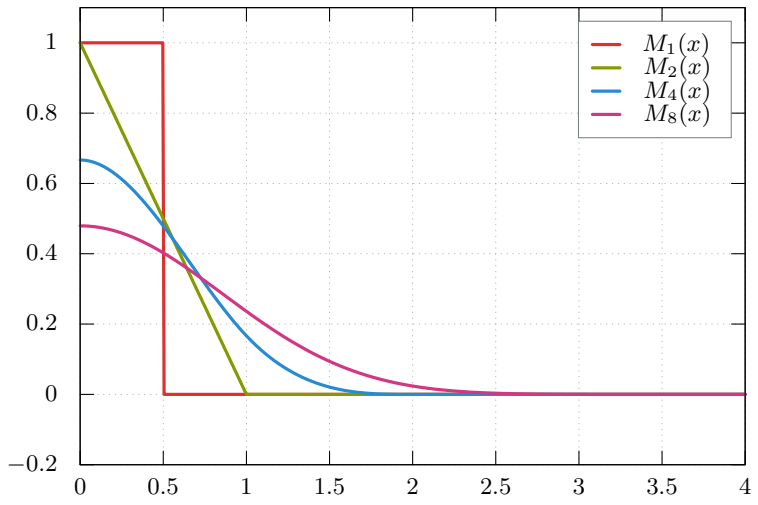


Figure 2.8. – Noyaux de remaillage de type M_p

Extrapolation à partir de noyaux réguliers

En utilisant l'extrapolation de Richarson, Monaghan (1985) produit des noyaux basés sur les fonctions M_p et leurs dérivées afin d'obtenir des ordres plus élevés en conservant leur régularité. On construit alors un noyau de la forme $W(x) = \sum_{c=0}^{p/2} c_k x^k M_p^{(k)}(x)$. Les coeffi-

cients c_k sont obtenus de telle sorte que le nouveau noyau vérifie les conditions suivantes :

$$\int W(x)\mathrm{d}x = 1,$$
$$\int x^\alpha W(x)\mathrm{d}x = 0, \quad 1 \leqslant \alpha \leqslant p,$$

ce qui conduit à la résolution d'un système linéaire. Seules les contraintes pour α pair sont à considérer, car les moments impairs sont nuls par symétrie des noyaux. Par exemple, pour le noyau M_4, on est ramené à la résolution du système :

$$\begin{pmatrix} \int M_4(x)\mathrm{d}x & \int xM_4'(x)\mathrm{d}x \\ \int x^2 M_4(x)\mathrm{d}x & \int x^3 M_4'(x)\mathrm{d}x \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 1/3 & -1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

dont la solution est $c_0 = 3/2$ et $c_1 = 1/2$. On obtient donc le noyau noté M_4'. Par construction, ce noyau est de classe C^1, comme M_4 est C^2, mais il conserve les moments jusqu'à l'ordre 3. Il est utilisé dans la plupart des implémentations depuis la fin des années 90 (Koumoutsakos, 1997 ; Ould Salihi, 1998).

$$M_4'(x) = \frac{1}{2}\left(3M_4(x) + x\frac{\mathrm{d}M_4}{\mathrm{d}x}(x)\right)$$

$$M_4'(x) = \begin{cases} 1 - \frac{5}{2}|x|^2 + \frac{3}{2}|x|^3 & \text{si } |x| \leqslant 1 \\ \frac{1}{2}(1-|x|)(2-|x|)^2 & \text{si } 1 < |x| \leqslant 2 \\ 0 & \text{sinon} \end{cases} \tag{2.12}$$

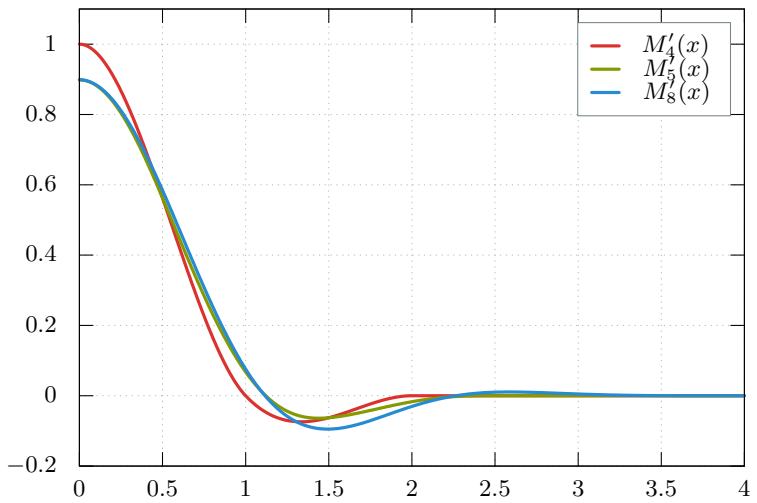


Figure 2.9. – Noyaux de remaillage type M_p'

Un comparatif de ces différents noyaux est donné dans le tableau 2.1 en termes de moments conservés, de régularité, de largeur de support et de degré polynomial.

Nom	Moment maximal conservé	Régularité	Support	Degré
Λ_1	1	C^0	$[-1;1]$	1
Λ_2	2	-	$[-1,5;1,5]$	2
Λ_3	3	C^0	$[-2;2]$	3
Λ_4	4	-	$[-2,5;2,5]$	4
M_1	1	-	$[-0,5;0,5]$	0
M_2	1	C^0	$[-1;1]$	1
M_4	1	C^2	$[-2;2]$	3
M_8	1	C^6	$[-4;4]$	7
M'_4	2	C^1	$[-2;2]$	3
M'_5	3	C^2	$[-2,5;2,5]$	4
M'_8	4	C^4	$[-4;4]$	7

Tableau 2.1. – Comparatif des différents noyaux de remaillage de la littérature.

2.1.6. Exemple d'implémentation

La principale implémentation de méthodes semi-Lagrangiennes (également appelées méthodes particules-grille) a été proposée par Sbalzarini *et al.* (2006). La librairie PPM *(Parallel Particle-Mesh)* repose sur une unification de méthodes de grilles et de méthodes particulaires à travers des interpolations entre les particules et la grille. Cette librairie propose une interface pour l'exploitation de machines à mémoire distribuée de manière transparente pour l'utilisateur. Le parallélisme est géré par des méthodes de décomposition de domaine et d'adaptation de charge. Des algorithmes efficaces pour l'évaluation des interactions entre particules, des méthodes de grilles globales utilisant des FFT et des solveurs multigrille ainsi que des interpolations entre les particules et la grille conduisent à de bonnes performances numériques.

Dans la suite de ce chapitre nous exposons une méthode de construction de formules de remaillage dont les propriétés de régularité et du nombre de moments conservés sont choisis arbitrairement. Nous considérons ensuite ces formules dans des études de consistance et de stabilité des schémas ainsi obtenus. Nous observons que l'ordre de la méthode semi-Lagrangienne dépend directement du nombre de moments conservés par la formule de remaillage ainsi que de sa régularité. Les détails techniques des étapes d'advection et remaillage propres à l'implémentation seront donnés dans les chapitres 5 et 6.

2.2. Advection semi-Lagrangienne d'ordre élevé

2.2.1. Construction de formules de remaillage d'ordre élevé

Une caractéristique intéressante des noyaux de type Λ_p est la propriété d'interpolation suivante :

$$\Lambda_p(x) = \begin{cases} 1, & \text{si } x = 0 \\ 0, & \text{si } x \in \mathbb{Z}^*. \end{cases} \tag{2.13}$$

Cette propriété permet une conservation exacte de la quantité remaillée pour les particules dont la position coïncide avec un point de grille, en particulier si la vitesse est nulle. En pratique, l'utilisation de noyaux possédant cette propriété donne des résultats plus précis. Cependant, les noyaux réguliers de type M_p et M'_p perdent cette propriété. Dans cette section, nous donnons une méthode de construction de noyaux de remaillage d'ordre élevé, réguliers et vérifiant la propriété (2.13). Le principe est similaire à celui employé pour la construction des noyaux de type Λ_p, donnée en Section 2.1.5. Il s'agit de trouver une fonction vérifiant les propriétés suivantes :

P1 : parité ;

P2 : support compact $[-S, S]$;

P3 : polynomiale par morceaux de degré q sur les intervalles entiers ;

P4 : régularité C^r ;

P5 : conservation les moments jusqu'à l'ordre p, selon l'égalité (2.9) ;

P6 : vérification de la propriété d'interpolation (2.13).

Ces propriétés conduisent à l'expression d'une régularité des noyaux $\Lambda_{p,r}$ en terme d'espaces fonctionnels :

$$\begin{cases} \Lambda_{p,r} \in C^\infty\left(]l, l+1[\right), \quad l \in \mathbb{Z}, \\ \Lambda_{p,r} \in W^{r+1,\infty}(\mathbb{R}) = \left\{ f \in L^\infty(\mathbb{R}), \forall l \in \mathbb{N}^*, l \leqslant r+1, \mathrm{D}^l f \in L^\infty \right\}, \end{cases} \tag{2.14}$$

où $\mathrm{D}^l f$ est la dérivée d'ordre l de f au sens des distributions.

Les propriété **P1**, **P2** et **P3** permettent de restreindre la recherche à seulement S polynômes de degré q, soit $S(q+1)$ coefficients inconnus. Le noyau s'écrit donc de manière générale :

$$\Lambda_{p,r}(x) \quad = \quad \begin{cases} Q_i(x), & \text{si } i \leqslant |x| < i+1,\ i = 0 \dots S-1 \\ 0 & \text{sinon,} \end{cases}$$

avec

$$Q_i(x) = \sum_{k=0}^{q} c_k^i |x|^k.$$

Avec ces notations, la propriété **P4** implique que $q \geqslant r$ ainsi que les $(r+1)(S+1)$ contraintes de raccords entre les polynômes aux points de coordonnées entières du support :

$$\begin{aligned} Q_i^{(j)}(i+1) &= Q_{i+1}^{(j)}(i+1), \quad 0 \leqslant j \leqslant r, \quad 1 \leqslant i \leqslant S-1, \\ Q_0^{(j)}(0) &= \begin{cases} 1 & \text{si } j = 0 \\ 0 & \text{sinon} \end{cases}, \quad 0 \leqslant j \leqslant r, \\ Q_{S-1}^{(j)}(S) &= 0, \quad 0 \leqslant j \leqslant r. \end{aligned}$$

Les $p+1$ égalités polynomiales imposées par la propriété **P5** se traduisent par $(1+p)(q+1)$ contraintes sur les coefficients polynomiaux. Cependant, par parité des noyaux, les moments

Nom	Moment maximal conservé	Régularité	Support	Degré
$\Lambda_{2,1}$	2	C^1	$[-2;2]$	3
$\Lambda_{2,2}$	2	C^2	$[-2;2]$	5
$\Lambda_{4,2}$	4	C^2	$[-3;3]$	5
$\Lambda_{4,4}$	4	C^4	$[-3;3]$	9
$\Lambda_{6,4}$	6	C^4	$[-4;4]$	9
$\Lambda_{6,6}$	6	C^6	$[-4;4]$	13
$\Lambda_{8,4}$	8	C^4	$[-5;5]$	9

Tableau 2.2. – Comparatif de quelques noyaux de remaillage de type $\Lambda_{p,r}$.

d'ordre impair ne conduisent pas à des contraintes supplémentaires car elles sont vraies quelque soient les coefficients c_k^i. On ne retient donc que $(1 + [p/2])(q + 1)$ contraintes. Enfin, la propriété d'interpolation conduit à $S - 1$ contraintes supplémentaires, les valeurs aux points de coordonnées 0 et S étant redondantes avec les contraintes issues de **P4**. Nous avons donc au total $S(r+2) + (1+[p/2])(q+1) + r$ contraintes. Des solutions uniques sont envisageables si :

$$S(q + 1) \leqslant S(r + 2) + (1 + [p/2])(q + 1) + r. \tag{2.15}$$

Pour un couple de (S, q) donné et en considérant l'ordre p et la régularité r comme fixés, l'ensemble de ces contraintes conduit à un système linéaire dont les inconnues sont les coefficients polynomiaux c_i^k. Comme les contraintes de conservation des moments d'ordre impair sont vérifiées par parité, un choix naturel est de prendre p pair. Dans ce cas, en prenant $S = 1 + p/2$, l'inégalité 2.15 est vérifiée car $S(r + 2) + r \geqslant 0$. Expérimentalement, nous obtenons des solutions uniques en prenant, en plus des deux hypothèses précédentes, $q = 2r + 1$. Les noyaux obtenus par cette méthode sont notés $\Lambda_{p,r}$. Nous retrouvons le noyau $M_4' = \Lambda_{2,1}$ ainsi que le noyau $\Lambda_{4,2}$ qui a été introduit par Bergdorf *et al.* (2006) sous la notation M_6'''. Les caractéristiques des noyaux utilisés dans la suite sont résumées dans le tableau 2.2 et leurs expressions sont données en annexe A. Une représentation de ces noyaux est donnée en figure 2.10.

Cette méthode de construction de formules de remaillage permet de contrôler explicitement le nombre de moments conservés p ainsi que la régularité du noyau r. Ces deux paramètres, nous le verrons au cours de l'analyse de consistance qui suit, sont en lien direct avec la précision du schéma et l'ordre numérique de la méthode. Ainsi, une méthode d'ordre élevé sera obtenue en utilisant des formules avec p et r grands. Cependant, l'augmentation du nombre de moments conservés par les formules de remaillage conduit à un élargissement du support des noyaux. Cela a pour conséquence qu'un plus grand nombre de points de grille seront contenus dans le support de la formule impliquant une plus grande complexité en terme d'accès aux données lors de l'implémentation. D'autre part, l'augmentation de la régularité de la formule implique une augmentation du degré des polynômes constituant les noyaux. L'incidence de cette augmentation se traduit par une plus grande complexité numérique lors de l'évaluation de ces fonctions pour le calcul des poids de remaillage.

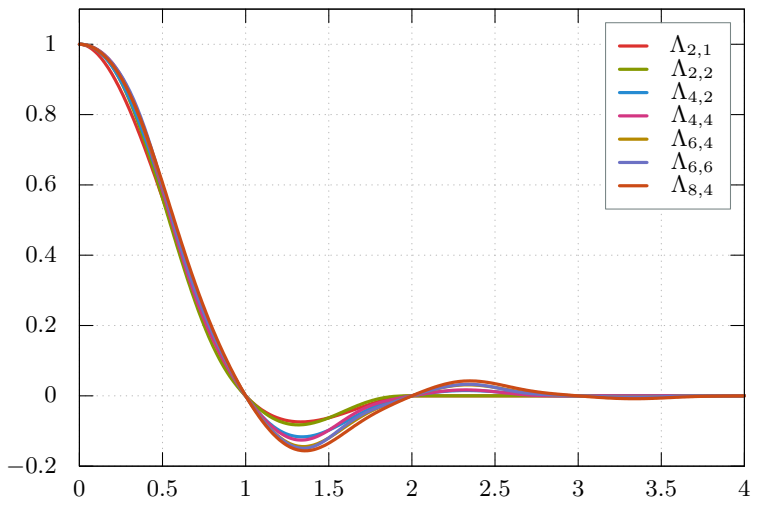


Figure 2.10. – Noyaux de remaillage de type $\Lambda_{p,r}$

Utilisation des poids de remaillage

En pratique, on exprime les poids de remaillage en fonction de la distance entre la particule et le point de grille le plus proche. Le support est découpé en intervalles de longueur égale à la distance entre deux points de grille sur lesquels la formule s'identifie à un polynôme. Ainsi, on a exactement un point de grille dans chaque morceau d'intervalle, ce qui conduit à l'expression de $2S$ poids de remaillage. La figure 2.11 illustre l'obtention des poids de remaillage pour une formule $\Lambda_{2,1}$, de support de longueur 4.

$$\Lambda_{2,1}(x) = \begin{cases} 1 - \frac{5}{2}|x|^2 + \frac{3}{2}|x|^3 = p_0(|x|) & 0 \leqslant |x| < 1 \\ 2 - 4|x| + \frac{5}{2}|x|^2 - \frac{1}{2}|x|^3 = p_1(|x|) & 1 \leqslant |x| < 2 \\ 0 & |x| \geqslant 2 \end{cases}$$

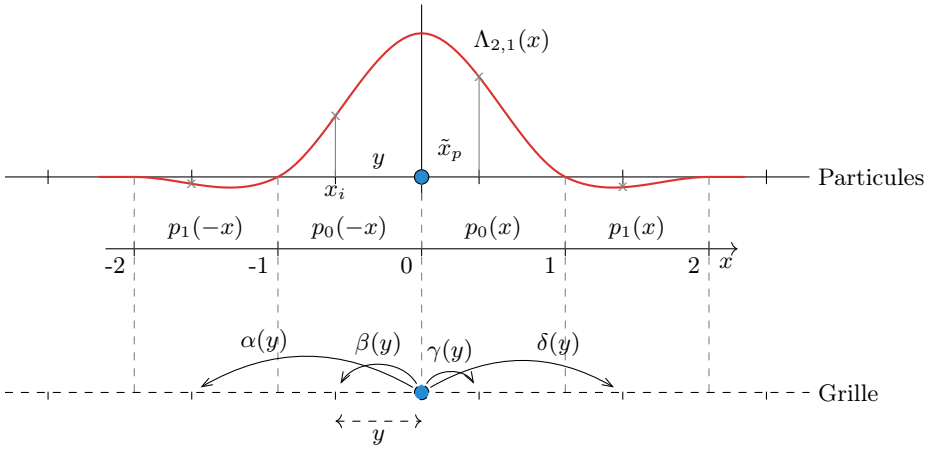


Figure 2.11. – Obtention des poids de remaillage pour une formule de support $[-2; 2]$

Lorsque x_i est le point de grille le plus proche de coordonnée inférieure à $\tilde{x}_p$, y est défini comme la distance normalisée entre ces deux points : $y = (\tilde{x}_p - x_i)/\Delta x$, $y \in [0;1]$. De ce fait, on obtient les poids en fonction de y :

$$\begin{cases} \alpha(y) = \Lambda_{2,1}(-1-y) = p_1(y) = y + y^2 - \frac{1}{2}y^3, \\ \beta(y) = \Lambda_{2,1}(-y) = p_0(y) = 1 - \frac{5}{2}y^2 + \frac{3}{2}y^3, \\ \gamma(y) = \Lambda_{2,1}(1-y) = p_0(1-y) = \frac{1}{2}y + 2y^2 - \frac{3}{2}y^3, \\ \delta(y) = \Lambda_{2,1}(2-y) = p_1(1-y) = -\frac{1}{2}y^2 + \frac{1}{2}y^3. \end{cases}$$

Les poids sont donnés pour les formules de type $\Lambda_{p,r}$ en annexe A. Pour les formules d'ordre élevé, les poids de remaillage sont de degré assez grand. Une implémentation efficace de l'évaluation de ces polynômes est réalisée par la méthode de Horner :

$$\begin{cases} \alpha(y) = y(y(-y+2)-1)/2, \\ \beta(y) = (y^2(3y-5)+2)/2, \\ \gamma(y) = y(y(-3y+4)+1)/2, \\ \delta(y) = y^2(y-1)/2. \end{cases}$$

Ainsi ce schéma permet de minimiser le nombre d'opérations arithmétiques pour l'évaluation de ces polynômes en évitant notamment le calcul des termes en y^n. L'évaluation d'un polynôme de degré q ne nécessite que $2q$ additions et multiplications au lieu des $q(q+1)/2$ pour un algorithme naïf.

Précision numérique du schéma d'Horner

Lors de l'évaluation d'un polynôme par le schéma d'Horner, la précision numérique atteinte, en tenant compte des erreurs d'arrondi, dépend du degré et du conditionnement du polynôme. L'erreur relative pour l'évaluation d'un polynôme $p(x) = \sum_{i=0}^{q} a_i x^i$ par un schéma de Horner est bornée par (Higham, 2002) :

$$\frac{|p(x) - \mathrm{Horner}(p,x)|}{|p(x)|} \leqslant K_q \,\mathrm{cond}(p,x) = K_q \frac{\sum_{i=0}^{q} |a_i||x^i|}{|\sum_{i=0}^{q} a_i x^i|}.$$

La constante K_q dépend du degré q du polynôme et de l'erreur relative lors de l'arrondi des nombres en virgule flottante, $u = 2^{-53}$ en double précision selon la norme IEEE-754. De manière générale, on a :

$$K_q = \gamma_{2q} = \frac{2qu}{1-2qu}\ ,$$

et lorsque des opérations FMA *(Fused Multiply and Add)* sont disponibles, comme c'est le cas sur la plupart des GPU, l'erreur est plus faible. En effet dans une FMA, la multiplication et l'addition sont effectuées en une seule opération arithmétique conduisant à une seule erreur d'arrondi au lieu de deux, la constante s'écrit alors :

$$K_q = \gamma_q = \frac{qu}{1-qu}\ .$$

Les polynômes permettant de calculer les poids de remaillage pour les formules $\Lambda_{p,r}$ ont un conditionnement qui augmente fortement au voisinage de 1. Cela conduit à des erreurs pouvant être assez grandes lorsque les particules s'approchent d'un point de grille par position inférieure ($y \to 1$). Ce phénomène est d'autant plus marqué que l'ordre des formules est élevé. Nous représentons sur la figure 2.12 la borne d'erreur relative pour les poids de remaillage pour trois des formules couramment utilisées.

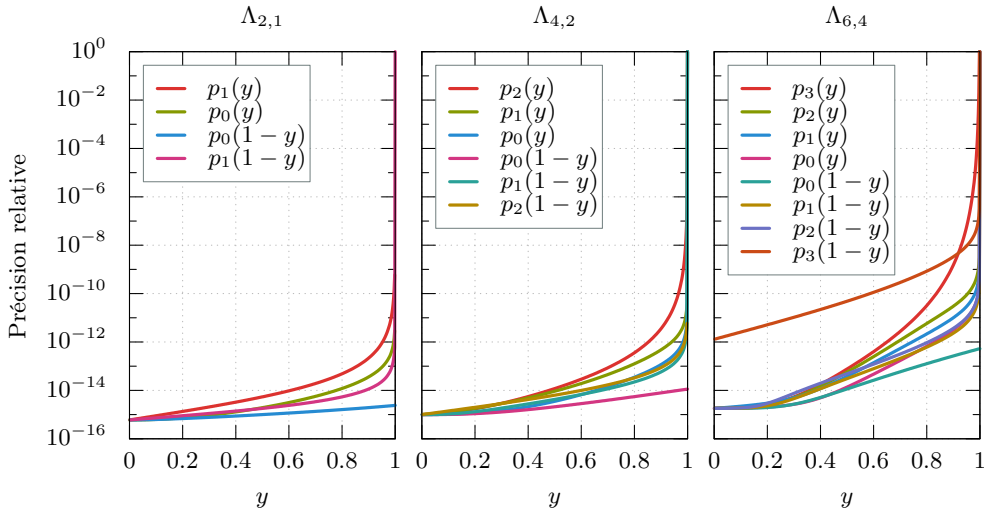


Figure 2.12. – Erreur numérique maximale du schéma de Horner pour l'évaluation des poids de remaillages pour les formules $\Lambda_{2,1}$, $\Lambda_{4,2}$ et $\Lambda_{6,4}$ en double précision (sans FMA)

Dans le cadre de ce travail, nous avons conservé le schéma de Horner pour l'implémentation de ces formules.

2.2.2. Consistance de la méthode semi-Lagrangienne

Dans ce paragraphe nous démontrons la consistance du schéma d'advection et remaillage (2.4) ou (2.5) et (2.6) lorsque la formule de remaillage est de type $\Lambda_{p,r}$, définie en section 2.2.1. Cette démonstration est présentée dans un article paru en 2014 dans *ESAIM : Mathematical Modelling and Numerical Analysis* pour un schéma d'Euler, inclus en annexe B. Nous détaillons ici le cas d'une approximation d'ordre deux de la vitesse lors de l'advection. La démonstration suit une démarche similaire à celle de l'article, la seule différence est que les développements sont conservés à un ordre quelconque.

Proposition 2.1

Soit un noyau de remaillage $\Lambda_{p,r}$ de support $[-S;S]$, vérifiant les contraintes de régularité (2.14), la propriété d'interpolation (2.13) ainsi que l'égalité de conservation des moments (2.9) jusqu'à un ordre p. Soient un champ de vitesse a et un scalaire u suffisamment réguliers tels que u soit solution de l'équation de transport (2.2). Lorsque la condition (2.7) est vérifiée, α est l'ordre d'approximation du champ de vitesse et $\beta = \inf(p,r)$, nous avons :

$$u(x_i, t^{n+1}) = u_i^{n+1} + \mathcal{O}(\Delta t^{\alpha+1}) + \mathcal{O}(\Delta t \Delta x^{\beta} + \Delta x^{\beta+1}). \tag{2.16}$$

De plus, dans le cas où chaque cellule contient une seule particule, $\beta = p$.

Démonstration de la proposition 2.1 : L'indice p d'une particule peut être référencé à partir de l'indice i et d'un décalage k de telle sorte que $x_p = x_i + k\Delta x$ et ainsi :

$$u(x_p, t^n) = \sum_{j=0}^{\beta} \frac{k^j \Delta x^j}{j!} \frac{\partial^j u}{\partial x^j}(x_i, t^n) + \mathcal{O}(\Delta x^{\beta+1}). \tag{2.17}$$

Ici, nous nous plaçons dans le cadre d'une grille cartésienne uniforme où $x_{i+k} = x_i + k\Delta x$. De même, la position des particules devient :

$$\tilde{x}_p^{n+1} = x_i + k\Delta x + \tilde{a}_{i+k}^n \Delta t,$$

qui se réécrit en posant $\lambda_i = \tilde{a}_i^n \Delta t / \Delta x$, le nombre CFL local :

$$\frac{\tilde{x}_p^{n+1} - x_i}{\Delta x} = k + \lambda_{i+k}.$$

La règle de dérivation des fonctions composées, se généralise par :

$$\begin{aligned}
(f \circ g)^{(m)}(x) &= \sum_{|q|=m} m! c_q \left(f^{(q_1+\cdots+q_m)} \circ g \right)(x) \prod_{s=1}^{m} \left(g^{(s)}(x) \right)^{q_s}, \\
c_q &= \frac{1}{q_1! 1!^{q_1} q_2! 2!^{q_2} \cdots q_m! m!^{q_m}},
\end{aligned}$$

où q est le multi-indice $q = (q_1, q_2, \ldots, q_m)$ tel que $|q| = q_1 + 2q_2 + \cdots + mq_m$. En utilisant cette formule, en posant $\nu = \Delta t / \Delta x$ et en prenant $g(x_i) = k + \lambda_i = k + \nu \tilde{a}^n(x_i)$, on obtient que :

$$\Lambda_{p,r}(k+\lambda_i)^{(m)} = \sum_{|q|=m} m! c_q \Lambda_{p,r}^{(q_1+\cdots+q_m)}(k+\lambda_i) \nu^{q_1+\cdots+q_m} \prod_{s=1}^{m} \left(\tilde{a}^{n\,(s)} \right)^{q_s}. \tag{2.18}$$

Le développement de $\Lambda_{p,r}(k+\lambda_{i+k})$ au voisinage de $k+\lambda_i$ à l'ordre r est obtenu en utilisant la relation (2.18). Cependant, comme $\Lambda_{p,r} \in W^{r+1,\infty}(\mathbb{R})$, cela implique que $||\Lambda_{p,r}^{(\beta+1)}||_\infty < \infty$ pour $0 \leqslant \beta \leqslant r$. Par conséquent, le développement est donc

effectué à l'ordre $\beta = \inf(p,r)$:

$$\Lambda_{p,r}(k+\lambda_{i+k}) = \sum_{m=0}^{\beta} \frac{k^m \Delta x^m}{m!} \sum_{|q|=m} m! c_q \Lambda_{p,r}^{(q_1+\cdots+q_m)}(k+\lambda_i)\nu^{q_1+\cdots+q_m} \prod_{s=1}^{m} \left(\tilde{a}^{n\,(s)}\right)^{q_s}$$
$$+ \mathcal{O}\left(\Delta x^{\beta+1} ||\Lambda_{p,r}^{(\beta+1)}||_\infty ||\tilde{a}^{(\beta+1)}||_\infty \sum_{|q|=\beta+1} \nu^{q_1+q_2+\cdots+q_{\beta+1}}\right). \tag{2.19}$$

D'autre part, comme $q_1+q_2+\cdots+q_{\beta+1} \leqslant |q|$, alors $\nu^{q_1+q_2+\cdots+q_{\beta+1}} \leqslant \nu^{\beta+1} \leqslant \nu^{\beta+1}+\nu$, le résidu précédent se réécrit en :

$$\mathcal{O}\left(\Delta t^{\beta+1} + \Delta x \Delta t^{\beta}\right).$$

Finalement, l'approximation du schéma (2.6) est obtenue à partir du produit des relations (2.17) et (2.19), noté E_k :

$$\begin{aligned}
u_i^{n+1} &= \sum_k u(x_k,t^n)\Lambda_{p,r}(k+\lambda_{i+k}) \\
&= \sum_k \left[\left(\sum_{m=0}^{\beta} k^m \Delta x^m \sum_{|q|=m} c_q \Lambda_{p,r}^{(q_1+\cdots+q_m)}(k+\lambda_i)\nu^{q_1+q_2+\cdots+q_m} \prod_{s=1}^{m}\left(\tilde{a}^{n\,(s)}\right)^{q_s}\right)\right. \\
&\qquad \left.\left(\sum_{j=0}^{\beta} \frac{k^j \Delta x^j}{j!}\frac{\partial^j u}{\partial x^j}(x_i,t^n)\right)\right] + \mathcal{O}\left(\Delta t^{\beta+1} + \Delta x \Delta t^{\beta} + \Delta x^{\beta+1}\right) \\
&= \sum_k E_k + \mathcal{O}\left(\Delta t^{\beta+1} + \Delta x \Delta t^{\beta} + \Delta x^{\beta+1}\right).
\end{aligned}$$

Le produit des deux sommes d'indices m et j des termes E_k se regroupent :

$$\sum_k E_k = \sum_{\substack{m=0\\ j=0}}^{\beta} \frac{\Delta x^{m+j}}{j!}\frac{\partial^j u}{\partial x^j}(x_i,t^n) \sum_{|q|=m} c_q$$
$$\sum_k k^{m+j} \Lambda_{p,r}^{(q_1+\cdots+q_m)}(k+\lambda_i)\nu^{q_1+q_2+\cdots+q_m} \prod_{s=1}^{m}\left(\tilde{a}^{n\,(s)}\right)^{q_s}.$$

Par dérivation de la relation (2.9), on a :

$$\sum_k k^\alpha \Lambda_{p,r}^{(\delta)}(k+x) = \frac{\alpha!}{(\alpha-\delta)!}(-1)^\delta (-x)^{\alpha-\delta} \quad \text{si } 0 \leqslant \delta \leqslant \alpha \leqslant \beta.$$

Par conséquent, on a également :

$$\sum_k k^{m+j} \Lambda_{p,r}^{(q_1+\cdots+q_m)}(k+\lambda_i)\nu^{q_1+q_2+\cdots+q_m} = d_{i,m+j,q}\nu^{m+j},$$

avec :

$$d_{i,s,q} = \frac{s!(-1)^{q_1+\cdots+q_m}}{(s-(q_1+\cdots+q_m))!}(-\tilde{a}_i^n)^{s-(q_1+\cdots+q_m)}, \text{ si } 0 \leqslant q_1+\cdots+q_m \leqslant s \leqslant \beta.$$

Ce qui permet d'aboutir à :

$$\begin{aligned}\sum_k E_k &= \sum_{\substack{m=0\\j=0}}^{\beta} \frac{\Delta x^{m+j}}{j!}\frac{\partial^j u}{\partial x^j}(x_i,t^n) \sum_{|q|=m} c_q d_{i,m+j,q} \nu^{m+j} \prod_{s=1}^{m} \left(\tilde{a}^{n\,(s)}\right)^{q_s} \\ &= \sum_{\substack{m=0\\j=0}}^{\beta} \frac{\Delta t^{m+j}}{j!}\frac{\partial^j u}{\partial x^j}(x_i,t^n) \sum_{|q|=m} c_q d_{i,m+j,q} \prod_{s=1}^{m} \left(\tilde{a}^{n\,(s)}\right)^{q_s}.\end{aligned}$$

Finalement, la troncature du schéma à l'ordre α en temps donne :

$$\begin{aligned}u_i^{n+1} = \sum_{\substack{m=0\\j=0}}^{\alpha} \frac{\Delta t^{m+j}}{j!}\frac{\partial^j u}{\partial x^j}(x_i,t^n) \sum_{|q|=m} c_q d_{i,m+j,q} \prod_{s=1}^{m} \left(\tilde{a}^{n\,(s)}\right)^{q_s} \\ + \mathcal{O}\left(\Delta t^{\alpha+1} + \Delta x \Delta t^{\beta} + \Delta x^{\beta+1}\right).\end{aligned} \tag{2.20}$$

Advection par un schéma d'Euler

Les couples (m,j) possibles pour la première somme tels que $m+j \leqslant \alpha+1$ sont $(0,0)$, $(0,1)$ et $(1,0)$. Pour $m=0$, le multi-indice q se réduit à un simple indice égal à 0. L'expression (2.20) devient :

$$u_i^{n+1} = u(x_i,t^n) - \Delta t \tilde{a}_i^n \frac{\partial u}{\partial x}(x_i,t^n) - \Delta t (\tilde{a}_i^n)' u(x_i,t^n) + \mathcal{O}\left(\Delta t^2 + \Delta x \Delta t^{\beta} + \Delta x^{\beta+1}\right).$$

D'autre part, si le champ de vitesse est approché à l'ordre 1, on a $\tilde{a}_i^n = a(x_i,t^n)$, ce qui donne :

$$u_i^{n+1} = u(x_i,t^n) - \Delta t \frac{\partial a_i^n u}{\partial x}(x_i,t^n) + \mathcal{O}\left(\Delta t^2 + \Delta x \Delta t^{\beta} + \Delta x^{\beta+1}\right). \tag{2.21}$$

Puis, en utilisant l'équation de transport (2.2) on aboutit au résultat :

$$u_i^{n+1} = u(x_i,t^{n+1}) + \mathcal{O}\left(\Delta t^2 + \Delta x \Delta t^{\beta} + \Delta x^{\beta+1}\right). \tag{2.22}$$

Advection par un schéma de Runge-Kutta du second ordre

Dans ce cas, en plus des termes présents pour le cas du schéma d'Euler, on ajoute les termes correspondants aux couples $(m,j) = (0,2)$, $(1,1)$ et $(2,0)$.

$$\begin{aligned}u_i^{n+1} = u(x_i,t^n) &- \Delta t \tilde{a}_i^n \frac{\partial u}{\partial x}(x_i,t^n) - \Delta t (\tilde{a}_i^n)' u(x_i,t^n) \\ &+ \frac{\Delta t^2}{2}\Bigg((\tilde{a}_i^n)^2 \frac{\partial^2 u}{\partial x^2}(x_i,t^n) + 4\tilde{a}_i^n (\tilde{a}_i^n)' \frac{\partial u}{\partial x}(x_i,t^n) \\ &\qquad + 2((\tilde{a}_i^n)'^2 + \tilde{a}_i^n (\tilde{a}_i^n)'')u(x_i,t^n)\Bigg) \\ &+ \mathcal{O}\left(\Delta t^3 + \Delta x \Delta t^{\beta} + \Delta x^{\beta+1}\right).\end{aligned}$$

Pour une approximation du second ordre, on a $\tilde{a}_i^n = a(x_i + a(x_i)\Delta t/2)$. Les développements de $\tilde{a}_i^n$ sont réalisés à l'ordre maximal puis injectés dans l'expression précédente :

$$\begin{aligned}
\Delta t \tilde{a}_i^n &= \Delta t a_i^n + \frac{\Delta t^2}{2} a(x_i) a'(x_i) + \mathcal{O}(\Delta t^3), \\
\Delta t (\tilde{a}_i^n)' &= \Delta t a'(x_i) + \frac{\Delta t^2}{2} a(x_i) a''(x_i) + \frac{\Delta t^2}{2} a'^2(x_i) + \mathcal{O}(\Delta t^3), \\
\Delta t^2 (\tilde{a}_i^n)'' &= \Delta t^2 a''(x_i) + \mathcal{O}(\Delta t^3),
\end{aligned}$$

$$\begin{aligned}
u_i^{n+1} = u(x_i, t^n) &- \Delta t \left(a_i^n \frac{\partial u}{\partial x}(x_i, t^n) + (a_i^n)' u(x_i, t^n) \right) \\
&+ \frac{\Delta t^2}{2} \Bigg((a_i^n)^2 \frac{\partial^2 u}{\partial x^2}(x_i, t^n) + 3 \tilde{a}_i^n (\tilde{a}_i^n)' \frac{\partial u}{\partial x}(x_i, t^n) \\
&\qquad + ((\tilde{a}_i^n)'^2 + \tilde{a}_i^n (\tilde{a}_i^n)'') u(x_i, t^n) \Bigg) \\
&+ \mathcal{O}\left(\Delta t^3 + \Delta x \Delta t^\beta + \Delta x^{\beta+1} \right).
\end{aligned}$$

Par simplifications on obtient :

$$\begin{aligned}
u_i^{n+1} = u(x_i, t^n) - \Delta t \frac{\partial a_i^n u}{\partial x}(x_i, t^n) + \frac{\Delta t^2}{2} \frac{\partial}{\partial x} \left(\frac{\partial a_i^n u}{\partial x} \right)(x_i, t^n) \\
+ \mathcal{O}\left(\Delta t^3 + \Delta x \Delta t^\beta + \Delta x^{\beta+1} \right).
\end{aligned} \tag{2.23}$$

Puis en utilisant l'équation de transport (2.2) on obtient finalement :

$$u_i^{n+1} = u(x_i, t^{n+1}) + \mathcal{O}\left(\Delta t^3 + \Delta x \Delta t^\beta + \Delta x^{\beta+1} \right). \tag{2.24}$$

Cas d'une seule particule par cellule

Dans ce cas, le développement (2.19) peut être effectué à n'importe quel ordre, en particulier à l'ordre $\beta = r$. La suite des calculs reste identique.

□

2.2.3. Stabilité de la méthode semi-Lagrangienne

Les démonstrations de la stabilité de la méthode sont données dans l'article Cottet *et al.* (2014) inclus en annexe B. Les preuves sont basées sur les propriétés de décroissance des noyaux qui sont vérifiées pour les méthodes d'ordre 2 et 4. Nous proposons ici une approche permettant de démontrer la propriété de stabilité linéaire de la méthode pour les noyaux de type $\Lambda_{p,r}$ jusqu'à $p = 8$.

Advection à vitesse constante

Dans un premier temps, la stabilité du schéma est analysée pour un champ de vitesse constant en espace. La démonstration se base sur le nombre CFL local : $\lambda = a\Delta t/\Delta x$, constant. Nous définissons les suites de fonctions suivantes, pour $k \in \mathbb{Z}$:

$$\alpha_k(\lambda) = \Lambda_{p,r}(k+\lambda), \tag{2.25}$$

$$A_k(\lambda) = \sum_i \alpha_i(\lambda)\alpha_{i+k}(\lambda). \tag{2.26}$$

Les noyaux de remaillage étant à support compact, seuls quelques éléments de ces suites sont non nuls.

Proposition 2.2

Les suites $\alpha_k(\lambda)$ et $A_k(\lambda)$ vérifient :

$$\sum_{k,l} (k-l)^q \alpha_k(\lambda)\alpha_l(\lambda) = 0, \quad \text{et} \quad \sum_k k^q A_k(\lambda) = 0, \quad q \leqslant p. \tag{2.27}$$

Démonstration de la proposition 2.2 : Le membre de gauche de la première équation se développe en :

$$\sum_{k,l} (k-l)^q \alpha_k(\lambda)\alpha_l(\lambda) = \sum_{k,l} \sum_{m=0}^{q} C_q^m k^m l^{q-m} \alpha_k(\lambda)\alpha_l(\lambda).$$

La propriété (2.9) étant vérifiée pour les formules de type $\Lambda_{p,r}$, on a :

$$\sum_{k,l} k^m l^{q-m} \alpha_k(\lambda)\alpha_l(\lambda) = (-\lambda)^m (\lambda)^{q-m},$$

et donc

$$\sum_{k,l} (k-l)^q \alpha_k(\lambda)\alpha_l(\lambda) = \sum_{m=0}^{q} C_q^m (-\lambda)^m (\lambda)^{q-m} = (\lambda-\lambda)^q = 0.$$

La seconde équation vient du fait que $A_k(\lambda) = A_{-k}(\lambda)$.

□

Proposition 2.3

Pour un champ de vitesse a constant en espace et pour un noyau de remaillage $\Lambda_{p,r}$ de support $[-S; S]$ vérifiant les contraintes de régularité (2.14), la propriété d'interpolation (2.13) ainsi que l'égalité de conservation des moments (2.9) jusqu'à un ordre p, le schéma de remaillage est inconditionnellement stable.

Démonstration de la proposition 2.3 : Soient I un entier tel que $\lambda \in [I; I+1[$ et $\mu = \lambda - I \in [0; 1[$. En posant $v_i = u_{i-I}^n$, le schéma (2.6) se réécrit :

$$u_i^{n+1} = \sum_k v_{i+k} \alpha_k(\mu).$$

Montrons que

$$\sum_i \left|u_i^{n+1}\right|^2 \leqslant \sum_i |u_i^n|^2.$$

En utilisant (2.27), il vient :

$$\sum_i \left|u_i^{n+1}\right|^2 = \sum_i \sum_{k,l} v_{i+k} v_{i+l} \alpha_k(\mu) \alpha_l(\mu). \tag{2.28}$$

La formule de remaillage possède un support $[-S; S]$, par conséquent, seuls quelques $\alpha_k(\mu)$ sont non nuls, pour $k \in [-S; S-1]$. Pour alléger les notations, comme $\mu \in [0, 1[$ ne joue aucun rôle, il sera omis dans la suite.

$$\begin{aligned}
\sum_i |u_i^{n+1}|^2 &= \sum_i \sum_{k,l} v_{i+k} v_{i+l} \alpha_k \alpha_l \\
&= \sum_i \left(\sum_k v_{i+k}^2 \alpha_k^2 + 2 \sum_{k<l} v_{i+k} v_{i+l} \alpha_k \alpha_l \right) \\
&= \sum_i v_i^2 \sum_k \alpha_k^2 + \sum_i \sum_{k<l} \left(v_{i+k}^2 + v_{i+l}^2 - |v_{i+k} - v_{i+l}|^2 \right) \alpha_k \alpha_l \\
&= \sum_i v_i^2 \left(\sum_k \alpha_k^2 + 2 \sum_{k<l} \alpha_k \alpha_l \right) - \sum_i \sum_{k<l} \alpha_k \alpha_l |v_{i+k} - v_{i+l}|^2.
\end{aligned}$$

La conservation du premier moment, par les équations (2.9), nous donne que

$$1 = \left(\sum_k \alpha_k \right)^2 = \sum_k \alpha_k^2 + 2 \sum_{k<l} \alpha_k \alpha_l,$$

et donc :

$$\sum_i |u_i^{n+1}|^2 = \sum_i v_i^2 - \sum_i \sum_{k<l} \alpha_k \alpha_l |v_{i+k} - v_{i+l}|^2.$$

On arrive finalement à :

$$\sum_i |u_i^{n+1}|^2 = \sum_i |u_i^n|^2 - R,$$

avec

$$\begin{aligned}
R &= \sum_i \sum_{k<l} \alpha_k \alpha_l |v_{i+k} - v_{i+l}|^2 \geqslant 0 \\
&= \sum_i \sum_k \sum_{j=1}^{2S-1} \alpha_k \alpha_{k+j} |v_{i+k} - v_{i+k+j}|^2 \\
&= \sum_k \sum_{j=1}^{2S-1} \alpha_k \alpha_{k+j} \sum_i |v_i - v_{i+j}|^2 \\
&= \sum_{j=1}^{2S-1} A_j \sum_i |v_i - v_{i+j}|^2.
\end{aligned}$$

La suite consiste à démontrer que $R \geqslant 0$. On définit alors les suites $\{\delta_{J,i}\}_{J=1,\dots,2S-1}$ de différences successives telles que :

$$\begin{cases} \delta_{1,i} = v_i - v_{i+1}, \\ \delta_{2,i} = \delta_{1,i} - \delta_{1,i+1}, \\ \delta_{J,i} = \delta_{J-1,i} - \delta_{J-1,i+1}, \end{cases} \tag{2.29}$$

ainsi que des sommes de termes de ces suites :

$$s_{J,j} = \sum_i \left| \sum_{k=0}^{j-1} \delta_{J,i+k} \right|^2 .$$

Pour $J = 1$, on a simplement $s_{1,j} = \sum_i |v_i - v_{i+j}|^2$, donc R se réécrit :

$$R = \sum_{j=1}^{2S-1} A_j s_{1,j}. \tag{2.30}$$

Par définition de $s_{J,j}$, on a :

$$\begin{aligned} s_{J,j} &= \sum_i \left| \sum_{k=0}^{j-1} \delta_{J,i+k} \right|^2 \\ &= \sum_i \left(\sum_{k=0}^{j-1} \delta_{J,i+k}^2 + \sum_{k=0}^{j-2} \sum_{l=1}^{j-1-k} 2\delta_{J,i+k}\delta_{J,i+k+l} \right) \\ &= j \sum_i \delta_{J,i}^2 + \sum_{l=1}^{j-1} (j-l) \sum_i 2\delta_{J,i}\delta_{J,i+l}. \end{aligned}$$

En utilisant la relation $2\delta_{J,i}\delta_{J,i+l} = \delta_{J,i}^2 + \delta_{J,i+l}^2 - |\delta_{J,i} - \delta_{J,i+l}|^2$, on obtient que :

$$\begin{aligned} s_{J,j} &= j \sum_i \delta_{J,i}^2 + \sum_{l=1}^{j-1} (j-l) \left(2 \sum_i \delta_{J,i}^2 - \sum_i |\delta_{J,i} - \delta_{J,i+l}|^2 \right) \\ &= \left(j + 2 \sum_{l=1}^{j-1} (j-l) \right) \sum_i \delta_{J,i}^2 - \sum_{l=1}^{j-1} (j-l) \sum_i |\delta_{J,i} - \delta_{J,i+l}|^2 \\ &= j^2 \sum_i \delta_{J,i}^2 - \sum_{l=1}^{j-1} (j-l) \sum_i |\delta_{J,i} - \delta_{J,i+l}|^2 \\ &= j^2 \sum_i \delta_{J,i}^2 - \sum_{l=1}^{j-1} (j-l) s_{J+1,l}. \end{aligned}$$

Ainsi, le second terme de l'expression précédente contient $j-1$ termes en $s_{J+1,l}$ dont les seconds membres contiennent au maximum $j-2$ termes, pour $s_{J+1,j-1}$. Le développement complet de $s_{J,j}$ fait intervenir $j-1$ niveaux de récurrence jusqu'au

niveau $J+j-1$. On obtient, en posant $D_J = \sum_i \delta_{J,i}^2$ pour alléger les notations, le développement suivant :

$$\begin{aligned} s_{J,j} &= j^2 D_J - \sum_{l=1}^{j-1} (j-l) s_{J+1,l} \\ &= j^2 D_J - \sum_{l=1}^{j-1} (j-l) l^2 D_{J+1} + \sum_{l=1}^{j-1} (j-l) \sum_{m=1}^{l-1} (l-m) s_{J+2,m} \\ &= \sum_{k=1}^{j} (-1)^{k+1} C_{k,j} D_{J+k-1}. \end{aligned}$$

Avec les coefficients $C_{k,j}$ définis par :

$$\begin{aligned} C_{1,j} &= j^2; \\ C_{2,j} &= \begin{cases} \displaystyle\sum_{l=1}^{j-1} (j-l) l^2, \text{ si } j \geqslant 2, \\ 0, \text{ sinon}; \end{cases} \\ C_{3,j} &= \begin{cases} \displaystyle\sum_{l=1}^{j-1} (j-l) \sum_{m=1}^{l-1} (l-m) m^2, \text{ si } j \geqslant 3, \\ 0, \text{ sinon}; \end{cases} \\ C_{4,j} &= \begin{cases} \displaystyle\sum_{l=1}^{j-1} (j-l) \sum_{m=1}^{l-1} (l-m) \sum_{n=1}^{m-1} (m-n) n^2, \text{ si } j \geqslant 4, \\ 0, \text{ sinon}; \end{cases} \\ &\vdots \end{aligned}$$

L'expression (2.30) se réécrit donc :

$$R = \sum_{j=1}^{2S-1} A_j \left(\sum_{k=1}^{j} (-1)^{k+1} C_{k,j} D_k \right),$$

puis en permutant les sommes, on obtient :

$$R = \sum_{k=1}^{2S-1} D_k \left(\sum_{j=k}^{2S-1} (-1)^{k+1} C_{k,j} A_j \right). \tag{2.31}$$

Deux remarques permettent de simplifier la recherche du signe de R. Premièrement, les D_k étant positifs comme sommes de carrés, le dernier terme de la somme, pour $k = 2S-1$, est toujours positif. En effet, ce terme se simplifie en $D_{2S-1} A_{2S-1} = D_{2S-1} \alpha_{-S} \alpha_{S-1}$ et les α_{-S} et α_{S-1} sont de même signe car on observe que les formules de type $\Lambda_{p,r}$ présentent une alternance de signe sur chaque intervalle entier. Les noyaux sont positifs sur l'intervalle $[-1;1]$, négatifs sur $[-2;-1] \cup [1;2]$ et ainsi sur l'ensemble des intervalles entiers du support. En particulier, il est positif sur l'intervalle $[-S;-S+1] \cup [S-1;S]$ si S est impair et négatif sinon. Deuxièmement, en

remarquant que les coefficients $C_{k,j}$ se réécrivent comme des combinaisons linéaires de puissances paires de j :

$$C_{2,j} = \frac{-j^2}{12} + \frac{j^4}{12}, \quad C_{3,j} = \frac{j^2}{90} - \frac{j^4}{72} + \frac{j^6}{360}, \quad C_{4,j} = \frac{-j^2}{560} + \frac{7j^4}{2880} - \frac{j^6}{1440} + \frac{j^8}{20160},$$

les termes de la forme $\sum C_{k,j} A_j$ s'annulent lorsque $k \leqslant [p/2]$ en utilisant la propriété de conservation des moments pairs (2.9). Ainsi, si la formule de remaillage conserve p moments alors les $[p/2]$ premiers termes de la somme d'indice k sont nuls. Ainsi on obtient la majoration suivante :

$$R \geqslant \sum_{k=[p/2]+1}^{2S-2} D_k \left(\sum_{j=k}^{2S-1} (-1)^{k+1} C_{k,j} A_j \right).$$

Pour toutes les formules de remaillage de type $\Lambda_{p,r}$, il a été vérifié numériquement que :

$$\sum_{j=k}^{2S-1} (-1)^{k+1} C_{k,j} A_j \geqslant 0 \quad \text{pour } k = [p/2]+1, \ldots, 2S-2$$

Ainsi nous obtenons que toutes les formules de type $\Lambda_{p,r}$ données en annexe A conduisent à une méthode stable car $R \geqslant 0$.

□

Advection à vitesse non constante

Nous donnons ici les grandes lignes de la démonstration de stabilité dans le cas général qui est détaillée dans l'article Cottet *et al.* (2014) inclus en annexe B.

Proposition 2.4

Pour un champ de vitesse a suffisamment régulier et pour un noyau de remaillage $\Lambda_{p,r}$ de support $[-S; S]$ vérifiant les contraintes de régularité (2.14), la propriété d'interpolation (2.13) ainsi que l'égalité de conservation des moments (2.9) jusqu'à un ordre p et lorsque le pas de temps est suffisamment petit vérifiant la condition (2.7), il existe une constance C indépendante de Δt et Δx telle que :

$$\sum_i \left| u_i^{n+1} \right|^2 \leqslant (1 + C\Delta t) \sum_i |u_i^n|^2. \tag{2.32}$$

Démonstration de la proposition 2.4 : Nous ne donnons ici que l'idée générale de la démonstration qui est détaillée dans l'article en annexe. Lorsque la vitesse est non constante et le pas de temps suffisamment petit (2.7), le nombre CFL local est différent pour chaque particule :

$$\lambda_p = \frac{a_p \Delta t}{\Delta x}.$$

Les particules peuvent alors être partitionnées selon l'intervalle entier auquel appartient λ_p :

$$J_I = \left\{ p \in \mathbb{Z}, \lambda_p \in [I; I+1[\right\}. \tag{2.33}$$

Le schéma de remaillage (2.6) se réécrit :

$$u_i^{n+1} = \sum_I \sum_{k \in J_I} u_k^n \Lambda_{p,r}(\lambda_k + k - i).$$

En posant

$$R_I(i) = \sum_{k \in J_I} u_k^n \Lambda_{p,r}(\lambda_k + k - i),$$

le membre de gauche de l'équation (2.32) devient :

$$\sum_i \left|u_i^{n+1}\right|^2 = \sum_i \left|\sum_I R_I(i)\right|^2 = \sum_i \sum_I R_I^2(i) + \sum_i \sum_{I \neq I'} R_I(i) R_{I'}(i).$$

L'étude du premier terme est similaire à l'analyse dans le cas à vitesse constante et se base sur le fait que dans chaque élément de la double somme, les particules ont un nombre CFL local dans le même intervalle entier. Dans ce cas, on aboutit au résultat :

$$\sum_i R_I^2(i) \leqslant \sum_{i \in J_I} |u_i^n|^2 + C\Delta t \tag{2.34}$$

Dans le second terme, il apparaît des produits de la forme :

$$\Lambda_{p,r}(\lambda_k + k - i)\Lambda_{p,r}(\lambda_{k'} + k' - i), \quad k \in J_I, k' \in J_{I'}, I \neq I'.$$

Ces produits sont non nuls seulement dans le cas où les particules son suffisamment proches après advection. De plus, chaque particule ne contribue que dans, au plus, $2S$ sommes $R_I(i)$. Ainsi on aboutit au résultat :

$$\sum_i \sum_{I \neq I'} R_I(i) R_{I'}(i) \leqslant C\Delta t \sum_i |u_i^n|^2 \tag{2.35}$$

Finalement, en combinant les résultats (2.34) et (2.35), on aboutit au résultat (2.32).

□

Conclusion

Dans ce chapitre, nous avons détaillé la méthode de résolution que nous employons pour résoudre une équation de transport. Nous avons vu comment l'étude des propriétés numériques de la méthode semi-Lagrangienne est réalisée en exploitant la similarité de la méthode avec celle des différences finies. En particulier, cela nous permet de produire des schémas numériques à un ordre arbitraire en utilisant des formules de remaillage appropriées. Ces formules possèdent, par construction, des propriétés de régularité et de conservation de moments des quantités transportées qui sont directement liées à l'ordre de la méthode.

Cette méthode semi-Lagrangienne permet de résoudre les équations de transport présentées au chapitre 1. Les autres éléments constituant le problème de transport de scalaire dans un fluide turbulent tels que les termes d'étirement et de diffusion ainsi que l'équation de Poisson sont traités par d'autres méthodes. Cela conduit à l'obtention d'une méthode hybride comme nous le verrons dans le chapitre 3.

3. Méthode hybride pour le transport turbulent

Dans ce chapitre nous proposons de construire une méthode hybride, utilisant la méthode semi-Lagrangienne détaillée dans le chapitre 2, pour résoudre un problème de transport d'un scalaire dans un fluide turbulent, exposé dans le chapitre 1. Comme nous l'avons mentionné précédemment, ce système présente à la fois un caractère multiéchelle et un faible couplage entre les sous-problèmes que sont la résolution du fluide et le transport du scalaire. Ces caractéristiques nous conduisent à l'élaboration d'une méthode hybride qui est d'une part adaptée à la résolution de ce type de problème et d'autre part capable de traiter des résolutions importantes sur des machines massivement parallèles.

Dans un premier temps, nous décrirons l'idée générale d'une méthode hybride puis nous donnerons quelques approches hybrides présentées dans la littérature avant de proposer la stratégie que nous emploierons par la suite.

3.1. Idée générale d'une méthode hybride

Le principe d'une méthode hybride est d'utiliser différentes méthodes conjointement afin de résoudre un problème complexe en exploitant les forces de chacune des méthodes. Ainsi, le problème initial est décomposé selon ses aspects physiques en différentes parties qui sont résolues avec des méthodes différentes. Même si chacune des méthodes est bien connue individuellement, comme pour les méthodes exposées en section 1.1.2, le couplage doit être étudié en détail.

La décomposition du problème global selon ses aspects physiques permet, par exemple dans le cadre de l'étude d'interactions entre un fluide et une structure, de séparer la résolution du fluide et du solide. De même pour le cas d'un transport de scalaire passif, il est possible de séparer la résolution du fluide de celle du transport du scalaire. Enfin, nous avons vu dans les chapitres précédents, que l'emploi d'une méthode semi-Lagrangienne pour la résolution d'un écoulement fait intervenir une équation de transport de la vorticité ainsi qu'une équation de Poisson permettant de calculer le champ de vitesse associé. Là encore une décomposition peut avoir lieu pour séparer la résolution de l'équation de Poisson de celle de l'équation de transport.

Ainsi dans une méthode hybride, les différentes parties du problème, issues de la décomposition selon les aspects physiques et numériques, sont résolues par différentes méthodes. Trois niveaux d'hybridation peuvent être distingués et sont détaillés dans la suite.

3.1.1. Méthode hybride en méthode numérique

Le premier niveau consiste en l'utilisation de différentes méthodes numériques. Par exemple, pour la résolution d'un écoulement turbulent, une méthode hybride est utilisée par Cottet *et al.* (2002), Cottet *et al.* (2009b) et Rees *et al.* (2011). Cette méthode consiste en une résolution semi-Lagrangienne du transport de la vorticité combinée à une méthode spectrale pour le calcul du champ de vitesse en fonction du champ de vorticité à partir de l'équation de Poisson. La résolution de l'équation de Poisson par une méthode spectrale est une alternative à celle basée sur la loi de Biot-Savart employée dans les méthodes Vortex. Cette dernière conduit à une résolution directe sur le champ de particules, ce qui constitue un problème à N corps à travers le calcul des interactions entre les paires de particules. Les implémentations efficaces nécessitent la gestion d'une structure de données complexe basée sur des arbres afin de regrouper les particules en fonction de leurs distances pour exploiter les approximations multipoles rapides (méthodes FMM). Ainsi, la méthode spectrale est bien plus simple à mettre en œuvre. Ce type de couplage entre méthodes semi-Lagrangienne et spectrale est utilisé également pour des simulations d'écoulements en présence de solides rigides, pris en compte par pénalisation (Coquerelle *et al.*, 2008 ; Rossinelli *et al.*, 2010).

D'autre part, dans le cas du transport turbulent d'un scalaire passif, des méthodes hybrides consistent en une résolution par méthode spectrale de l'écoulement puis un transport du scalaire par différences finies (Gotoh *et al.*, 2012) ou par une méthode semi-Lagrangienne (Lagaert *et al.*, 2012 ; Lagaert *et al.*, 2014).

3.1.2. Méthode hybride en résolution

Un second aspect des méthodes hybrides est d'employer des grilles de résolutions différentes pour discrétiser les équations présentes dans le modèle. Par exemple, Cottet *et al.* (2009a) proposent l'utilisation d'une méthode particulaire hybride en résolution pour le transport de scalaire. Les auteurs utilisent la même méthode semi-Lagrangienne pour résoudre l'écoulement et le scalaire. Le couplage est réalisé par interpolation du champ de vitesse pour le transport du scalaire. Dans cette méthode, le scalaire est discrétisé à une résolution plus fine que les champs de vitesse et de vorticité. Généralement, des modèles sous-maille sont utilisés dans le cadre de simulations des grandes échelles (LES) pour prendre en compte la dissipation d'énergie par son transfert vers les petites échelles. Cependant, ce modèle n'est pas suffisant pour prendre en compte les effets des plus petites échelles de fluctuations du scalaire qui ne sont pas résolues car de taille inférieure au maillage. Cottet *et al.* (2009a) montrent que l'emploi d'une méthode multiéchelle remédie naturellement à ce problème par une simulation directe des petites échelles du scalaire tout en conduisant à un coût de calcul similaire.

Les études menées par Gotoh *et al.* (2012), Lagaert *et al.* (2012) et Lagaert *et al.* (2014) dans le cadre de l'analyse du comportement d'un scalaire transporté par un écoulement de

turbulence homogène montrent que cette approche hybride en résolution permet d'obtenir des résultats cohérents avec le cas d'une méthode spectrale classique, en simple résolution. En effet, comme nous l'avons souligné dans la partie 1.3, ce problème se caractérise par la présence de différentes échelles et la résolution du fluide à l'échelle du scalaire n'apporte pas d'amélioration de la prise en compte de la physique de l'écoulement. Les méthodes utilisées dans ces travaux seront détaillées plus loin, en section 3.2.

3.1.3. Méthode hybride en matériel

Comme nous l'avons vu dans la partie 1.2, les machines de calcul présentent de nombreuses architectures différentes. Un dernier niveau de construction d'une méthode hybride consiste en l'exploitation des différentes architectures d'une même machine. L'idée est de distribuer les différentes parties du problème sur différents matériels. Par exemple, dans le cas du transport de scalaire, la résolution du fluide est effectuée sur les processeurs CPU et le transport du scalaire est résolu sur les cartes graphiques d'une machine hétérogène. Cette approche sera complétée plus loin dans ce chapitre et sa mise en œuvre détaillée dans le chapitre 6.

Une combinaison de ces différents niveaux est évidemment possible. En effet, une méthode hybride peut être définie par des résolutions et des méthodes différentes pour les parties du problème. Les approches semblables de Gotoh *et al.* (2012) et Lagaert *et al.* (2014) illustrent parfaitement ce cas de figure. Dans leurs études, le fluide est résolu par une méthode spectrale à une résolution grossière et le scalaire par différences finies ou méthode semi-Lagrangienne à une résolution plus fine. Les nombreuses possibilités de construction de méthodes hybrides nécessitent quelques précisions quant à l'application au transport de scalaire passif dans un écoulement turbulent. En effet, les caractéristiques du problème permettent d'aiguiller les choix pour la mise en place d'une méthode hybride.

3.2. Application au transport turbulent

3.2.1. Méthodes spectrale et différences finies

Dans le cas du transport de scalaire passif, Gotoh *et al.* (2012) proposent une comparaison de leur méthode hybride avec une méthode entièrement spectrale. La méthode hybride consiste en la résolution de l'écoulement à l'aide d'une méthode spectrale alors que le transport du scalaire est résolu par différences finies d'ordre élevé. L'intérêt majeur de cette approche est que les opérations non locales sur le scalaire, issues des transformées de Fourier de la méthode spectrale, sont remplacées par des schémas locaux. Cela permet de limiter les communications lors de l'exploitation d'un grand nombre de cœurs de calcul. Les différentes échelles physiques présentes dans les cas à nombre de Schmidt élevé permettent à Gotoh *et al.* de réaliser des simulations en multirésolution. Le fluide est calculé sur 256^3 points de grille alors que $1\,024^3$ points sont utilisés pour discrétiser le scalaire.

Une réduction du temps de calcul est obtenue à travers ces différentes simulations. En particulier, les auteurs obtiennent un facteur d'accélération de 1,4 grâce à l'utilisation de leur méthode hybride au lieu d'une méthode entièrement spectrale. D'autre part, le fait de réduire le nombre de points de $1\,024^3$ à 256^3 pour la résolution du fluide, pour $Sc = 50$, leur permet d'obtenir un facteur d'accélération supplémentaire de 3. L'ensemble des calculs ont été réalisés en utilisant 4 096 cœurs de calcul.

Une limite de cette approche est que le pas de temps reste limité par la résolution du maillage du scalaire à travers l'expression d'une condition de CFL. Ainsi, lorsque le problème est résolu en utilisant des grilles résolutions différentes, les auteurs sont limités à l'utilisation d'un pas de temps dépendant de celle du scalaire.

3.2.2. Méthodes spectrale et semi-Lagrangienne

Une approche similaire a été proposée par Lagaert *et al.* (2012) et Lagaert *et al.* (2014). Les auteurs utilisent une méthode hybride où le fluide est résolu de façon spectrale couplée à une méthode semi-Lagrangienne pour le transport du scalaire. L'intérêt majeur est de pouvoir bénéficier de l'avantage des méthodes particulaires dont le pas de temps n'est pas contraint par une condition aussi restrictive que la condition CFL des méthodes Eulériennes, dans certains cas. Cela permet à Lagaert *et al.* de choisir un pas de temps pour le transport du scalaire dix fois plus grand que celui du fluide. Ainsi, dans le cas d'un grand nombre de Schmidt, non seulement les résolutions spatiales sont différentes mais les pas de temps aussi. Cela permet d'atteindre un facteur d'accélération approchant 40 par rapport à une méthode entièrement spectrale avec une seule résolution, dans le cas $Sc = 50$ avec 256^3 points pour le fluide et $1\,024^3$ points pour le scalaire.

Toutefois, la méthode semi-Lagrangienne utilisée par Lagaert *et al.* reste d'un ordre spatial plus faible (ordre 4) que le schéma aux différences finies d'ordre 8 employé par Gotoh *et al.*

3.2.3. Méthodes semi-Lagrangiennes et architecture hybride

Dans ce travail, nous reprenons l'idée de Cottet *et al.* (2009a) pour l'utilisation d'une méthode hybride. La méthode que nous utilisons exploite les trois niveaux d'hybridation exposés plus haut : maillages de résolutions différentes pour le scalaire et l'écoulement, couplage de méthodes de natures différentes et utilisation d'architectures matérielles différentes pour la résolution de l'écoulement et du transport du scalaire. Nous exploitons le caractère multiéchelle du problème avec une méthode hybride en résolution dans laquelle le scalaire transporté est résolu de manière plus fine que la vorticité. L'écoulement est résolu par une méthode semi-Lagrangienne pour le transport de la vorticité, avec des différences finies pour le terme d'étirement et une diffusion spectrale. Nous calculons le champ de vitesse à partir de la vorticité à l'aide d'un solveur spectral pour l'équation de Poisson. Le scalaire est également transporté par une méthode semi-Lagrangienne avec une diffusion en différences finies. Enfin au niveau matériel, le fluide sera résolu sur une architecture multicœurs CPU classique et nous tirons parti de la puissance de calcul disponible dans les cartes graphiques pour résoudre le transport du scalaire.

Comme toutes les résolutions en temps sont effectuées par méthode semi-Lagrangienne, cela nous permet de choisir de grands pas de temps contraints uniquement par la condition de CFL Lagrangienne qui fait intervenir le gradient du champ de vitesse et non la taille de la grille. D'autre part, comme nous l'avons vu en section 2.2, la méthode d'ordre élevé conduit à l'utilisation de formules de remaillage d'une forte complexité arithmétique. En effet, le remaillage d'une particule nécessite l'évaluation de polynômes de degré relativement élevés en chacun des nombreux point du support. Cette complexité est maîtrisée par l'utilisation de cartes graphiques. L'intérêt de cette approche est de pouvoir accélérer, à l'aide des GPU, la partie la plus intensive du code dont la résolution pourra être réalisée en même temps que l'écoulement par l'utilisation de matériels différents pour chaque partie. D'un point de vue pratique, lors de l'exploitation de machines de calcul, il est d'usage de réserver les cartes graphiques par nœuds complets ce qui, sur la plupart des machines, donne accès à un nombre de cœurs CPU bien plus grand que le nombre de GPU. L'objectif d'une exploitation efficace des ressources est envisageable à travers une utilisation simultanée des GPU et des cœurs CPU pour résoudre respectivement le transport du scalaire et le fluide.

Conclusion

Dans ce chapitre, nous avons vu comment une méthode hybride permet d'exploiter les forces des différentes méthodes en fonction des caractéristiques du problème à résoudre. Ainsi, la méthode que nous utiliserons par la suite combine la méthode semi-Lagrangienne exposée dans le chapitre 2 avec des méthodes de grille classiques, différences fines et spectrales. L'idée principale de cette méthode est de profiter à la fois de la puissance de calcul développée par les cartes graphiques et de l'algorithme de splitting dimensionnel pour réaliser des simulations d'ordre élevé pour le transport du scalaire. Les résultats de cette méthode seront présentés dans le chapitre 6.

4. Développement d'un code multiarchitectures

L'un des objectifs de cette thèse consiste en l'implémentation sur cartes graphiques de la méthode particulaire avec remaillage introduite dans le chapitre 2. Cette étape s'inscrit dans un cadre plus global de construction d'une méthode hybride, dont les principes ont été donnés dans le chapitre 3, afin d'explorer les possibilités d'une exploitation les ressources de machines de calcul hétérogènes telles que décrites dans le chapitre 1. Comme nous l'avons souligné dans le chapitre 2, la méthode semi-Lagrangienne semble être bien adaptée à une implémentation sur cartes graphiques du fait de la régularité des structures de données et des algorithmes qui en découlent. De plus, l'utilisation de formules de remaillage d'ordre élevé conduit à l'exécution de nombreuses opérations arithmétiques simples notamment pour le calcul des poids de remaillage. Actuellement, de nombreuses machines de calcul sont équipées de cartes graphiques. Cependant, comme nous l'avons vu dans le chapitre 1, cette tendance n'est pas immuable et peut éventuellement changer dans les prochaines années du fait des évolutions technologiques des composants. Par conséquent, dans un souci de pérennité du code, la librairie que nous développons doit pouvoir être facilement adaptée à une nouvelle architecture tout en conservant la possibilité d'une exploitation de machines classiques. D'autre part, la volonté d'une forte portabilité permettant de réaliser des simulations à la fois sur une machine de bureau, un portable ou une machine de calcul offre également une grande souplesse pour les différentes étapes de développement, de parallélisation et de production.

Dans ce chapitre, nous présentons la librairie de calcul que nous utiliserons dans la suite pour les simulations numériques. Dans un premier temps, nous exposerons la conception de cette librairie qui est basée sur une stratégie de découpage sémantique en concepts mathématiques puis en différents niveaux d'abstraction, comme suggéré par Labbé *et al.* (2004). Dans un second temps, l'architecture et les moyens de programmation des cartes graphiques sont détaillés puis nous introduirons les principes du standard de programmation OpenCL. Enfin, l'intégration des modèles de programmation des GPU à la librairie de calcul sera rapidement décrite. Cette étape du développement est largement facilitée par la conception globale de la librairie.

4.1. Développement d'une librairie de calcul scientifique

4.1.1. Conception préliminaire

Bien que de nombreux codes numériques soient encore implémentés dans un paradigme de programmation impérative, une tendance récente consiste à employer un paradigme de programmation orientée objet pour le développement de codes de calcul scientifique. L'intérêt majeur de cette technique est de pouvoir atteindre une forte flexibilité à la fois pour le développement mais aussi pour l'utilisation. La programmation orientée objet repose sur une représentation des concepts et des entités du domaine d'application sous la forme d'objets dont les interactions constituent l'exécution du programme. Le code regroupe un ensemble de classes permettant de décrire les attributs et les méthodes des objets. L'ensemble des valeurs des attributs constituent l'état d'un objet et les méthodes expriment les interactions possibles avec cet objet.

La programmation orientée objet se base sur les quatre concepts fondamentaux que sont l'encapsulation, l'héritage, l'abstraction et le polymorphisme. Leur mise en œuvre permet, lors de la conception, de décrire les relations entre les différentes classes. Selon le principe de l'encapsulation, les interactions avec les objets se font uniquement en utilisant ses méthodes et non directement ses données. Cela permet de garantir l'intégrité des propriétés de l'objet en cachant ses données. Par le principe d'héritage, il est possible de créer une classe à partir de classes existantes pour éventuellement leur ajouter de nouvelles propriétés ou fonctionnalités. De manière similaire, l'abstraction permet d'exposer uniquement les informations utiles et de laisser les détails pour des niveaux d'abstractions inférieurs. Cela permet d'accroître l'efficacité des objets ainsi que de réduire leur complexité. Enfin le polymorphisme consiste en la redéfinition de méthodes à travers les classes. L'exécution d'une méthode est réalisée par son plus proche parent dans la hiérarchie de classes si elle n'est pas définie par la classe de l'objet. L'intérêt de l'utilisation de ce type de paradigme de programmation est que cela permet aux utilisateurs et développeurs de manipuler des objets aussi proches que possibles des concepts physiques ou mathématique qu'ils représentent.

La conception d'un programme orienté objet consiste en la description de l'ensemble des classes avec leurs attributs et leurs méthodes ainsi que leurs relations. En plus de l'héritage, des relations de composition et d'agrégation figurent parmi les plus couramment utilisées. Elles permettent de décrire la combinaison de plusieurs éléments pour en former un plus complexe. La composition se distingue de l'agrégation par une notion de propriété forte. Les objets qui composent un autre plus complexe n'ont pas d'existence en dehors de ce dernier.

Découpage sémantique des concepts mathématiques

Pour le développement de codes de calculs scientifiques, il est nécessaire d'employer une méthodologie de conception plus spécifique, comme celle décrite par Labbé *et al.* (2004). Dans leur approche, les auteurs se focalisent sur la résolution de systèmes d'équations différentielles par des méthodes d'algèbre linéaire. Ils aboutissent à une décomposition d'un problème de résolution d'équations aux dérivées partielles générique en différents concepts :

Problème : Il correspond au problème mathématique global que l'on cherche à résoudre ;

Opérateur : Généralisation de la notion mathématique d'opérateur, il définit un opérateur mathématique élémentaire comme le gradient ou le laplacien et, plus généralement, un terme d'une équation du problème. Il peut être vu comme un sous-problème élémentaire.

Variable : Ce sont les entités mathématiques mises en relation par les opérateurs au sein du problème, qu'elles soient données ou inconnues.

Domaine : Il décrit le contexte formel sur lequel sont définies les variables. Il comprend la spécification d'une géométrie et des conditions aux limites.

Différents niveaux d'abstraction

Toujours en suivant la méthodologie de Labbé *et al.* (2004), nous introduisons différents niveaux d'abstractions pour les concepts définis ci-dessus. Le point clé de cette conception est un découplage des concepts abstraits, des méthodes de résolution et des algorithmes. Ainsi, l'utilisateur manipule les éléments du niveau d'abstraction le plus élevé, qui correspondent aux concepts mathématiques des problèmes, opérateurs, variables et domaines. Un second niveau est constitué des algorithmes de résolution et de la discrétisation des variables. Enfin, un dernier niveau regroupe les implémentations des méthodes spécifiques au stockage des données des variables ainsi qu'à l'architecture permettant de réaliser les calculs. L'intérêt majeur est que les opérations réalisées à un niveau ne dépendent pas de la manière dont fonctionnent les autres niveaux. En particulier, il est aisé de changer de méthode de résolution ou d'architecture sans que l'utilisateur n'ait à modifier son code.

Couplage faible et cohésion forte

La structure basée sur un découpage sémantique et en niveaux d'abstraction permet d'obtenir une forte cohésion des différents composants. Les fonctionnalités de haut niveau (employées par l'utilisateur) font appel à des fonctions de niveau inférieur jusqu'à arriver au niveau le plus bas où les calculs sont effectués dans des fonctions élémentaires réalisant une tâche précise.

Parallèlement, cela permet un couplage faible des différentes parties. A un niveau donné, les fonctionnalités ne sont pas interdépendantes. Ainsi, les effets de bords sont limités lors du développement et de la maintenance du code. Du point de vue de l'utilisation, les éléments sont autosuffisants à travers leurs niveaux d'abstraction.

En suivant ces principes, il est possible de changer d'algorithme pour une méthode donnée sans avoir à modifier le reste des éléments. De même, plusieurs versions d'un même algorithme peuvent cohabiter, du moment qu'ils implémentent la même interface. Comme remarqué dans le chapitre 1, les architectures évoluent rapidement et une implémentation peut devenir inutilisable en changeant d'architecture. D'où l'importance d'une grande flexibilité car il sera aisé d'introduire une nouvelle version pour une architecture donnée ou, au contraire, d'écarter une version incompatible avec l'architecture courante au moment du déploiement. D'autre part, les choix de conception et de langages doivent garantir une por-

tabilité du code suffisante. Ainsi, face à une nouvelle architecture, le code existant devrait pouvoir être simplement reconfiguré et non réécrit.

Schéma d'utilisation

Du point de vue de l'utilisation, le code se décompose selon les étapes suivantes :

1. Description d'un problème à partir d'un domaine sur lequel sont définies des variables utilisées par des opérateurs.
2. Initialisation du problème à travers la discrétisation et l'initialisation des variables. Cette étape consiste essentiellement en l'allocation des zones mémoires ainsi qu'au branchement des différentes méthodes numériques parmi l'ensemble des versions disponibles.
3. Résolution du problème en appliquant ses opérateurs sur leurs variables.

Ces étapes s'organisent dans le diagramme d'utilisation de la figure 4.1.

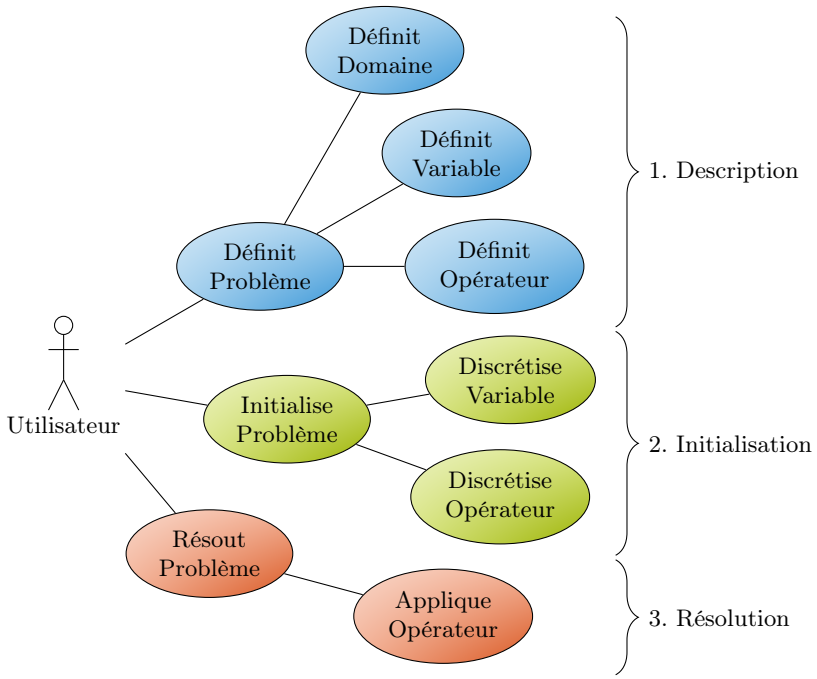


Figure 4.1. – Diagramme de cas d'utilisation.

4.1.2. Conception détaillée

Les entités issues du découpage en concepts et en différents niveaux d'abstraction sont décrites dans le diagramme de classe présenté en figure 4.2.

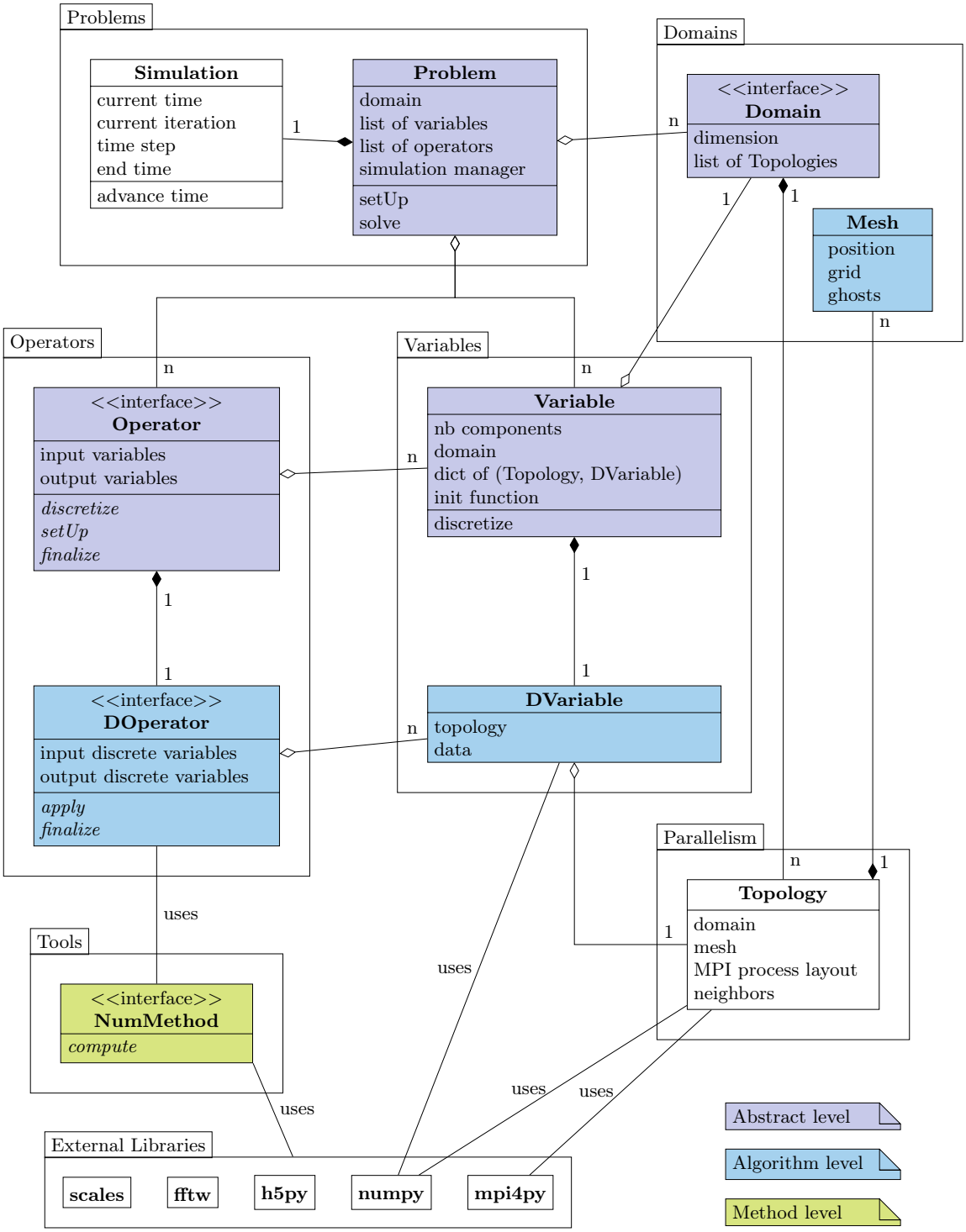


Figure 4.2. – Diagramme de classe.

Domaine

Le domaine correspond à la définition mathématique sur lequel sont définies les variables et équations du problème. Cette entité nous permet de gérer la distribution des données et des calcul dans les simulations parallèles. En effet, dans ce cadre, le domaine est distribué à travers l'ensemble des processeurs et chaque sous-domaine est affecté à un processus.

Variables

Une variable est définie sur un domaine spatial. Les vecteurs et les scalaires ne sont pas distingués et diffèrent uniquement par leur nombre de composantes. Il est possible d'avoir différentes discrétisations d'une même variable selon leurs résolutions et leurs découpages à travers les différents processus. Les données discrètes des variables sont référencées indépendemment de leur stockage par un pointeur.

Opérateurs

Un opérateur est défini à partir de ses variables d'entrée et de sortie. Un opérateur discret est créé en fonction d'une paramétrisation de la méthode numérique, et fait appel à un ou plusieurs algorithmes de calcul et méthodes numériques en utilisant les données des variables discrètes.

Méthodes numériques

Ces entités de bas niveau permettent de réaliser des calculs sur les données numériques indépendemment de leur emploi et de leur provenance. Les schémas numériques issus des méthodes de résolution, comme les intégrateurs en temps, les schémas de différences finies ou les formules de remaillage sont implémentés de manière à traiter des données numériques à partir d'un espace mémoire générique. En pratique, diverses versions d'une même méthode selon le stockage des données employé peuvent cohabiter, la version appropriée étant choisie à l'initialisation des opérateurs.

Problèmes

Comme son équivalent mathématique, un problème est construit à partir d'une liste d'opérateurs, de variables et de domaines. Il constitue une partie de l'interface utilisateur de notre librairie et permet de discrétiser l'ensemble des entités ainsi que de piloter la simulation numérique tout au long de la boucle en temps par l'application successive des différents opérateurs.

4.1.3. Fonctionnement de la librairie

Langages de programmation

La conception de la librairie de calcul, donnée en sections 4.1.1 et 4.1.2, est indépendante de tout langage de programmation. Étant données les caractéristiques requises par

notre code, le choix s'est rapidement tourné vers le langage Python. C'est un langage qui permet l'utilisation des paradigmes de programmation impératifs, objets et fonctionnels, ce qui contribue à la flexibilité du code. Python est un langage interprété, ce qui facilite le développement par l'absence d'étape de compilation. Python est présent par défaut sur de nombreux systèmes et grâce à un mécanisme de gestionnaire de modules, il est aisé d'installer les dépendances du code sur n'importe quelle machine et sans les droits d'administrateur du système. Ce langage est caractérisé par une large base de bibliothèques relatives à de nombreux domaines, en particulier pour le calcul scientifique. Il tend à être de plus en plus utilisé pour le développement de codes de calcul parallèles. Toutefois, il présente l'inconvénient, dû à l'aspect interprété, de conduire à des performances numériques de base assez médiocres dans le cadre d'une utilisation naïve. Il existe de nombreux moyens d'améliorer ces performances comme les modules `NumPy` et `Cython`. Le premier fournit un grand nombre d'outils et d'algorithmes pour le calcul scientifique. Le second propose des optimisations plus poussées permettent de retrouver des performances équivalentes à celles obtenues avec des langages compilés. D'autre part, Python est généralement utilisé comme interface pour l'utilisation de différentes librairies compilées et optimisées. Le surcoût de la couche Python est négligeable et on obtient directement les performances de ces librairies. De nombreux exemples de code de calcul scientifique sont écrits en Python (Pérez *et al.*, 2011 ; Rashed *et al.*, 2012 ; Logg *et al.*, 2012).

Dépendances externes

Fort de toutes ces caractéristiques, ce langage est en plein essor dans la communauté scientifique. Cela se traduit par une très forte activité des développeurs qui fournissent de très nombreux modules sans cesse améliorés. La plupart des librairies traditionnelles possèdent une interface Python qui sont bien souvent augmentées de fonctionnalités ou de simplifications inhérentes au langage. En particulier, nous utilisons le module `MPI4Py` qui permet d'utiliser la librairie d'envoi de message `MPI` pour le parallélisme à mémoire distribuée. De même, la librairie `H5py` permet l'écriture et la lecture de données numériques au format `HDF5`.

Dans le but de l'exploitation de machines de calcul parallèles, nous intégrons différents niveaux de parallélisme. Nous verrons par la suite que l'utilisation d'une carte graphique revient à l'expression d'un parallélisme à mémoire partagée. Nous détaillerons également les mécanismes mis en place pour l'utilisation de machines à mémoire distribuée, à travers la librairie MPI.

D'autre part, Python permet de réaliser simplement des interfaces avec des langages compilés comme le C ou Fortran. Cela nous permet d'utiliser les librairies `fftw` pour les transformées de Fourier des méthodes spectrales ainsi que l'implémentation en Fortran de la méthode semi-Lagrangienne développée par Lagaert *et al.* (2014).

Fonctionnement global

Nous achevons cette partie par une illustration de son utilisation à travers un exemple simple permettant la résolution d'un problème de transport. Un exemple de script minimal dont la majorité du code concerne l'étape de description du problème est donné en figure 4.3. Les deux fonctions permettent d'initialiser les variables. Le problème est créé avec un seul opérateur dont les calculs s'effectuent à une résolution de 128^3 lors de chacune des 10 itérations. Dans cet exemple simple, l'opérateur d'advection se construit sur ses paramètres de résolution par défaut. En pratique, il convient de donner les détails de la méthode numérique comme l'ordre d'intégration en temps ou la formule de remaillage et éventuellement des spécifications en rapport avec l'architecture employée.

Comme le montre l'exemple, l'interface utilisateur ainsi développée permet d'exprimer, à travers les classes Python, une sémantique proche du langage mathématique. Cela rend l'utilisation proche de celle proposée par un langage dédié. Un tel langage constitue l'expression idéale d'une interface utilisateur car il consiste en la définition d'un nouveau langage de programmation dont la sémantique est issue du domaine d'application du programme. Ainsi, l'utilisateur exprime ses besoins dans un langage ayant un sens pour l'application, ce qui conduit à une grande simplicité d'utilisation. Cependant, le développement de ce langage est assez difficile. Une approche plus simple consiste à embarquer seulement de nouveaux éléments de syntaxe dans le langage du programme (Mernik *et al.*, 2005).

La librairie décrite précédemment nous permet, de par sa conception, d'intégrer un module pour l'exploitation de cartes graphiques sans perte de portabilité sur les machines dépourvues de ce matériel. Les GPU diffèrent sensiblement des processeurs classiques tant par leurs architectures que par leurs modèles de programmation. Dans cette seconde partie, nous détaillons le fonctionnement de ces cartes.

4.2. Calcul générique sur carte graphique

4.2.1. Description du matériel

Fonctionnement des cartes graphiques

Les cartes graphiques que nous utilisons aujourd'hui ont été introduites à la fin des années 90. La fonction principale de ces cartes est de générer des images bidimensionnelles destinées être affichées sur un écran. Dès le milieu des années 90, les cartes sont capables de gérer un contenu tridimensionnel et sont principalement pour l'usage des jeux vidéo et à l'infographie. Dans les années 2000, l'augmentation des performances des cartes grand public rend ce matériel très compétitif en termes de capacité de calcul, relativement à leur coût d'achat. Cela conduit à l'émergence du GPGPU, acronyme anglais signifiant calcul générique sur processeur graphique.

Dans le cadre du rendu 3D, une carte graphique effectue un traitement en chaîne, représenté en figure 4.4. Premièrement, le *Vertex processor*, calcule un ensemble de points

Script Python :

```
import parmepy as pp

def s0(res, x, y, z, t=0.):
    res = ...
    return res

def u0(res, x, y, z, t=0.):
    res = ...
    return res

# Domaine
O = pp.Box(origin=[0, 0, 0],
           length=[1, 1, 1])

T = pp.Simu(tinit=0., tend=1.,
            nbIter=10))
# Variables
u = pp.Field(domain=O,
             vector=True,
             formula=v_init)
s = pp.Field(domain=O,
             vector=False,
             formula=s_init)
# Operateurs
advec = pp.Advection(
    s, velocity=u,
    resolution=[128, ]*3)
# Problème
pb = pp.Problem(
    operators=[advec, ],
    simu=T)
# Initialisation
pb.setup()
# Résolution
pb.solve()
```

Formalisme mathématique équivalent :

$s_0(\boldsymbol{x}) = \ldots$

$\boldsymbol{u}_0(\boldsymbol{x}) = \ldots$

$\Omega = [0;1]^3$

$T = [0;1]$

$\boldsymbol{u} : \Omega \times T \mapsto \mathbb{R}^3$
$\boldsymbol{u}(\boldsymbol{x}, t = 0) = \boldsymbol{u}_0(\boldsymbol{x})$

$s : \Omega \times T \mapsto \mathbb{R}$
$s(\boldsymbol{x}, t = 0) = s_0(\boldsymbol{x})$

$$\frac{\partial s}{\partial t} + (\boldsymbol{u} \cdot \nabla) s = 0$$

Figure 4.3. – Exemple de script utilisateur

à partir de la géométrie à afficher. Ensuite, les objets tridimensionnels sont projetés sur un ensemble de pixels afin d'éliminer les parties cachées par la perspective. Les pixels sont ensuite transformés par le *Fragment processor*, notamment par l'application de textures. Enfin, les pixels sont assemblés dans la grille de pixels qui constituent l'image à afficher sur l'écran. Aux débuts du calcul générique sur GPU, la programmation des cartes s'effectuait en détournant cet enchaînement d'étapes par le développement de fonctions spécifiques destinées à être exécutées par le *Vertex processor* et le *Fragment processor*.

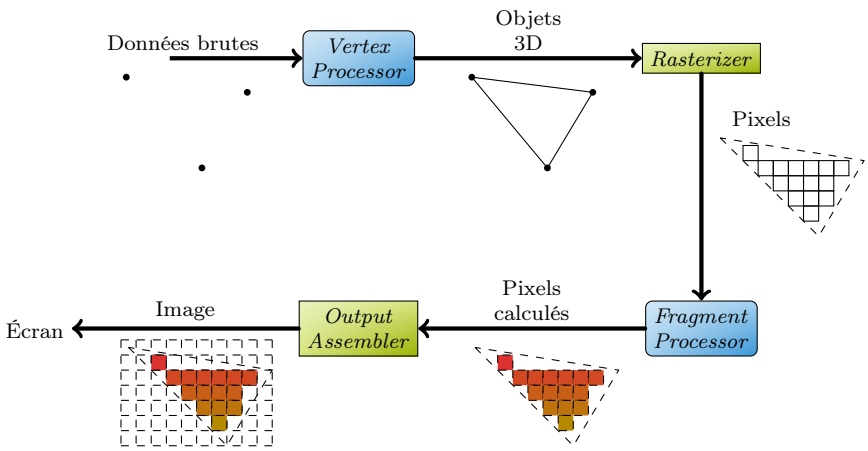
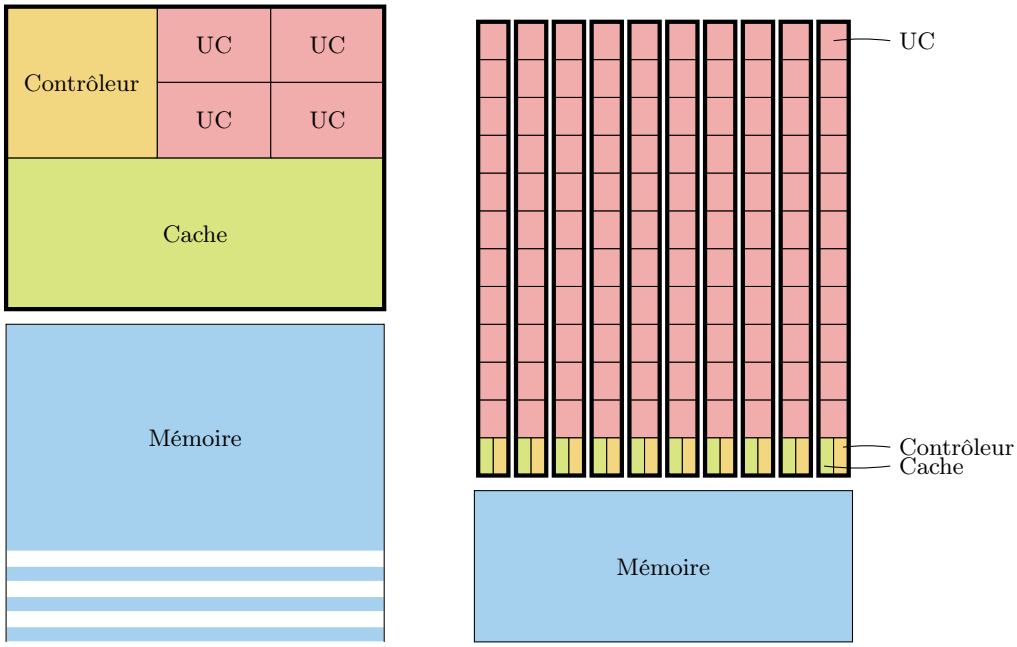


Figure 4.4. – Transformations en chaîne pour le rendu graphique

Architecture des cartes graphiques

Ce matériel spécifique est composé de nombreux éléments dont l'architecture diffère des processeurs classiques. Les composants principaux d'une carte graphique sont le processeur graphique et la mémoire vidéo. Les processeurs graphiques se caractérisent par un très grand nombre d'unités de calcul permettant de réaliser les opérations sur les sommets de la géométrie ainsi que des transformations des pixels. Ces unités sont spécialisées dans le traitement de données vectorielles jusqu'à quatre composantes (coordonnées spatiales ou canaux de couleurs d'un pixel). D'autre part, les unités de calculs réalisent de manière efficace des opérations simples sur ces données comme les additions, multiplications et produits scalaires. La mémoire vidéo est généralement divisée en deux zones distinctes. Une première partie permet des accès classiques en lecture et écriture et une seconde est spécialisée en lecture ou en écriture pour le stockage de données de textures 2D ou 3D. Comparativement à un processeur classique, un GPU permet un très haut débit de tâches élémentaires et indépendantes effectuées en parallèle par les unités de calcul (UC dans la figure 4.5). Ces dernières se partagent différents niveaux de cache. La figure 4.5 compare les représentations d'un processeur multicœur, 4.5a et d'un GPU, 4.5b. Dans les deux cas, seul le niveau de cache le plus haut est représenté. Alors que sur CPU, le nombre de cœurs dépasse rarement la dizaine, il atteint plusieurs milliers sur un GPU. D'autre part, les CPU peuvent accéder à la mémoire globale du système alors que la mémoire disponible sur un GPU est généralement inférieure à 8 GByte.



(a) CPU **(b)** GPU

Figure 4.5. – Comparaison haut niveau de l'architecture d'un CPU et d'un GPU

L'intégration de la carte graphique au système principal se fait par une connexion à la carte mère par PCI Express dont le débit varie de 1 à 16 GByte/s selon la version. Cela constitue bien souvent un goulet d'étranglement pour les applications car le temps de transfert des données est bien plus long que le temps de leur traitement sur la carte.

4.2.2. Différentes méthodes de programmation

Le calcul générique sur GPU a été grandement simplifié par les interfaces de programmations. Elles permettent de s'affranchir de la chaîne de traitement graphique, représentée en figure 4.4, au profit d'une exploitation directe des ressources des cartes. Le principe général d'un calcul sur GPU est de traiter des données régulières (proches d'une image) dont les algorithmes se décomposent selon un modèle hiérarchique en sous-blocs qui sont eux-mêmes constitués d'un ensemble de tâches élémentaires par unité de donnée. Il existe deux grandes familles de modèles de programmation des GPU dont les principaux représentants sont évoqués ci-après.

Programmation par directives

La programmation par directive sur GPU est très similaire à l'utilisation de l'interface de programmation parallèle OpenMP pour les architectures à mémoire partagée. Les standards de programmation les plus répandus sont HMPP, introduit par Dolbeau *et al.* (2007), et OpenACC, qui est désormais inclus dans OpenMP depuis sa version 4.0 (2013). Ils permettent d'exécuter des parties du code, le plus souvent des boucles, sur des accéléra-

teurs par l'insertion de directives de compilation spécifiques. L'intérêt majeur de ce modèle est de pouvoir utiliser ces accélérateurs sans avoir à réécrire le code concerné. Cependant cela empêche une gestion fine de la transcription vers les instructions GPU, cette partie du travail étant effectuée par le compilateur. Le support est généralement assuré par des compilateurs commerciaux fournis notamment par CAPS Entreprise pour HMPP ou PGI pour OpenACC. Toutefois, ce dernier est en cours l'intégration au compilateur libre GCC.

Programmation directe

À l'opposé de la famille précédente, se trouve la programmation utilisant un langage spécifique. Il en existe plusieurs pour les GPU dont les plus utilisés sont CUDA (Nvidia, 2014) et OpenCL (Khronos, 2014). Le premier est développé par la société Nvidia pour l'utilisation exclusive de ses cartes. Le second est un standard de programmation ouvert permettant l'exploitation de processeurs multicœurs, que ce soit des processeurs traditionnels, des cartes graphiques ou des coprocesseur. Les deux langages sont assez proches et se basent sur un modèle de programmation qui reflète l'architecture du matériel. Cependant, OpenCL bénéficie d'une portabilité bien plus grande car il permet d'exploiter non seulement les cartes graphiques mais aussi les processeurs multicœurs. CUDA comme OpenCL bénéficient d'interfaces performantes en Python (Klöckner *et al.*, 2012).

4.2.3. Les modèles de programmation OpenCL

Le standard de programmation OpenCL permet de développer des modèles de parallélisme par tâches et de données sur processeurs multicœurs. La librairie fournit une interface de programmation de haut niveau permettant la gestion des calculs et des ressources ainsi que le profilage des applications. Les calculs de bas niveau sont implémentés en un langage basé sur le C99. Une application OpenCL se doit de définir les quatre modèles hiérarchiques suivants. Les termes techniques sont systématiquement traduits dans cette partie mais pourront être utilisés en anglais dans la suite.

Le modèle de plateforme

Du point de vue OpenCL, une plateforme est constituée d'un hôte *(host)* et de un ou plusieurs appareils *(devices)*. Ces derniers regroupent, de manière générique, bon nombre de matériel multicœur comme les cartes graphiques mais aussi les processeurs classiques ou les processeurs spécifiques. L'implémentation consiste en une partie de code exécutée par la machine hôte *(host)* et capable d'envoyer et d'exécuter des noyaux de calcul *(kernels)* aux appareils *(devices)*. Le code des noyaux est compilé de manière spécifique aux appareils sur lequel il sera exécuté. Les noyaux sont exécutés par les cœurs *(processing elements)* de chaque unité de calcul *(compute unit)*, ce qui reflète l'organisation matérielle des unités de calculs d'un GPU, comme illustré par la figure 4.6.

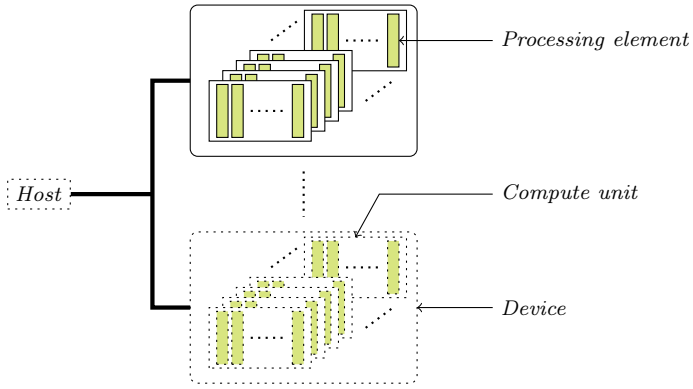


Figure 4.6. – Modèle de plateforme OpenCL

Le modèle d'exécution

On distingue deux catégories d'unités d'exécution : le programme exécuté par la machine hôte et les noyaux. À chaque appareil *(device)* est associé une ou plusieurs files d'attente de commandes dans lesquelles sont insérées les transferts de données et les appels aux noyaux. Ces derniers sont définis par un état d'exécution (soumis, en attente, démarré, terminé, ...) dont les durées entre transitions sont mesurables par l'outil de profilage intégré. Des dépendances entre les éléments peuvent être données explicitement, ce qui conduit à l'expression d'un parallélisme par tâche à travers une exécution asynchrone. La caractéristique principale est que chaque noyau est exécuté, par un cœur de calcul, comme une fonction en un point d'un espace d'indice représenté sur la figure 4.7. Cet espace est décomposé en un ensemble de groupes de travail *(work-groups)* contenant plusieurs éléments de travail *(work-items)*. À chaque élément de travail *(work-item)* est associé un cœur de calcul.

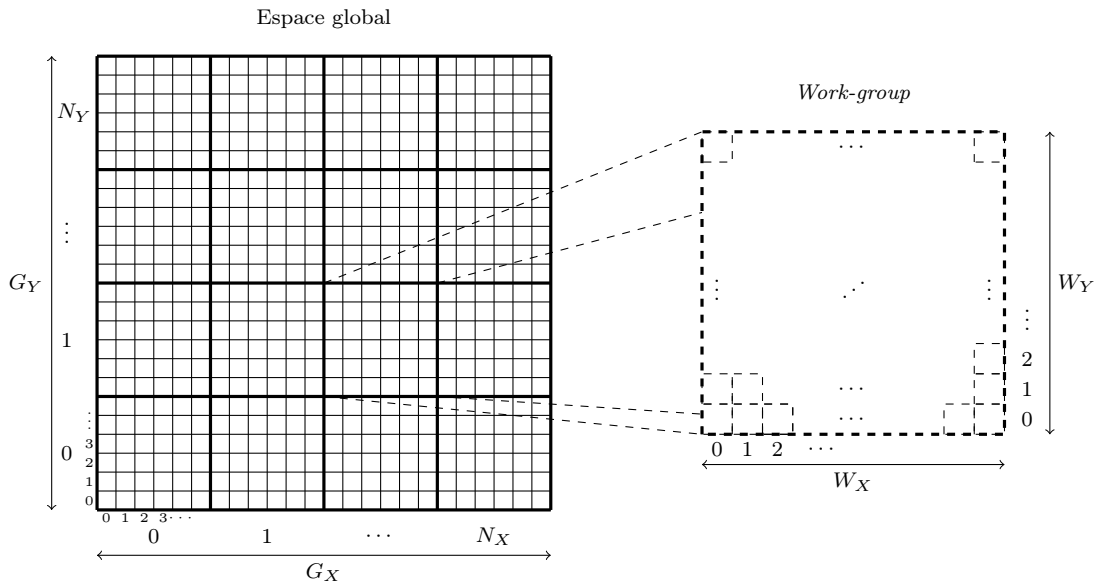


Figure 4.7. – Espace d'indices OpenCL. Espace 2D de taille $G_X \times G_Y$ impliquant $N_X \times N_Y$ *work-groups* composés de $W_X \times W_Y$ *work-items*, avec $G_\bullet = N_\bullet W_\bullet$

L'espace d'indices est un espace cartésien mono-, bi- ou tridimensionnel. Chaque élément de travail *(work-item)* est identifié de manière unique par ses coordonnées dans l'espace global ou par le couple des coordonnées du groupe *(work-group)* et d'une coordonnée locale au groupe *(work-groups)*, comme illustré par la figure 4.7. Les tailles de l'espace global et des groupes de travail sont données en paramètres à chaque exécution d'un noyau. Le parallélisme de données est obtenu en reliant l'espace d'indices et l'indexation des données.

Différents niveaux de synchronisation sont possibles. L'exécution des éléments de travail *(work-items)* est concurrente au sein d'un groupe *(work-group)*. Il est possible de synchroniser les éléments au sein d'un même groupe à l'aide de barrières. Par contre il est impossible de synchroniser les groupes entre eux, de même qu'il n'est pas possible de spécifier l'ordre dans lequel ils seront exécutes ni de savoir lesquels seront exécutés simultanément. Du côté de l'hôte, la synchronisation globale se fait par l'intermédiaire de la file d'attente et de l'état de chaque noyau de calcul.

Le modèle de mémoire

Dans ce modèle, la mémoire de l'hôte est dissociée de celle des appareils. Cette dernière est décomposée en quatre région distinctes :

Globale : niveau le plus haut, il est partagé en lecture et écriture par tous les éléments de travail de tous les groupes ($\sim 1\,\text{GByte}$).

Constante : mémoire accessible en lecture seule par les éléments de travail au cours de l'exécution. De taille assez faible, elle permet de stocker des constantes numériques ($\sim 10\,\text{kByte}$).

Locale : niveau intermédiaire partagé au sein d'un groupe de travail. Ce niveau peut être utilisé explicitement comme un cache ($\sim 10\,\text{kByte}$).

Privée : niveau le plus bas, il est dédié à un élément de travail et inaccessible aux autres ($< 1\,\text{kByte}$).

La mémoire globale ne peut contenir que des *Buffers* ou des *Images*. Les premiers sont des zones mémoires contiguës destinées à un usage générique comme un tableau d'éléments. Les seconds sont stockés dans la mémoire dédiée aux textures sous la forme d'une structure de données complexe et dont les accès, en lecture seule ou écriture seule, sont gérés par des fonctions OpenCL prédéfinies. Ce modèle hiérarchique est schématisé en figure 4.8. Comme nous le verrons par la suite, une gestion précise des transferts de données entre ces différents niveaux de mémoire est fondamentale pour l'amélioration les performances.

Le modèle de programmation

Afin d'exploiter tous les modèles définis précédemment, une application OpenCL se décompose selon trois couches dont il est nécessaire de décrire tous les niveaux. Au niveau de la plateforme, le programme hôte explore l'architecture disponible et un contexte OpenCL doit être créé afin de lier le ou les appareils utilisés avec l'hôte et pour créer les files de tâches associées à chaque appareil. Le compilateur OpenCL permet de compiler les sources des noyaux de calcul spécifiquement pour un appareil ciblé. Au niveau des modèles de mémoire et d'exécution, le programme est constitué de la gestion des files d'évènements ainsi que de

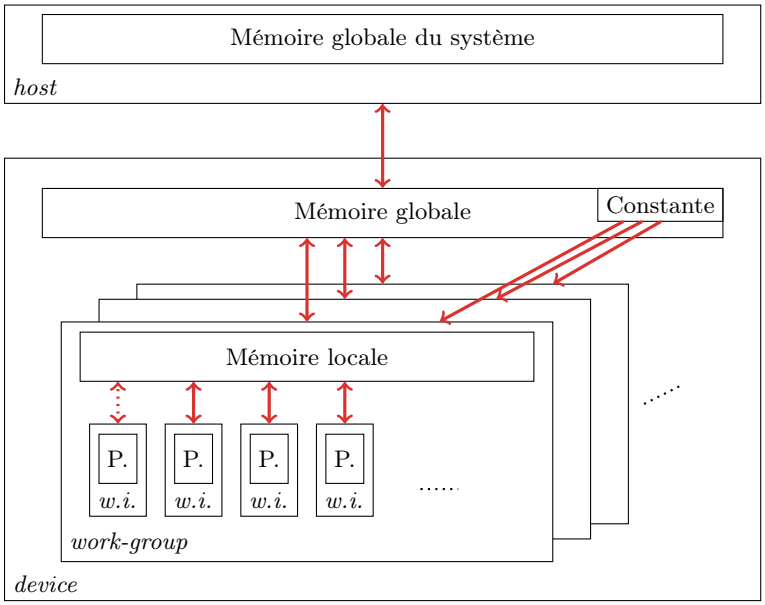


Figure 4.8. – Modèle hiérarchique OpenCL pour la mémoire

transferts de données entre les mémoires de l'hôte et des appareils. Un schéma d'exécution est représenté en figure 4.9. Dans l'idéal, la conception des algorithmes doit permettre une gestion asynchrone des évènements afin de maximiser l'utilisation des composants. Cette dernière est maximale lorsque des calculs sont effectués simultanément sur les appareils et sur l'hôte ainsi que pendant des transferts de données.

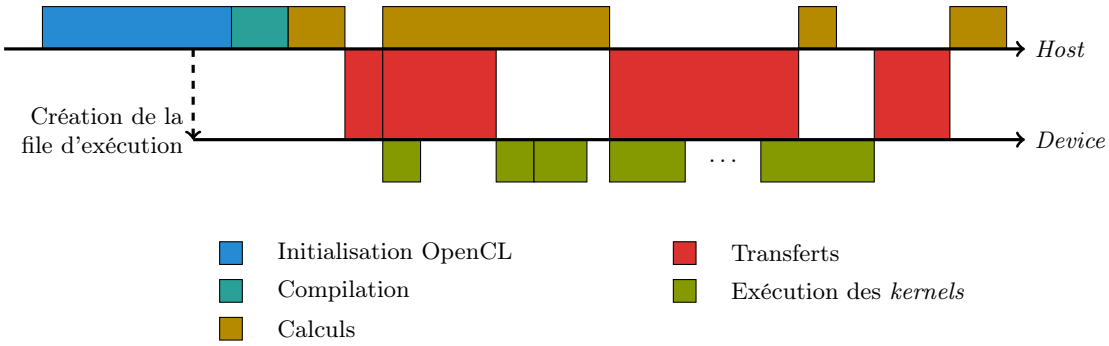


Figure 4.9. – Schéma d'exécution OpenCL

La figure 4.9 représente les durées d'exécution à titre d'illustration car les proportions entre les différentes étapes ne reflètent pas nécessairement les temps observés dans une application.

4.2.4. Analyse de performances par le modèle roofline

En préambule à l'étude des performances de l'implémentation, il est nécessaire d'explorer les performances atteignables par le matériel. Le code est développé et exploité sur différents types de machines depuis un portable jusqu'à un serveur de calcul. Une partie des caractéristiques techniques des cartes graphiques sur lesquelles le code a été exploité sont présentées dans le tableau 4.1. Les deux premières cartes sont comparables et ont permis essentiellement les développements du début de ce travail. La carte K20m est bien plus récente que les deux autres et bénéficie de technologies bien plus avancées.

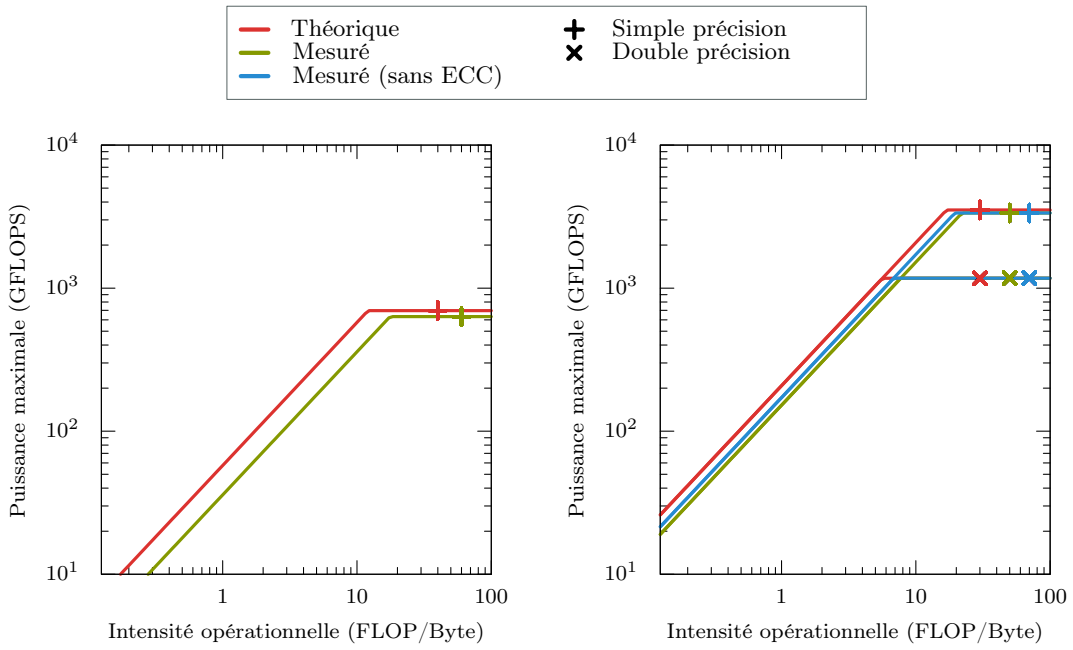
Carte	AMD Radeon HD 6770M	AMD Radeon HD 6900 Series	Nvidia K20m
Type de machine	portable	bureau	serveur
Date de mise sur le marché	01/2011	12/2010	01/2013
Taille mémoire (GByte)	1	1	5
Bande passante max. (GByte/s)	57.6	160	208
Puissance (SP) (GFLOPS)	696	2 253	3 519
Puissance (DP) (GFLOPS)	Non supporté	563	1 173

Tableau 4.1. – Comparaison de différentes cartes graphiques

Les études de performances des noyaux de calcul présentées dans la suite sont réalisées à travers le modèle roofline introduit en section 1.2.2. Nous donnons ici le modèle pour les cartes Radeon 6770M et K20m sur les figures 4.10 en utilisant les données théoriques mais aussi des mesures de performances par des programmes de test. Les deux données nécessaires à l'élaboration du roofline sont la bande passante maximale et la puissance de calcul maximale. La première est obtenue à l'aide d'un *kernel* réalisant une copie d'un *buffer* à un autre. La puissance maximale, quand à elle, est obtenue à partir d'un *kernel* ayant une grande intensité opérationnelle. Le *kernel* que nous utilisons réalise 50 instructions d'additions et multiplications sur 4 variables de type vecteur à 4 composantes en simple ou en double précision, le tout dans une boucle de 20 itérations, soit 32 000 opérations par *work-item*. Les performances mesurées sont résumées dans le tableau 4.2.

La capacité de certaines cartes graphiques à effectuer des calculs en double précision est assez récente. La raison principale est que la précision numérique n'est pas une priorité dans le cadre de l'affichage d'un contenu graphique. La principale contrainte est d'afficher des images sur l'écran en un temps le plus court possible. Cela est poussé à l'extrême dans certains modèles qui ne sont pas conformes à la norme IEEE 754 pour le calcul en virgule flottante. Sur la majorité des cartes, des options de compilations spécifiques permettent d'autoriser le compilateur à ne pas respecter cette norme. Toutefois, les cartes qui la supportent possèdent généralement moins de cœurs capables de traiter la double précision que de cœurs pour la simple précision ce qui conduit à des performances en double précision bien inférieures à la simple précision.

Le mécanisme ECC, de l'anglais *Error Correction Code*, activé par défaut sur les cartes Nvidia permet de vérifier et corriger, à l'exécution, les accès aux données en mémoire et ainsi remédier à d'éventuelles erreurs dues à des facteurs extérieurs. Ce mécanisme consomme



(a) ATI Radeon 6770M **(b)** Nvidia K20m

Figure 4.10. – Modèle roofline

Bande passante (GByte/s)	ATI Radeon 6770M		Nvidia K20m	
Théorique	57.6		208	
Mesurée	36.9	(64%)	152	(73%)
ECC désactivé	–		172	(83%)
Puissance, simple précision (GFLOPS)				
Théorique	696		3 519	
Mesurée	632	(91%)	3 350	(95%)
Puissance, double précision (GFLOPS)				
Théorique	–		1 173	
Mesurée	–		1 173	(100%)

Tableau 4.2. – Performances mesurées.

12.5% de la mémoire de la carte et réduit d'environ 12% la bande passante, comme le montre les résultats du tableau 4.2.

Dans cette dernière partie, nous détaillons la mise en pratique de l'utilisation des cartes graphiques dans la librairie de calcul ainsi que la principale mesure de performances.

4.3. Utilisation de cartes graphiques dans la librairie

L'implémentation de ces modèles de programmation est réalisée à travers le module Python `PyOpenCL` et s'insère dans le diagramme de classe présenté en figure 4.2, page 69. La figure 4.11 donne les nouvelles classes ainsi que leurs relations avec le modèle existant. Le modèle de mémoire est implémenté dans la classe `GPUDVariable` et encapsule un pointeur vers un objet mémoire GPU ainsi qu'une interface pour les transferts de données avec l'hôte. Le modèle de plateforme est intégré dans la classe outil `ClEnvironment` et permet de créer et de gérer les objets du le contexte et de la file d'attente des appareils ainsi que la compilation du code OpenCL. Enfin, le modèle d'exécution est décrit par les interactions entre les `GPUDOperators`. Les évènements créés par les appels aux kernels sont associés aux variables et permettent d'exprimer un parallélisme par tâches. L'intérêt majeur de séparer l'implémentation GPU dans un module distinct est de pouvoir s'affranchir de cette partie du code sur les machines dépourvues d'accélérateurs et d'implémentation OpenCL et ainsi, garantir une portabilité suffisante à la librairie. D'autre part, grâce à cette séparation, un module GPU basé sur le langage CUDA, par exemple, pourrait être intégré à la librairie sans aucun impact sur le reste du code.

Nous démontrons ici la simplicité d'ajout de la prise en compte d'une nouvelle architecture particulière à la librairie de calcul. En effet, seuls les éléments dépendants de cette architecture sont à implémenter.

Conclusion

Dans ce chapitre nous avons obtenu une librairie de calcul présentant une grande souplesse d'utilisation ainsi qu'une forte capacité d'adaptation à des architectures hétérogènes, comme les GPU. L'obtention de ces caractéristiques est possible à travers l'utilisation d'un paradigme de programmation orientée objet ainsi qu'une conception basée sur une stratégie de découpage sémantique des concepts mathématiques exprimés à travers différents niveaux d'abstraction. La librairie ainsi conçue permet une grande flexibilité pour le développement grâce à un faible couplage des différentes parties et à une forte cohésion à travers les niveaux d'abstraction. Le choix du langage de programmation a été fait en fonction des caractéristiques de la librairie, qui s'expriment à travers la conception, et les objectifs d'utilisation du code. Ainsi, le choix s'est porté sur le langage Python dont un des avantages est de proposer de nombreux modules pour le calcul scientifique comme le module `NumPy` ainsi que des interfaces vers des librairies externes comme les modules `PyOpenCL` pour OpenCL et `MPI4Py` pour MPI

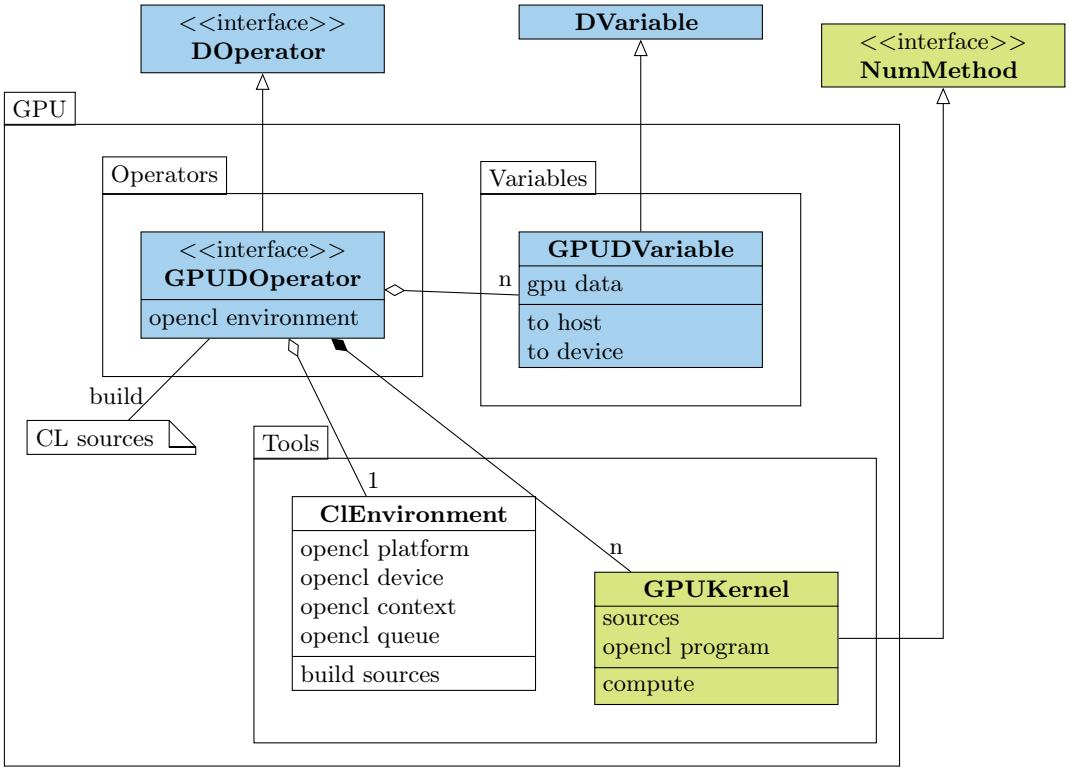


Figure 4.11. – Diagramme de classe pour le module GPU

L'exploitation des cartes graphiques semble pertinent au regard des reports de puissance de calcul développée dans la littérature que ce soit de manière théorique par le matériel en lui-même ou à travers des applications concrètes. Comme nous le verrons dans le chapitre 5, la structure de donnée nécessaire à la mise en œuvre de la méthode particulaire avec remaillage s'adapte bien à un traitement par cartes graphiques. D'autre part, les schémas numériques associés à la méthode sont compacts et locaux du fait de l'utilisation du splitting dimensionnel. Le traitement d'un point ne requiert seulement que quelques points voisins dans une seule direction de l'espace. Enfin, la majeure partie des opérations du remaillage consistent en des additions et multiplications en nombre réels qui sont effectuées de manière très efficace par les cartes graphiques.

5. Mise en œuvre sur cartes graphiques

La méthode numérique, exposée au chapitre 2, semble bien adaptée à l'utilisation de cartes graphiques dont les principales caractéristiques ainsi que les techniques d'exploitation ont été détaillés au cours du chapitre 4. L'objectif de ce chapitre est, dans un premier temps, de présenter les différentes implémentations existantes de méthodes particulaires avec remaillage sur cartes graphiques afin d'identifier leurs avantages et inconvénients. À travers l'exploration des travaux existants, l'accent est mis sur les interpolations entre les particules et le maillage. Une progression se dégage de l'évolution de l'exploitation des matériels depuis 2008. En effet, les travaux les plus anciens se basent essentiellement sur les primitives propres au traitement d'images à travers la chaîne de rendu graphique alors que la tendance actuelle est plutôt à utiliser des structures de données génériques, moins dépendantes du matériel.

Dans un second temps, nous détaillerons les choix effectués dans le cadre spécifique de la méthode semi-Lagrangienne d'ordre élevé avec splitting dimensionnel. Ces différents choix ne dépendent pas du matériel utilisé mais s'appliquent à n'importe quelle carte graphique. Ainsi, nous conservons le caractère portable de notre code. Une grande partie de ce chapitre est dédiée à l'étude des performances des principales fonctions des algorithmes issus de la méthode et de l'impact des optimisations réalisées. Ces dernières sont présentées dans le cadre concret de la carte graphique K20m mais, dans un souci de portabilité, elles ne sont généralement appliquées que lors de l'exécution, pendant la phase d'initialisation. En effet, nous exploitons un des aspects de la programmation sur GPU qui consiste à compiler le code à l'exécution. Ainsi, le code est développé de manière générique puis, après une prise de connaissance de la carte employée et de ses caractéristiques, certaines optimisations sont activées pour la compilation. Sans implémenter une auto-optimisation du code, cette étape nous permet d'adapter, entre autres, les tailles de l'espace d'indices OpenCL, la charge par work-item ou encore le schéma d'accès aux données aux spécifications de la carte utilisée.

Enfin, nous illustrerons l'utilisation de la méthode de résolution sur quelques exemples simples de transport de scalaire en 2D et 3D. Ces exemples permettent une validation de la méthode de manière qualitative ainsi qu'une description de la répartition des temps de calcul en vue d'éventuelles optimisations plus poussées.

5.1. Implémentations GPU de méthodes semi-Lagrangiennes

5.1.1. Méthodes semi-Lagrangiennes

Une première implémentation sur GPU d'une méthode semi-Lagrangienne a été proposée par Rossinelli *et al.* (2008) pour la simulation de fluides incompressibles en 2D. La diffusion et l'équation de Poisson sont résolues par différences finies sur la grille et les particules sont transportées de manière Lagrangienne puis remaillées à l'aide d'une formule du second ordre M_4'. Cette implémentation se base sur l'utilisation de la mémoire de textures. Cette mémoire spécifique aux GPU permet un stockage de pixels de telle sorte que les données voisines correspondent aux pixels voisins en 2D. Les particules sont représentées par des pixels où les composantes RGB sont associées respectivement aux positions et la vorticité des particules. Les auteurs montrent une performance globale de 10 itérations par seconde pour un problème de taille $1\,024^2$ en simple précision en utilisant une méthode de Runge-Kutta du second ordre.

Plus récemment, Rossinelli *et al.* (2010) présentent des résultats pour des simulations 2D d'écoulements en présence d'obstacles solides. La méthode employée est similaire à celle utilisée précédemment avec l'ajout de la prise en compte des obstacles par une méthode de pénalisation. Là encore, le remaillage est effectué par la formule M_4' en utilisant les fonctions intrinsèques OpenGL pour le rendu graphique et des textures sont utilisées pour le stockage des données. L'équation de Poisson est résolue par un solveur spectral utilisant l'implémentation de la FFT fournie par Nvidia en CUDA dont la résolution occupe 96% du temps de calcul total. Les simulations considérées pour ces études de performances impliquent jusqu'à $2\,048^2$ particules en simple précision.

L'inconvénient majeur de l'utilisation des textures est que les données sont arrangées en tableaux de structures. Ce stockage n'est pas le plus approprié pour des opérations n'impliquant pas toutes les composantes des éléments. D'autre part, les équivalents 3D de cette mémoire sont uniquement proposés par les dernières générations de cartes graphiques. Ainsi, le passage à une méthode de résolution de problèmes 3D nécessite une attention particulière pour l'exploitation de structures de données essentiellement 2D.

5.1.2. Deux types d'interpolations

Les méthodes semi-Lagrangiennes utilisent deux algorithmes majeurs pour les interactions entre les particules et le maillage basés sur des interpolations. Une interpolation maillage-particule permet d'évaluer sur les particules une quantité connue sur le maillage et inversement pour une interpolation particule-maillage. Le premier type d'interpolation est utilisé lors de l'advection des particules pour connaître leur vitesse alors que le remaillage exploite le second type. Stantchev *et al.* (2008) introduisent les stratégies *particle-push* et *particle-pull* pour réaliser l'étape d'interpolation particule-maillage. Ces deux stratégies diffèrent au niveau de l'organisation de la charge de travail. La première consiste en un traitement centré sur les particules dont le principe est de les remailler sur les points de grille de leur voisinage. Elle permet un calcul simple des points de grille contenus dans le support

de la particule à partir de ses coordonnées et permet également, dans un cas tensoriel, une réutilisation des poids de remaillage. Cependant, des accès conflictuels en mémoire peuvent survenir lors du traitement en parallèle de plusieurs particules situées dans la même cellule. La seconde est centrée sur le maillage de telle sorte qu'en chaque point de grille on agrège les contributions des particules voisines. Le seul avantage de la seconde stratégie est de ne pas conduire à des accès concurrents en mémoire. Dans cette stratégie, une recherche des particules les plus proches est nécessaire.

Les méthodes de type *particle-in-cell* sont basées sur ces interpolations. Pour les interpolations particule-maillage, la majeure partie des implémentations exploitent des stratégies *particle-pull* et relaxent les éventuels problèmes d'accès concurrents par un regroupement des particules voisines en blocs qui sont traités de manière séquentielle par un même processus. Ainsi, un tri des particules à deux niveaux est nécessaire, d'abord par blocs puis au sein d'un bloc. Différentes techniques sont employées comme par exemple le maintien d'une structure de données triée (Rossi *et al.*, 2012) ou un tri à chaque itération (Kong *et al.*, 2011). Quelque soit la méthode employée pour le tri, cela implique des réarrangements de données au gré des déplacements des particules afin d'assurer un haut niveau de localité des données. Les interpolations de type maillage-particule sont plus simples à mettre en œuvre puisque l'interpolation se fait sur la grille dont la structure est régulière.

Pour ce qui est des méthodes particulaires avec remaillage, les premières implémentations GPU exploitaient les primitives OpenGL du rendu graphique. En particulier, Rossinelli *et al.* (2008) montrent que le remaillage des particules est bien plus efficace, avec une facteur d'accélération de 1.5, lorsque qu'il est effectué par le moteur de rendu OpenGL plutôt que par une implémentation CUDA. Cette implémentation se base sur des objets *point-sprites* permettant la génération d'une texture contenant, après exécution du rendu en chaîne, les sommes des contributions de l'ensemble des *point-sprites*. Ainsi, l'étape exécutée par le *fragment processor* consiste à calculer les pixels de la texture par la formule de remaillage en fonction de la distance aux *point-sprites* et de leur valeur. Les auteurs soulignent ici un inconvénient de l'utilisation du langage CUDA qui est de ne pas permettre une exploitation des fonctionnalités natives du matériel, en particulier les opérations du rendu graphique. L'étape du *vertex processeur* permet, en amont du remaillage, de traiter les conditions aux bords périodiques par une réplication des particules situées à proximité des bords ainsi que d'écarter les particules transportant une quantité nulle ou inférieure à un certain seuil.

Cette approche est ensuite complétée (Rossinelli *et al.*, 2010) pour le cas de domaines ouverts. Les auteurs traitent les particules quittant le domaine local en les écartant au moment de l'étape du *vertex processor* qui précède le remaillage. D'autre part, dans leur implémentation, un remaillage systématique est effectué à chaque itération ce qui fait que les particules sont toujours situées sur un point de grille au début de l'étape d'advection et permet ainsi d'éviter une interpolation maillage-particule du champ de vitesse. Une interpolation reste nécessaire pour la seconde étape du schéma d'intégration Runge-Kutta du second ordre.

Une étude spécifique de l'interpolation maillage-particule est menée par Rossinelli *et al.* (2011) sur des architectures multicœurs CPU et GPU. Comme précédemment, l'implémentation GPU se base sur un stockage des particules en mémoire de textures. Les données d'une même particule ne sont plus stockées dans un pixel mais par composantes dans plusieurs textures. Chaque pixel contient les données de quatre points de grille dans la direction

des pixels. Dans leur étude, les auteurs utilisent la formule M_4' pour une interpolation tensorielle en 2D. Ainsi dans un traitement ligne par ligne, deux pixels consécutifs contiennent nécessairement toute l'information pour traiter une particule. Parmi les 8 données lues, seules 4 sont nécessaires, les auteurs utilisent des poids booléens afin de s'affranchir de branchements conditionnels. L'inconvénient de cette approche est qu'il est nécessaire de charger en mémoire 8 éléments (2 pixels) dans la direction des pixels pour n'utiliser que les quatre éléments du support de la particule. Les auteurs atteignent 72% de la performance maximale atteignable en simple précision et les performances chutent fortement en double précision.

Plus récemment, Büyükkeçeci *et al.* (2012) réalisent une étude des deux types d'interpolations maillage-particule et particule-maillage basées sur des stratégies respectivement *mesh-pull* et *particle-push* ce qui conduit à des algorithmes centrés sur les particules dans les deux cas. Les auteurs proposent une implémentation générique dans laquelle les particules sont réarrangées dans une structure de donnée hiérarchique permettant un stockage contigu des particules de la même région de l'espace. Cette structure se base sur un traitement séquentiel des particules dans une même cellule en utilisant autant de copies du domaine que du nombre maximal de particules par cellule. Cela conduit à n'avoir qu'une seule particule par cellule et par copie ainsi que des cellules vides. Les cellules vides sont peuplées de particules factices, écartées à la fin de l'interpolation, ce qui permet d'éviter des branchements conditionnels. Des études systématiques de performances sont réalisées sur plusieurs cartes graphiques pour chaque interpolation avec des formules linéaire ou M_4' et pour des problèmes 2D et 3D. Les auteurs soulignent que les meilleures performances sont obtenues pour les cas 3D utilisant la formule M_4'. Ils montrent des accélérations significatives du temps de calcul entre leur implémentation GPU et une référence multicœur CPU. Même si cette implémentation est efficace, les performances globales restent limitées par l'intégration à la librairie PPM. En effet, l'utilisation de ce remaillage nécessite un transfert des données en entrée ainsi que de retourner les résultats sur la mémoire de la machine hôte avant de réaliser les autres parties de la résolution. Le facteur d'accélération obtenu est généralement inférieur à 10, et même inférieur à 1 dans certains cas, en prenant en compte ce temps de transfert.

À partir des différentes implémentations existantes de méthodes semi-Lagrangiennes et en particulier des interpolations entre le maillage et les particules, nous proposons une implémentation qui exploite les différents aspects inhérents à la méthode considérée. En effet, l'utilisation d'un splitting dimensionnel est à l'origine de bon nombre de choix présentés dans ce qui suit. Les données sont stockées dans des tableaux classiques, sans aucune utilisation de structures de données spécifiques aux cartes graphiques comme les textures ou les pixels. Après une description de ces choix, nous présentons les performances de chaque fonction de la méthode pour la carte Nvidia K20m. La restriction à une seule carte permet de simplifier la présentation des résultats et l'extension à d'autres cartes se fait par adaptation des configurations aux caractéristiques techniques de ce nouveau matériel. Les performances sont données dans les métriques adaptées à chaque cas.

5.2. Implémentation et performances

5.2.1. Adéquation de la méthode au matériel

Les implémentations réalisées dans le cadre de ce travail sont basées, à l'instar de Büyükkeçeci *et al.* (2012), sur des stratégies *mesh-pull* pour les interpolations de type maillage-particule ainsi que *particle-push* pour le type particule-maillage. Toutefois, l'utilisation d'un splitting dimensionnel nous permet d'envisager l'emploi de formules de remaillage d'ordre élevé ayant un large support. Du point de vue de l'implémentation, cela conduit à un algorithme avec une localité des données plus forte que dans le cadre de méthodes classiques. En effet, le traitement d'une donnée (particule ou point de grille) ne nécessite que quelques données voisines uniquement dans la direction courante. La méthode revient à la résolution de nombreux sous-problèmes monodimensionnels ce qui pousse naturellement à l'emploi d'une structure de donnée également 1D. Ainsi, nous utilisons des objets mémoire linéaires comme les *buffers* qui correspondent à des tableaux classiques. Le découpage par splitting dimensionnel permet également de considérer séparément les composantes des vecteurs, en particulier de la vitesse, lors de l'étape d'advection. Nous employons donc une structure de donnée en tableaux par composantes et non pas en tableau de structures. En pratique les composantes d'un vecteur sont stockées sous la forme de plusieurs tableaux de scalaires.

Dans la suite de cette thèse, nous prenons comme convention, sans que cela n'implique aucune restriction, que les données sont stockées dans un ordre en colonne, à l'instar du langage Fortran. Le stockage monodimensionnel des données d'un tableau de taille (N_X, N_Y, N_Z) dans un espace de taille $N_X \times N_Y \times N_Z$ est illustré en figure 5.1.

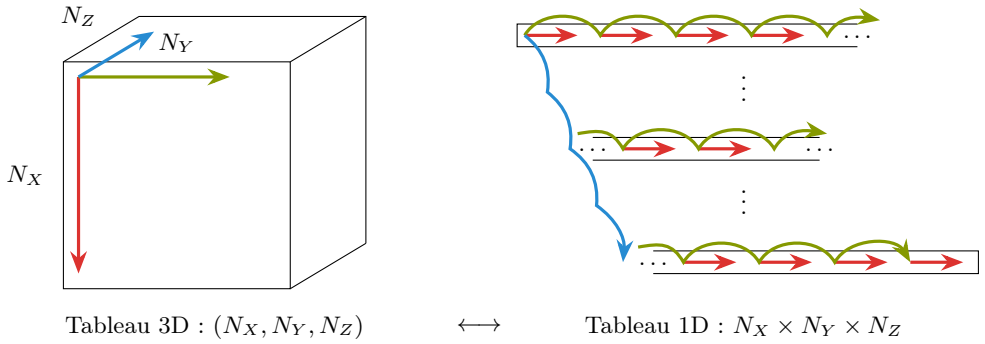


Figure 5.1. – Convention de stockage des données multidimensionnelles

Ainsi, lors des différentes étapes de splitting, le traitement de la première direction de stockage des données conduit à des accès contigus en mémoire. Ce qui n'est plus le cas lors du traitement des autres directions. Afin d'éviter une dégradation des performances due à ces accès, nous mettons en place des transpositions des données pour le traitement des différentes directions de l'espace. En effet, les performances des accès en mémoire globale sont généralement plus faibles lors d'accès avec un saut, même constant, comme représenté par les flèches vertes et bleues sur la figure 5.1. Ainsi, l'accès aux données est toujours contigu en mémoire pour toutes les directions. Ce choix conduit à trois remarques à propos de la mise en œuvre de ces transpositions. Premièrement, le champ de vitesse $\boldsymbol{u}$ n'est

utilisé que composantes par composantes dans les étapes de splitting. Lors de l'advection dans la direction X, seule la composante $\boldsymbol{u}_X$ est utilisée. Par conséquent, les composantes des vecteurs, en particulier de la vitesse, sont toujours stockées de manière a avoir un accès contigu dans la direction concernée. Deuxièmement, les positions des particules à l'issue de l'étape d'advection ne changent que dans leur composante correspondant à la direction courante, les autres composantes restent inchangées car le déplacement est 1D. Cela nous permet de réduire le vecteur position des particules à un scalaire correspondant à la composante de la direction courante et ainsi de réduire l'emprunte mémoire de la méthode. Enfin, en dehors des transpositions, toutes les directions sont traitées de manière identique. En pratique, le même code s'applique dans toutes les directions moyennant une paramétrisation des dimensions des tableaux dans les cas de grilles non cubiques.

Comme nous l'avons détaillé au chapitre 2, les trois étapes de résolution consistent en une initialisation en chaque point de grille, un déplacement puis un remaillage des particules. Lors de l'étape d'initialisation, la position des particules est directement donnée par la coordonnée du point de grille associé et la quantité transportée par la particule est simplement une copie de la valeur au point de grille. Ainsi dans notre implémentation, cette initialisation est remplacée par une transposition lorsque la direction courante change. En effet, une simple copie suffit dans le cas contraire car les données sont déjà dans le bon ordre de stockage.

Les algorithmes 1 à 4 détaillent les étapes d'une itération complète de la méthode. Les données manipulées sont limitées aux composantes du champ de vitesse, aux valeurs sur la grille des quantités transportées, à un scalaire pour la position des particules et aux valeurs transportées par les particules. La figure 5.2 illustre l'algorithme 2 dans le cas d'un splitting du second ordre en 2D et pour un domaine rectangulaire. Les données manipulées par les opérations d'advection et de remaillage pour un sous-problème 1D sont représentées par une bande rouge verticale dans les tableaux.

Algorithme 1 : 2D, splitting d'ordre 1

Direction X :
1. $\tilde{u} \leftarrow$ Copie (u)
2. $\tilde{x} \leftarrow$ Advection $(dt,\ \boldsymbol{a}_X)$
3. $u \leftarrow$ Remaillage $(\tilde{x},\ \tilde{u})$

Direction Y :
4. $\tilde{u} \leftarrow$ TranspositionXY (u)
5. $\tilde{x} \leftarrow$ Advection $(dt,\ \boldsymbol{a}_Y)$
6. $u \leftarrow$ Remaillage $(\tilde{x},\ \tilde{u})$

Réarrangement des données :
7. $\tilde{u} \leftarrow$ TranspositionXY (u)
8. $u \leftarrow$ Copie $(\tilde{u})$

Algorithme 2 : 2D, splitting d'ordre 2

Direction X :
1. $\tilde{u} \leftarrow$ Copie (u)
2. $\tilde{x} \leftarrow$ Advection $(dt/2,\ \boldsymbol{a}_X)$
3. $u \leftarrow$ Remaillage $(\tilde{x},\ \tilde{u})$

Direction Y :
4. $\tilde{u} \leftarrow$ TranspositionXY (u)
5. $\tilde{x} \leftarrow$ Advection $(dt,\ \boldsymbol{a}_Y)$
6. $u \leftarrow$ Remaillage $(\tilde{x},\ \tilde{u})$

Direction X :
7. $\tilde{u} \leftarrow$ TranspositionXY (u)
8. $\tilde{x} \leftarrow$ Advection $(dt/2,\ \boldsymbol{a}_X)$
9. $u \leftarrow$ Remaillage $(\tilde{x},\ \tilde{u})$

Dans tous les algorithmes (1 à 4), les deux premières étapes de chaque direction sont indépendantes. Ainsi, lors de l'exécution, un parallélisme par tâches pourra être exprimé et conduira à une exécution simultanée des étapes d'advection et des étapes de copie et de transposition. Une étape de réarrangement des données apparaît en fin d'algorithme

Algorithme 3 : 3D, splitting d'ordre 1

Direction X :
1. $\tilde{u} \leftarrow$ Copie (u)
2. $\tilde{x} \leftarrow$ Advection $(dt, \boldsymbol{a}_X)$
3. $u \leftarrow$ Remaillage $(\tilde{x}, \tilde{u})$

Direction Y :
4. $\tilde{u} \leftarrow$ TranspositionXY (u)
5. $\tilde{x} \leftarrow$ Advection $(dt, \boldsymbol{a}_Y)$
6. $u \leftarrow$ Remaillage $(\tilde{x}, \tilde{u})$

Direction Z :
7. $\tilde{u} \leftarrow$ TranspositionXZ (u)
8. $\tilde{x} \leftarrow$ Advection $(dt, \boldsymbol{a}_Z)$
9. $u \leftarrow$ Remaillage $(\tilde{x}, \tilde{u})$

Réarrangement des données :
10. $\tilde{u} \leftarrow$ TranspositionXZ (u)
11. $u \leftarrow$ TranspositionXY $(\tilde{u})$

Algorithme 4 : 3D, splitting d'ordre 2

Direction X :
1. $\tilde{u} \leftarrow$ Copie (u)
2. $\tilde{x} \leftarrow$ Advection $(dt/2, \boldsymbol{a}_X)$
3. $u \leftarrow$ Remaillage $(\tilde{x}, \tilde{u})$

Direction Y :
4. $\tilde{u} \leftarrow$ TranspositionXY (u)
5. $\tilde{x} \leftarrow$ Advection $(dt/2, \boldsymbol{a}_Y)$
6. $u \leftarrow$ Remaillage $(\tilde{x}, \tilde{u})$

Direction Z :
7. $\tilde{u} \leftarrow$ TranspositionXZ (u)
8. $\tilde{x} \leftarrow$ Advection $(dt, \boldsymbol{a}_Z)$
9. $u \leftarrow$ Remaillage $(\tilde{x}, \tilde{u})$

Direction Y :
10. $\tilde{u} \leftarrow$ TranspositionXZ (u)
11. $\tilde{x} \leftarrow$ Advection $(dt/2, \boldsymbol{a}_Y)$
12. $u \leftarrow$ Remaillage $(\tilde{x}, \tilde{u})$

Direction X :
13. $\tilde{u} \leftarrow$ TranspositionXY (u)
14. $\tilde{x} \leftarrow$ Advection $(dt/2, \boldsymbol{a}_X)$
15. $u \leftarrow$ Remaillage $(\tilde{x}, \tilde{u})$

dans les cas d'un splitting du premier ordre. Cette étape est nécessaire pour retrouver le scalaire dans l'ordre de stockage initial de l'algorithme, afin de rendre ces transformations transparentes. En effet, dans tous les cas, les données du scalaire sur la grille sont stockée en fin d'algorithme dans le même ordre qu'au début.

Les kernels sont donc développés indépendemment de la direction de résolution. Nous séparons les développements de kernels de l'étape d'initialisation de ceux du calcul. Les premiers regroupent les opérations de copie et les transpositions qui ne nécessitent aucun calcul. Leurs performances sont analysées en terme de bande passante. Les performances des seconds sont données dans le modèle roofline.

5.2.2. Préambule à l'étude des performances

Sauf mention contraire, dans la suite de ce chapitre, nous présenterons des résultats en double précision pour la carte Nvidia K20m.

En l'absence d'outils d'optimisation automatique, nous exploitons les caractéristiques du matériel pour paramétrer les exécutions des kernels. En particulier, le lien entre la taille des données et l'espace d'indice OpenCL ne suffit pas à déterminer complètement le nombre et la taille des *work-groups*. Dans l'état actuel de la librairie, cette paramétrisation des noyaux est réalisée de manière explicite pour les différents cas d'utilisation tels que le nombre de dimensions du problème, la précision utilisée et le matériel employé. Les paramètres donnant les meilleures performances ont été obtenus par petites variations autour de valeurs en

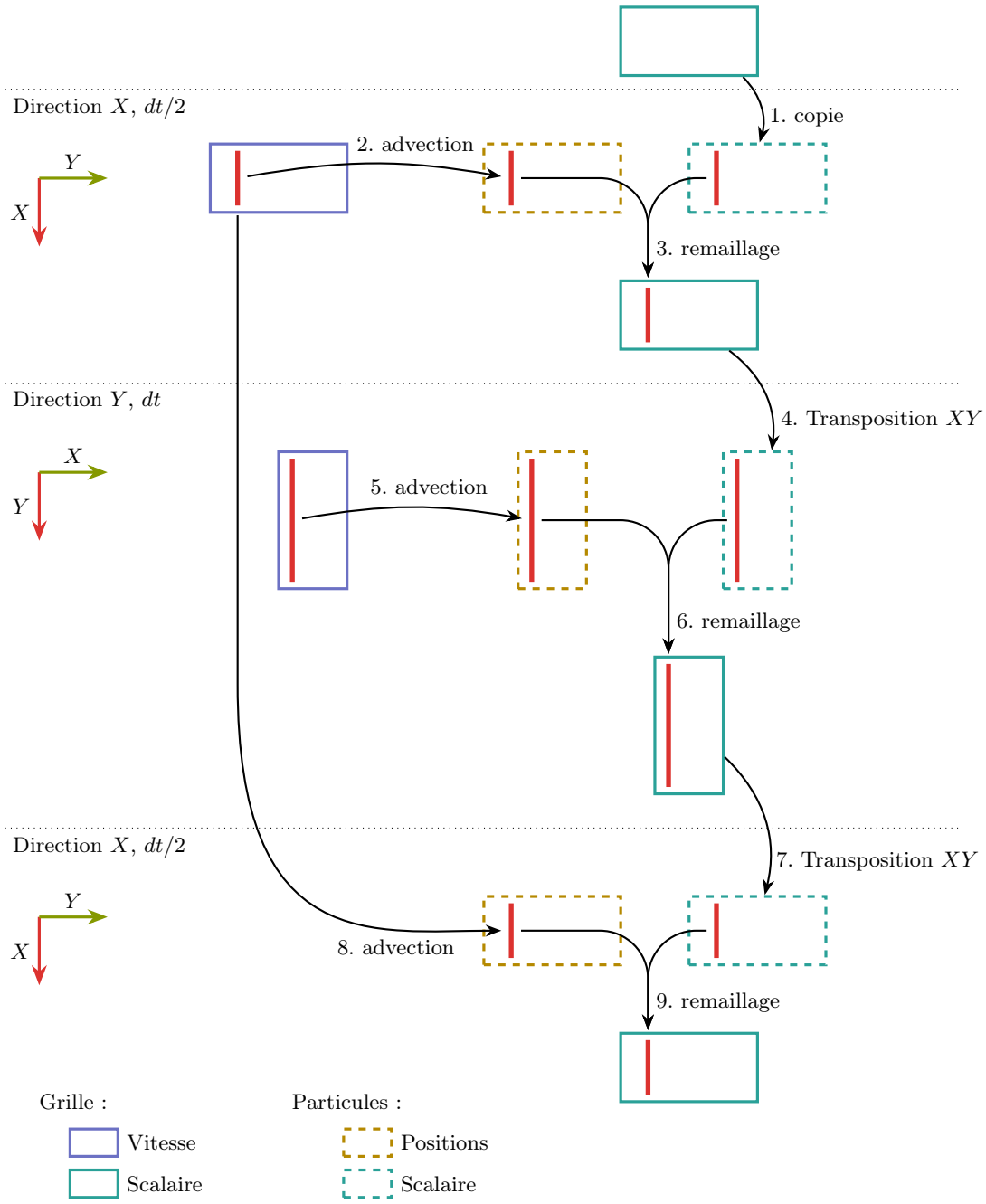


Figure 5.2. – Algorithme pour une itération en 2D, splitting de Strang

adéquation avec le matériel. Le tableau 5.1 rappelle et détaille les principales caractéristiques pour la carte K20m. Le vocabulaire employé dans ce tableau est celui du langage CUDA pour les cartes Nvidia. Pour les cartes ATI, l'architecture est similaire et les *warps* sont appelés *wavefronts*. D'autre part les termes *thread* et *thread block* sont les équivalents CUDA de ce que sont les *work-items* et *work-groups* pour OpenCL.

Puissance double (simple) précision	1 173 (3 519) GFLOPS
Cœurs double (simple) précision	832 (2496)
Bande passante	208 GByte/s
Multiprocesseurs (MP)	13
Cœurs double (simple) précision par MP	64 (192)
Mémoire partagée par MP	48 kByte
Registres 32-bit par MP	65536
Nb. *Threads* par *warp*	32
Nb. max. de *warps* par MP	64
Nb. max. de *threads* par MP	2048
Nb. max. de *thread blocks* par MP	16
Nb. max. de registres par *thread*	255
Nb. max. de *threads* par *thread blocks*	1024

Tableau 5.1. – Caractéristiques techniques de la carte Nvidia K20m (Nvidia, 2012b ; Nvidia, 2012a)

De manière générale l'obtention de bonnes performances sur GPU se fait en considérant finement les caractéristiques du matériel. Comme nous l'avons souligné précédemment, la mesure des performances se fait par diverses métriques souvent complémentaires comme la puissance de calcul ou la bande passante. L'occupation est une métrique spécifique aux cartes graphiques qui permet de mesurer l'utilisation des multiprocesseurs. Lors de l'exécution d'un kernel, les *work-item* sont associés aux cœurs physiques de la carte par groupes constituant un *warp*. Les multiprocesseurs possèdent un planificateur permettant une exécution simultanée de plusieurs *warps*. L'occupation est donc simplement le rapport entre le nombre de *warps* exécutés simultanément par multiprocesseur et le nombre maximal supporté. Ainsi, lors d'une exécution d'un kernel, la spécification de l'espace d'index ainsi que la quantité de mémoire partagée et le nombre de registres utilisés peuvent limiter l'occupation. Des outils de calcul d'occupation sont proposés à la fois par Nvidia [1] et AMD [2]. Le calcul de l'occupation nécessite la connaissance du nombre de registres utilisés par *work-item*. Cette information peut être obtenue par l'analyse des binaires produits par les compilateurs pour une carte spécifique. L'option de compilation `-cl-nv-verbose` permet d'obtenir ce nombre lors de la compilation avec OpenCL sur une carte Nvidia.

Toutefois, comme le montre Volkov (2010) sur des exemples simples, la maximisation de l'occupation ne permet pas toujours de maximiser les performances. En effet, le temps que les unités de calculs mettent à effectuer les opérations arithmétiques ou à réaliser un accès en mémoire peut être exploité afin d'augmenter les performances. La latence arithmétique peut

1. `http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls`
2. `http://developer.amd.com/tools-and-sdks/archive/amd-app-profiler/user-guide/app-profiler-kernel-occupancy/`

être cachée en augmentant la charge de travail par *work-item* afin de bénéficier d'opérations non interdépendantes pouvant être exécutées simultanément. De même, il est possible de cacher la latence mémoire en utilisant d'avantage de registres et en réduisant le nombre de *work-item*. Ces optimisations au niveau des instructions (*Instruction Level Parallelism*) impliquent généralement une augmentation du nombre de registres et une diminution du nombre de *work-item* ce qui peut conduire à une diminution de l'occupation. De manière générale, il n'est pas possible de connaître a priori quelles optimisations sont préférables entre les instructions et l'occupation. C'est pourquoi dans les résultats qui vont suivre, les noyaux n'ont pas été optimisés systématiquement à différents niveaux. De plus, dans un souci de portabilité et de non spécialisation à une carte en particulier, les optimisations restent générales et basées uniquement sur les connaissances des caractéristiques techniques.

5.2.3. Initialisation des particules

Les étapes d'initialisation des particules se caractérisent par une absence totale de calculs. Les opérations consistent uniquement en des transferts de données. Ainsi, les performances sont mesurées en terme de bande passante.

Copie

Cette opération consiste en une simple copie des données entre deux zones mémoire. Une fonction de copie directe est proposée par l'interface OpenCL. Dans cette comparaison, nous utilisons la fonction `clEnqueueCopyBuffer` pour la copie d'un buffer complet vers un autre. La figure 5.3 illustre le fonctionnement simple de ce kernel. La copie est réalisée directement par blocs successifs, en pointillés sur la figure. Chaque bloc est traité par un *work-group* dont les *work-items*, en pointillés, traitent un seul élément du tableau. La taille du *work-group* est fixée à 256 *work-item* ce qui permet d'atteindre une occupation maximale avec 8 *warps* par multiprocesseur.

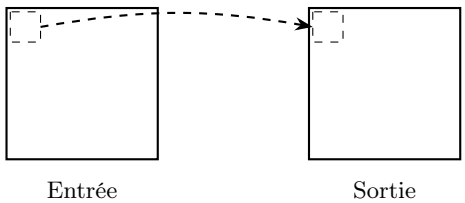
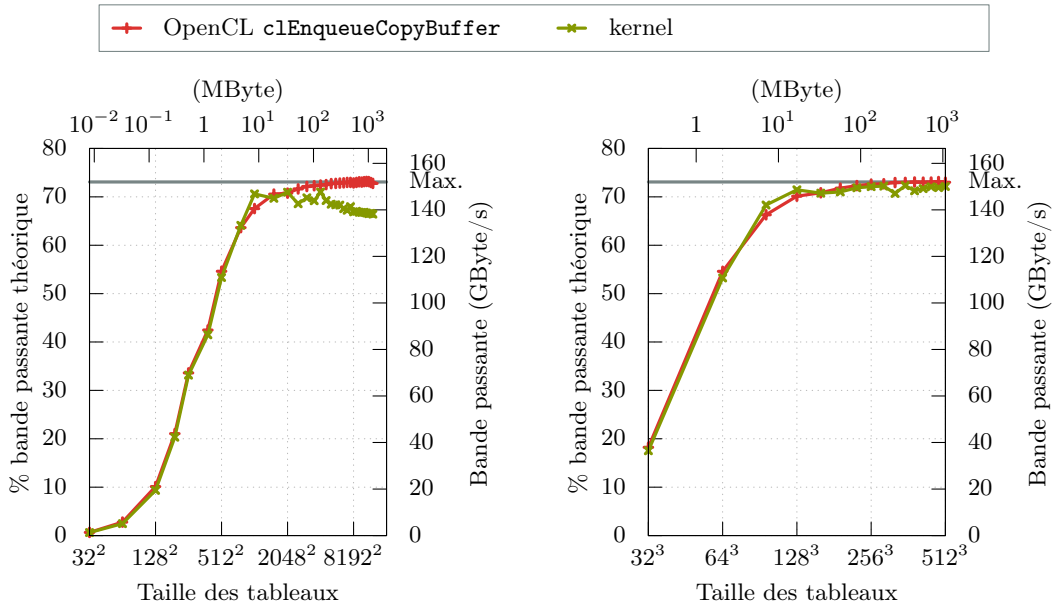


Figure 5.3. – Fonctionnement du kernel de copie

La figure 5.4 présente les performances atteintes par le kernel et la fonction OpenCL pour différentes tailles de tableaux en 2D (5.4a) et 3D (5.4b). Les résultats obtenus sont comparables et atteignent la bande passante maximale mesurée dès lors que la taille des tableaux est suffisamment grande. En effet, l'exécution sur des données trop petites ne permettent pas une occupation maximale de la carte et donc conduit à de mauvaises performances. Ainsi, nous donnons les performances pour les petites tailles, inférieures à 1024^2 ou à 128^3, seulement à titre indicatif.



(a) 2D **(b)** 3D

Figure 5.4. – Performances des copies

Transposition XY

Nous appelons transposition XY une transposition 2D dans le plan XY. Dans le cas de données 2D, cela revient à une transposition classique. Cette opération est bien connue et l'implémentation proposée ici se base sur l'exemple donné par Ruetsch *et al.* (2009) pour la transposition d'une matrice de taille $2\,048^2$ en simple précision. Les auteurs présentent différentes optimisations permettant d'obtenir une bande passante pour la transposition égale à 70% de celle obtenue pour une copie. Nous étendons cette approche à différentes tailles de tableaux 2D et 3D en double précision. Pour les données 3D, une telle transposition est effectuée pour chaque plan dans la direction Z.

La transposition optimale est schématisée en figure 5.5 et fait apparaître les trois niveaux d'optimisation que nous détaillons ci-après. Les données représentées en rouge et en vert sont traitées par les *work-items* d'un même *work-group* et une boucle permet de compléter les sous-tableaux en pointillés.

Une transposition naïve, similaire au kernel de copie, réalisant une écriture directe dans le tableau de sortie conduit à de très mauvaises performances. En effet, une telle implémentation implique des accès en mémoire globale non contigus. Une solution consiste à utiliser un tableau intermédiaire en mémoire partagée permettant de réaliser des accès contigus à la fois en lecture et en écriture pour les tableaux en mémoire globale. Dans ce cas, l'écriture des données en colonnes dans le tableau de sortie revient à lire des lignes en mémoire locale.

Cependant, cette approche provoque des conflits de banc lors de la lecture des données en ligne dans la mémoire partagée. Ce phénomène vient du fait que la mémoire partagée est organisée en différents bancs, dont le nombre est en général égal à 32. Ces bancs ne permettent qu'un accès séquentiel aux données et on assiste à un conflit lorsque plusieurs

processus accèdent simultanément à des données situées dans le même banc, les accès sont alors effectués de manière séquentielle. Du fait du choix de la taille de cette zone mémoire, 16^2 ou 32^2, les données d'une même ligne sont situées dans le même banc. En suivant l'exemple de Ruetsch *et al.* (2009), on alloue une zone de taille 17×16 ou 33×32 pour décaler les données dans les bancs et ainsi avoir des accès sans conflits. Une autre approche consiste à accéder aux données du tableau en mémoire partagée dans un système de coordonnées diagonale (Baqais *et al.*, 2013). Si x et y sont respectivement les indices de ligne et de colonne du sous-tableau traité par un *work-group*, les données correspondantes sont stockées aux indices x' et y' avec :

$$\begin{aligned} x' &= (x+y)\%T \quad \text{avec } T \text{ la taille du sous-tableau,} \\ y' &= x. \end{aligned}$$

Ainsi, les accès ne provoquent pas de conflits et la mémoire allouée est entièrement utilisée. Cependant, cette approche n'a pas été envisagée pour l'instant car la version avec extension du tableau conduit à des performances satisfaisantes. D'autre part, la réduction d'utilisation de mémoire partagée n'est pas suffisamment importante pour influencer l'occupation de manière significative.

Enfin, comme la mémoire partagée, la mémoire globale est organisée en partitions si bien que le traitement simultanée d'une colonne de sous-tableaux conduit à la lecture de données dans différentes partitions et à une écriture dans les même partitions. Une structuration des sous-tableaux en coordonnées diagonales permet des accès non restreints à un petit nombre de partitions (flèches pointillées sur la figure). Cependant, nos expérimentations sur des cartes plus récentes que celles utilisées par Ruetsch *et al.* (2009) montrent que ce réarrangement en diagonale ne conduit pas à une augmentation significative des performances.

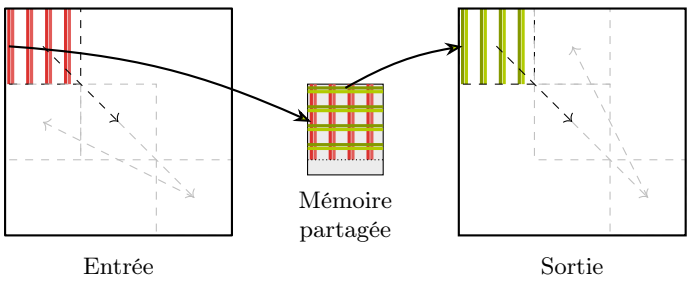
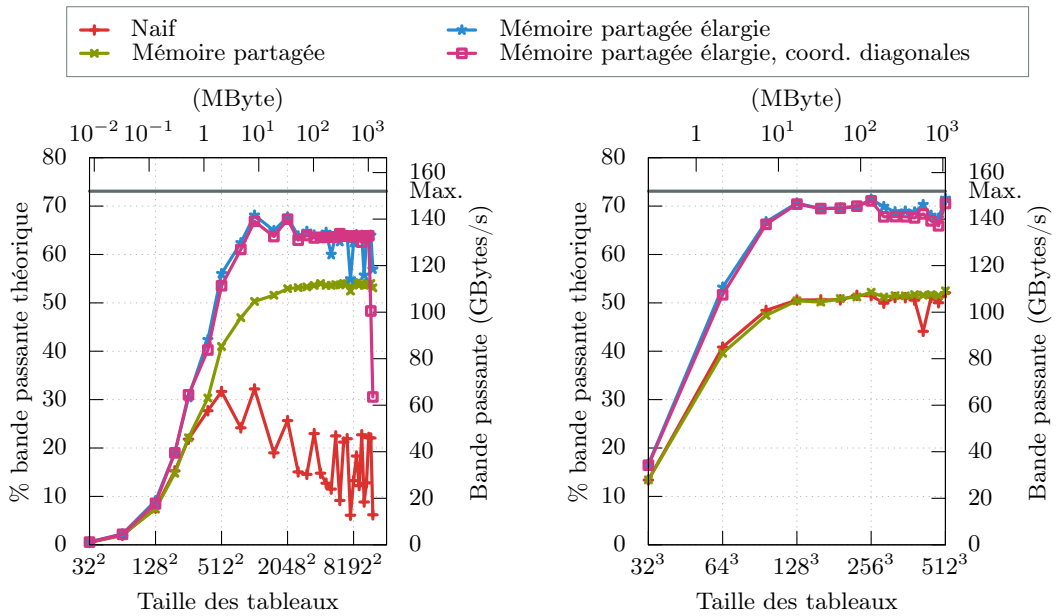


Figure 5.5. – Fonctionnement du kernel de transposition XY

La figure 5.6 présente les performances obtenues pour les trois niveaux d'optimisation de la transposition XY en 2D et 3D pour différentes tailles de tableaux ainsi qu'une implémentation naïve. La mise en place de ces niveaux d'optimisation conduit à une bande passante pour la transposition supérieure à 80% de la bande passante maximale mesurée pour la copie.



(a) 2D **(b)** 3D

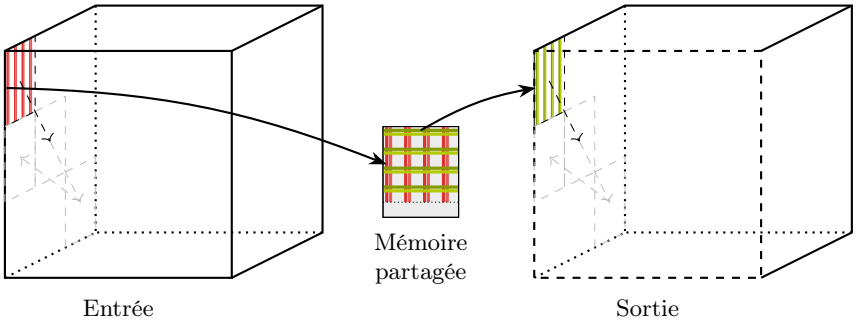
Figure 5.6. – Performances des transpositions XY

Lorsque la taille du sous-tableau alloué en mémoire partagée est fixé à 33×32, soit 8448Byte, et que les *work-groups* contiennent 512 *work-items*, 12 registres sont utilisés ce qui conduit à une occupation maximale avec l'exécution simultanée de 4 *work-groups* par multiprocesseurs.

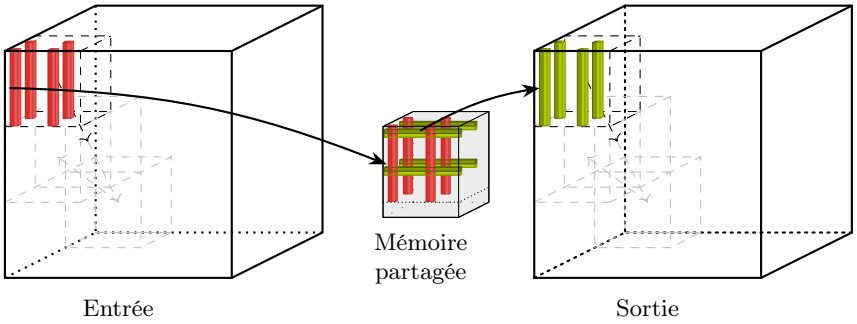
Transposition XZ

Cette transposition est réalisée de manière similaire à la transposition XY comme illustré en figure 5.7a. L'opération se réalise par le traitement de sous-tableaux 2D formés dans les plans XZ successifs. Chacun de ces plans est traité séparément par plusieurs sous-tableaux avec l'utilisation d'une mémoire partagée élargie et avec des coordonnées diagonales pour les différents blocs. Cependant, bien que les accès soient contigus dans une même colonne des sous-tableaux, le passage d'une colonne à une autre dans les tableaux 3D se fait avec un saut bien plus important que pour la transposition XY. Ce phénomène pénalise grandement les performances sur des cartes plus anciennes et une seconde transposition a été implémentée par sous-tableau 3D, illustré par la figure 5.7b. Le principe reste le même mais les données sont lues également dans la direction Y, avec un saut bien plus faible. Comparativement à la version 2D, cela revient à réaliser les transpositions simultanées de plusieurs plans XZ. Cependant, les dimensions de la zone en mémoire partagée imposent une taille plus faible pour les *work-groups*.

Les performances de cette transposition sont représentées sur la figure 5.8. Il apparaît clairement que l'utilisation d'une transposition par bloc 3D n'est pas optimale du fait d'une trop faible occupation (16%) induite par une très grande taille de mémoire partagée par



(a) Par tranches 2D



(b) Par blocs 3D

Figure 5.7. – Fonctionnement du kernel de transposition XZ

work-groups, (16^3 éléments). La meilleure configuration pour cette transposition est obtenue pour une occupation de 63%.

5.2.4. Advection et remaillage

Advection

Cette fonction permet de résoudre l'équation de mouvement des particules à partir du champ de vitesse connu sur la grille. Les particules sont initialisées en chaque point de grille, leur vitesse en ce point est donc directement lue sur la grille. Les schémas de Runge-Kutta d'ordre deux et quatre, notés respectivement RK2 et RK4, nécessitent des évaluations du champ de vitesse à plusieurs positions intermédiaires qui sont réalisées par des interpolations linéaires de type maillage-particule. Une stratégie centrée sur les particules permet de réaliser ces interpolations à la volée au cours de l'advection. Ainsi, une particule est traitée en une seule fois mais nécessite plusieurs accès aux données du champ de vitesse. De plus le schéma d'accès est inconnu car il dépend du champ de vitesse. Toutefois, du fait de la condition de non croisement des particules, ces accès restent ordonnés de la même manière que les particules.

La stratégie de distribution des calculs sur l'espace d'indices des GPU est choisie de telle sorte qu'un sous-problème 1D est traité par les *work-items* d'un seul *work-group*. Ces sous-problèmes étant indépendants, plusieurs pourront être traités simultanément en

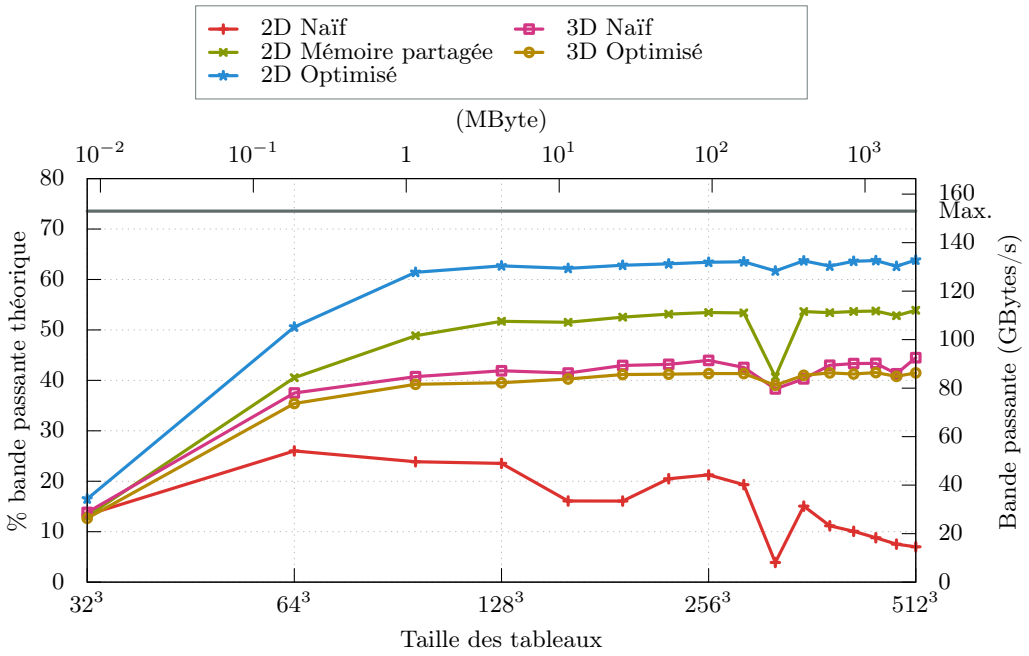


Figure 5.8. – Performances des transpositions XZ

fonction de l'occupation des multiprocesseurs de la carte. Les multiples accès aux données du champ de vitesse peuvent être évités en utilisant une zone en mémoire partagée, comme illustré sur la figure 5.9. Cela permet, après un remplissage par accès contigus en mémoire globale, à des accès plus rapides et dont le schéma inconnu est moins pénalisant. Les *work-items* réalisent, dans une premier temps, un chargement en mémoire partagée des données du champ de vitesse nécessaires au traitement d'un problème 1D. Cette opération, qui s'apparente à une mise en cache, est réalisée de manière collaborative par les *work-items* et est nécessairement suivie d'une barrière de synchronisation des processus du *work-group*. Dans un second temps, l'intégration du champ de vitesse est réalisée à l'aide d'interpolations.

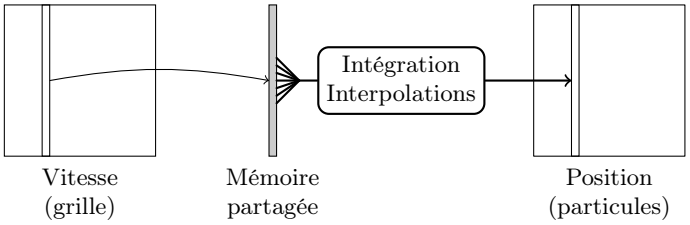


Figure 5.9. – Algorithme pour le noyau d'advection

Dans le cas où la grille est identique pour le champ de vitesse et pour les particules, les 3 ou 7 accès aux données de vitesse respectivement pour les schémas RK2 et RK4 nécessaires au calcul des positions des particules sont effectués en mémoire partagée et bénéficient ainsi de son temps de latence plus faible. La quantité de données lues et écrites en mémoire globale pour une étape d'advection est donc indépendante du schéma et est donnée par :

$$M_{A,s} = 2N^u P, \tag{5.1}$$

où N^u est le nombre de points de grille du scalaire et P la taille d'un nombre en virgule flottante ($P = 8$ en double précision).

On note $N^{\boldsymbol{a}}$ le nombre de points de grille du champ de vitesse. Pour ce qui est du cas multiéchelle, deux interpolations supplémentaires sont nécessaires. Le cache pour le champ de vitesse, de $N_X^{\boldsymbol{a}}$ éléments, est complété par une première interpolation linéaire tensorielle des points de grille dans les directions Y et Z en la coordonnée du sous-problème 1D. Ainsi, les données du champ de vitesse sont lues plusieurs fois, et conduisent à une quantité totale exprimée par :

$$M_{A,m} = \begin{cases} (2N^{\boldsymbol{a}} + N^u)P & \text{pour un problème 2D,} \\ (4N^{\boldsymbol{a}} + N^u)P & \text{pour un problème 3D.} \end{cases} \tag{5.2}$$

La seconde interpolation se fait dans la mémoire partagée pour l'évaluation de la vitesse initiale des particules dont la position est un point de la grille fine.

Le calcul d'un problème 1D étant effectué par un *work-group*, plusieurs particules peuvent être traitées par le même *work-item*. Ainsi les opérations effectuées en dehors de la boucle interne sont mutualisées pour ces particules. On note n_{wi} le nombre de *work-item* par *work-group*. Les nombres d'opérations nécessaires à la réalisation d'une étape d'advection simple échelle sont données par :

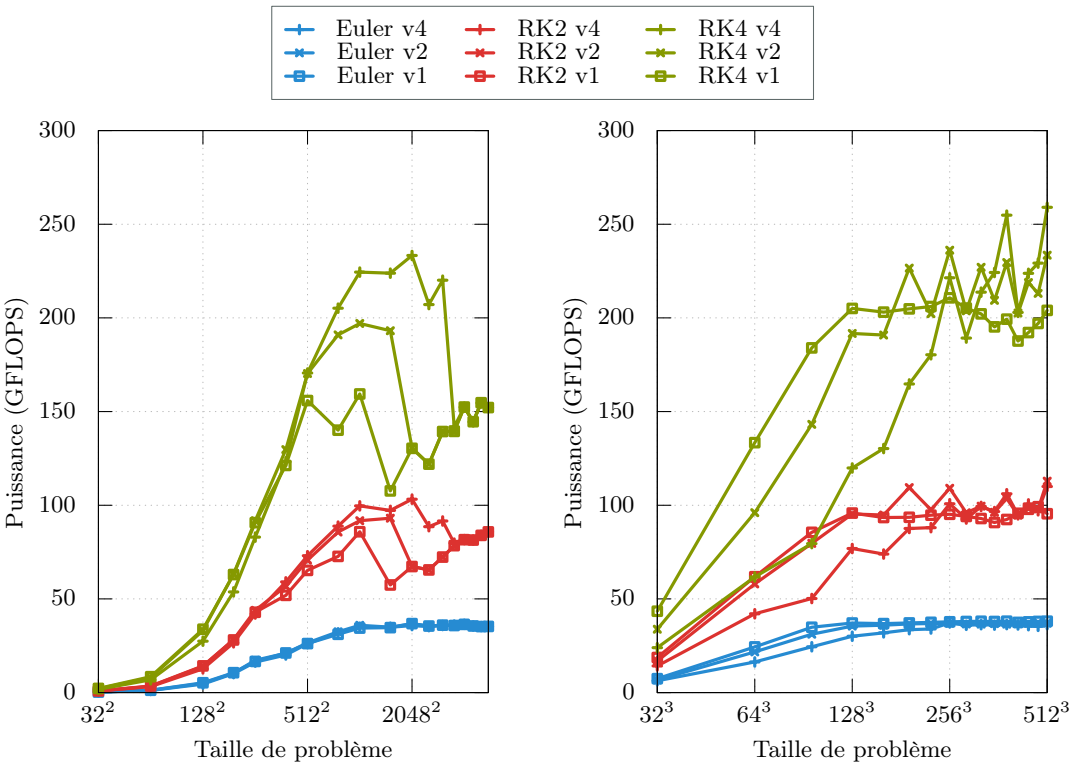
$$F_{A,s} = \begin{cases} 4N^u & \text{pour un schéma d'Euler} \\ 12N^u & \text{pour un schéma de Runge-Kutta d'ordre 2} \\ 32N^u & \text{pour un schéma de Runge-Kutta d'ordre 4} \end{cases} \tag{5.3}$$

Pour les cas multiéchelle, le nombre d'opérations est :

$$F_{A,m} = F_{A,s} + \begin{cases} (4n_{wi} + 3N_X^{\boldsymbol{a}})N^u/N_X^u & \text{en 2D} \\ (8n_{wi} + 11N_X^{\boldsymbol{a}})N^u/N_X^u & \text{en 3D} \end{cases} \tag{5.4}$$

Le choix de l'espace d'index se fait en fixant le nombre de particules traitées par un *work-item*. Les figures 5.10 représentent les performances pour 1, 2 ou 4 particules respectivement notées v1, v2 et v4 dans la légende. Dans un souci de clarté, nous ne présentons pas ici les autres choix qui ont été envisagés d'autant qu'ils ont conduit à des performances plus faibles. L'augmentation de la complexité arithmétique due à l'augmentation de l'ordre des schémas se traduit directement par une hausse de la puissance obtenue. Cela sera analysé par la suite à l'aide du modèle roofline. Une différence structurelle entre les problèmes 2D et 3D est que la résolution est constituée de sous-problèmes 1D bien plus petits en 3D. Ainsi, l'usage de la mémoire partagée est plus faible en 3D qu'en 2D et implique donc une plus grande occupation de la carte pour les grands problèmes. Un traitement de plusieurs particules par *work-item* permet de diminuer la taille des *work-groups* ce qui, là encore permet une augmentation de l'occupation. Cependant la limite des 1 024 *work-items* est rapidement atteinte en 2D. Ce qui explique la chute de performances à partir de $1\,024^2$. L'effet inverse se produit en 3D où trop peu de *work-items* sont assignés ce qui conduit à une faible occupation en raison de la limite du nombre de *work-group* exécutables simultanément par multiprocesseur.

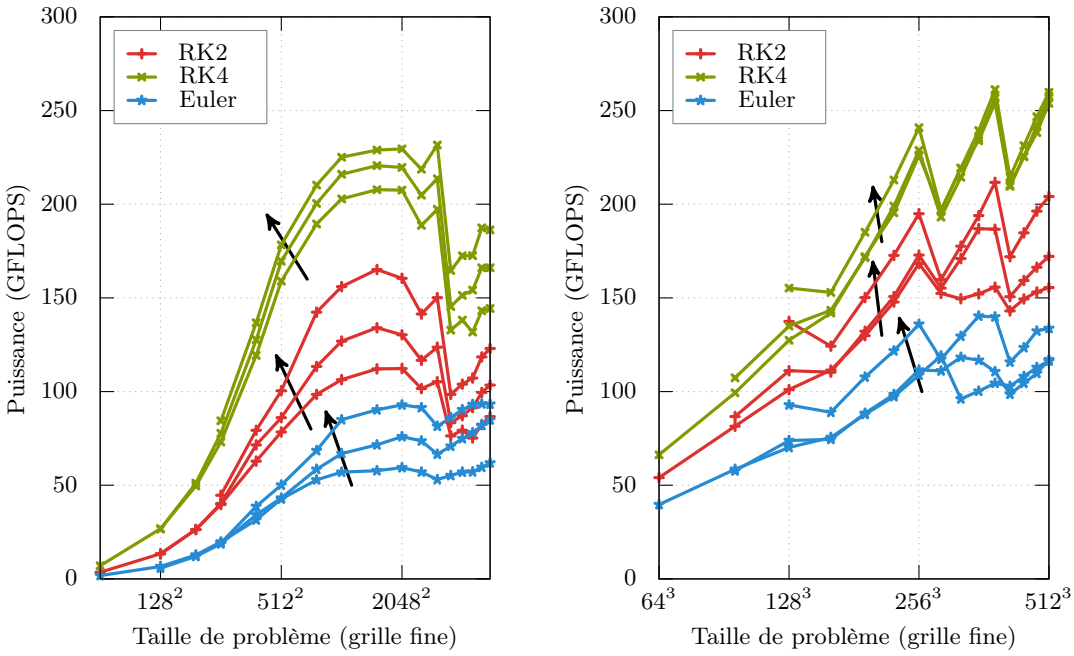
Pour ce qui est de l'advection multiéchelle, nous donnons les performances atteintes pour différentes tailles de problèmes sur les figures 5.11. Seules les performances de la version



(a) 2D **(b)** 3D

Figure 5.10. – Performances des noyaux d'advection

pour laquelle chaque *work-item* traite 4 particules sont présentées. Les rapports d'échelle entre les grilles des particules et celle du champ de vitesse sont fixés à 2, 4 et 8 et conduisent aux courbes dans l'ordre indiqué par les flèches. Ces résultats montrent que les performances suivent l'évolution du rapport d'échelle, à taille de grille fine fixée. Cette évolution est, en partie, due au fait que l'augmentation du rapport d'échelle implique une diminution à la fois du nombre de données nécessaires pour le calcul, d'après l'équation (5.2), et du nombre d'opérations arithmétique à réaliser, équation (5.4).



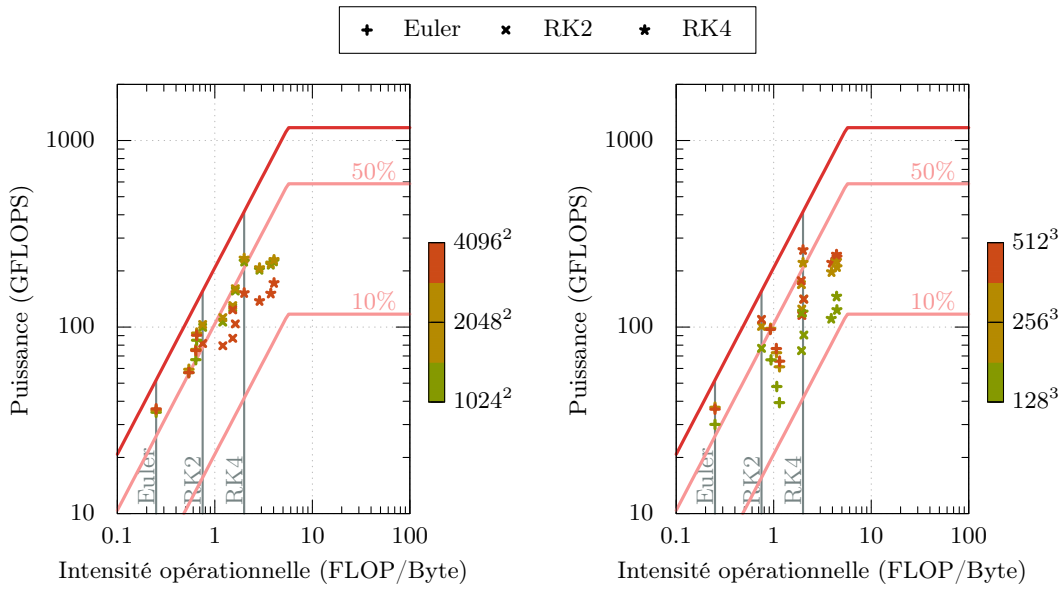
(a) 2D **(b)** 3D

Figure 5.11. – Performances des noyaux d'advection multiéchelle

Le modèle roofline nous permet de réaliser une analyse de performances relativement à la puissance maximale atteignable. Nous représentons sur les figures 5.12 les résultats pour quelques tailles de problèmes. En simple échelle, l'intensité opérationnelle $F_{A,s}/M_{A,s}$ est constante lorsque la taille du problème change. Ces intensités sont représentées par un trait vertical sur les figures 5.12. Dans le cas multiéchelle, les intensités ne sont plus constantes et donc les points qui leur correspondent ne sont pas sur les segments verticaux. Afin de remédier aux effets visuels des échelles logarithmiques du modèle roofline, nous représentons également les niveaux 0,5 et 0,1 de la puissance maximale atteignable.

Remaillage

Cette étape permet de réaliser l'interpolation particule-maillage en employant une stratégie centrée sur les particules. À partir de la coordonnée d'une particule, les points de la grille sur lesquels se remaille la particule sont calculés en fonction de la largeur du support de la formule utilisée. Les positions sont dépendantes du champ de vitesse donc le schéma de remaillage en termes d'accès aux données de la grille ne peut pas être connu à priori.



(a) 2D **(b)** 3D

Figure 5.12. – Modèle roofline pour les noyaux d'advection

Par conséquent, deux particules consécutives peuvent avoir exactement les mêmes points de remaillage. Cependant, du fait de la régularité des champs de vitesse traités, nous pouvons poser l'hypothèse que le nombre maximal de particules par cellule reste petit. On supposera que ce nombre ne dépasse pas deux.

La distribution de l'espace d'index est similaire à celle de l'advection et chaque sous-problème 1D est traité par un *work-group*. La charge par *work-item* est modifiée pour remédier aux éventuels accès concurrents lorsque plusieurs particules sont situées dans la même cellule. Ainsi, chaque *work-item* traite au moins deux particules consécutives. En conservant les mêmes notations que dans le chapitre 2, le support des formules de remaillage est de largeur $2S$. Par conséquent, le remaillage d'une particule nécessite $2S$ accès en lecture et autant en écriture pour les points de grille contenus dans son support. Ces nombreux accès sont effectués dans un tableau temporaire correspondant au sous-problème 1D en mémoire partagée, comme illustré sur la figure 5.13. Une fois toutes les particules traitées et après une barrière de synchronisation, les données sont copiées en mémoire globale.

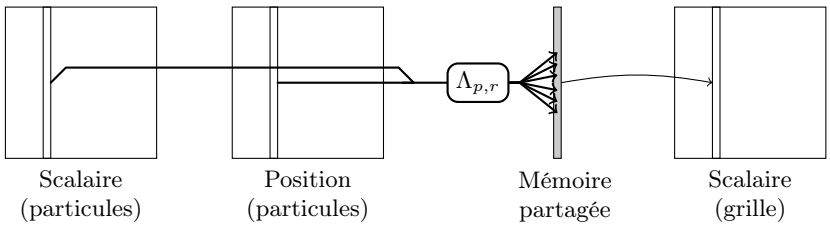


Figure 5.13. – Algorithme pour le noyau de remaillage

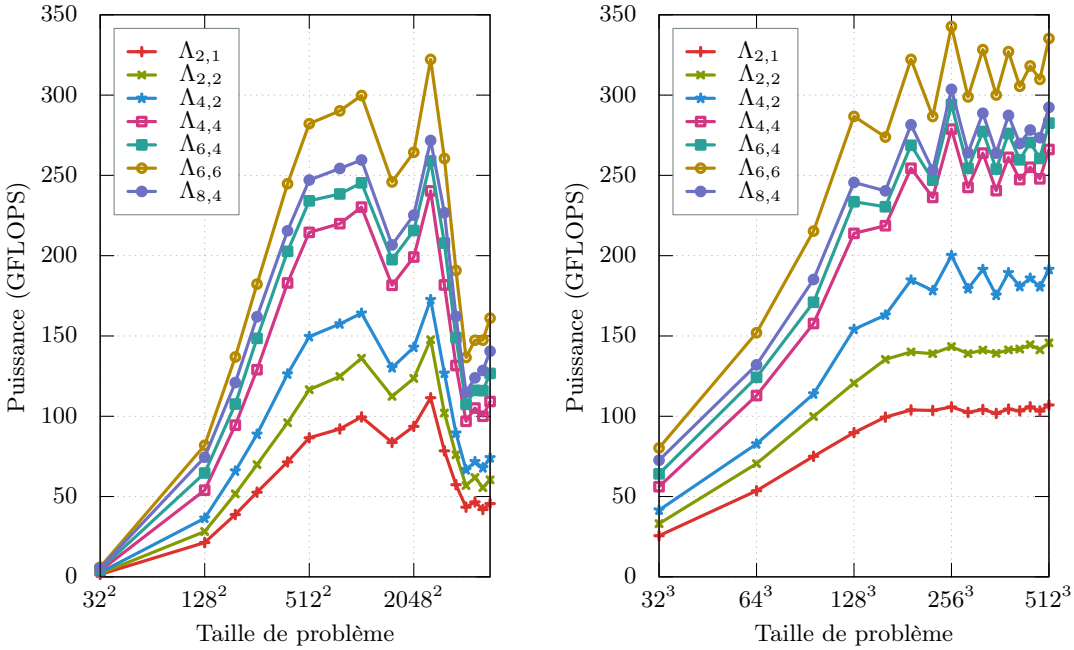
Le nombre d'accès en mémoire globale est donné par l'expression suivante, en fonction du nombre de composantes remaillées c :

$$M_R = (2c+1)N^u P. \tag{5.5}$$

Pour ce qui est du nombre d'opérations en virgule flottante, il dépend de la largeur du support $2S$, de la formule de remaillage et du degré q des polynômes dont l'évaluation est effectuée par la méthode de Horner en utilisant les opérations FMA de la carte.

$$F_R = (2S(2c+2q+1)+5)\,N^u \tag{5.6}$$

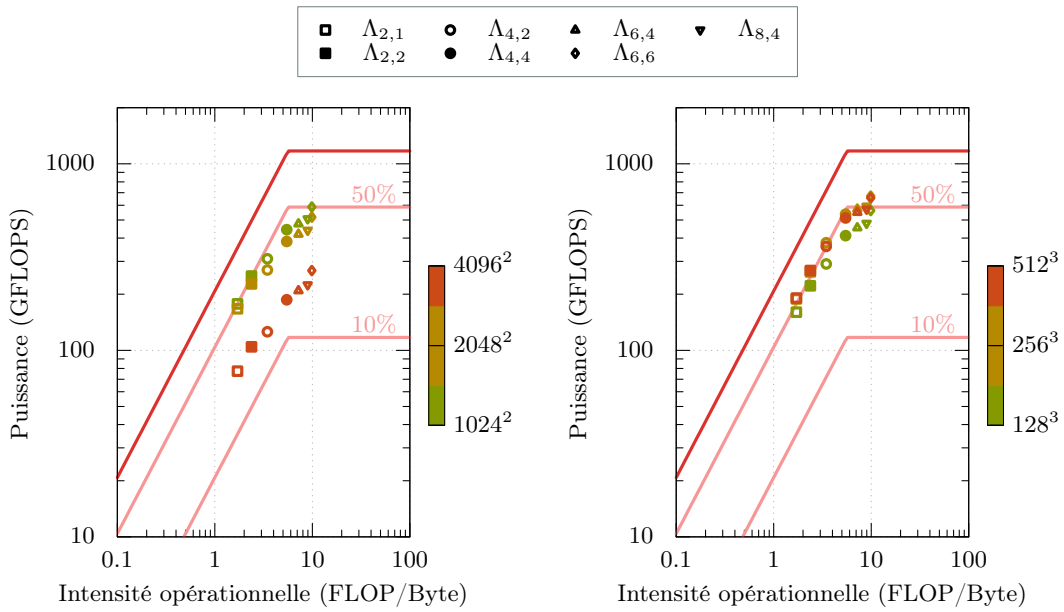
Nous donnons les performances des noyaux pour quelques formules de remaillage. Les meilleurs résultats ont été obtenus lorsque chaque *work-item* traite exactement deux particules. Une augmentation du nombre de particules par *work-item* implique une occupation plus faible car la quantité de mémoire employée ne change pas et la taille des *work-groups* diminue. Un nombre plus grand implique une utilisation d'un grand nombre de registres et donc une diminution de l'occupation des multiprocesseurs. Comme pour les noyaux d'advection on assiste à une chute des performances en 2D à partir d'une taille de $4\,096^2$ à cause d'une trop grande consommation de mémoire partagée.



(a) 2D **(b)** 3D

Figure 5.14. – Performances des noyaux de remaillage

Les performances relatives à la puissance maximale atteignable sont représentées dans un modèle roofline sur les figures 5.15. À l'instar de Büyükkeçeci *et al.* (2012), nous observons de plus hautes performances en 3D qu'en 2D essentiellement à cause d'une utilisation du tableau en mémoire partagée plus efficace du fait de la différence des tailles des sous-problèmes 1D.



(a) 2D **(b)** 3D

Figure 5.15. – Modèle roofline pour les noyaux de remaillage

Noyau complet

Une approche regroupant les opérations d'advection et de remaillage en un seul noyau est également envisagée. Le principal intérêt est de supprimer le tableau en mémoire globale pour la position des particules. En effet, cette variable intermédiaire est simplement locale au processus. Cependant, cette approche nécessite deux tableaux temporaires en mémoire partagée, comme illustré sur la figure 5.16. En pratique, cette implémentation n'est pas

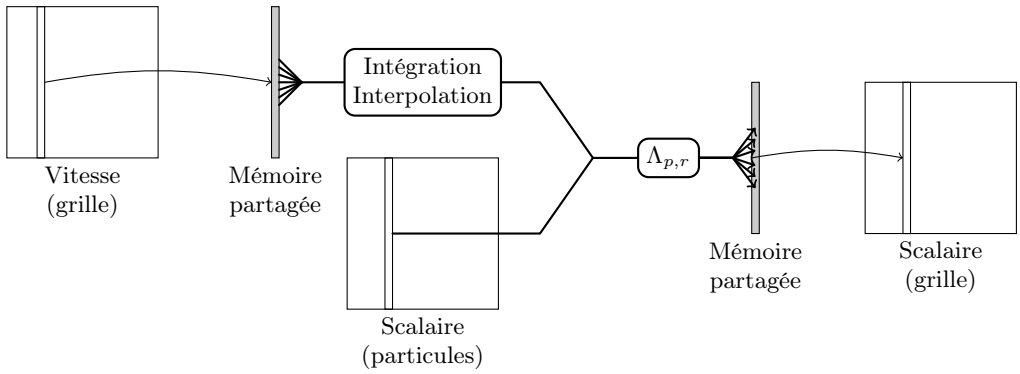


Figure 5.16. – Algorithme pour le noyau d'advection et remaillage

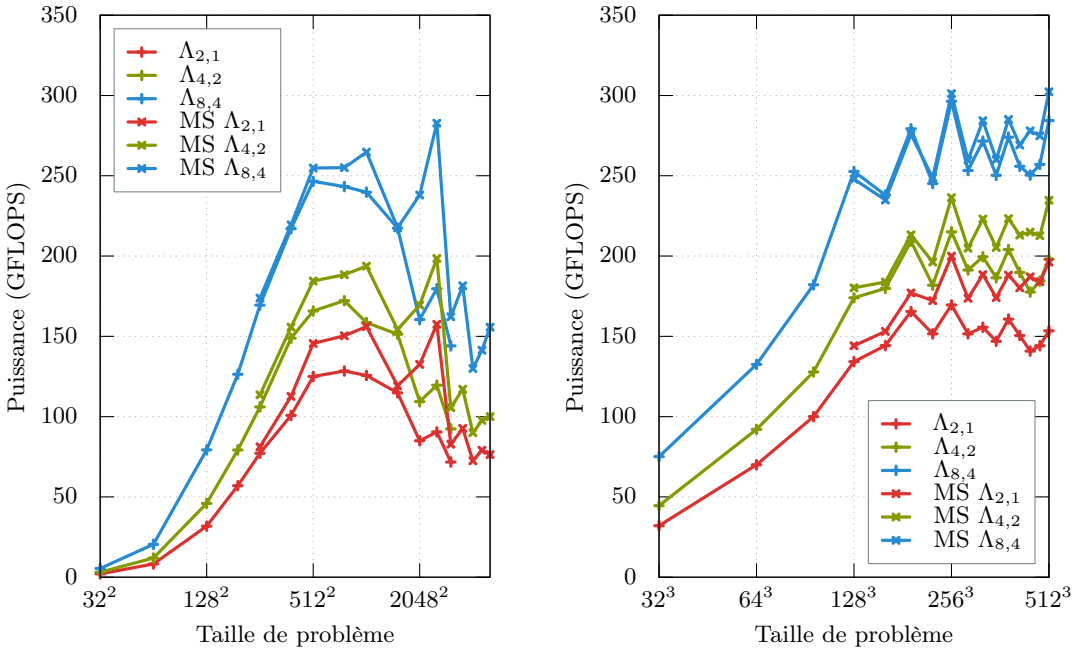
pertinente pour les grandes tailles de problèmes 2D. Cependant, comme la mémoire partagée n'est pas un facteur limitant en 3D, son utilisation semble intéressante. Dans ce noyau, nous réutilisons les fonctions employées précédemment, ce qui conduit à une quantité d'accès

mémoire en simple et multiéchelle :

$$M_{AR,s} = M_{R,s} = (1+2c)N^u P, \tag{5.7}$$

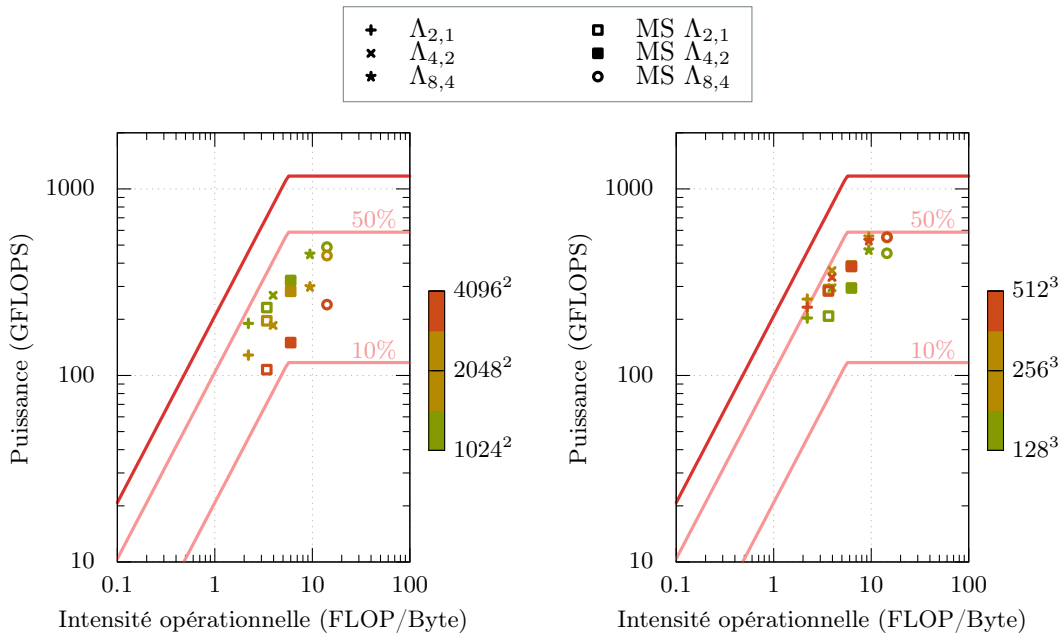
$$M_{AR,m} = \begin{cases} (2N^{\boldsymbol{a}} + 2cN^u)P & \text{en 2D} \\ (4N^{\boldsymbol{a}} + 2cN^u)P & \text{en 3D.} \end{cases} \tag{5.8}$$

Pour ce qui est du nombre d'opérations, il est simplement donné par la somme des opérations des noyaux d'advection et de remaillage. Les performances obtenues sont représentées en figures 5.17 et dans un modèle roofline en figures 5.18. Nous considérons le cas d'une advection par un schéma Runge-Kutta d'ordre 2 ainsi que l'utilisation des formules de remaillage $\Lambda_{2,1}$, $\Lambda_{4,2}$ et $\Lambda_{8,4}$. L'évolution des performances selon la formule de remaillage utilisée est identique à celle observée pour le noyau de remaillage seul. De même, les performances pour une advection avec un schéma d'Euler et Runge-Kutta d'ordre 4 sont respectivement inférieures et supérieures à celles représentées lorsque la formule de remaillage est fixée. Les courbes notées MS dans la légende correspondent à une advection multiéchelle pour un facteur d'échelle égal à 4. Comparativement au cas simple échelle, cela induit une petite augmentation de performances ainsi qu'une augmentation de l'intensité opérationnelle du noyau qui est renforcée par l'effet de la complexité de la formule de remaillage. Cependant, une comparaison des performances relatives des modèles roofline avec les noyaux simples montrent que cette approche n'apporte pas une meilleure efficacité. Toutefois, comme nous le verrons par la suite, l'utilisation de ce noyau conduit à un temps de calcul légèrement inférieur à la somme des deux noyaux simples ce qui permet de bénéficier de la réduction de l'empreinte en mémoire globale.



(a) 2D **(b)** 3D

Figure 5.17. – Performances des noyaux d'advection et remaillage pour un schéma RK2



(a) 2D **(b)** 3D

Figure 5.18. – Modèle roofline pour les noyaux d'advection et remaillage pour un schéma RK2

Les performances individuelles des différents noyaux se retrouvent directement dans les exécutions du code complet. Ainsi, nous exposons la validité de la méthode et de l'implémentation à travers quelques exemples simples de transport de scalaire. Dans un premier exemple, la vitesse est donnée par une fonction analytique dépendant de l'espace et du temps alors qu'elle ne dépend pas du temps dans le second exemple. Le cas multiéchelle est illustré sur l'exemple 3D. Les résultats des exemples sont analysés de manière qualitative permettant une validation de la méthode. Les résultats en terme de temps de calcul sont présentés et comparés entre les différentes fonctions. Dans tous les cas, la vitesse est donnée en entrée sous la forme des valeurs aux points de grille, permettant ainsi de se placer dans un cadre général indépendant de la manière dont est calculé le champ de vitesse.

5.3. Application simple GPU

5.3.1. Transport de scalaire 2D

Un premier exemple d'utilisation de la méthode compète sur GPU est réalisé pour le transport d'une fonction levelset dans un champ de vitesse analytique 2D. Ce champ correspond à un mouvement de rotation incompressible périodique en temps. Ainsi, la fonction levelset est déformée au cours du temps puis retrouve sa valeur initiale après un nombre entier de périodes. Le champ de vitesse est défini par l'expression suivante :

$$\boldsymbol{a}(x,y,t) = \begin{pmatrix} -\sin^2(\pi x)\sin(2\pi y) \\ \sin(2\pi x)\sin^2(\pi y) \end{pmatrix} \cos\left(\frac{\pi t}{12}\right) \tag{5.9}$$

Le scalaire est initialisé par une fonction indicatrice d'un disque de rayon 0,15 centré au point de coordonnée $(0,5; 0,75)$. À chaque itération de la méthode, le champ de vitesse analytique est évalué sur les points de grille puis la méthode décrite précédemment s'applique avec une étape d'advection suivie d'un remaillage des particules. Le contour de niveau 0,5 de la fonction transportée est représenté à différents instants sur la figure 5.19. À l'instant $t = 12$, soit après une période, le contour a retrouvé sa forme initiale. La difficulté de cet exemple réside en la représentation du contour à l'extrémité de la spirale. En effet, selon la qualité de la méthode employée, la séparation du contour ainsi que l'apparition d'éventuelles oscillations numériques ne permettent plus de retrouver la condition initiale. Nous rappelons que le pas de temps est contraint par la condition de CFL Lagrangienne suivante :

$$\Delta t \leqslant \frac{M}{|\nabla \boldsymbol{a}|_\infty}.$$

La simulation suivante a été réalisée en utilisant $1\,024^2$ points de grille. La constante M est prise égale à 0,35, ce qui pour cette résolution, conduit à une CFL équivalente de $|\boldsymbol{a}|_\infty \Delta t/\Delta x = 57$. Ainsi, une première validation de l'implémentation est effectuée car à l'instant $t = 12$, on retrouve bien la condition initiale en utilisant de grands pas de temps, comme le montre la figure 5.19. La simulation est réalisée en utilisant un schéma d'intégration RK2 et une formule de remaillage $\Lambda_{6,4}$.

Les temps de calcul par particule, hors initialisation et hors calcul du champ de vitesse, sont donnés en figure 5.20 pour différentes formules de remaillage et schémas d'advection. Comme prévu par les performances individuelles des noyaux, les petites résolutions conduisent à des temps de calcul assez grands. Nous les considérons comme non significatives car elles ne sont jamais utilisées en pratique dans les applications. Un temps de traitement compris entre 1,5 et 3 ns est obtenu pour un remaillage avec la formule $\Lambda_{2,1}$ et augmente avec l'ordre et la complexité de la formule pour atteindre des valeurs entre 2 et 4 ns pour $\Lambda_{8,4}$. Les autres formules conduisent à des temps intermédiaires. Cette figure propose également une comparaison de la version à un seul noyau de calcul avec celle à deux noyaux séparés. Cette dernière conduit à des temps de calculs plus élevés mais permet de traiter des problèmes de plus grande taille. Cela est dû, à l'utilisation d'une plus grande zone de mémoire partagée pour la version à un noyau. Enfin, la hausse des temps de calcul lorsque les problèmes traités sont de taille supérieure à $2\,048^2$ est conforme avec la chute de puissance observée dans la section précédente. Là encore, l'utilisation de la mémoire partagée est à l'origine de cette évolution notamment à travers une baisse de l'occupation des multiprocesseurs dans les étapes de calcul.

La figure 5.21 donne la répartition du temps de calcul dans les différentes fonctions présentées dans la section précédente. Les couleurs claires correspondent à l'étape de splitting dans la direction X et occupent 2/3 du temps de calcul alors que les teintes foncées représentent la direction Y et occupent le 1/3 restant. Ces simulations ont été réalisées avec un splitting de Strang où deux étapes sont nécessaires dans la direction X. Les portions du temps de calcul attribuées à l'hôte deviennent négligeables lorsque la taille dépasse $4\,096^2$. Elles correspondent à des opérations réalisées par le CPU pour la gestion des étapes de splitting et des lancements de kernels OpenCL. Les opérations d'initialisation à l'aide de copies et de transpositions n'occupent que 10 à 20% du temps de calcul.

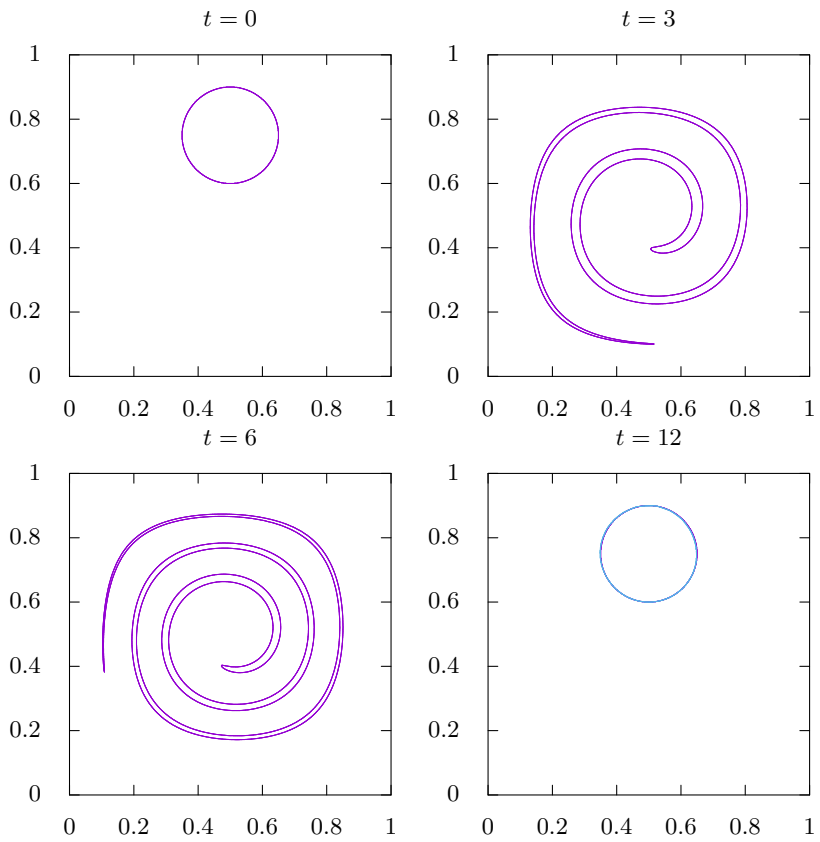


Figure 5.19. – Exemple de transport de fonction levelset 2D

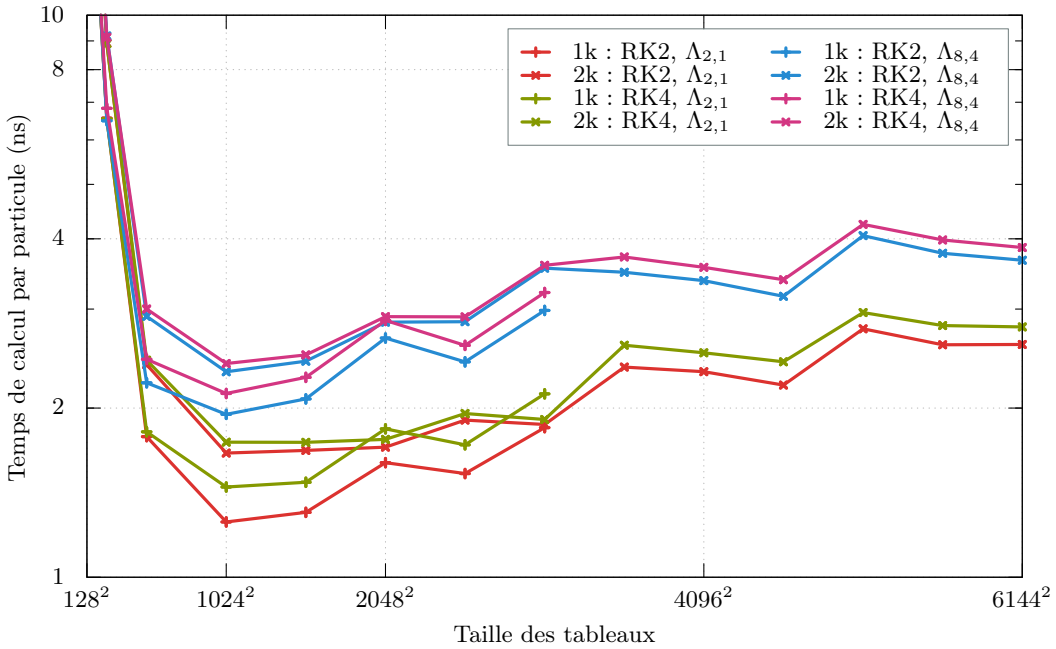


Figure 5.20. – Performances globales de la méthode pour un transport de scalaire 2D

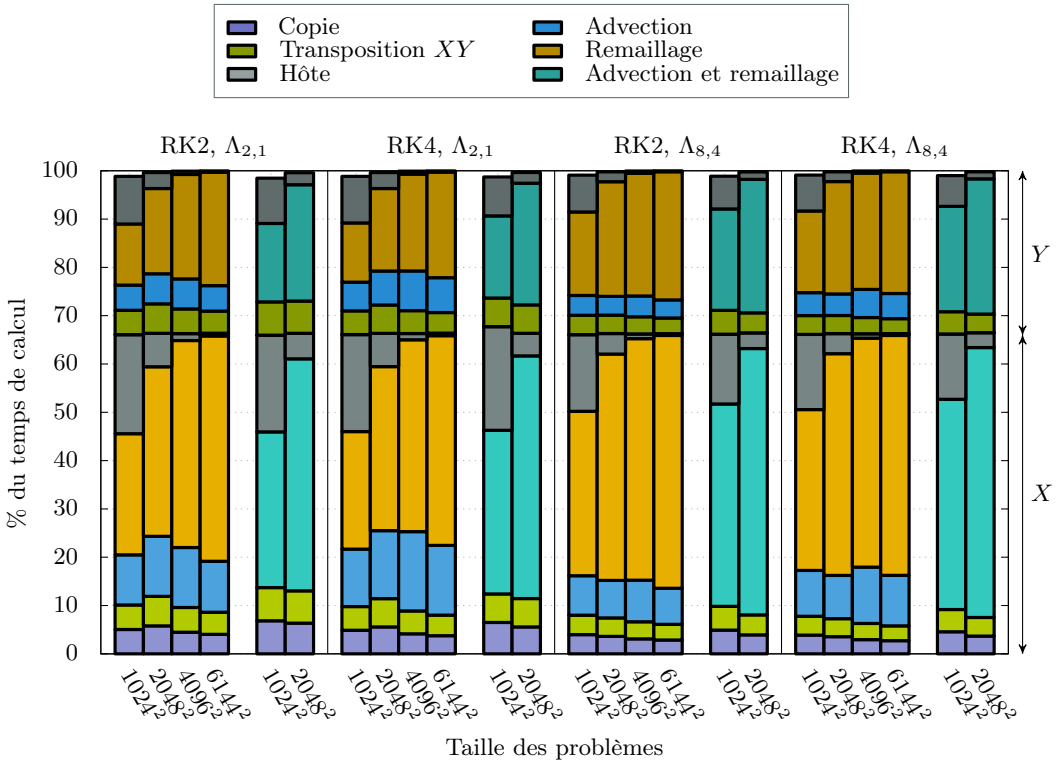


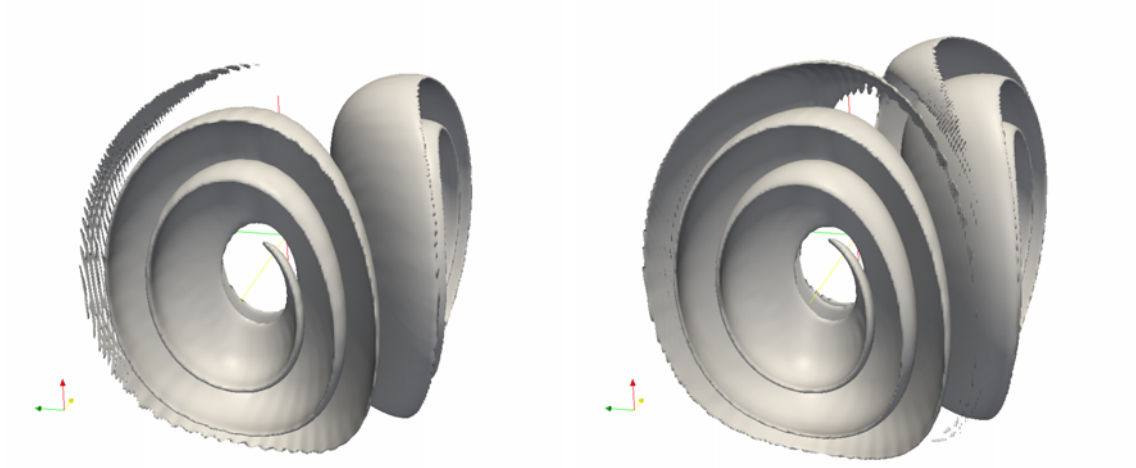
Figure 5.21. – Répartition du temps de calcul des différentes fonctions pour un transport de scalaire 2D

5.3.2. Transport de scalaire 3D

Une application similaire est réalisée en 3D. Le scalaire est initialisé comme la fonction caractéristique d'une sphère de centre $(0,35; 0,35; 0,35)$ et de rayon 0,15. Le champ de vitesse, correspondant à une rotation incompressible, est défini par :

$$\boldsymbol{a}(x,y,z,t) = \begin{pmatrix} 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z) \\ -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z) \\ -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z) \end{pmatrix} \tag{5.10}$$

Nous nous intéressons ici à l'évolution de la surface de niveau 0,5 au cours du temps qui est soumise à une déformation de plus en plus forte. Nous comparons les effets des différentes formules de remaillage à travers l'évolution du volume représenté par cette surface. La figure 5.22 représente la surface de niveau 0,5 à l'instant $t = 4$, pour deux formules de remaillage différentes. L'utilisation d'une formule d'ordre plus élevé, sur la figure 5.22b, retarde l'apparition de « trous » ainsi que leur disparition comparativement à l'utilisation de la formule $\Lambda_{2,1}$, sur la figure 5.22a.



(a) Remaillage par $\Lambda_{2,1}$ **(b)** Remaillage par $\Lambda_{8,4}$

Figure 5.22. – Exemple de transport de fonction levelset en 3D

L'évolution du volume délimité par la surface de niveau 0,5 est donnée en fonction du temps pour quelques noyaux de remaillage sur la figure 5.23. Ainsi, les formules d'ordre élevé préservent mieux le volume ce qui est conforme à l'observation des figures 5.22. Ces simulations ont été réalisées pour une constante LCFL égale à 0,35 et pour une résolution de 320^3. Cela équivaut à une CFL de 17,8.

Les temps de calcul pour cette simulation sont donnés sur les figures 5.24 en fonction de la taille du problème pour différentes formules de remaillage dans les cas simple échelle et multiéchelle. Les temps de calculs sont indépendants de la taille du problème et sont de l'ordre de 1 à 2 ns, excepté pour les petites tailles non significatives. Comme pour le cas 2D, seules les formules de remaillage $\Lambda_{2,1}$ et $\Lambda_{8,4}$ sont représentées, les autres se répartissent sur des temps de calculs intermédiaires. Là encore, la version avec un seul noyau de calcul pour l'advection et le remaillage donne de meilleurs temps de calcul. Contrairement au

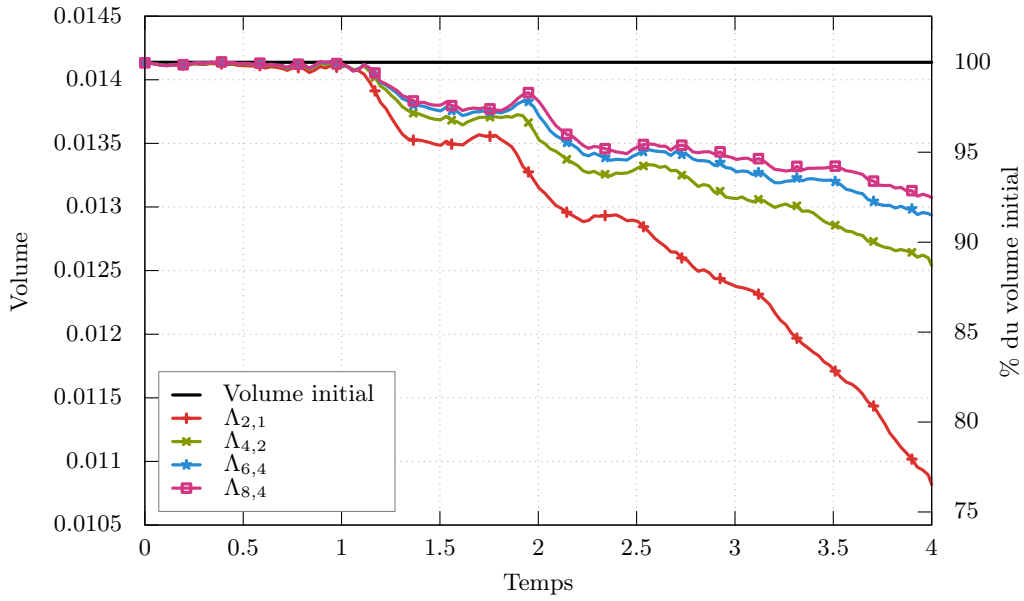
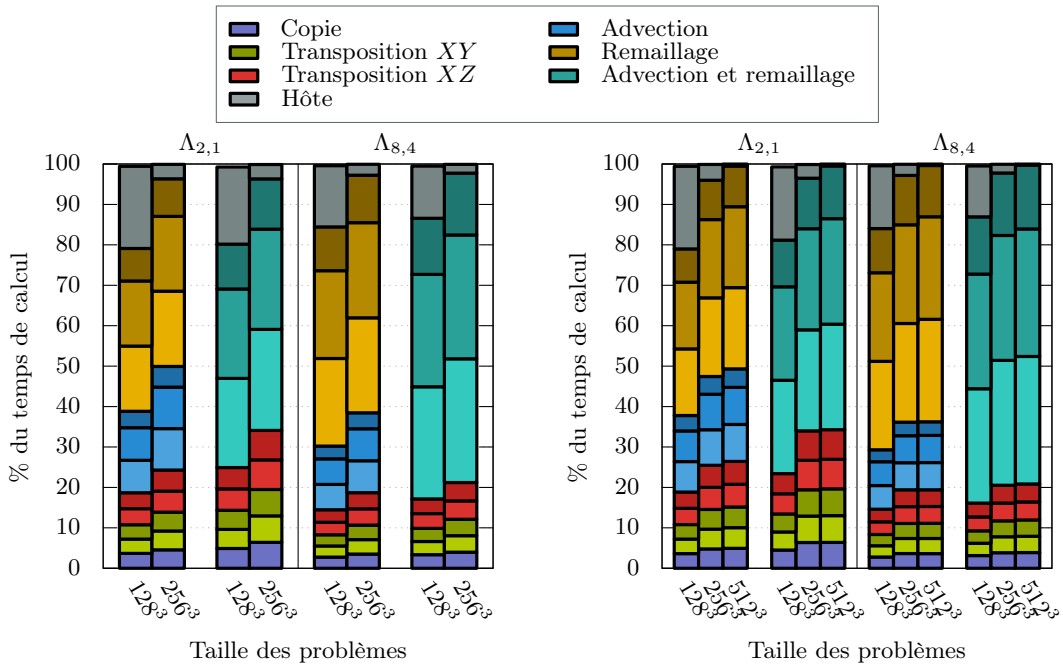


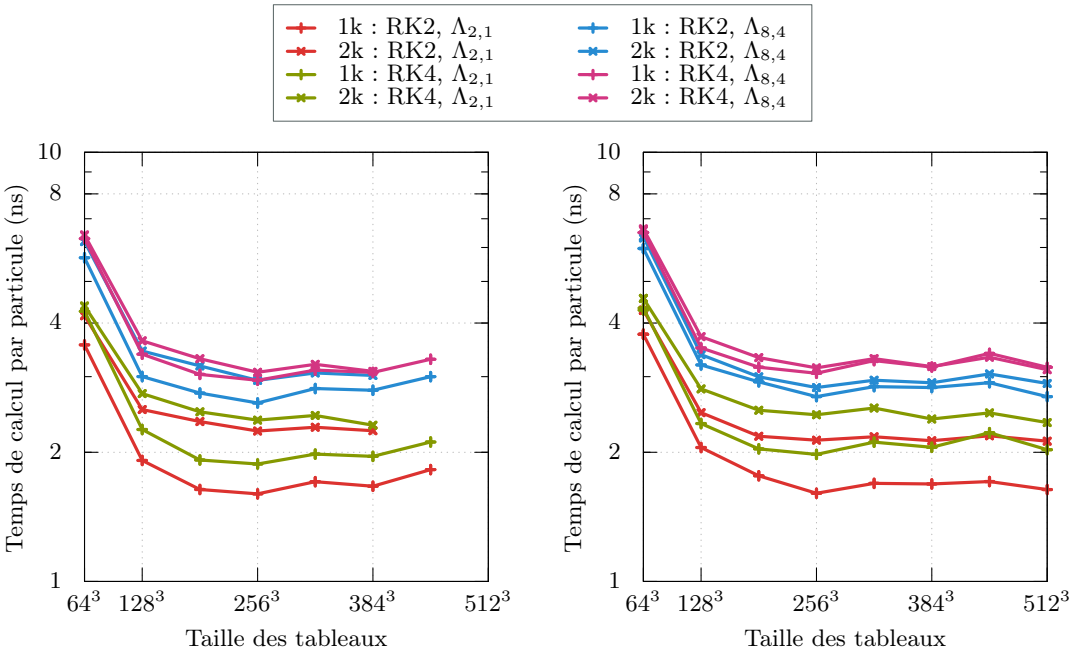
Figure 5.23. – Évolution du volume défini par le niveau 0,5 d'une fonction levelset

cas 2D, cette version permet de traiter des résolutions plus grandes car l'empreinte en mémoire globale est plus faible et la mémoire partagée n'est pas limitante. Ainsi la résolution maximale en simple échelle est de 448^3 mais il est possible de traiter une résolution de 512^3 pour le scalaire lorsque la résolution de la vitesse est inférieure à 384^3.



(a) Simple échelle **(b)** Multiéchelle ($N^u = 4^3 N^a$)

Figure 5.25. – Temps de calcul des différentes fonctions pour un transport de scalaire 3D



(a) Simple échelle

(b) Multiéchelle ($N^u = 4^3 N^{\boldsymbol{a}}$)

Figure 5.24. – Performances globales de la méthode pour un transport de scalaire 3D

Comme pour le cas 2D, nous donnons, sur la figure 5.25, les répartitions du temps de calcul à travers les différentes fonctions de la méthode. Les contributions issues des différentes directions de splitting sont différenciées par les nuances de couleurs. Comme pour le cas 2D, les directions X, Y et Z se partagent le temps de calcul total et en occupent respectivement les 2/5, 2/5 et 1/5. Les deux premières sont traitées deux fois sur des demi pas de temps. Les opérations d'initialisation occupent entre 15 et 30% du temps de calcul.

Conclusion

Dans ce chapitre, nous avons présenté une implémentation spécifique aux architectures multicœurs telle que les cartes graphiques de la méthode semi-Lagrangienne. La stratégie d'implémentation a été élaborée en se basant sur les études similaires existantes. Les différentes fonctions de la méthode ont été étudiées à travers leurs performances respectivement à des optimisations génériques basées sur les caractéristiques techniques des cartes graphiques. Ces optimisation peuvent également conduire à de bonnes performances sur d'autres cartes que celle présentée ici.

Les performances individuelles des noyaux d'initialisation, comme fonctions de manipulation de données, atteignent jusqu'à 73% de la bande passante maximale pour la copie. Les transpositions permettent d'obtenir une bande passante respectivement de 64%, 71% et 63% pour la transposition XY en 2D et 3D et pour la transposition XZ. Ces performances sont assez satisfaisantes en comparaison avec celles obtenues pour la copie. En effet, les bandes passantes atteintes pour les transpositions sont supérieures à 86% de celle de la meilleure

copie. Comme pour les noyaux d'initialisation, les performances des noyaux de calcul sont fortement dépendantes des tailles de tableaux traitées. Pour les tailles les plus adaptées, les puissances de calcul développées par les noyaux d'advection dans le modèle roofline sont supérieures 50% de la puissance maximale atteignable et légèrement inférieures dans les cas multiéchelle. L'augmentation de l'ordre des formules de remaillage implique une augmentation de puissance de calcul des kernels de remaillage pour obtenir des performances proches de 50% de la puissance maximale. Pour ce qui est des noyaux réalisant les opérations d'advection et de remaillage en une seule passe, les performances se situent au voisinage de 20% de la puissance maximale. Toutefois, ces noyaux montrent un temps de calcul global généralement meilleur que lorsque deux noyaux séparés sont employés. La quantité de mémoire partagée requise par ce noyau interdit son usage pour les grandes tailles en 2D. Au contraire, en 3D, ce noyau permet de traiter de plus grandes résolutions du fait de l'utilisation d'une plus faible quantité de mémoire globale. Par conséquent, il est préférable d'utiliser la version en deux noyaux pour les simulations 2D du fait de la limitation en mémoire partagée. De même, le noyau complet est mieux adapté aux problèmes 3D car il permet de diminuer l'empreinte en mémoire globale. Enfin, les deux exemples de transport de fonction levelset proposent une validation qualitative de la méthode.

Bien que ces résultats soient obtenus sur une machine de calcul, des expériences conduisant à l'obtention de performances similaires sont menées sur des machines de bureau ou portables, montrant ainsi la portabilité de notre implémentation. Toutefois, nous ne présentons pas ces résultats dans ce manuscrit par souci de simplification car ces machines sont dotées de cartes d'anciennes générations n'offrant pas les même facilités de développement. De plus les cartes sont utilisées par les systèmes d'exploitation pour l'affichage à l'écran. Sur ces configurations, le calcul est en concurrence avec l'affichage pour l'utilisation des ressources, ce qui donne lieu à des variations de performances non reproductibles et pouvant être importantes.

6. Implémentation sur architectures hétérogènes

Nous avons vu dans le chapitre 5 que la méthode semi-Lagrangienne, telle que présentée au chapitre 2 avec splitting dimensionnel et formules de remaillage d'ordre élevé, est bien adaptée pour tirer profit de l'architecture des cartes graphiques. En effet, l'analyse des performances dans un modèle roofline montre des puissances de calcul généralement supérieures à 50% de la puissance maximale atteignable. Les temps de calculs ainsi obtenus sont assez faibles et incitent à passer à l'échelle avec une exécution parallèle sur plusieurs cartes graphiques.

L'objectif de ce chapitre est de réaliser des simulations de transport de scalaire passif dans un écoulement turbulent tel que décrit dans le chapitre 1. La stratégie de résolution, exposée dans le chapitre 3, vise à exploiter les différentes ressources des machines de calcul hétérogène en réalisant le calcul du fluide sur des architectures multi-CPU et le transport du scalaire sur plusieurs GPU. La conception de la librairie, détaillée au chapitre 4, facilite l'intégration des méthodes de résolution des autres éléments des équations du problème complet de transport de scalaire passif dans un écoulement turbulent tels que les termes de d'étirement, de diffusion et l'équation de poisson.

Dans un premier temps, nous détaillerons l'architecture hétérogène considérée ainsi que les différentes stratégies de parallélisme usuelles. Notre approche se base sur l'exploitation de toutes les ressources de calcul disponibles en maximisant leur occupation. Cela se traduit par l'emploi combiné de parallélismes à mémoire distribuée et partagée. Dans un second temps, nous détaillerons l'adaptation de la méthode à la résolution d'un problème de transport de scalaire sur une architecture multi-GPU. L'accent sera mis sur l'analyse des performances des communications entre les cartes à travers l'illustration sur l'exemple de transport de fonction levelset 3D du chapitre 5. Enfin, nous présenterons la méthode de résolution pour l'application au transport turbulent de scalaire passif à travers un exemple d'application à la simulation d'un jet plan turbulent.

6.1. Lien entre la méthode et l'architecture hybride

6.1.1. Description d'une machine hybride

Comme nous l'avons évoqué dans le chapitre 1, les machines de calcul parallèles actuelles sont caractérisées par des architectures hétérogènes regroupant de nombreux processeurs multicœurs et accélérateurs. Généralement, elles présentent une organisation hiérarchique de leurs composants. Une machine de calcul est constituée d'un ensemble de nœuds connectés par un réseau local à très haut débit permettant des vitesses de transfert de plusieurs dizaines de GByte/s. Un nœud de calcul possède les mêmes caractéristiques qu'un ordinateur personnel utilisé comme serveur. Ainsi, il est constitué, au minimum, d'un processeur, de mémoire vive et d'un espace disque. Ces ressources sont gérées par un système d'exploitation qui lui est propre. Les composants présents sur un nœud varient d'une machine à l'autre et éventuellement d'un type de nœud à l'autre sur une même machine. De manière générale, ils contiennent plusieurs processeurs multicœurs et éventuellement des accélérateurs tels que les cartes graphiques ou des coprocesseurs. Un exemple d'une telle architecture est donnée en figure 6.1 où les nœuds sont connectés à travers le même réseau et certains contiennent des accélérateurs.

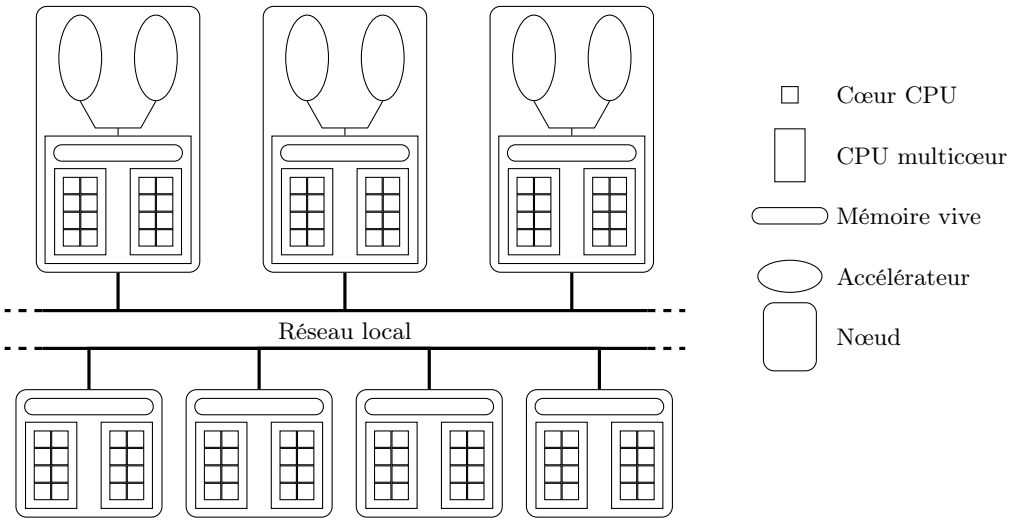


Figure 6.1. – Exemple d'architecture hybride

Dans la suite de ce chapitre, nous utilisons la machine de calcul Froggy du mésocentre CIMENT [1] qui possède notamment sept nœuds contenant chacun deux processeurs octo-cœurs `Intel Sandy Bridge EP E5-2670` partageant 32 GByte de mémoire vive et deux cartes graphiques Nvidia K20m.

1. `http://ciment.ujf-grenoble.fr`

6.1.2. Différents niveaux de parallélisme

Parmi les paradigmes de programmation parallèle usuels, le parallélisme de données et le parallélisme par tâches se distinguent par l'organisation des calculs par rapport aux données à traiter (Dongarra *et al.*, 2003). Dans le premier, on réalise l'exécution simultanée de tâches identiques sur des données différentes. En revanche, le second consiste en l'exécution simultanée de tâches différentes sur des données identiques ou non. Ces deux approches sont couramment employées pour l'exploitation de machines de calcul parallèles hétérogènes, à travers l'utilisation des librairies telles que MPI et OpenMP. Différentes techniques sont à considérer pour la mise en œuvre de ces paradigmes selon la nature de la mémoire, partagée ou distribuée. La littérature étant abondante sur le sujet, dans cette section nous ne donnerons que quelques éléments concernant l'implémentation de ces paradigmes.

Au sein d'un nœud, la mémoire est partagée par l'ensemble des cœurs des processeurs. Dans ce contexte, les taches sont distribuées sur l'ensemble des cœurs physiques à l'aide d'une librairie telle que OpenMP. Ce paradigme permet l'exploitation, au maximum, de l'ensemble des cœurs CPU d'un seul nœud car les données manipulées doivent être référencées dans le même espace d'adressage

En complément de cette stratégie se trouve le parallélisme par envoi de message tel que proposé par le standard MPI. Il permet de réaliser des communications d'un processus à un autre dans un contexte de mémoire distribuée. Dans le cas général, il n'est pas possible à un processus de lire ou d'écrire dans une zone mémoire attribuée à un autre processus. Le principe de ce paradigme est que les données à transmettre sont empaquetées dans un message qui est envoyé au processus destinataire à travers le réseau de la machine. Une gestion fine de l'envoi et de la réception de ces messages est nécessaire afin que les données soient correctement traitées, éventuellement par l'utilisation de zones mémoires temporaires. Une très grande précision des communications est possible à travers l'utilisation des nombreuses variantes proposées par la librairie telles que, entre autres, les communications point à point ou globales ainsi que leurs versions bloquantes ou non. Ce paradigme s'utilise dans un environnement à mémoire distribuée mais fonctionne également avec une mémoire partagée.

Une implémentation hybride associe plusieurs stratégies de parallélisme pour exploiter au mieux une architecture donnée. Ainsi, sans tenir compte des accélérateurs dans un premier temps, les utilisations courantes suivent deux stratégies distinctes pour l'exploitation de l'ensemble des cœurs CPU d'une machine. L'intégration d'une librairie d'envoi de message permet une utilisation d'un ensemble de processeurs multicœurs de plusieurs nœuds d'une machine. Ce fonctionnement se base sur fait que les librairies d'envoi de message, comme MPI, sont utilisables dans un contexte mixte de mémoire partagée et distante. Cependant, l'intégration d'un tel parallélisme implique généralement d'importants changements dans le code. Une réécriture de certaines parties est nécessaire lorsque l'utilisation de ce paradigme n'a pas été prévue à la conception. La seconde stratégie consiste combiner des paradigmes spécifiques à un environnement à mémoire partagée et à mémoire distribuée. Une approche basée sur les librairies MPI et OpenMP est couramment utilisée. Ces deux stratégies que l'on notera respectivement MPI et MPI+OpenMP permettent d'utiliser l'ensemble des cœurs CPU d'une machine de calcul. La principale différence est que la mémoire partagée par les processus d'un même nœud est gérée par un parallélisme spécifique dans le cas MPI+OpenMP, ce qui permet une plus grande finesse d'optimisation et donc

potentiellement l'obtention de meilleures performances. En revanche, le développement et le débogage sont généralement plus complexes.

L'ensemble des processus déployés sur les cœurs physiques des nœuds peuvent ne pas réaliser des opérations similaires sur des données différentes. Un paradigme de parallélisme par tâches permet d'affecter des ensembles de processus à différentes tâches. Dans ce cas, au sein d'un même groupe, une tâche est traitée de manière collaborative. En revanche, deux processus associés à deux tâches distinctes sont amenés à exécuter des instructions totalement différentes. Toutefois, des communications sont généralement nécessaires pour échanger des données entre les processus des différentes tâches lorsqu'elles ne sont pas totalement indépendantes.

L'exploitation des cartes graphiques comme accélérateurs se réalise de manière identique au cas d'une utilisation exclusive de l'accélérateur, comme présentée au chapitre 5. Un processus peut exploiter un accélérateur à l'aide des différentes techniques telles que CUDA ou OpenCL. Les cartes graphiques récentes sont capables de gérer l'exécution d'un ensemble de tâches provenant de plusieurs processus. Par conséquent, tous les processus placés sur un même nœud peuvent exploiter une même carte graphique. De même, dans le cas où plusieurs cartes sont présentes, un même processus peut utiliser simultanément l'ensemble des cartes graphiques.

Ainsi, les machines hybrides doivent être exploitées en utilisant à la fois un parallélisme à mémoire distribuée, à mémoire partagée ainsi que des accélérateurs. On complète les notations précédentes par MPI+OpenCL ou MPI+OpenMP+OpenCL si OpenCL est utilisé sur les accélérateurs.

6.1.3. Stratégie d'utilisation

La stratégie que nous employons dans ce travail se limite à une exploitation des processeurs multicœurs à l'aide du seul niveau de parallélisme par envoi de message. Cette approche, complétée par l'utilisation de la librairie OpenCL, nous permet de traiter à la fois les cas multi-GPU et hétérogènes. En effet, pour une exploitation de plusieurs cartes graphiques, il est nécessaire de placer un processus par nœud pour exploiter l'ensemble des cartes du nœud. Ces différents processus, se situent nécessairement dans un environnement à mémoire distribuée, d'où le choix de l'utilisation de MPI pour la gestion du parallélisme des processus hôtes.

En pratique, dans notre implémentation, chaque GPU n'est géré que par un et un seul processus hôte. En effet, des études préliminaires ont montré une dégradation des performances lorsque plusieurs processus exploitent la même carte graphique de manière concurrente comparativement au cas où une carte est associée à un processus. Ce phénomène est en partie dû au fait que notre implémentation est plus efficace pour traiter de grandes tâches comparativement au traitement de plusieurs tâches plus petites. D'autre part, il nous semble plus avantageux de ne pas exploiter plusieurs cartes à partir du même processus hôte car le découpage des tâches est déjà réalisé au niveau des différents processus hôtes. Il apparaît donc inutile d'ajouter un second niveau de découpage interne au processus. Ainsi, l'exploitation de n cartes graphiques est obtenue par le placement de n processus MPI répartis sur l'ensemble des nœuds contenant les cartes.

L'inconvénient majeur de cette stratégie est qu'un nœud de calcul dispose généralement d'un plus grand nombre de cœurs CPU que de cartes graphiques. Dans l'exemple de la machine Froggy, le rapport est de 8. Lors d'une utilisation multi-GPU, de nombreux cœurs CPU sont inutilisés. L'approche que nous proposons dans la section 6.3 à travers la mise en œuvre de la méthode hybride nous permettra d'explorer une solution à ce problème d'occupation.

La suite de ce chapitre est consacrée à la mise en œuvre de cette stratégie dans un cadre multi-GPU puis hybride multi-CPU et multi-GPU.

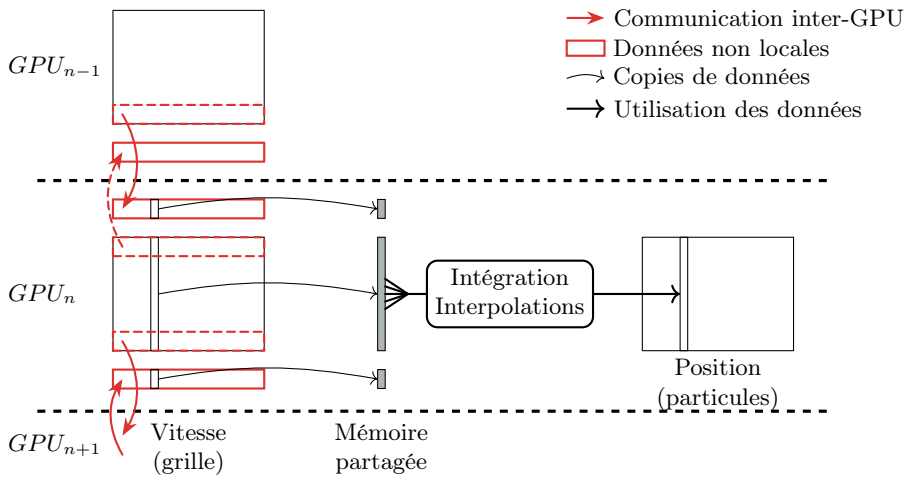
6.2. Application multi-GPU

6.2.1. Mécanisme de communication

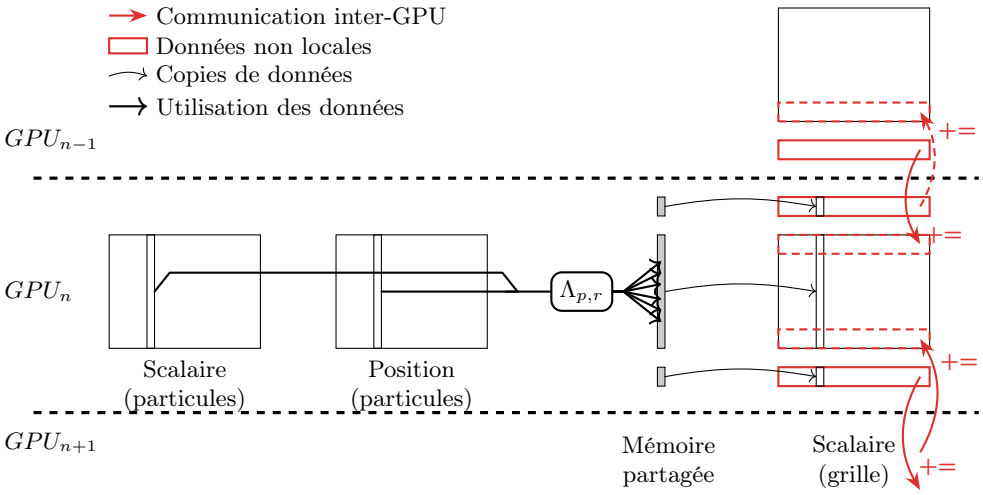
La résolution de l'équation de transport sur plusieurs cartes graphiques nécessite des communications entre ces GPU. À chaque carte est associée une partie du domaine de calcul. Ainsi, le domaine de calcul est séparé le long de plans orthogonaux à une direction de découpe. Nous appelons direction de découpe une direction dont le traitement nécessite des communications. Le principal avantage de ce découpage est de conserver une méthode de résolution identique au cas simple GPU lors du traitement des directions de l'espace autres que celles de découpe. Dans ces dernières, les données correspondant aux sous-problèmes 1D sont distribuées sur plusieurs cartes ce qui implique des communications entre ces cartes. Nous distinguons alors les données locales calculées par une carte et les données non locales qui sont calculées par une autre carte et qui doivent être transférées.

Deux communications sont nécessaires : une pour l'étape d'advection et l'autre pour le remaillage. Pour l'advection, les interpolations du champ de vitesse nécessaires au calcul des positions des particules sont faites sur des données non locales pour les particules ayant quitté le domaine. Ces données devront être transférées au préalable depuis les cartes voisines. La quantité de ces données dépend directement du champ de vitesse lui-même. Toutefois, une majoration de leur taille est possible à partir du nombre CFL maximal. En effet, ce dernier correspond au nombre maximal de cellules traversées par les particules en une itération. Une particule au bord du domaine n'aura pas besoin de plus de données distantes qu'elle ne peut traverser de cellules. Cet algorithme est schématisé sur la figure 6.2a. De la même manière, lors du remaillage, la taille des zones de communications dépend du nombre CFL maximal ainsi que de la largeur du support de la formule de remaillage employée. Lors du calcul, le domaine local est bordé par des zones mémoire qui représentent les points de grille voisins distants et récupèrent les contributions des particules sur ces points. Ces derniers sont parfois appelés points fantômes ou *ghosts*. Les données de ces zones sont ensuite envoyées aux processus voisins puis ajoutées aux données locales. L'adaptation de cet algorithme est schématisé sur la figure 6.2b. Les communications se font exclusivement avec les processus voisins dans la direction de calcul. Dans les deux algorithmes, une utilisation de la mémoire partagée comme un cache similaire au cas simple GPU est employée pour les données du champ de vitesse non local ainsi que pour les points de grille extérieurs. Les échanges de données à l'issue du remaillage se caractérisent par un ajout des

valeurs distantes du scalaire aux valeurs locales, alors que pour le champ de vitesse, il s'agit simplement d'une copie des données qui sont utilisées en lecture uniquement.



(a) Advection



(b) Remaillage

Figure 6.2. – Adaptation des algorithmes d'advection et remaillage au cas multi-GPU

Ces communications inter-GPU consistent en des échanges de données entre les mémoires globales des cartes graphiques. Généralement, les implémentations MPI ne sont pas capables de lire et écrire des données directement dans la mémoire des cartes sans faire intervenir l'hôte. Toutefois, certaines implémentations spécifiques telles que MPI-ACC (Aji *et al.*, 2012) ou MVAPICH (Wang *et al.*, 2011) permettent de réaliser des envois de messages directement de la mémoire d'une carte à l'autre. Dans cette dernière, les transferts utilisent les technologies telles que proposées par les versions récentes de CUDA sur les cartes Nvidia *(GPUDirect)*. La librairie MVAPICH est disponible au téléchargement [2] mais ne permet qu'une exploitation des seules cartes NVIDIA et, à notre connaissance, la librairie générique MPI-ACC n'a pas été encore distribuée. Ces travaux font état de gains en termes de per-

2. http ://mvapich.cse.ohio-state.edu/downloads/

formances des communications pouvant atteindre 30% par rapport à des transferts naïfs. En l'absence de telles librairies, la méthode naïve consiste en une succession de transferts depuis la mémoire globale de la carte vers le CPU, entre les CPU hôtes et enfin du CPU vers la mémoire de la carte de destination. Ce mécanisme est illustré sur la figure 6.3 par l'enchaînement des flèches vertes et bleues.

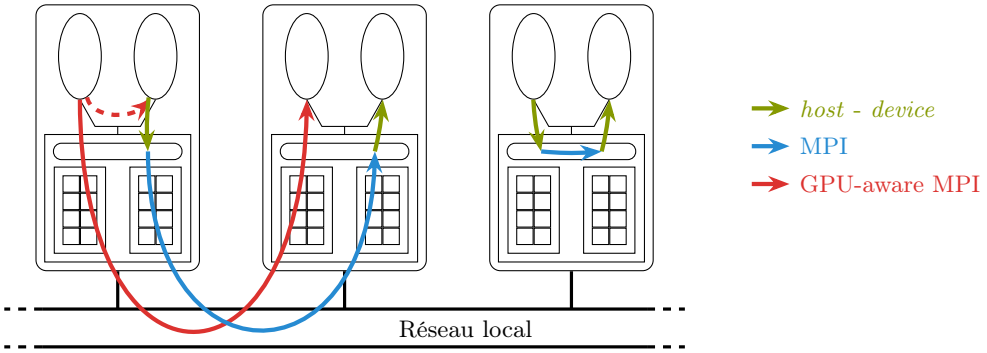


Figure 6.3. – Schémas de communications inter-GPU

Les résultats de cette thèse sont obtenus à l'aide de transferts naïfs combinant explicitement un usage des fonctions OpenCL pour les transferts entre les cartes et les CPU hôtes ainsi que des envois de message MPI entre CPU. Nous précisons que l'utilisation technologies spécifiques aux cartes Nvidia n'a pas encore été intégré au standard OpenCL. De manière générale, ces transferts peuvent être optimisés par un découpage des données en blocs dont les envois sont traités en chaîne (Aji *et al.*, 2013), comme illustré par la figure 6.4. Ainsi, le message original est traité par un ensemble d'envois de plus petite taille permettant un recouvrement des opérations. La taille des blocs est un des paramètres qui doit être ajusté en fonction du matériel utilisé (PCIe et réseau).

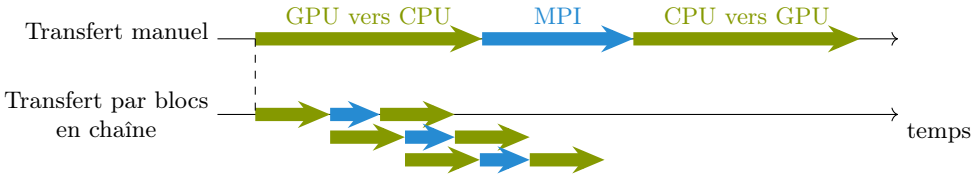


Figure 6.4. – Optimisation des communications inter-GPU

6.2.2. Performances

Les performances de cette implémentation dans un contexte multi-GPU sont étudiées à travers l'exemple de transport de fonction levelset 3D présenté en section 5.3.2.

Des études de scalabilité, dont le principe a été introduit en section 1.2.2, permettent d'observer l'évolution du temps de calcul face à l'augmentation du volume total de communication induite par l'augmentation du nombre de ressources utilisées. La scalabilité forte d'un code démontre sa capacité à traiter un problème plus rapidement lorsque le nombre de

ressources augmente. Idéalement, on attend que le temps de calcul soit réduit d'un facteur n lorsque les ressources sont augmentées d'un facteur n. Cette étude n'est pas adaptée à notre implémentation multi-GPU car elle conduit à la résolution de problèmes de tailles de plus en plus petite à mesure que le nombre de cartes utilisées augmente. Dans ce type d'étude, la taille du problème total est fixée et doit pouvoir être traitée par un seul GPU. De plus, comme nous avons vu dans le chapitre précédent, les meilleures performances sont obtenues lorsque le problème traité sur un GPU est suffisamment grand.

Une étude de scalabilité faible est préférable dans ce contexte. Elle conduit à l'étude de l'évolution du temps de calcul lorsque le nombre de ressources augmente en conservant une charge de travail par GPU constante. Ainsi, en partant d'une taille de problème compatible avec les caractéristiques d'une carte graphique, l'influence des communications est directement mesurée. Les résultats de scalabilité, présentés sur la figure 6.5, sont obtenus en en utilisant un schéma RK2 et une formule de remaillage $\Lambda_{6,4}$ pour résoudre des problèmes de taille (N_X, N_Y, nN_Z) où n est le nombre de GPU employés. Ainsi, chaque GPU est chargé d'un problème de taille (N_X, N_Y, N_Z). Nous ne considérons ici qu'une seule direction de découpe, dans la direction Z. La figure 6.5 représente l'évolution de la scalabilité en fonction du nombre de cartes graphiques utilisées. Deux cas sont considérés : le premier est une résolution identique de 256^3 points de grille par GPU pour la vitesse et le scalaire. Le second est un cas multiéchelle où la vitesse est connue sur une grille de taille 128^3 alors que le scalaire est résolu sur 512^3 points.

Les oscillations qui apparaissent pour les cas à 256^3 par GPU sont dues à la distance variable entre deux cartes voisines. En effet, les communications au sein d'un même nœud sont plus rapides que d'un nœud à l'autre. En pratique, lorsqu'un GPU est ajouté à un nombre impair, on ajoute la seconde carte du dernier nœud à l'ensemble des cartes utilisées. Dans ce cas, les communications supplémentaires sont réalisées entre les deux cartes du même nœud, ce qui conduit à une amélioration de la scalabilité. La perte importante de scalabilité entre 1 et 2 GPU provient de l'absence totale de communications dans le cas 1 GPU ainsi que du manque d'optimisation des cas multi-GPU. Comme nous l'avons évoqué précédemment, leur taille dépend du nombre CFL de la simulation. Pour les cas $N^{\boldsymbol{a}} = N^u = 256^3$, le nombre CFL maximal est égal à 8 et implique des communications de taille 8×256^2 éléments par voisins pour le champ de vitesse et le scalaire. Dans le cas multiéchelle $N^{\boldsymbol{a}} = 128^3$ et $N^u = 512^3$, le pas de temps est identique et les tailles de communications par voisins sont de 4×128^3 et 15×512^3 respectivement pour la vitesse et le scalaire. D'autre part, dans l'état actuel du code, nous n'avons pas encore mis en place de stratégies de recouvrement des communications par des calculs, ce qui implique que les temps de calculs sont augmentés de la totalité des temps de communications. Ainsi la scalabilité du cas 2 GPU par rapport au cas 1 GPU est assez faible car elle revient à comparer les temps de communications à un cas sans aucune communication. Toutefois, la scalabilité par rapport au cas utilisant 2 GPU est satisfaisante car supérieure à 0.85 jusqu'à 12 GPU. La version avec un seul kernel donne des résultats similaires à celle avec deux kernels séparés avec des temps de calcul inférieurs de 3%. D'autre part, la chute de scalabilité entre 1 et 2 GPU peut être atténuée en diminuant le volume de communications par une réduction du pas de temps.

La répartition du temps de calcul pour quelques cas tirés de l'étude précédente est représentée sur la figure 6.6. Elle donne la répartition du temps de calcul global d'une

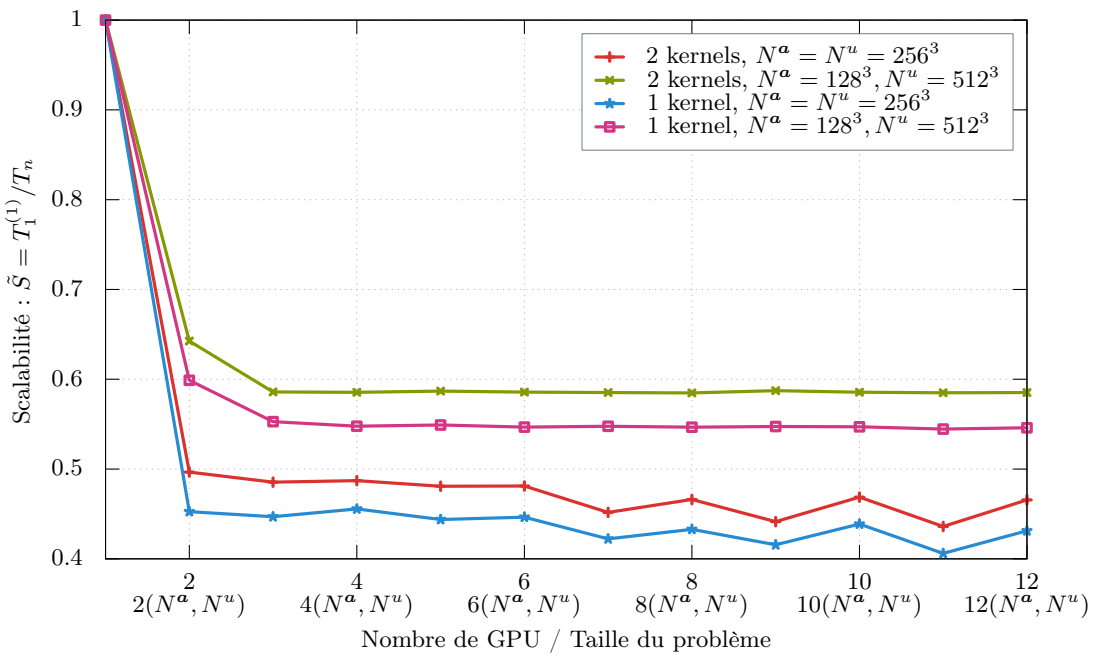


Figure 6.5. – Performances du transport de scalaire 3D multi-GPU

itération entre les différentes directions. Comme annoncé précédemment, les temps de calcul issus des communications dans la direction Z apparaissent comme ajoutés au temps de calcul obtenu sans communications et sont inclus dans la partie associée à l'hôte pour cette direction. Ainsi, seul le temps de communication varie avec le nombre de GPU et les temps d'exécutions des kernels restent constants.

Nous détaillons sur la figure 6.7a les éléments constituant la direction Z en distinguant les kernels OpenCL pour l'initialisation, les kernels d'advection et de remaillage ainsi que les étapes de transferts de données. Les communications entre CPU sont notées MPI et celles entre les cartes et les hôtes sont notées D→H ou H→D selon le sens de transfert. Enfin, l'étape d'ajout des contributions provenant des processus voisins est notée += et est effectuée par l'hôte de chaque carte. Les dépendances entre toutes ces étapes sont représentées sur la figure 6.8.

Cette représentation de l'enchaînement des différentes étapes lors du traitement d'une direction de découpe montre que certaines peuvent être réalisées simultanément. En effet, l'initialisation des particules est indépendante du transfert des données du champ de vitesse. De même, la copie des données du scalaire correspondant à une partie du domaine local est indépendante des communications des contributions pour les domaines voisins. Le parallélisme par tâches OpenCL permet de démarrer ces opérations sans en attendre la complétion. Les détails du profilage des temps de calculs pour la figure 6.7a sont réalisés en forçant une synchronisation explicite des tâches permettant leur comparaison.

Comme le montre la figure 6.7a, les étapes de calculs ne suffisent pas à couvrir les communications. La figure 6.7b illustre cette remarque en reprenant la figure 6.7a avec un code de couleur indiquant les étapes de calcul et celles de communication. Ces résultats suggèrent la nécessité d'une réduction des temps de communications. Une optimisation

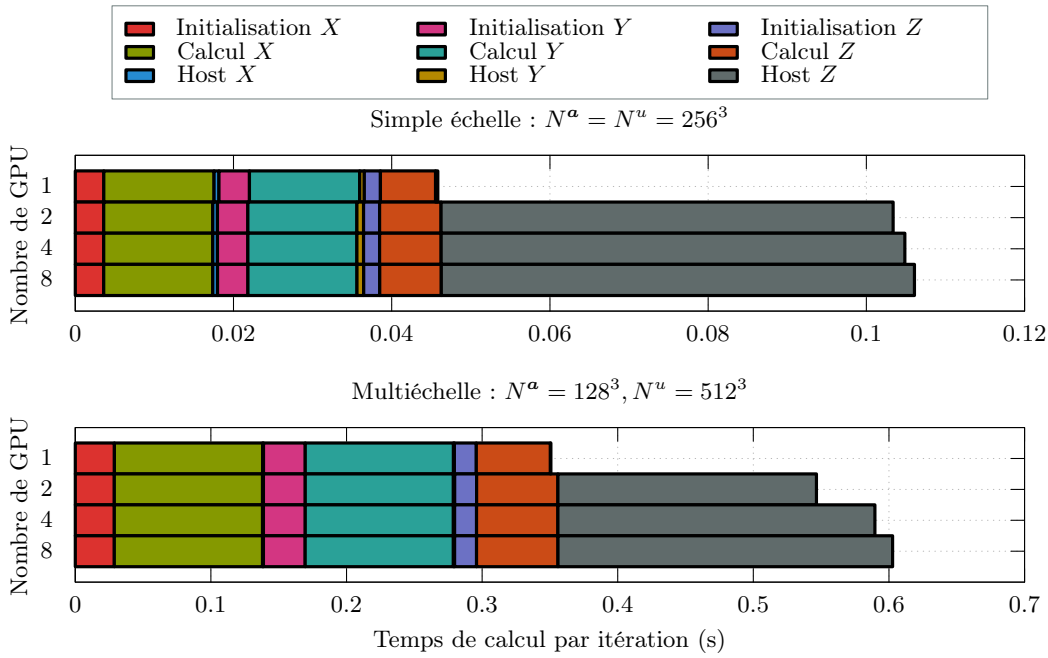
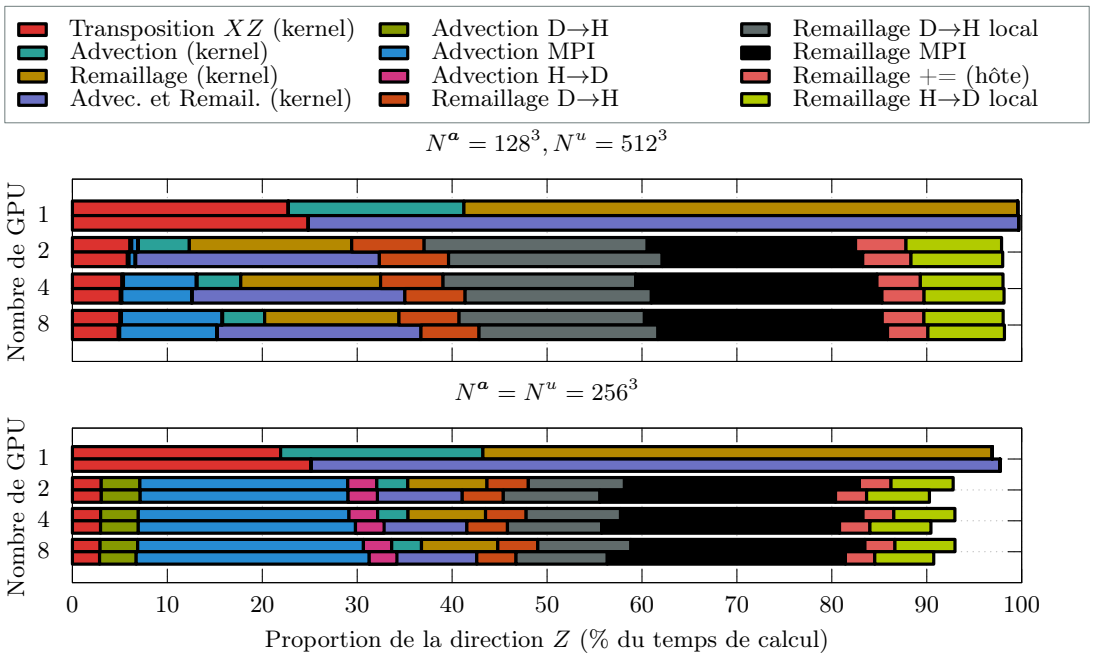


Figure 6.6. – Temps de calcul par itération

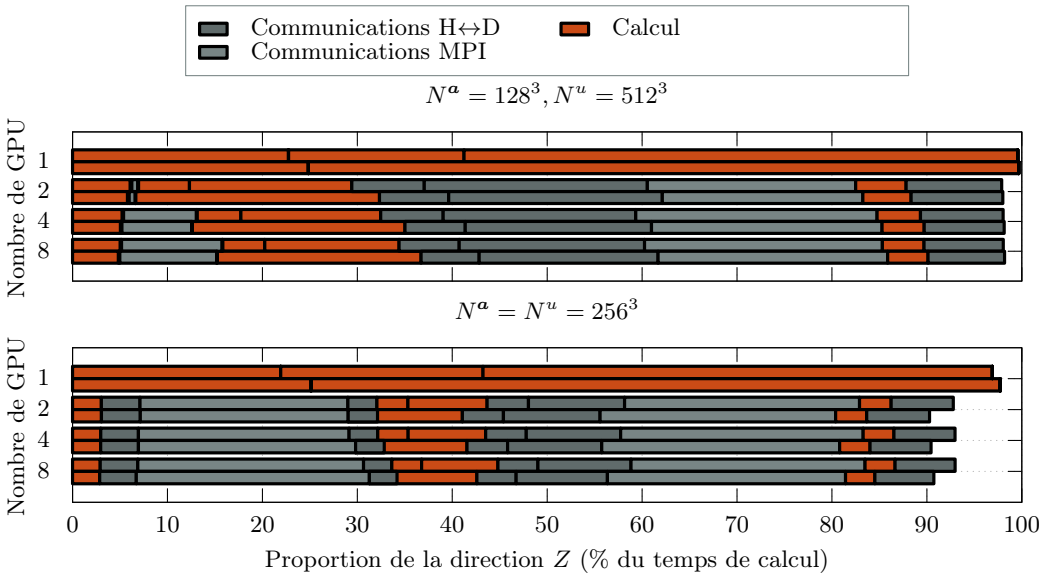
envisageable consiste à améliorer les temps de transfert entre GPU en utilisant, par exemple, un traitement par blocs en chaîne. Après cette première optimisation, une stratégie de recouvrement des communications par les calculs pourra être envisagée. En particulier, les étapes de calculs peuvent être séparées en deux parties à l'issue d'un redécoupage du domaine local. Une première partie entièrement locale, ne nécessitant pas l'usage de données distantes, peut s'exécuter pendant les communications. La seconde partie consiste à traiter le reste du domaine à l'aide des données transférées. L'optimisation des communications ainsi que leur recouvrement constituent une des perspectives immédiates de ce travail.

Les résultats présentés ici sont réalisés uniquement dans le cas d'une seule direction de découpe, par plans XY. L'implémentation que nous proposons ne se restreint pas à cette approche et permet notamment de considérer une découpe dans autre direction ainsi que dans plusieurs directions. Cette dernière est utile essentiellement dans les cas où un grand nombre de cartes graphiques est employé afin de réduire les disparités de nombre de points des domaines locaux. Par exemple, pour la résolution d'un problème global de taille (N_X, N_Y, N_Z) sur n cartes graphiques la résolution des domaines locaux est égale à $(N_X, N_Y, N_Z/n)$ dans le cas d'une découpe dans la direction Z. Elle passe à $(N_X, N_Y/n_1, N_Z/n_2)$, avec $n = n_1 n_2$ pour une découpe dans les directions Y et Z. L'avantage est de conduire à des résolutions plus grandes dans les directions de découpes que dans le premier cas. En revanche un plus grand nombre de communication est nécessaire. Les configurations considérées dans ce manuscrit ne nécessitent pas l'utilisation d'un découpage dans plusieurs directions car le nombre de cartes graphiques reste relativement faible.

Cette implémentation multi-GPU pour le transport de scalaire peut être employée en parallèle d'une résolution CPU de l'écoulement pour la simulation du problème complet.



(a) Étapes de l'algorithme



(b) Calculs et communications

Figure 6.7. – Profilage de la direction de découpe

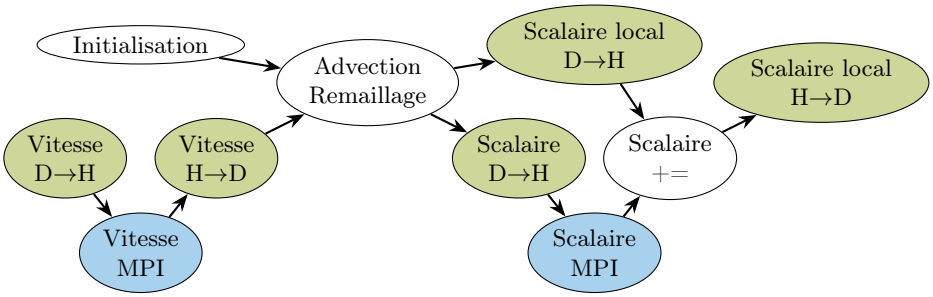


Figure 6.8. – Dépendances des étapes de communication multi-GPU

6.3. Transport turbulent d'un scalaire passif

6.3.1. Application hybride

Le problème que nous considérons dans cette partie est le transport de scalaire passif dans un jet plan turbulent tridimensionnel, tel que présenté par Magni *et al.* (2012). Le domaine de calcul est une boite unitaire périodique $[0;1]^3$. Le problème est modélisé par le couplage entre les équations de Navier-Stokes, en formulation vitesse vorticité, et d'une équation de transport du scalaire θ, rappelé ci-après :

$$\begin{cases} \operatorname{div} \boldsymbol{u} = 0, \quad \boldsymbol{\omega} = \operatorname{rot} \boldsymbol{u}, \\ \dfrac{\mathrm{D}\boldsymbol{\omega}}{\mathrm{D}t} = \left(\dfrac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{\omega} \right) = (\boldsymbol{\omega} \cdot \nabla)\, \boldsymbol{u} + \nu \Delta \boldsymbol{u}, \\ \dfrac{\mathrm{D}\theta}{\mathrm{D}t} = \dfrac{\partial \theta}{\partial t} + (\boldsymbol{u} \cdot \nabla)\theta = \kappa \Delta \theta. \end{cases} \tag{6.1}$$

L'écoulement est constitué d'un jet plan, modélisé par un champ de vitesse initialement nul dans tout le domaine excepté dans une région d'épaisseur w le long d'un plan XZ. Le champ de vitesse initial est perturbé par un champ aléatoire $\boldsymbol{P}$ de magnitude 0,05. Le scalaire passif est initialisé de la même manière que le champ de vitesse. Les valeurs initiales du champ de vitesse et du scalaire sont données par les expressions :

$$\theta(x,y,z) = (1 + 0.3\sin(8\pi x)) \left(1 + \tanh\left(\frac{0.1 - 2|y - 0.5|}{4w} \right) \right) /2 \tag{6.2}$$

$$\begin{aligned} \boldsymbol{u}^X(x,y,z) &= \theta(x,y,z)\boldsymbol{P}^X, \\ \boldsymbol{u}^Y(x,y,z) &= \boldsymbol{P}^Y \\ \boldsymbol{u}^Z(x,y,z) &= \boldsymbol{P}^Z \end{aligned} \tag{6.3}$$

La condition initiale du champ de vitesse est représentée en figure 6.9 pour la composante $\boldsymbol{u}^X$. L'écoulement se développe dans le plan du jet, dans la direction X. Les perturbations provoquent l'établissement d'un régime turbulent qui se traduit par l'évolution tridimensionnelle de l'écoulement. Le scalaire est ainsi transporté dans tout le domaine.

L'écoulement est caractérisé par la largeur du jet $w = 0.1$, ce qui donne un nombre de Reynolds égal à 10^3 pour une viscosité $\nu = 10^{-4}$. Différentes valeurs de nombre de Schmidt

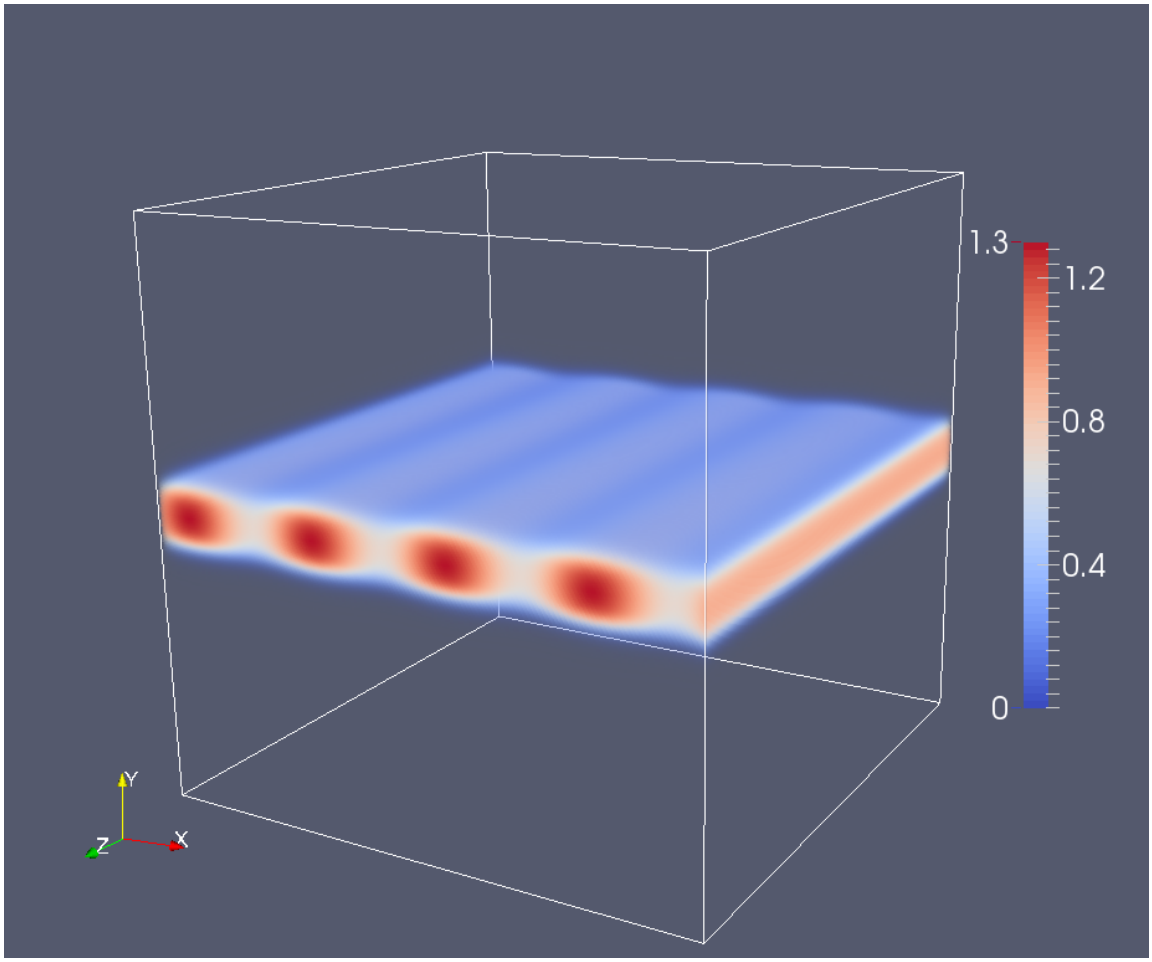


Figure 6.9. – Champ de vitesse initial, composante $\boldsymbol{u}^X$

seront considérées en fonction du rapport d'échelle entre les grilles de résolutions du fluide et du scalaire. En effet, comme nous l'avons décrit au chapitre 1, le rapport des échelles de dissipation d'énergie est lié au nombre de Schmidt : $\eta_\kappa\sqrt{Sc} = \eta_\nu$, avec $Sc = \nu/\kappa$ où ν est la viscosité du fluide et κ la diffusivité du scalaire. Ces différentes échelles sont prises en compte par la résolution des discrétisations des variables. Le champ de vitesse est résolu sur une grille plus grossière que celle du scalaire.

Nous rappelons que la méthode numérique hybride employée pour résoudre ces problèmes consiste en l'utilisation d'une méthode semi-Lagrangienne pour le transport du scalaire et de différences finies pour sa diffusion. L'écoulement est résolu en employant une méthode semi-Lagrangienne pour le transport de vorticité, des différences finies pour le terme d'étirement et une méthode spectrale pour la diffusion de vorticité. L'équation de Poisson pour le calcul du champ de vitesse est également résolue par une méthode spectrale.

6.3.2. Exploitation d'une machine hétérogène

Comme nous l'avons évoqué précédemment, le transport du scalaire et le fluide sont suffisamment indépendants pour être résolus simultanément sur différents matériels. En effet, le couplage de ces sous-problèmes consiste simplement en une communication du champ de vitesse en début d'itération de la partie fluide vers le transport. Pour l'exemple de l'utilisation de la machine Froggy présentée en section 6.1, nous utilisons n GPU et les n cœurs CPU hôtes associés pour résoudre le transport du scalaire ainsi que les $7n$ cœurs CPU disponibles sur les nœuds pour calculer le champ de vitesse de l'écoulement. Cette distribution des tâches est représentée sur la figure 6.10.

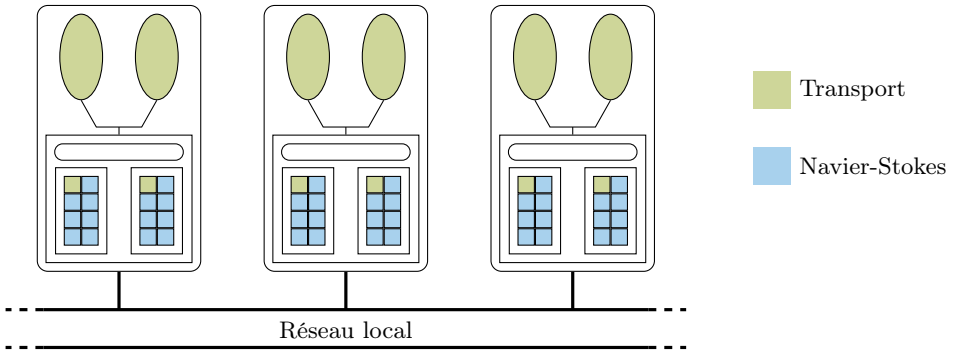


Figure 6.10. – Distribution des tâches pour l'application hybride

En pratique, nous introduisons une notion de parallélisme par tâche. Les différents processus MPI sont associés à une tâche à travers un découpage du communicateur global. Une synchronisation globale est effectuée à l'occasion de la communication du champ de vitesse. Dans cette exemple, deux tâches sont définies : une pour l'écoulement et l'autre pour le transport. Pour simplifier la distribution des tâches sur l'ensemble des processus, la résolution globale du scalaire doit être un multiple de n. De même, les résolutions de la vitesse et de la vorticité doivent être égales et multiples du nombre de cœurs CPU associés à leur calcul. D'autre part, les opérations de FFT du solveur spectral donnent de meilleures performances pour des résolutions multiples d'une puissance de 2. Ainsi, sauf

mention contraire, la résolution du fluide sera égale à un multiple de $4n$ et seulement 4 des 7 cœurs seront utilisés.

6.3.3. Résultats et performances

L'évolution du jet est caractérisée par une phase de diffusion suivie d'une phase de création d'enstrophie qui correspond à l'apparition d'instabilités tridimensionnelles s'accompagnant de la production de petites échelles. L'enstrophie $\mathcal{E}$ décrit l'évolution des pertes d'énergie cinétique de l'écoulement E par la relation :

$$\frac{\mathrm{d}\mathcal{E}}{\mathrm{d}t} = -\nu E, \tag{6.4}$$

avec

$$E = \frac{1}{2}\int_\Omega ||\boldsymbol{u}||^2 \mathrm{d}\boldsymbol{x} \quad \text{et} \quad \mathcal{E} = \int_\Omega ||\boldsymbol{\omega}||^2 \mathrm{d}\boldsymbol{x}$$

La figure 6.11a donne la variation de l'enstrophie de l'écoulement dont l'augmentation rapide, vers $t = 2$, indique l'apparition du régime turbulent. Les effets de dissipation d'énergie par diffusion deviennent ensuite prépondérants et se traduisent par une diminution de l'enstrophie et des valeurs extrêmes du champ de vitesse.

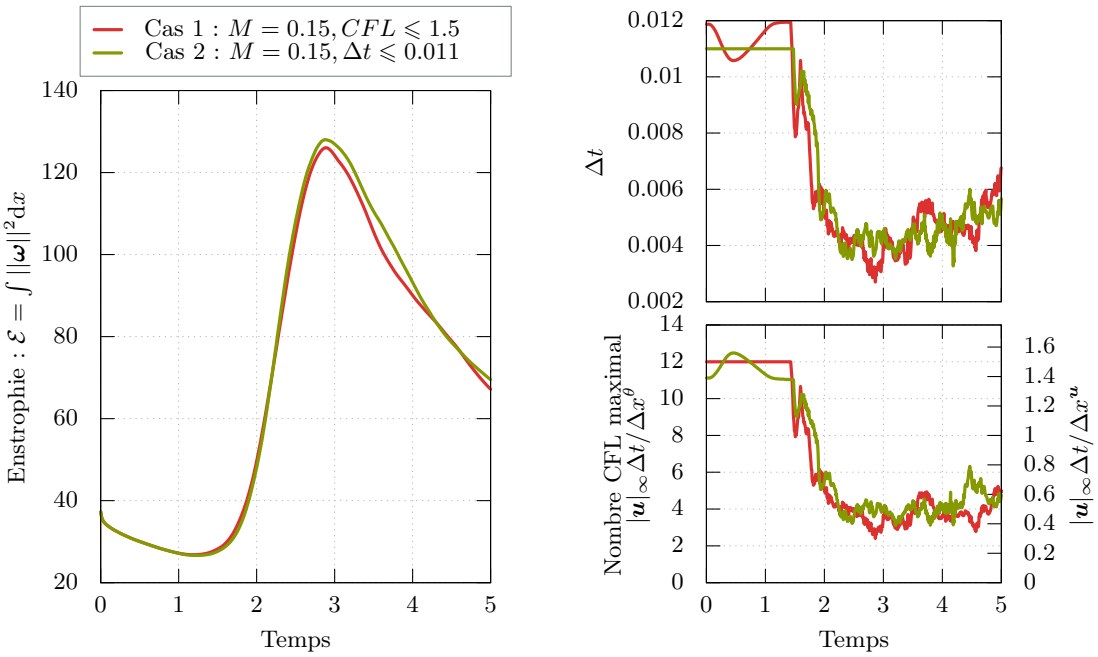
Ces simulations sont réalisées en employant des formules de remaillage $\Lambda_{6,4}$ et une intégration en temps par un schéma Runge-Kutta d'ordre deux pour les transports de la vorticité et du scalaire. Des schémas aux différences finies sont utilisés à l'ordre 4 pour le terme d'étirement et au premier ordre pour la diffusion du scalaire. Enfin, le pas de temps est calculé à chaque itération de manière à être maximal, tout en vérifiant la condition de CFL Lagrangienne :

$$\Delta t \leqslant \frac{M}{|\nabla \boldsymbol{u}|_\infty}.$$

Le choix de la constante $M < 1$ est laissé à l'utilisateur. Pour de grandes valeurs il peut conduire, selon les problèmes traités, à l'apparition d'instabilités numériques pouvant perturber l'écoulement de manière non physique. En pratique, une calibration de cette constante par rapport au problème et à ses conditions initiales est nécessaire. Dans cet exemple, nous constatons expérimentalement que des valeurs $M > 0.2$ ne permettent plus de capter la physique de l'écoulement surtout lorsque le régime turbulent est établi, pour $t > 3$. D'autre part, la perturbation aléatoire appliquée à la condition initiale conduit à des valeurs du gradient du champ de vitesse peu représentatives de l'écoulement, en début de simulation, et entraîne l'utilisation de pas de temps trop grands. Dans ce cas, il est nécessaire d'employer une stratégie de calcul différente en début de simulation. En particulier, une technique simple consiste à fixer le pas de temps à une valeur arbitraire. Une autre idée est d'ajouter une contrainte supplémentaire sous la forme d'une condition de CFL par rapport au champ de vitesse.

L'évolution du pas de temps et du nombre CFL maximal de l'écoulement sont présentés sur la figure 6.11b pour deux modifications du calcul du pas de temps. Lorsque l'écoulement n'a pas encore atteint un régime turbulent, nous appliquons une condition de type CFL (cas 1) tel que : $\Delta t \leqslant 1.5\Delta x^{\boldsymbol{u}}/|\boldsymbol{u}|_\infty$ ou bien (cas 2) une restriction à un pas de temps inférieur à 0.011. Dans tous les cas, le pas de temps utilisé vérifie la condition de CFL Lagrangienne.

Cette dernière devient la contrainte la plus restrictive à partir de l'instant $t = 1.5$ et conduit à une diminution du pas de temps. Enfin, il augmente à nouveau sur la fin de la simulation, à mesure que l'énergie est dissipée par les petites échelles. Toutefois, les pas de temps obtenus sont assez grands et le nombre CFL varie entre 2 et 13, par rapport à la grille fine de résolution $1\,024^3$. L'avantage de l'approche couplant une condition de CFL et la condition Lagrangienne est qu'elle permet de conserver l'intérêt d'un pas de temps adaptatif en suivant la dynamique de l'écoulement.



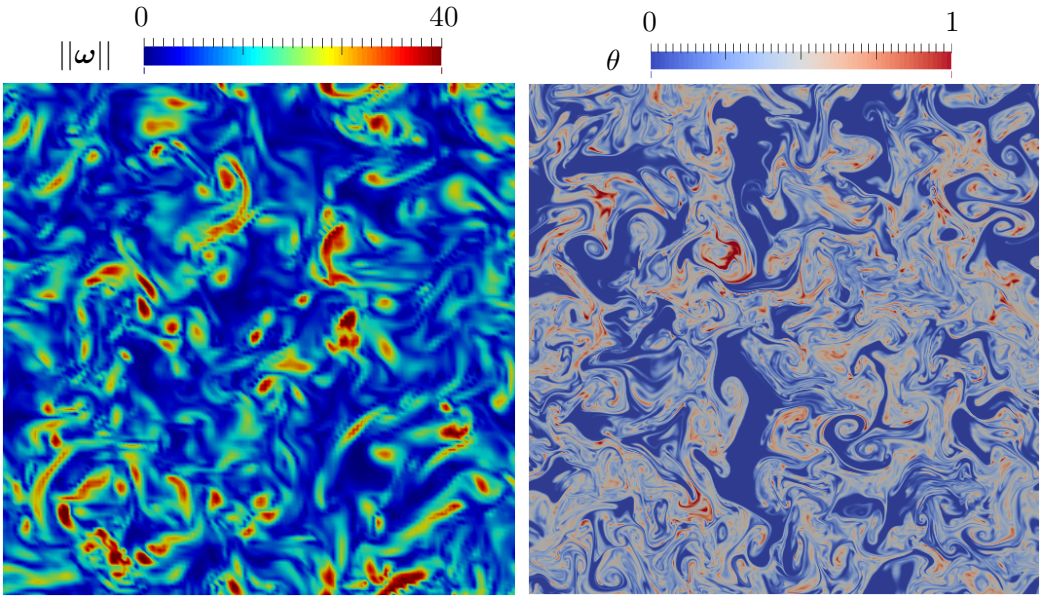
(a) Enstrophie **(b)** Pas de temps et CFL max.

Figure 6.11. – Configuration du jet turbulent

Le nombre CFL obtenu pour le transport du scalaire peut paraître relativement faible pour une méthode particulaire. En effet, dans cet exemple, nous employons le même pas de temps à la fois pour la résolution du fluide et celle du scalaire. Théoriquement, une condition moins restrictive que celle employée ici pour le transport du scalaire est envisageable et permettrait d'utiliser de plus grands pas de temps. Cela est illustré par les exemples de transports de fonctions levelset donnés dans le chapitre 5 où le nombre CFL atteint des valeurs de plusieurs dizaines. Une solution, exploitée notamment par Lagaert *et al.* (2014), consiste à réaliser des sous-itérations pour la partie fluide. Elle permet d'ajuster les contraintes sur le pas de temps indépendemment pour chaque sous-problème. En pratique, une condition LCFL est imposée pour la résolution du transport du scalaire, avec M grand, et un nombre de sous-itérations est calculé afin d'obtenir un pas de temps suffisamment petit pour la résolution du fluide. Dans cet exemple, il serait envisageable d'employer des pas de temps 6 fois plus grands pour le transport du scalaire, en prenant $M = 0.9$. Toutefois, l'intérêt de cette approche est limité ici car une itération du transport du scalaire est, dans la majorité des cas, totalement recouverte par le calcul de l'écoulement. Par conséquent, relativement au temps total de simulation, des sous-itérations pour la partie fluide condui-

raient à un renforcement du déséquilibre de charge entre les CPU et les GPU, comme nous le verrons plus loin.

Les figures 6.12 donnent les valeurs de la norme de la vorticité et du scalaire dans un plan XZ au centre du jet à l'instant $t = 4.5$. L'écoulement est complètement turbulent et on observe de nombreuses structures de tailles variables. Une comparaison de ces deux images permet d'illustrer les différentes échelles qui caractérisent ce problème pour un nombre de Schmidt élevé. En effet, les plus petites structures de la figure de gauche, correspondant au fluide, sont bien plus grandes que celles de la figure de droite, pour le scalaire. Le rapport entre les tailles des structures est égal à $\sqrt{Sc}$ et est égal à 8 dans cet exemple.



(a) Norme de la vorticité **(b)** Scalaire

Figure 6.12. – Champs instantanés dans le plan XZ, au centre du jet, à $t = 4.5$. $N^{\boldsymbol{u}} = 128^3$, $N^{\theta} = 1\,024^3$, $Sc = 64$

Cette simulation a été réalisée en employant 32 cœurs CPU pour la résolution de l'écoulement et 8 GPU pour le transport du scalaire à raison de 1,4 secondes par itérations, soit 23 minutes de calcul pour atteindre $t = 5$. La répartition des temps de calcul pour les différentes parties de la résolution sont présentées en figure 6.13 pour différentes résolutions et configurations. Par notre stratégie de programmation hybride, les résolutions des deux parties du problème sont exécutées simultanément sur différentes ressources. Ainsi, le temps de calcul total est donné par la plus longue des parties. Les trois premières lignes de la figure correspondent à une étude de scalabilité forte car les résolutions restent identiques. La troisième ligne fait apparaître une large part de communication pour l'advection multi-GPU dans la direction Z. Le temps de calcul associé à cette étape est encore augmenté lorsque la résolution du scalaire passe à $1\,024^3$ points.

La nécessité d'une optimisation des communications entre GPU telle que suggérée par les résultats de la section 6.2 est ici moins prononcée. En effet, dans la plupart des configurations employées pour cette application, la résolution du fluide est la partie limitante du temps de calcul global. Ainsi, les communications, même non optimisées, générées par

le transport multi-GPU sont recouvertes par les calculs de l'écoulement sur CPU. Comme nous l'avons détaillé précedemment, notre implémentation se base sur une majoration de la quantité de données à échanger pour les communications inter-GPU. Dans ces exemples, cela conduit à un volume de communications par GPU pour les cas $N^{\boldsymbol{u}} = 128^3$ et $N^{\boldsymbol{u}} = 256^3$ respectivement de 2×128^2 et 4×256^2 points pour les données du champ de vitesse. Pour le scalaire, le nombre de points est de 10×512^2 et $17 \times 1\,024^2$ pour les cas $N^{\theta} = 512^3$ et $N^{\theta} = 1\,024^3$.

Cette estimation des tailles de communications est réalisée à priori par l'utilisateur à partir d'une majoration du pas de temps et du champ de vitesse. Une limite à cette approche est que les données échangées ne sont pas toujours entièrement utilisées. En effet, au gré de l'évolution du pas de temps, la taille des données nécessaires peut diminuer par rapport à l'estimation. De même, le maximum de vitesse n'est pas nécessairement atteint au voisinage des bords du domaine local. La mise en place d'un calcul de la taille des données à échanger à chaque itération a été envisagé mais n'a pas été retenu. En effet, la complexité algorithmique supplémentaire ne permet pas nécessairement de conduire à une réduction des temps de calcul liés à la diminution du volume des communications. Le code GPU que nous développons exploite des tableaux en mémoire partagée dans laquelle l'allocation dynamique n'est pas possible. Par conséquent, un changement dans la taille de ces tableaux nécessite une recompilation des noyaux OpenCL au cours de l'exécution. Cette recompilation peut être réalisée seulement dans les cas où la taille des tableaux augmente. Cependant, les communications et les transferts seraient effectués sur des données non contiguës, ce qui nuit aux performances.

On notera que le temps de calcul associé aux communications induites par le découpage par tâches de l'implémentation hybride sont assez faibles car elles ne concernent que le champ de vitesse. Enfin, les temps de calculs pour l'advection de la vorticité sont assez longs relativement à la résolution du champ de vitesse. Une des causes de ce comportement est que l'implémentation multi-CPU développée par Lagaert *et al.* (2014) que nous utilisons est plutôt destinée à traiter de larges résolutions sur un très grand nombre de processeurs, jusqu'à $3\,064^3$ points sur 2 048 cœurs. De plus, traiter un problème de taille 128^3 sur 32 cœurs conduit à une charge par cœur de seulement $128^2 \times 4$ points. Le volume de communications généré dans la direction Z devient relativement important car toutes les particules nécessitent un envoi, le support de la formule de remaillage étant plus large que le domaine local. Le cas 4 est le seul où le calcul du transport du scalaire est plus long que le calcul de l'écoulement sur CPU. Ce déséquilibre sera vraisemblablement atténué par une optimisation des communications entre GPU.

Comme pour l'étude dans le cas multi-GPU, nous réalisons une étude de scalabilité faible, sur la figure 6.14a, dont les temps de calculs sont donnés sur la figure 6.14b. Deux cas d'occupation des nœuds sont considérés. Le premier consiste en la résolution d'un problème de taille $N^{\boldsymbol{u}} = 128^3$ et $N^{\theta} = 512^3$ pour un ensemble de ressources composé de 4 cœurs CPU pour la résolution du fluide et 1 GPU pour le transport du scalaire, ce qui correspond à une occupation de $5/8 = 62,5\%$. Le second cas nous permet d'atteindre une occupation maximale en utilisant 7 cœurs. La taille du sous-problème de la partie fluide $N^{\boldsymbol{u}} = 112^3 = (7 \times 16)^3$ est ajustée pour être multiple de 7. Comme prévu par les résultats précédents, la résolution du fluide est effectuée en un temps plus long que le transport du scalaire. Dans les deux cas, l'évolution du temps de calcul du solveur multi-GPU donne lieu à une

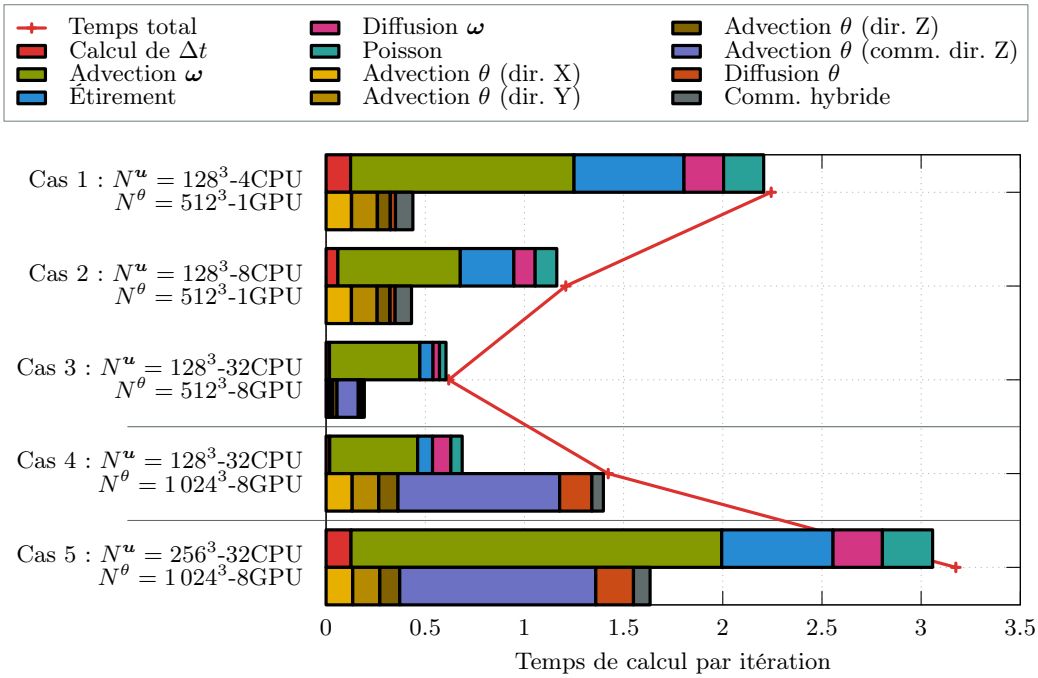
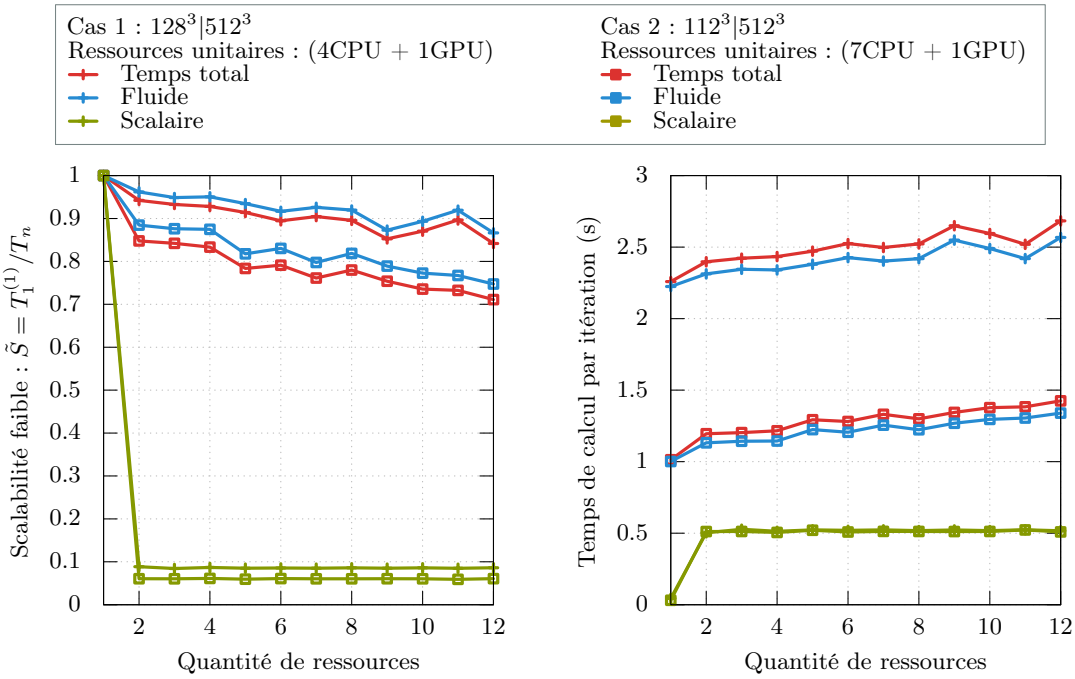


Figure 6.13. – Répartition du temps de calcul pour le cas hybride

scalabilité similaire à celle obtenue pour la figure 6.5. Les communications entre GPU sont pénalisées par la présence des communications induites par la résolution de l'écoulement sur CPU, en plus des raisons évoquées en section 6.2. En effet, toutes les communications internœuds passent nécessairement par le même réseau et se partagent sa bande passante, ce qui limite les performances en comparaison au cas où seuls les GPU sont utilisés. En revanche, la scalabilité reste supérieure à 85 et 70% respectivement dans les deux cas sur 48 et 84 cœurs pour le calcul de l'écoulement.

Conclusion

Dans ce chapitre nous avons vu comment les mécanismes habituels utilisés pour l'exploitation des CPU multicœurs des machines à mémoire distribuée s'étendent au cas des accélérateurs. L'approche étudiée consiste à compléter le parallélisme classique pour la gestion de la mémoire distribuée par un niveau de parallélisme spécifique à l'utilisation des accélérateurs. Ainsi, notre implémentation est capable d'exploiter l'ensemble des ressources d'une machine hétérogène, composée de CPU multicœurs et de cartes graphiques. Dans le modèle de programmation OpenCL, les cartes graphiques sont associées à un processus hôte sur le CPU. Dans le cas d'une utilisation seule des GPU, un grand nombre de cœurs sont donc inutilisés. Nous avons exploré une solution à ce problème d'occupation à l'occasion de l'exemple de transport de scalaire dans un écoulement turbulent. L'idée est de placer des processus sur l'ensemble des cœurs des nœuds et de les associer à différentes tâches. Un ensemble de processus est identifié comme hôtes des cartes graphiques permettent de



(a) Scalabilité faible **(b)** Temps de calcul

Figure 6.14. – Scalabilité faible pour le cas hybride

calculer le transport du scalaire alors que les autres processus sont affectés à la résolution de l'écoulement.

Ce chapitre nous a permis d'analyser les performances de la méthode dans un cadre multi-GPU à travers une étude de scalabilité faible. Il apparaît dans les résultats que la part de temps de calcul induite par les communications entre cartes graphiques nécessite des optimisations. En effet, diverses stratégies ont été évoquées telles qu'un découpage en blocs pour un traitement en chaîne ou encore un recouvrement par les calculs ne nécessitant pas de communications. Toutefois, lors de la résolution hybride du problème complet, le temps de calcul est généralement limité par la résolution du fluide sur les cœurs CPU.

Le problème de transport de scalaire passif par un jet plan turbulent a été réalisé sur diverses configurations matérielles jusqu'à l'exploitation de 6 nœuds complets de la machine Froggy, soit 96 cœurs CPU et 8 GPU et pour des résolution de 256^3 points pour le fluide et $1\,024^3$ pour le scalaire. Une limitation de cette approche est que l'équilibrage de la charge entre les CPU et les GPU n'est pas possible autrement que par une modification des résolutions ou des paramètre des méthodes. Ainsi, le meilleur équilibrage de charge est obtenu dans le cas 4 de la figure 6.13 et pourrait être amélioré par l'optimisation des communications inter-GPU. Pour ce qui est du cas 5, correspondant aux résolutions les plus importantes, une solution consisterait à employer des ressources CPU supplémentaires, des nœuds sans GPU par exemple.

Conclusion générale

L'objectif principal de cette thèse était d'explorer les possibilités d'une exploitation de machines de calcul hétérogènes par une méthode hybride pour la résolution de problèmes multiéchelles de transport de scalaire passif.

La méthode de résolution employée repose sur une méthode particulaire avec remaillage. Elle s'identifie à une classe de méthodes semi-Lagrangiennes dans laquelle les particules suivent les trajectoires de l'écoulement mais en traitant l'interpolation particule-grille de manière explicite par remaillage. L'intérêt majeur est de combiner les avantages des méthodes Lagrangiennes et Eulériennes. En effet, cette méthode permet l'utilisation de pas de temps plus grands que ceux obtenus sous une contrainte de CFL classique pour les méthodes de grille. De plus, elle est parfaitement adaptée à la résolution d'équations de conservation et permet une montée en ordre des schémas numériques lorsque des formules de remaillages conservant un grand nombre de moments et suffisamment régulières sont utilisées. Nous avons exposé, dans le chapitre 2, une méthode de construction de telles formules dont les caractéristiques influent directement sur l'ordre de la méthode. Dans ce même chapitre, nous avons exploité l'analogie avec la méthode des différences finies pour démontrer la consistance et la stabilité des schémas numériques ainsi obtenus.

Le principe d'une méthode hybride tel que rappelé dans le chapitre 3 est de coupler différentes méthodes de résolution adaptées aux différents aspects physiques et numériques du problème considéré. Dans cette thèse, nous avons repris l'idée d'un couplage de la méthode semi-Lagrangienne avec des schémas aux différences finies et des méthodes spectrales pour la résolution de problèmes de transport de scalaire passif dans un écoulement turbulent. Cette approche consiste à combiner les trois niveaux d'hybridation : grilles de résolutions différentes pour l'écoulement et le transport du scalaire, méthodes numériques de diverses natures et calcul sur architectures hétérogènes. Le calcul hybride est un aspect émergent du calcul à hautes performances et découle directement de la volonté de s'adapter aux architectures des machines de calcul parallèles. Le constat de leur évolution rapide nous a poussé à développer un code multiarchitecture dans lequel l'accent a été mis sur la portabilité et la souplesse d'utilisation (chapitre 4). Ce code est ainsi capable de s'exécuter sur divers types d'architectures (CPU multicœur et GPU) et de machines (d'un portable jusqu'à un serveur de calcul parallèle). Le langage Python a été choisi pour sa portabilité, sa simplicité d'interfaçage avec d'autres langages et ses performances. Il nous permet d'orchestrer les simulations en utilisant, entre autres, les librairies MPI et OpenCL ainsi que des routines en Fortran.

L'utilisation de cartes graphiques est particulièrement bien adaptée à la méthode particulaire avec remaillage d'ordre élevé. En effet, comme nous l'avons montré dans le chapitre 5, la régularité des structures de données ainsi que l'intensité opérationnelle des schémas numériques nous permettent d'obtenir des performances de calcul intéressantes. En particulier, dans une analyse par le modèle roofline, elles atteignent des performances généralement supérieures à 50% de la puissance de calcul maximale atteignable. Les faibles temps de calculs ainsi obtenus incitent à traiter des résolutions importantes. Cependant, elles sont rapidement limitées par les tailles des différents niveaux de mémoire des cartes graphiques. C'est pourquoi, nous avons développé une version multi-GPU de la méthode dont les performances, en termes de scalabilité, ont été analysées dans le chapitre 6. Les résultats ont montré un fort potentiel de réduction du temps de calcul induit par les communications entre GPU sous réserve de leur optimisation et de leur recouvrement par des calculs. Cette implémentation multi-GPU nous permet d'atteindre l'objectif principal de réalisation de simulations de transport d'un scalaire passif dans un écoulement turbulent en exploitant une méthode hybride et une architecture hétérogène multi-CPU et multi-GPU. Nous avons proposé une approche permettant d'augmenter l'occupation des nœuds de calcul basée sur un parallélisme par tâches. Elle conduit à la résolution simultanée du transport du scalaire sur plusieurs cartes graphiques et de l'écoulement sur les cœurs CPU disponibles. Malgré des temps de communications entre GPU relativement importants, le calcul du transport du scalaire reste généralement plus rapide que celui de l'écoulement. Une limite à cette stratégie est que la résolution de deux sous-problèmes sur des architectures différentes peut conduire à un déséquilibre de charge qu'il est possible d'ajuster par une modification des paramètres des méthodes et des tailles de grilles.

De nombreuses perspectives se dégagent des aspects mathématiques, numériques et applicatifs de ce travail. Au niveau de la méthode numérique, il serait intéressant d'étendre l'étude de montée en ordre aux schémas d'intégration des trajectoires des particules et de splitting dimensionnel. Pour les premiers, nous avons réalisé l'analyse de consistance pour des schémas d'advection d'ordre 1 et 2 ainsi que l'analyse de stabilité au premier ordre. Une poursuite de ces études pour le cas de schémas d'ordres plus élevés serait nécessaire afin de s'assurer de la validité de l'utilisation de schémas de type Runge-Kutta d'ordre 4, par exemple. En ce qui concerne le splitting, nous avons essentiellement employé un splitting de Strang d'ordre 2. La mise en œuvre d'un splitting d'ordre 3 ou 4 serait intéressante du point de vue numérique et algorithmique.

D'autre part, lors de l'étude de la construction de formules de remaillage d'ordre élevés $\Lambda_{p,r}$, nous avons tenté, de produire des formules diffusives, sans obtenir de résultats satisfaisants. En pratique de telles formules ont été obtenues seulement à partir des noyaux $\Lambda_{2,r}$. L'intérêt de ces formules serait de réaliser une diffusion de la quantité transportée par les particules directement dans l'étape de remaillage et ainsi d'éviter un second calcul dédié. En effet, le traitement de la diffusion du scalaire passif, par différences finies dans nos simulations, nécessite un parcours de l'ensemble des données en lecture et écriture dont la suppression conduirait à une diminution du temps de calcul. Cette amélioration serait vraisemblablement significative sur GPU mais devrait également être sensible sur des architectures CPU plus classiques. La méthode de construction des formules, développée dans le chapitre 2, consiste à résoudre un système linéaire dont les inconnues sont les coefficients polynômiaux. Ce système traduit l'expression des différentes contraintes parmi lesquelles la conservation des p premiers moments. Une étude plus fine que la simple modification de la

contrainte de conservation du moment d'ordre 2 serait nécessaire pour la prise en compte de la diffusion.

Du point de vue numérique, une perspective se dégage directement de l'analyse des performances de la résolution du transport de scalaire dans un cas multi-GPU. Comme nous l'avons souligné dans le chapitre 6, les communications entre cartes graphiques font état d'un temps de traitement assez long comparativement au temps de calcul total. Une réduction de ce temps de traitement est envisageable en deux étapes. Dans une première étape, il serait intéressant d'optimiser les transferts de données entre GPU en utilisant une approche similaire à celle employée dans la librairie MPI-ACC (Aji *et al.*, 2012). L'idée est de traiter les messages en plusieurs envois successifs. Ce traitement par blocs en chaîne devrait permettre un recouvrement des différentes étapes et ainsi accélérer la communication. En effet, pendant l'envoi du message correspondant à un bloc, il est possible de réaliser la copie depuis l'hôte vers la carte de destination du bloc précédent ainsi que celle du bloc suivant depuis la carte vers l'hôte émetteur. Dans une seconde étape, un recouvrement de ces transferts par des calculs est possible en réalisant, par exemple, le calcul en deux passes. Les calculs locaux, indépendants des données distantes, peuvent être réalisés pendant leurs transferts.

En ce qui concerne l'implémentation multi-GPU, l'occupation des ressources CPU est un problème délicat pour lequel nous avons proposé une solution dans le chapitre 6. En effet, les nœuds de calculs présentent généralement plus de cœurs CPU que de GPU et comme notre implémentation se base sur l'exploitation d'une carte par un seul processus, cela conduit à une inutilisation des cœurs restants. L'approche basée sur un parallélisme par tâches consiste, dans notre application, à réaliser le calcul de l'écoulement sur ces cœurs CPU disponibles. Sur l'exemple d'utilisation de la machine Froggy, le fluide est calculé sur 4 des 7 cœurs disponibles, soit une occupation de $5/8 = 62.5\%$. Une occupation maximale a été obtenue artificiellement en distribuant cette tâche sur l'ensemble des 7 cœurs en ajustant la taille de la grille de résolution. De même, une tentative de placement de 8 processus pour cette même tâche sur les 8 cœurs physiques sur lesquels est ajouté un 9e processus pour le calcul du scalaire a conduit à des performances largement dégradées. Nous atteignons ainsi les limites de notre parallélisme par tâche qui, par choix de simplification des communications, se restreint à la possibilité d'une seule tâche par processus. Cependant, une autre stratégie pour l'augmentation de l'occupation est envisageable. Dans le cas des octocœurs utilisés, les meilleurs résultats ont été obtenus en employant 1 cœur CPU associé à 1 GPU pour le calcul du scalaire et 4 cœurs CPU pour le calcul de l'écoulement. Les trois cœurs restants peuvent être assignés à d'autres tâches que celles de calcul telles que des opérations de post-traitement, de sorties sur fichiers ou encore pour la gestion de reprises des calculs en cours de simulation.

D'autre part, l'implémentation que nous avons proposé est basée exclusivement sur un parallélisme par envoi de messages pour les tâches n'utilisant pas de GPU. Une perspective intéressante serait d'envisager l'ajout d'un paradigme à mémoire partagée pour l'exploitation des cœurs CPU. Ainsi, les cœurs d'un même nœuds seraient utilisés à travers un parallélisme spécifique tel que OpenMP, ou même OpenCL. Un parallélisme à trois niveaux serait donc obtenu en combinant l'utilisation de MPI entre les nœuds, OpenMP (ou OpenCL) pour les cœurs CPU d'un même nœud et OpenCL pour les GPU. La conception a été réalisée dans le but d'obtenir un code robuste dans lequel ces modifications peuvent

être apportées sans impacter l'ensemble de la librairie et de manière transparente pour l'utilisateur.

Enfin, une perspective à ce travail, en termes d'application, serait de réaliser des simulations de transport de scalaire actif ayant un effet sur l'écoulement. Par exemple, lorsque le scalaire représente une densité dans un écoulement de fluide multiphasique. Ce problème se modélise par une équation de transport de la densité dont la vitesse est donnée par les équations de Navier-Stokes. Dans ces dernières un terme barotrope apparaît pour permettre la prise en compte de la densité comme force extérieure au modèle. En pratique la résolution du transport de la densité est toujours envisagée sur GPU et celle du terme barotrope sera mise en œuvre soit directement sur GPU soit sur CPU, selon la nature de la méthode numérique utilisée. Dans le premier cas, cela implique une modification du champ de vitesse sur GPU et donc son transfert vers le CPU pour la résolution de l'écoulement. Dans le second cas, un simple transfert de la densité vers le CPU est nécessaire. L'intérêt de cette application serait de conserver le caractère multiéchelle du problème en utilisant des grilles de résolutions différentes pour l'écoulement et la densité. Dans ce cas, il serait nécessaire de filtrer la densité, après son transport, à l'échelle de l'écoulement. Ce filtre serait réalisé sur les GPU afin de ne transférer vers les CPU qu'une densité filtrée, donc de taille réduite.

Bibliographie

Aji, A. M., J. Dinan, D. Buntinas, P. Balaji, W.-c. Feng, K. R. Bisset et R. Thakur (2012). « MPI-ACC: An Integrated and Extensible Approach to Data Movement in Accelerator-based Systems ». *IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems*.

Aji, A. M., L. S. Panwar, F. Ji, M. Chabbi, K. Murthy, P. Balaji, K. R. Bisset, J. Dinan, W.-c. Feng, J. Mellor-Crummey, X. Ma et R. Thakur (2013). « On the Efficacy of GPU-Integrated MPI for Scientific Applications ». *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*.

Baqais, A., M. Assayony, A. Khan et M. Al-Mouhamed (2013). « Bank Conflict-Free Access for CUDA-Based Matrix Transpose Algorithm on GPUs ». *International Conference on Computer Applications Technology*.

Batchelor, G. K. (1958). « Small-scale variation of convected quantities like temperature in turbulent fluid Part 1. General discussion and the case of small conductivity ». *Journal of Fluid Mechanics* 5.1.

Bergdorf, M., G.-H. Cottet et P. Koumoutsakos (2005). « Multilevel Adaptive Particle Methods for Convection-Diffusion Equations ». *Multiscale Modeling & Simulation* 4.1.

Bergdorf, M. et P. Koumoutsakos (2006). « A Lagrangian Particle-Wavelet Method ». *Multiscale Modeling & Simulation* 5.3.

Bergman, K., S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams et K. Yelick (2008). *ExaScale Computing Study : Technology Challenges in Achieving Exascale Systems*. Rapport technique. DARPA IPTO.

Büyükkeçeci, F., O. Awile et I. F. Sbalzarini (2012). « A portable OpenCL implementation of generic particle–mesh and mesh–particle interpolation in 2D and 3D ». *Parallel Computing* 39.2.

Canuto, C., M. Y. Hussaini, A. Quarteroni et T. A. Zang (1987). *Spectral Methods in Fluid Dynamics (Scientific Computation)*.

Chen, S. et G. D. Doolen (1998). « Lattice boltzmann method for fluid flows ». *Annual Review of Fluid Mechanics* 30.0.

Cheng, H., L. Greengard et V. Rokhlin (1999). « A Fast Adaptive Multipole Algorithm in Three Dimensions ». *Journal of Computational Physics* 155.2.

Chorin, A. J. (1968). « Numerical Solution of the Navier-Stokes Equations ». *Mathematics of Computation* 22.

Chorin, A. J. (1973). « Numerical study of slightly viscous flow ». *Journal of Fluid Mechanics* 57.4.

Cocle, R., G. Winckelmans et G. Daeninck (2008). « Combining the vortex-in-cell and parallel fast multipole methods for efficient domain decomposition simulations ». *Journal of Computational Physics* 227.21.

Coquerelle, M. et G.-H. Cottet (2008). « A vortex level set method for the two-way coupling of an incompressible fluid with colliding rigid bodies ». *Journal of Computational Physics* 227.21.

Cottet, G.-H. (1991). « Particle-grid domain decomposition methods for the Navier-Stokes equations in exterior domains ». *Lectures in Applied Mathematics* 28.

Cottet, G.-H. et S. Mas-Gallic (1990). « A particle method to solve the Navier-Stokes system ». *Numerische Mathematik* 57.1.

Cottet, G.-H., P. Koumoutsakos et M. L. Ould Salihi (2000). « Vortex Methods with Spatially Varying Cores ». *Journal of Computational Physics* 162.1.

Cottet, G.-H., B. Michaux, S. Ossia et G. VanderLinden (2002). « A Comparison of Spectral and Vortex Methods in Three-Dimensional Incompressible Flows ». *Journal of Computational Physics* 175.2.

Cottet, G.-H. et L. Weynans (2006). « Particle methods revisited: a class of high order finite-difference methods ». *Comptes Rendus Mathematique* 343.1.

Cottet, G.-H., G. Balarac et M. Coquerelle (2009a). « Subgrid particle resolution for the turbulent transport of a passive scalar ». *Advances in Turbulence XII, Proceedings of the 12th EUROMECH European Turbulence Conference, September, 2009* 132.1.

Cottet, G.-H. et A. Magni (2009b). « TVD remeshing formulas for particle methods ». *Comptes Rendus de l'Academie des Sciences Paris* Serie I 34.23.

Cottet, G.-H., J.-M. Etancelin, F. Perignon et C. Picard (2014). « High order semi-lagrangian particles for transport equations: numerical analysis and implementations issues. » *ESAIM: Mathematical Modelling and Numerical Analysis* 48.4.

Crouseilles, N., T. Respaud et E. Sonnendrücker (2009). « A forward semi-Lagrangian method for the numerical solution of the Vlasov equation ». *Computer Physics Communications* 180.10.

Degond, P. et S. Mas-Gallic (1989). « The Weighted Particle Method for Convection-Diffusion Equations Part 1 : The Case of an Isotropie Viscosity ». *Mathematics of Computation* 53.188.

Dolbeau, R., S. Bihan et F. Bodin (2007). « HMPP: A hybrid multi-core parallel programming environment ». *Workshop on General Purpose Processing on Graphics Processing Units.*

Dongarra, J., I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon et A. White (2003). *The Sourcebook of Parallel Computing.*

Dongarra, J., P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. van der Steen, J. Vetter, P. Williams, R. Wisniewski et K. Yelick (2011). « The Internatio-

nal Exascale Software Project roadmap ». *International Journal of High Performance Computing Applications* 25.1.

Dongarra, J., J. Hittinger, J. Bell, L. Chac\'{o}n, R. Falgout, M. Heroux, P. Hovland, E. Ng, C. Webster et S. Wild (2014). *Applied Mathematics Research for Exascale Computing.* Rapport technique. U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program.

Fermi, E., J. Pasta et S. Ulam (1955). *Studies of nonlinear problems.* Rapport technique.

Gotoh, T., S. Hatanaka et H. Miura (2012). « Spectral compact difference hybrid computation of passive scalar in isotropic turbulence ». *Journal of Computational Physics* 231.21.

Green500 (2014). *http://www.green500.org/.*

Gustafson, J. L. (1988). « Reevaluating Amdahl's Law ». *Communications of the ACM* 31.5.

Higham, N. J. (2002). *Accuracy and Stability of Numerical Algorithms.* Second edi. SIAM.

Ishiyama, T., K. Nitadori et J. Makino (2012). « 4.45 Pflops astrophysical N-body simulation on K computer - The gravitational trillion-body problem ». *International Conference for High Performance Computing, Networking, Storage and Analysis, SC.*

Jansson, N., J. Hoffman et M. Nazarov (2011). « Adaptive simulation of turbulent flow past a full car model ». *State of the Practice Reports on - SC '11* 0.

Khronos (2014). *The OpenCL Specification, version 2.0.*

Klöckner, A., N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov et A. Fasih (2012). « PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation ». *Parallel Computing* 38.3.

Kong, X., M. C. Huang, C. Ren et V. K. Decyk (2011). « Particle-in-cell simulations with charge-conserving current deposition on graphic processing units ». *Journal of Computational Physics* 230.4.

Koumoutsakos, P. (1997). « Inviscid Axisymmetrization of an Elliptical Vortex ». *Journal of Computational Physics* 138.2.

Koumoutsakos, P. et A. Leonard (1995). « High-resolution simulations of the flow around an impulsively started cylinder using vortex methods ». *Journal of Fluid Mechanics* 296.

Kuczaj, A., E. Komen et M. Loginov (2010). « Large-Eddy Simulation study of turbulent mixing in a T-junction ». *Nuclear Engineering and Design* 240.9.

Labbé, S., J. Laminie et V. Louvet (2004). *Méthodologie et environment de développement orientés objets : de l'analyse mathématique à la programmation.*

Lagaert, J.-B., G. Balarac, G.-H. Cottet et P. Bégou (2012). « Particle method: an efficient tool for direct numerical simulations of a high Schmidt number passive scalar in turbulent flow ». *Center for Turbulence Research Proceedings of the Summer Program 2012.* 1959.

Lagaert, J.-B., G. Balarac et G.-H. Cottet (2014). « Hybrid spectral-particle method for the turbulent transport of a passive scalar ». *Journal of Computational Physics* 260.0.

Leonard, A. (1980). « Vortex methods for flow simulation ». *Journal of Computational Physics* 37.3.

Lesieur, M. (2008). *Turbulence in fluids.*

Lesieur, M., O. Métais et P. Comte (2005). *Large-Eddy Simulations of Turbulence.*

Liu, M. B. et G. R. Liu (2010). « Smoothed Particle Hydrodynamics (SPH): an Overview and Recent Developments ». *Archives of Computational Methods in Engineering* 17.1.

Logg, A., K.-A. Mardal et G. Wells (2012). *Automated Solution of Differential Equations by the Finite Element Method.*

Magni, A. et G.-H. Cottet (2012). « Accurate, non-oscillatory, remeshing schemes for particle methods ». *Journal of Computational Physics* 231.1.

Mernik, M., J. Heering et A. M. Sloane (2005). « When and how to develop domain-specific languages ». *ACM Computing Surveys* 37.4.

Michioka, T. et F. K. Chow (2008). « High-Resolution Large-Eddy Simulations of Scalar Transport in Atmospheric Boundary Layer Flow over Complex Terrain ». *Journal of Applied Meteorology and Climatology* 47.12.

Monaghan, J. (1985). « Extrapolating B splines for interpolation ». *Journal of Computational Physics* 60.2.

Moore, G. E. (1998). « Cramming More Components onto Integrated Circuits ». *Proceedings of the IEEE*. Tome 86. 1.

Moore, G. E. (2005). *Excerpts from A Conversation with Gordon Moore : Moore's Law*.

Murtis, J., J. S. Elkinton et R. T. Cardé (1992). « Odor plumes and how insects use them ». *Annual review of entomology* 37.1.

Nvidia (2012a). *Kepler GK110*. Rapport technique.

Nvidia (2012b). *Tesla K20 GPU accelerator, board specification*. Rapport technique.

Nvidia (2014). *Cuda C programming guide*.

OpenACC (2013). *The OpenACC Application Programming Interface*.

Ould Salihi, M. L. (1998). « Couplage de méthodes numériques en simulation directe d'écoulements incompressibles ». Thèse de doctorat. Université Joseph Fourier.

Ould Salihi, M. L., G.-H. Cottet et M. El Hamraoui (2000). « Blending finite-difference and vortex methods for incompressible flow computations ». *SIAM Journal on Scientific Computing* 22.5.

Pérez, F., B. E. Granger et J. D. Hunter (2011). « Python: An Ecosystem for Scientific Computing ». *Computing in Science & Engineering* 13.2.

Pitsch, H., O. Desjardins, G. Balarac et M. Ihme (2008). « Large-eddy simulation of turbulent reacting flows ». *Progress in Aerospace Sciences* 44.6.

Rashed, G. et R. Ahsan (2012). « Python in computational science: applications and possibilities ». *International Journal of Computer Applications* 46.20.

Rees, W. M. van, A. Leonard, D. I. Pullin et P. Koumoutsakos (2011). « A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers ». *Journal of Computational Physics* 230.8.

Rodgers, D. P. (1985). « Improvements in Multiprocessor System Design ». *SIGARCH Computer Architecture News* 13.3.

Rossi, F., P. Londrillo, A. Sgattoni, S. Sinigardi et G. Turchetti (2012). « Towards robust algorithms for current deposition and dynamic load-balancing in a GPU particle in cell code ». *AIP Conference Proceedings*. Tome 184.

Rossinelli, D. et P. Koumoutsakos (2008). « Vortex methods for incompressible flow simulations on the GPU ». *The Visual Computer* 24.7-9.

Rossinelli, D., M. Bergdorf, G.-H. Cottet et P. Koumoutsakos (2010). « GPU accelerated simulations of bluff body flows using vortex particle methods ». *Journal of Computational Physics* 229.9.

Rossinelli, D., C. Conti et P. Koumoutsakos (2011). « Mesh-particle interpolations on graphics processing units and multicore central processing units. » *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 369.1944.

Rossinelli, D., B. Hejazialhosseini, P. Hadjidoukas, C. Bekas, A. Curioni, A. Bertsch, S. Futral, S. J. Schmidt, N. A. Adams, P. Koumoutsakos et L. Livermore (2013). « 11

PFLOP/s Simulations of Cloud Cavitation Collapse ». *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*.

Ruetsch, G. et P. Micikevicius (2009). *Optimizing Matrix Transpose in CUDA*. Rapport technique January. NVIDIA.

Sagaut, P. (2006). *Large Eddy Simulation for Incompressible Flows - An introduction*.

Sbalzarini, I. F., J. H. Walther, M. Bergdorf, S. E. Hieber, E. M. Kotsalis et P. Koumoutsakos (2006). « PPM – A highly efficient parallel particle–mesh library for the simulation of continuum systems ». *Journal of Computational Physics* 215.2.

Schoenberg, I. J. (1946). « Contributions to the problem of approximation of equidistant data by analytic functions ». *Quarterly of Applied Mathematics* 4.2.

Sethian, J. a. et P. Smereka (2003). « Level set methods for fluid interfaces ». *Annual Review of Fluid Mechanics* 35.1.

Shalf, J., S. Dosanjh et J. Morrison (2010). « Exascale Computing Technology Challenges ». *High Performance Computing for Computational Science–VECPAR 2010*.

Shraiman, B. I. et E. D. Siggia (2000). « Scalar turbulence ». *Nature* 405.6787.

Staniforth, A. et J. Côté (1991). « Semi-Lagrangian Integration Schemes for Atmospheric Models - A Review ». *Monthly weather review* 119.9.

Stantchev, G., W. Dorland et N. Gumerov (2008). « Fast parallel Particle-To-Grid interpolation for plasma PIC simulations on the GPU ». *Journal of Parallel and Distributed Computing* 68.10.

Subramaniam, B., W. Saunders, T. Scogland et W.-c. Feng (2013). « Trends in energy-efficient computing: A perspective from the Green500 ». *4th International Green Computing Conference*.

Taylor, J. R. et R. Stocker (2012). « Trade-offs of chemotactic foraging in turbulent water. » *Science* 338.6107.

Temam, R. (1968). « Une méthode d'approximation de la solution des équations de Navier-Stokes ». *Bulletin de la Société Mathématique de France* 96.

Top500 (2014). *http://www.top500.org/*.

Tryggvason, G., R. Scardovelli et S. Zaleski (2011). *Direct Numerical Simulations of Gas–Liquid Multiphase Flows*. Cambridge : Cambridge University Press.

Volkov, V. (2010). « Better performance at lower occupancy ». *Proceedings of the GPU Technology Conference,*

Wang, H., S. Potluri, M. Luo, A. K. Singh, S. Sur et D. K. Panda (2011). « MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters ». *Computer Science - Research and Development* 26.3-4.

Weynans, L. et A. Magni (2013). « Consistency, accuracy and entropic behaviour of remeshed particle methods ». *ESAIM: Mathematical Modelling and Numerical Analysis* 47.1.

Williams, S., A. Waterman et D. Patterson (2009). « Author : Roofline : An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures ». *Communications of the ACM* 52.4.

Yokota, R., L. A. Barba, T. Narumi et K. Yasuoka (2013). « Petascale turbulence simulation using a highly parallel fast multipole method on GPUs ». *Computer Physics Communications* 184.3.

A. Formules de remaillage de type $\Lambda_{p,r}$

Cette annexe décrit l'ensemble des formules de remaillage de type $\Lambda_{p,r}$ construites par la méthode décrite dans la section 2.2. Les caractéristiques des formules sont résumées dans le tableau A.1. Les formules sont regroupées en fonction de la largeur de leur support et pour chaque noyau, nous donnons leur expression. Les poids de remaillage sont également donnés et exprimés en fonction de la distance y de la particule au point de grille le plus proche à gauche.

A.1. Formules de type $\Lambda_{2,r}$

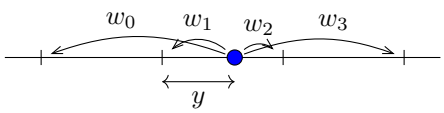


Figure A.1. – Poids de remaillage pour les formules de type $\Lambda_{2,r}$

Formule $\Lambda_{2,1}$

$$\Lambda_{2,1}(x) = M_4'(x) = \begin{cases} 1 - \frac{5}{2}|x|^2 + \frac{3}{2}|x|^3 & 0 \leqslant |x| < 1 \\ 2 - 4|x| + \frac{5}{2}|x|^2 - \frac{1}{2}|x|^3 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases} \tag{A.1}$$

```
w[0] = (y * (y * (-y + 2.) - 1.)) / 2.
w[1] = (y * y * (3. * y - 5.) + 2.) / 2.
w[2] = (y * (y * (-3. * y + 4.) + 1.)) / 2.
w[3] = (y * y * (y - 1.)) / 2.
```

Nom	Moment maximal conservé	Régularité	Support	Degré
$\Lambda_{2,1}$	2	C^1	$[-2;2]$	3
$\Lambda_{2,2}$	2	C^2	$[-2;2]$	5
$\Lambda_{2,3}$	2	C^3	$[-2;2]$	7
$\Lambda_{2,4}$	2	C^4	$[-2;2]$	9
$\Lambda_{4,2}$	4	C^2	$[-3;3]$	5
$\Lambda_{4,3}$	4	C^3	$[-3;3]$	7
$\Lambda_{4,4}$	4	C^4	$[-3;3]$	9
$\Lambda_{6,3}$	6	C^3	$[-4;4]$	7
$\Lambda_{6,4}$	6	C^4	$[-4;4]$	9
$\Lambda_{6,5}$	6	C^5	$[-4;4]$	11
$\Lambda_{6,6}$	6	C^6	$[-4;4]$	13
$\Lambda_{8,4}$	8	C^4	$[-5;5]$	9

Tableau A.1. – Comparatif des formules de remaillage de type $\Lambda_{p,r}$.

Formule $\Lambda_{2,2}$

$$\Lambda_{2,2}(x) = \begin{cases} 1 - |x|^2 - \frac{9}{2}|x|^3 + \frac{15}{2}|x|^4 - 3|x|^5 & 0 \leqslant |x| < 1 \\ -4 + 18|x| - 29|x|^2 + \frac{43}{2}|x|^3 - \frac{15}{2}|x|^4 + |x|^5 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases} \tag{A.2}$$

```
w[0] = (y * (y * (y * (y * (2. * y - 5.) + 3.) + 1.) - 1.)) / 2.
w[1] = (y * y * (y * (y * (-6. * y + 15.) - 9.) - 2.) + 2.) / 2.
w[2] = (y * (y * (y * (y * (6. * y - 15.) + 9.) + 1.) + 1.)) / 2.
w[3] = (y * y * y * (y * (-2. * y + 5.) - 3.)) / 2.
```

Formule $\Lambda_{2,3}$

$$\Lambda_{2,3}(x) = \begin{cases} 1 - |x|^2 - 15|x|^4 + \frac{75}{2}|x|^5 - \frac{63}{2}|x|^6 + 9|x|^7 & 0 \leqslant |x| < 1 \\ 32 - 168|x| + 376|x|^2 - 460|x|^3 + 330|x|^4 - \frac{277}{2}|x|^5 + \frac{63}{2}|x|^6 - 3|x|^7 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases} \tag{A.3}$$

```
w[0] = (y * (y * (y * y * (y * (y * (-6. * y + 21.) - 25.) + 10.) + 1.) - 1.)) / 2.
w[1] = (y * y * (y * y * (y * (y * (18. * y - 63.) + 75.) - 30.) - 2.) + 2.) / 2.
w[2] = (y * (y * (y * y * (y * (y * (-18. * y + 63.) - 75.) + 30.) + 1.) + 1.)) / 2.
w[3] = (y * y * y * y * (y * (y * (6. * y - 21.) + 25.) - 10.)) / 2.
```

Formule $\Lambda_{2,4}$

$$\Lambda_{2,4}(x) = \begin{cases} 1 - |x|^2 - \frac{105}{2}|x|^5 + \frac{357}{2}|x|^6 - 231|x|^7 + 135|x|^8 - 30|x|^9 & 0 \leqslant |x| < 1 \\ -208 + 1432|x| - 4304|x|^2 + 7420|x|^3 - 8085|x|^4 & \\ \qquad + \frac{11543}{2}|x|^5 - \frac{5397}{2}|x|^6 + 797|x|^7 - 135|x|^8 + 10|x|^9 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases} \tag{A.4}$$

```
w[0] = (y * (y * (y * y * y * (y * (y * (y * (20. * y - 90.) + 154.) - 119.) + 35.) + 1.) - 1.)) / 2.
w[1] = (y * y * (y * y * y * (y * (y * (y * (-60. * y + 270.) - 462.) + 357.) - 105.) - 2.) + 2.) / 2.
w[2] = (y * (y * (y * y * y * (y * (y * (y * (60. * y - 270.) + 462.) - 357.) + 105.) + 1.) + 1.)) / 2.
w[3] = (y * y * y * y * y * y * (y * (y * (y * (-20. * y + 90.) - 154.) + 119.) - 35.)) / 2.
```

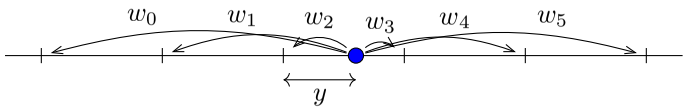


Figure A.2. – Poids de remaillage pour les formules de type $\Lambda_{4,r}$

A.2. Formules de type $\Lambda_{4,r}$

Formule $\Lambda_{4,2}$

$$\Lambda_{4,2}(x) = M_6'(x) = \begin{cases} 1 - \frac{5}{4}|x|^2 - \frac{35}{12}|x|^3 + \frac{21}{4}|x|^4 - \frac{25}{12}|x|^5 & 0 \leqslant |x| < 1 \\ -4 + \frac{75}{4}|x| - \frac{245}{8}|x|^2 + \frac{545}{24}|x|^3 - \frac{63}{8}|x|^4 + \frac{25}{24}|x|^5 & 1 \leqslant |x| < 2 \\ 18 - \frac{153}{4}|x| + \frac{255}{8}|x|^2 - \frac{313}{24}|x|^3 + \frac{21}{8}|x|^4 - \frac{5}{24}|x|^5 & 2 \leqslant |x| < 3 \\ 0 & 3 \leqslant |x| \end{cases} \tag{A.5}$$

```
w[0] = (y * (y * (y * (y * (-5. * y + 13.) - 9.) - 1.) + 2.)) / 24.
w[1] = (y * (y * (y * (y * (25. * y - 64.) + 39.) + 16.) - 16.)) / 24.
w[2] = (y * y * (y * (y * (-50. * y + 126.) - 70.) - 30.) + 24.) / 24.
w[3] = (y * (y * (y * (y * (50. * y - 124.) + 66.) + 16.) + 16.)) / 24.
w[4] = (y * (y * (y * (y * (-25. * y + 61.) - 33.) - 1.) - 2.)) / 24.
w[5] = (y * y * y * (y * (5. * y - 12.) + 7.)) / 24.
```

Formule $\Lambda_{4,3}$

$$\Lambda_{4,3}(x) = \begin{cases} 1 - \frac{5}{4}|x|^2 - \frac{28}{3}|x|^4 + \frac{145}{6}|x|^5 - \frac{245}{12}|x|^6 + \frac{35}{6}|x|^7 & 0 \leqslant |x| < 1 \\ 31 - \frac{1945}{12}|x| + \frac{2905}{8}|x|^2 - \frac{5345}{12}|x|^3 + \frac{1281}{4}|x|^4 - \frac{1615}{12}|x|^5 + \frac{245}{8}|x|^6 - \frac{35}{12}|x|^7 & 1 \leqslant |x| < 2 \\ -297 + \frac{3501}{4}|x| - \frac{8775}{8}|x|^2 + \frac{3029}{4}|x|^3 - \frac{3731}{12}|x|^4 + \frac{911}{12}|x|^5 - \frac{245}{24}|x|^6 + \frac{7}{12}|x|^7 & 2 \leqslant |x| < 3 \\ 0 & 3 \leqslant |x| \end{cases} \tag{A.6}$$

```
w[0] = (y * (y * (y * (y * (y * (y * (14. * y - 49.) + 58.) - 22.) - 2.) - 1.) + 2.)) / 24.
w[1] = (y * (y * (y * (y * (y * (y * (-70. * y + 245.) - 290.) + 111.) + 4.) + 16.) - 16.)) / 24.
w[2] = (y * y * (y * y * (y * (y * (140. * y - 490.) + 580.) - 224.) - 30.) + 24.) / 24.
w[3] = (y * (y * (y * (y * (y * (y * (-140. * y + 490.) - 580.) + 226.) - 4.) + 16.) + 16.)) / 24.
w[4] = (y * (y * (y * (y * (y * (y * (70. * y - 245.) + 290.) - 114.) + 2.) - 1.) - 2.)) / 24.
w[5] = (y * y * y * y * (y * (y * (-14. * y + 49.) - 58.) + 23.)) / 24.
```

Formule $\Lambda_{4,4}$

$$\Lambda_{4,4}(x) = \begin{cases} 1 - \frac{5}{4}|x|^2 + \frac{1}{4}|x|^4 - \frac{100}{3}|x|^5 + \frac{455}{4}|x|^6 - \frac{295}{2}|x|^7 + \frac{345}{4}|x|^8 - \frac{115}{6}|x|^9 & 0 \leqslant |x| < 1 \\ -199 + \frac{5485}{4}|x| - \frac{32975}{8}|x|^2 + \frac{28425}{4}|x|^3 - \frac{61953}{8}|x|^4 + \frac{33175}{6}|x|^5 & \\ \qquad - \frac{20685}{8}|x|^6 + \frac{3055}{4}|x|^7 - \frac{1035}{8}|x|^8 + \frac{115}{12}|x|^9 & 1 \leqslant |x| < 2 \\ 5913 - \frac{89235}{4}|x| + \frac{297585}{8}|x|^2 - \frac{143895}{4}|x|^3 + \frac{177871}{8}|x|^4 - \frac{54641}{6}|x|^5 & \\ \qquad + \frac{19775}{8}|x|^6 - \frac{1715}{4}|x|^7 + \frac{345}{8}|x|^8 - \frac{23}{12}|x|^9 & 2 \leqslant |x| < 3 \\ 0 & 3 \leqslant |x| \end{cases} \tag{A.7}$$

```
w[0] = (y * (y * (y * (y * (y * (y * (y * (y * (-46. * y + 207.) - 354.) + 273.) - 80.) + 1.) - 2.)
         - 1.) + 2.)) / 24.
w[1] = (y * (y * (y * (y * (y * (y * (y * (y * (230. * y - 1035.) + 1770.) - 1365.) + 400.) - 4.) + 4.)
         + 16.) - 16.)) / 24.
w[2] = (y * y * (y * y * (y * (y * (y * (y * (-460. * y + 2070.) - 3540.) + 2730.) - 800.) + 6.) - 30.)
```

```
        + 24.) / 24.
w[3] = (y * (y * (y * (y * (y * (y * (y * (y * (460. * y - 2070.) + 3540.) - 2730.) + 800.) - 4.) - 4.)
        + 16.) + 16.)) / 24.
w[4] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (-230. * y + 1035.) - 1770.) + 1365.) - 400.) + 1.) + 2.)
        - 1.) - 2.)) / 24.
w[5] = (y * y * y * y * y * (y * (y * (y * (46. * y - 207.) + 354.) - 273.) + 80.)) / 24.
```

A.3. Formules de type $\Lambda_{6,r}$

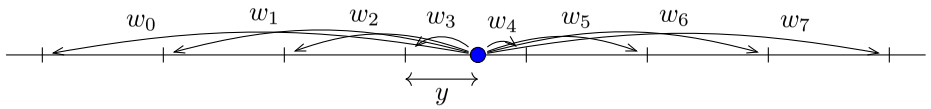


Figure A.3. – Poids de remaillage pour les formules de type $\Lambda_{6,r}$

Formule $\Lambda_{6,3}$

$$\Lambda_{6,3}(x) = \begin{cases} 1 - \frac{49}{36}|x|^2 - \frac{959}{144}|x|^4 + \frac{2569}{144}|x|^5 - \frac{727}{48}|x|^6 + \frac{623}{144}|x|^7 & 0 \leqslant |x| < 1 \\ \frac{138}{5} - \frac{8617}{60}|x| + \frac{12873}{40}|x|^2 - \frac{791}{2}|x|^3 + \frac{4557}{16}|x|^4 - \frac{9583}{80}|x|^5 + \frac{2181}{80}|x|^6 - \frac{623}{240}|x|^7 & 1 \leqslant |x| < 2 \\ -440 + \frac{25949}{20}|x| - \frac{117131}{72}|x|^2 + \frac{2247}{2}|x|^3 - \frac{66437}{144}|x|^4 + \frac{81109}{720}|x|^5 - \frac{727}{48}|x|^6 + \frac{623}{720}|x|^7 & 2 \leqslant |x| < 3 \\ \frac{3632}{5} - \frac{7456}{5}|x| + \frac{58786}{45}|x|^2 - 633|x|^3 + \frac{26383}{144}|x|^4 - \frac{22807}{720}|x|^5 + \frac{727}{240}|x|^6 - \frac{89}{720}|x|^7 & 3 \leqslant |x| < 4 \\ 0 & 4 \leqslant |x| \end{cases} \tag{A.8}$$

```
w[0] = (y * (y * (y * (y * (y * (y * (-89. * y + 312.) - 370.) + 140.) + 15.) + 4.) - 12.)) / 720.
w[1] = (y * (y * (y * (y * (y * (y * (623. * y - 2183.) + 2581.) - 955.) - 120.) - 54.) + 108.)) / 720.
w[2] = (y * (y * (y * (y * (y * (y * (-1869. * y + 6546.) - 7722.) + 2850.) + 195.) + 540.) - 540.)) / 720.
w[3] = (y * y * (y * y * (y * (y * (3115. * y - 10905.) + 12845.) - 4795.) - 980.) + 720.) / 720.
```

```
w[4] = (y * (y * (y * (y * (y * (y * (-3115. * y + 10900.) - 12830.) + 4880.) - 195.) + 540.) + 540.)) / 720.
w[5] = (y * (y * (y * (y * (y * (y * (1869. * y - 6537.) + 7695.) - 2985.) + 120.) - 54.) - 108.)) / 720.
w[6] = (y * (y * (y * (y * (y * (y * (-623. * y + 2178.) - 2566.) + 1010.) - 15.) + 4.) + 12.)) / 720.
w[7] = (y * y * y * y * (y * (y * (89. * y - 311.) + 367.) - 145.)) / 720.
```

Formule $\Lambda_{6,4}$

$$\Lambda_{6,4}(x) = \begin{cases} 1 - \frac{49}{36}|x|^2 + \frac{7}{18}|x|^4 - \frac{3521}{144}|x|^5 + \frac{12029}{144}|x|^6 - \frac{15617}{144}|x|^7 + \frac{1015}{16}|x|^8 - \frac{1015}{72}|x|^9 & 0 \leqslant |x| < 1 \\ -\frac{877}{5} + \frac{72583}{60}|x| - \frac{145467}{40}|x|^2 + \frac{18809}{3}|x|^3 - \frac{54663}{8}|x|^4 + \frac{390327}{80}|x|^5 & \\ \qquad -\frac{182549}{80}|x|^6 + \frac{161777}{240}|x|^7 - \frac{1827}{16}|x|^8 + \frac{203}{24}|x|^9 & 1 \leqslant |x| < 2 \\ 8695 - \frac{656131}{20}|x| + \frac{3938809}{72}|x|^2 - \frac{158725}{3}|x|^3 + \frac{2354569}{72}|x|^4 - \frac{9644621}{720}|x|^5 & \\ \qquad +\frac{523589}{144}|x|^6 - \frac{454097}{720}|x|^7 + \frac{1015}{16}|x|^8 - \frac{203}{72}|x|^9 & 2 \leqslant |x| < 3 \\ -\frac{142528}{5} + \frac{375344}{5}|x| - \frac{3942344}{45}|x|^2 + \frac{178394}{3}|x|^3 - \frac{931315}{36}|x|^4 + \frac{5385983}{720}|x|^5 & \\ \qquad -\frac{1035149}{720}|x|^6 + \frac{127511}{720}|x|^7 - \frac{203}{16}|x|^8 + \frac{29}{72}|x|^9 & 3 \leqslant |x| < 4 \\ 0 & 4 \leqslant |x| \end{cases} \tag{A.9}$$

```
w[0] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-1006. * y + 5533.) - 12285.) + 13785.) - 7829.)
         + 1803.) - 3.) - 5.) + 15.) + 4.) - 12.)) / 720.
w[1] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (7042. * y - 38731.) + 85995.) - 96495.) + 54803.)
         - 12620.) + 12.) + 60.) - 120.) - 54.) + 108.)) / 720.
w[2] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-21126. * y + 116193.) - 257985.) + 289485.)
         - 164409.) + 37857.) - 15.) - 195.) + 195.) + 540.) - 540.)) / 720.
w[3] = (y * y * (y * y * (y * y * (y * (y * (y * (y * (35210. * y - 193655.) + 429975.) - 482475.) + 274015.)
         - 63090.) + 280.) - 980.) + 720.) / 720.
w[4] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-35210. * y + 193655.) - 429975.) + 482475.)
         - 274015.) + 63085.) + 15.) - 195.) - 195.) + 540.) + 540.)) / 720.
w[5] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (21126. * y - 116193.) + 257985.) - 289485.)
         + 164409.) - 37848.) - 12.) + 60.) + 120.) - 54.) - 108.)) / 720.
w[6] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-7042. * y + 38731.) - 85995.) + 96495.) - 54803.)
```

```
        + 12615.) + 3.) - 5.) - 15.) + 4.) + 12.)) / 720.
w[7] = (y * y * y * y * y * y * (y * (y * (y * (y * (1006. * y - 5533.) + 12285.) - 13785.) + 7829.)
        - 1802.)) / 720.
```

Formule $\Lambda_{6,5}$

$$\Lambda_{6,5}(x) = \begin{cases} 1 - \frac{49}{36}|x|^2 + \frac{7}{18}|x|^4 - \frac{701}{8}|x|^6 + \frac{54803}{144}|x|^7 - \frac{32165}{48}|x|^8 + \frac{9555}{16}|x|^9 - \frac{38731}{144}|x|^{10} + \frac{3521}{72}|x|^{11} & 0 \leqslant |x| < 1 \\ 1233 - \frac{617533}{60}|x| + \frac{1544613}{40}|x|^2 - \frac{515179}{6}|x|^3 + \frac{502579}{4}|x|^4 - \frac{3809911}{30}|x|^5 & \\ \qquad + \frac{3618099}{40}|x|^6 - \frac{10894163}{240}|x|^7 + \frac{251685}{16}|x|^8 - \frac{172123}{48}|x|^9 + \frac{38731}{80}|x|^{10} - \frac{3521}{120}|x|^{11} & 1 \leqslant |x| < 2 \\ -181439 + \frac{16709441}{20}|x| - \frac{125352311}{72}|x|^2 + \frac{13002493}{6}|x|^3 - \frac{64445353}{36}|x|^4 + \frac{30912301}{30}|x|^5 & \\ \qquad - \frac{3373567}{8}|x|^6 + \frac{88345523}{720}|x|^7 - \frac{1194095}{48}|x|^8 + \frac{160657}{48}|x|^9 - \frac{38731}{144}|x|^{10} + \frac{3521}{360}|x|^{11} & 2 \leqslant |x| < 3 \\ 1188352 - \frac{19108864}{5}|x| + \frac{250837216}{45}|x|^2 - \frac{14600752}{3}|x|^3 + \frac{25437902}{9}|x|^4 - \frac{17195278}{15}|x|^5 & \\ \qquad + \frac{13253241}{40}|x|^6 - \frac{49136309}{720}|x|^7 + \frac{471205}{48}|x|^8 - \frac{45083}{48}|x|^9 + \frac{38731}{720}|x|^{10} - \frac{503}{360}|x|^{11} & 3 \leqslant |x| < 4 \\ 0 & 4 \leqslant |x| \end{cases} \tag{A.10}$$

```
w[0] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-1006. * y + 5533.) - 12285.) + 13785.) - 7829.)
        + 1803.) - 3.) - 5.) + 15.) + 4.) - 12.)) / 720.
w[1] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (7042. * y - 38731.) + 85995.) - 96495.) + 54803.)
        - 12620.) + 12.) + 60.) - 120.) - 54.) + 108.)) / 720.
w[2] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-21126. * y + 116193.) - 257985.) + 289485.)
        - 164409.) + 37857.) - 15.) - 195.) + 195.) + 540.) - 540.)) / 720.
w[3] = (y * y * (y * y * (y * y * (y * (y * (y * (y * (35210. * y - 193655.) + 429975.) - 482475.) + 274015.)
        - 63090.) + 280.) - 980.) + 720.) / 720.
w[4] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-35210. * y + 193655.) - 429975.) + 482475.)
        - 274015.) + 63085.) + 15.) - 195.) - 195.) + 540.) + 540.)) / 720.
w[5] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (21126. * y - 116193.) + 257985.) - 289485.)
        + 164409.) - 37848.) - 12.) + 60.) + 120.) - 54.) - 108.)) / 720.
w[6] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-7042. * y + 38731.) - 85995.) + 96495.) - 54803.)
```

```
        + 12615.) + 3.) - 5.) - 15.) + 4.) + 12.)) / 720.
w[7] = (y * y * y * y * y * y * (y * (y * (y * (y * (1006. * y - 5533.) + 12285.) - 13785.) + 7829.)
        - 1802.)) / 720.
```

Formule $\Lambda_{6,6}$

$$\Lambda_{6,6}(x) = \begin{cases} 1 - \frac{49}{36}|x|^2 + \frac{7}{18}|x|^4 - \frac{1}{36}|x|^6 - \frac{46109}{144}|x|^7 + \frac{81361}{48}|x|^8 - \frac{544705}{144}|x|^9 + \frac{655039}{144}|x|^{10} \\ \qquad - \frac{223531}{72}|x|^{11} + \frac{81991}{72}|x|^{12} - \frac{6307}{36}|x|^{13} & 0 \leqslant |x| < 1 \\ -\frac{44291}{5} + \frac{1745121}{20}|x| - \frac{15711339}{40}|x|^2 + \frac{32087377}{30}|x|^3 - \frac{7860503}{4}|x|^4 + \frac{38576524}{15}|x|^5 \\ \qquad - \frac{24659323}{10}|x|^6 + \frac{84181657}{48}|x|^7 - \frac{74009313}{80}|x|^8 + \frac{17159513}{48}|x|^9 - \frac{7870247}{80}|x|^{10} \\ \qquad + \frac{438263}{24}|x|^{11} - \frac{81991}{40}|x|^{12} + \frac{6307}{60}|x|^{13} & 1 \leqslant |x| < 2 \\ 3905497 - \frac{424679647}{20}|x| + \frac{3822627865}{72}|x|^2 - \frac{2424839767}{30}|x|^3 + \frac{3009271097}{36}|x|^4 - \frac{930168127}{15}|x|^5 \\ \qquad + \frac{305535494}{9}|x|^6 - \frac{9998313437}{720}|x|^7 + \frac{203720335}{48}|x|^8 - \frac{137843153}{144}|x|^9 + \frac{22300663}{144}|x|^{10} \\ \qquad - \frac{6126883}{360}|x|^{11} + \frac{81991}{72}|x|^{12} - \frac{6307}{180}|x|^{13} & 2 \leqslant |x| < 3 \\ -\frac{255622144}{5} + \frac{971097344}{5}|x| - \frac{15295867328}{45}|x|^2 + \frac{5442932656}{15}|x|^3 - \frac{2372571796}{9}|x|^4 + \frac{2064517469}{15}|x|^5 \\ \qquad - \frac{9563054381}{180}|x|^6 + \frac{2210666335}{144}|x|^7 - \frac{796980541}{240}|x|^8 + \frac{76474979}{144}|x|^9 - \frac{43946287}{720}|x|^{10} \\ \qquad + \frac{343721}{72}|x|^{11} - \frac{81991}{360}|x|^{12} + \frac{901}{180}|x|^{13} & 3 \leqslant |x| < 4 \\ 0 & 4 \leqslant |x| \end{cases} \tag{A.11}$$

```
w[0] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (3604. * y - 23426.) + 63866.)
        - 93577.) + 77815.) - 34869.) + 6587.) + 1.) - 3.) - 5.) + 15.) + 4.) - 12.)) / 720.
w[1] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-25228. * y + 163982.) - 447062.)
        + 655039.) - 544705.) + 244083.) - 46109.) - 6.) + 12.) + 60.) - 120.) - 54.) + 108.)) / 720.
w[2] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (75684. * y - 491946.) + 1341186.)
        - 1965117.) + 1634115.) - 732249.) + 138327.) + 15.) - 15.) - 195.) + 195.) + 540.) - 540.)) / 720.
w[3] = (y * y * (y * y * (y * y * (y * (y * (y * (y * (y * (y * (y * (-126140. * y + 819910.) - 2235310.)
        + 3275195.) - 2723525.) + 1220415.) - 230545.) - 20.) + 280.) - 980.) + 720.) / 720.
```

```
w[4] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (126140. * y - 819910.) + 2235310.)
         - 3275195.) + 2723525.) - 1220415.) + 230545.) + 15.) + 15.) - 195.) - 195.) + 540.) + 540.)) / 720.
w[5] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (-75684. * y + 491946.) - 1341186.)
         + 1965117.) - 1634115.) + 732249.) - 138327.) - 6.) - 12.) + 60.) + 120.) - 54.) - 108.)) / 720.
w[6] = (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (y * (25228. * y - 163982.) + 447062.)
         - 655039.) + 544705.) - 244083.) + 46109.) + 1.) + 3.) - 5.) - 15.) + 4.) + 12.)) / 720.
w[7] = (y * y * y * y * y * y * y * (y * (y * (y * (y * (y * (y * (-3604. * y + 23426.) - 63866.) + 93577.)
         - 77815.) + 34869.) - 6587.)) / 720.
```

A.4. Formules de type $\Lambda_{8,r}$

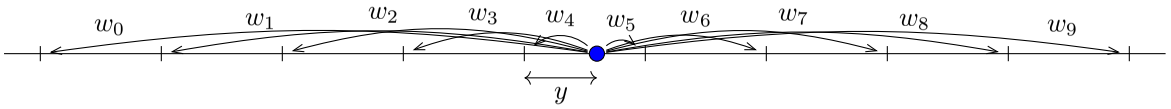


Figure A.4. – Poids de remaillage pour les formules de type $\Lambda_{8,r}$

Formule $\Lambda_{8,4}$

$$\Lambda_{8,4}(x) = \begin{cases} 1 - \frac{205}{144}|x|^2 + \frac{91}{192}|x|^4 - \frac{6181}{320}|x|^5 + \frac{6337}{96}|x|^6 - \frac{2745}{32}|x|^7 + \frac{28909}{576}|x|^8 - \frac{3569}{320}|x|^9 & 0 \leqslant |x| < 1 \\ -154 + \frac{12757}{12}|x| - \frac{230123}{72}|x|^2 + \frac{264481}{48}|x|^3 - \frac{576499}{96}|x|^4 + \frac{686147}{160}|x|^5 & \\ \qquad - \frac{96277}{48}|x|^6 + \frac{14221}{24}|x|^7 - \frac{28909}{288}|x|^8 + \frac{3569}{480}|x|^9 & 1 \leqslant |x| < 2 \\ \frac{68776}{7} - \frac{1038011}{28}|x| + \frac{31157515}{504}|x|^2 - \frac{956669}{16}|x|^3 + \frac{3548009}{96}|x|^4 - \frac{2422263}{160}|x|^5 & \\ \qquad + \frac{197255}{48}|x|^6 - \frac{19959}{28}|x|^7 + \frac{144545}{2016}|x|^8 - \frac{3569}{1120}|x|^9 & 2 \leqslant |x| < 3 \\ -56375 + \frac{8314091}{56}|x| - \frac{49901303}{288}|x|^2 + \frac{3763529}{32}|x|^3 - \frac{19648027}{384}|x|^4 + \frac{9469163}{640}|x|^5 & \\ \qquad - \frac{545977}{192}|x|^6 + \frac{156927}{448}|x|^7 - \frac{28909}{1152}|x|^8 + \frac{3569}{4480}|x|^9 & 3 \leqslant |x| < 4 \\ \frac{439375}{7} - \frac{64188125}{504}|x| + \frac{231125375}{2016}|x|^2 - \frac{17306975}{288}|x|^3 + \frac{7761805}{384}|x|^4 - \frac{2895587}{640}|x|^5 & \\ \qquad + \frac{129391}{192}|x|^6 - \frac{259715}{4032}|x|^7 + \frac{28909}{8064}|x|^8 - \frac{3569}{40320}|x|^9 & 4 \leqslant |x| < 5 \\ 0 & 5 \leqslant |x| \end{cases} \tag{A.12}$$

```
w0 = (y * (y * (y * (y * (y * (y * (y * (y * (-3569. * y + 16061.) - 27454.) + 21126.) - 6125.)
         + 49.) - 196.) - 36.) + 144.)) / 40320.
w1 = (y * (y * (y * (y * (y * (y * (y * (y * (y * (32121. * y - 144548.) + 247074.) - 190092.)
         + 55125.) - 672.) + 2016.) + 512.) - 1536.)) / 40320.
w2 = (y * (y * (y * (y * (y * (y * (y * (y * (y * (-128484. * y + 578188.) - 988256.) + 760312.)
         - 221060.) + 4732.) - 9464.) - 4032.) + 8064.)) / 40320.
w3 = (y * (y * (y * (y * (y * (y * (y * (y * (y * (299796. * y - 1349096.) + 2305856.) - 1774136.)
         + 517580.) - 13664.) + 13664.) + 32256.) - 32256.)) / 40320.
w4 = (y * y * (y * y * (y * (y * (y * (y * (-449694. * y + 2023630.) - 3458700.) + 2661540.)
         - 778806.) + 19110.) - 57400.) + 40320.) / 40320.
w5 = (y * (y * (y * (y * (y * (y * (y * (y * (y * (449694. * y - 2023616.) + 3458644.) - 2662016.)
         + 780430.) - 13664.) - 13664.) + 32256.) + 32256.)) / 40320.
w6 = (y * (y * (y * (y * (y * (y * (y * (y * (y * (-299796. * y + 1349068.) - 2305744.) + 1775032.)
         - 520660.) + 4732.) + 9464.) - 4032.) - 8064.)) / 40320.
w7 = (y * (y * (y * (y * (y * (y * (y * (y * (y * (128484. * y - 578168.) + 988176.) - 760872.)
         + 223020.) - 672.) - 2016.) + 512.) + 1536.)) / 40320.
```

```
w8 = (y * (y * (y * (y * (y * (y * (y * (y * (-32121. * y + 144541.) - 247046.) + 190246.)
         - 55685.) + 49.) + 196.) - 36.) - 144.)) / 40320.
w9 = (y * y * y * y * y * (y * (y * (y * (3569. * y - 16060.) + 27450.) - 21140.) + 6181.)) / 40320.
```

B. Publications

ESAIM: M2AN 48 (2014) 1029–1060
DOI: 10.1051/m2an/2014009

ESAIM: Mathematical Modelling and Numerical Analysis
www.esaim-m2an.org

HIGH ORDER SEMI-LAGRANGIAN PARTICLE METHODS FOR TRANSPORT EQUATIONS: NUMERICAL ANALYSIS AND IMPLEMENTATION ISSUES

G.-H. Cottet[1], J.-M. Etancelin[1], F. Perignon[1] and C. Picard[1]

Abstract. This paper is devoted to the definition, analysis and implementation of semi-Lagrangian methods as they result from particle methods combined with remeshing. We give a complete consistency analysis of these methods, based on the regularity and momentum properties of the remeshing kernels, and a stability analysis of a large class of second and fourth order methods. This analysis is supplemented by numerical illustrations. We also describe a general approach to implement these methods in the context of hybrid computing and investigate their performance on GPU processors as a function of their order of accuracy.

Mathematics Subject Classification. 65M12, 65M75, 65Y05, 65Y20.

Received October 22, 2013. Revised January 28, 2014.
Published online June 30, 2014.

1. Introduction

Particle methods are Lagrangian methods that have been designed for advection dominated problems, with applications mostly in plasma physics [10], incompressible flows [4,7,16], or gas dynamics [20]. Following [12,13], particle methods are often associated with remeshing, in order to maintain the regularity of the particle distribution and, as a consequence, to control the accuracy of the method in presence of strong strain in the flow carrying the particles. Remeshing removes the grid free nature of particle methods but maintains some of its features like conservation, locality, and stability. It not only guarantees accuracy but also allows to combine in a seamless fashion particle methods with grid-based techniques, like domain decomposition methods and fast FFT-based Poisson solvers for field evaluations [5,22], multi-resolution and adaptive mesh refinement [1,2].

In practice particle remeshing occurs every *few* time steps. In that case, and as long as the remeshing frequency does not increase when the discretization parameters tend to zero, Remeshed Particle Methods (RPM in the sequel) can be viewed as *conservative forward semi-lagrangian* methods. Classical semi-lagrangian methods for transport equations combine tracking of trajectories originating at grid points and interpolation from the grid. Because they operate on local point values, semi-lagrangian methods in general do not conserve mass, unless some specific treatment is done. By contrast, particle methods transport masses which makes them inherently conservative.

Keywords and phrases. Advection equations, particle methods, semi-Lagrangian methods, GPU computing.

[1] Université Grenoble Alpes and CNRS, Laboratoire Jean Kuntzmann, BP 53 38041, Grenoble Cedex 9, France.
georges-henri.cottet@imag.fr

Article published by EDP Sciences
© EDP Sciences, SMAI 2014

Forward Semi Lagrangian methods have already been designed and used with success in plasma physics [9] and geophysical flows [6]. Semi Lagrangian particles, as they come out of RPM, have some distinctive features. the projection on the grid is explicit, based on explicit remeshing kernels, and does not require to solve a linear system to recover grid values. The computational effort can thus be restricted to the support of the solution and efficient parallel algorithms can be implemented. Moreover, remeshed particle methods can be derived at any order in a systematic fashion, whereas the methods derived in [6, 9] seem to be restricted to first order in space [23]. One goal of this paper is indeed to derive remeshed particle methods of arbitrary order, independently of any CFL condition, and to give a complete numerical analysis of a large class of second and fourth order methods.

RPM only involve local operations and are therefore well adapted to parallel implementations. In [24] an implementation of remeshed particle methods on GPU processors was described for the two-dimensional Navier–Stokes equations with penalization to account for solid obstacles. In the present paper, following [18], we consider implementations of two or three-dimensional RPM combined with directional splitting, that is where particles are pushed and remeshed successively in the three directions. Directional splitting allows to use high order remeshing kernels, with large stencils, at a minimal cost. We in particular discuss how directional splitting impacts the memory management to optimize implementations of RPM on GPU.

The outline of this paper is as follows. In Section 2 we give the definition of RPM, recall how remeshing kernels are derived and produce a list of high order kernels. In Section 3 we give the numerical analysis of RPM, based on the regularity and moment properties of the remeshing kernels. In Section 4 we present numerical illustrations and refinement studies of RPM in one to three dimensions. Section 5 is devoted to our implementation of RPM on GPU, in the context of a software library devoted to hybrid computations where different solvers can be implemented on different platforms. Finally Section 6 is devoted to concluding remarks and we give in the appendix the analytical formulas of the remeshing kernels that are considered in the paper.

2. Remeshed particle methods

In all the sequel we consider advection equations in the following conservation form

$$\frac{\partial u}{\partial t} + \text{div } (au) = 0, x \in \mathbb{R}^d, t > 0, \tag{2.1}$$

where a is a given smooth velocity field.

2.1. Definition

In this paper we consider the case of uniform grids (we refer to [1, 2] for the design and applications of adaptive RPM with variable grid-size). RPM consist of alternating particle motion and remeshing. Particles are at the beginning of every time-step on a regular grid $x_i = i\Delta x$, $i \in \mathbb{Z}^d$, then move with the following equation

$$x_i^{n+1} = x_i + \tilde{a}_i^n \Delta t. \tag{2.2}$$

In the above equation $\tilde{a}_i^n$ denotes an evaluation of the velocity field at time $t_n = n\Delta t$ and location x_i which depends on the chosen time-stepping scheme. Remeshing follows, through interpolation with a remeshing kernel Γ which satisfies $\Gamma(-x) = \Gamma(x)$. If u_i^n denotes the value approximating $u(x_i, t_n)$, this gives the following formula:

$$u_i^{n+1} = \sum_j u_j^n \Gamma\left(\frac{x_j^{n+1} - x_i}{\Delta x}\right), i \in \mathbb{Z}^d, n \geqslant 0. \tag{2.3}$$

In traditional remeshing schemes for multidimensional problems, multidimensional kernels Γ are derived as tensor products of 1D formulas. In the present study we follow the approach initiated in [18] which consists of reducing the advection problem (2.1) into one-dimensional advection equations through directional splitting. One the one hand, this approach has the drawback that increasing the order in time of the method is less

straightforward than for the original multidimensional problem. On the other hand, it offers the advantage to significantly reduce the computational cost of the method if high order kernels, with large stencils, are used. In short, if the kernel Γ involves a stencil with N_s grid points, in three dimensions a multidimensional remeshing method will cost $O(N_s^3)$ per particle, while a directional splitting will cost $O(3N_s)$ operations. For values of $N_s \gtrsim 6$, a typical case we will consider in the sequel, the ratio is larger than 10. Directional splitting also allows to derive limiting techniques to reduce oscillations [18] and, for the numerical analysis of the method, one only needs to consider the case $d = 1$.

2.2. Derivation of remeshing kernels

We consider the equation (2.1) and its approximation formulas (2.2), (2.3) for $d = 1$. As will be clear from the numerical analysis below, one key feature which controls the accuracy of RPM is the conservation of momentum up to a given order p. More precisely, one seeks to satisfy for the scheme (2.3) the following identity

$$\sum_i u_i^{n+1} x_i^\alpha = \sum_i u_i^n x_i^\alpha, 0 \le \alpha \le p, \tag{2.4}$$

for all sequences (u_i^n). It is readily seen that this is equivalent to the following moments conditions for the remeshing kernel Γ

$$\sum_{k\in\mathbb{Z}} k^\alpha \Gamma(x-k) = x^\alpha,\ 0 \le \alpha \le p,\ x \in \mathbb{R} \tag{2.5}$$

or, equivalently,

$$\sum_{k\in\mathbb{Z}} (x-k)^\alpha \Gamma(x-k) = \begin{cases} 1 & \text{if } \alpha = 0 \\ 0 & \text{if } 1 \le \alpha \le p \end{cases},\ x \in \mathbb{R}. \tag{2.6}$$

Note that for $\alpha = 0$ these conditions enforce the conservation of mass. Using these identities for a given value of p and assuming that the kernel Γ remeshes particle weights among the $p+1$ nearest grid points, one obtains a piecewise polynomial function of degree p. For $p = 1$ this gives the piecewise linear tent function. With $p = 2$ one obtains a piecewise quadratic function, the so-called Λ_2 formula, that has been used with success in the first particle simulation of the Navier–Stokes equations using remeshing in a systematic fashion [13].

This method to derive kernels is straightforward but has the drawback that it does not deliver smooth kernels. The kernel Λ_2 is not even continuous (this is the case more generally for kernels corresponding to even values of p; for odd values, the kernels are continuous but their derivatives are discontinuous). This lack of smoothness may result in a loss of accuracy or in oscillations, in particular if large time-steps are used. In [18] local correction techniques were derived to guarantee at least first order for the kernel Λ_2 and third order for the analogous kernel Λ_4 corresponding to $p = 4$.

To derive smooth kernels, one option is to use extrapolation techniques starting from smooth B-splines. If we denote by M_1 the top-hat filter with support in $[-1/2, +1/2]$, and by M_n its successive convolution: $M_n = M_1^{(*n)} \in W^{n,\infty}(\mathbb{R})$, the idea is to derive linear combinations of M_n, xM_n', $x^2 M_n''$,... to cancel the successive continuous moments of Γ. More precisely, given an even integer $p \ge 2$ and $k > 1 + p/2$, one looks for coefficients $\alpha_1, \ldots, \alpha_{p/2+1}$ such that $\Gamma = \sum_{l=0}^{p/2} \alpha_{l+1} x^l M_k^{(l)}$ satisfies

$$\int y^\alpha \Gamma(y)\,\mathrm{d}y = \begin{cases} 1 & \text{if } \alpha = 0, \\ 0 & \text{if } 1 \le \alpha \le p. \end{cases} \tag{2.7}$$

For symmetry reasons these conditions need only be enforced for even values of α. This leads to a square linear system of size $1 + p/2$ the coefficients of which only involve the even moments of M_k. With this method, one for instance readily obtains the following kernels

$$\Gamma = \frac{1}{2}(3M_4 + xM_4'), \tag{2.8}$$

and

$$\Gamma = \frac{1}{8}(15M_8 + 9xM_8' + x^2M_8''). \tag{2.9}$$

The first formula corresponds to a kernel of class C^1, piecewise cubic, with a support of size 4 and $p = 2$. It was derived under the name of M_4' in [19] and has been and still is extensively used in particle simulations, in particular of vortex flows. The second formula corresponds to a kernel of class C^4, piecewise polynomial of degree 7, with a support of size 8 and $p = 4$.

The moment conditions (2.7) concern *continuous* moments of the kernel. To check that the *discrete* moment conditions (2.6) are satisfied as well, one can use an equivalent condition using the Fourier transform of the kernel [7, 29]. If

$$\widehat{\Gamma}(\xi) = \int \Gamma(y)\mathrm{e}^{-\mathrm{i}\xi y}\,\mathrm{d}y,$$

it can be shown ([7]) that the properties (2.5) are satisfied provided $\widehat{\Gamma}$ fulfills the following conditions:

$$\widehat{\Gamma}(\xi) - 1 \text{ has a zero of order } p+1 \text{ at } \xi = 0$$

$$\widehat{\Gamma}(\xi) \text{ has a zero of order } p+1 \text{ at all } \xi = 2\pi m, m \neq 0.$$

Since $\widehat{\Gamma}^{(\alpha)}$ is proportional to $y^\alpha\Gamma(y)$, the first condition above is clearly equivalent to the conditions (2.7). Moreover one has

$$\widehat{M_1}(\xi) = \frac{\sin(\xi/2)}{\xi/2}$$

and thus

$$\widehat{M_n}(\xi) = \left[\frac{\sin(\xi/2)}{\xi/2}\right]^n.$$

As a result, the Fourier transform of the functions $x^l M_k^{(l)}$ has a zero of order $k-l$ at all non zero multiple of π. From these observations, one can easily check that the kernels (2.8), and (2.9) do satisfy the discrete moment conditions (2.5) with $p = 2$ and $p = 4$ respectively.

The approach just presented leads to smooth and high order kernels. However the kernels derived in this fashion do not necessarily satisfy the following interpolation property

$$\Gamma(i) = \begin{cases} 1 \text{ if } i = 0, \\ 0 \text{ otherwise.} \end{cases} \tag{2.10}$$

Indeed the so-called kernel M_4' given in (2.8) does satisfy this condition, but not the one derived in (2.9) from M_8. Property (2.10) is natural as it ensures that, if the velocity is zero, the exact solution is algebraically conserved. Although this property does not enter the numerical analysis that follows, in practice it seems to have some importance, in particular to represent accurately the smallest scales in turbulent flows.

One can derive kernels which satisfy simultaneously and to any given order, regularity, moment conditions and the interpolation property. For a sake of simplicity, in the following we restrict ourselves to kernels with a stencil covering an even number $2M_s$ of grid points. We seek kernels Γ that have the following properties:

P1. Γ has support in $[-M_s, +M_s]$,
P2. Γ is even and piecewise polynomial of degree M in intervals of the form $[i, i+1]$,
P3. Γ is of class C^r,
P4. Γ satisfies the moment properties (2.5) for a given value of p,
P5. Γ satisfies the interpolation property (2.10).

TABLE 1. Kernels of various regularity, moment properties and complexity. In bold, the kernels that are considered in the numerical experiments of Section 4 and for which analytical formulas are given in the appendix.

	Moments (p in 2.5)	Regularity	Nb of grid points in stencil	degree	Support
$\boldsymbol{\Lambda_{2,1}}$	2	C^1	4	3	$[-2;2]$
$\boldsymbol{\Lambda_{2,2}}$	2	C^2	4	5	$[-2;2]$
$\Lambda_{2,3}$	2	C^3	4	7	$[-2;2]$
$\Lambda_{2,4}$	2	C^4	4	9	$[-2;2]$
$\boldsymbol{\Lambda_{4,2}}$	4	C^2	6	5	$[-3;3]$
$\Lambda_{4,3}$	4	C^3	6	7	$[-3;3]$
$\boldsymbol{\Lambda_{4,4}}$	4	C^4	6	9	$[-3;3]$
$\Lambda_{6,3}$	6	C^3	8	7	$[-4;4]$
$\boldsymbol{\Lambda_{6,4}}$	6	C^4	8	9	$[-4;4]$
$\Lambda_{6,5}$	6	C^5	8	11	$[-4;4]$
$\boldsymbol{\Lambda_{6,6}}$	6	C^6	8	13	$[-4;4]$
$\boldsymbol{\Lambda_{8,4}}$	8	C^4	10	9	$[-5;5]$

Such kernels are determined by $M_s(M+1)$ coefficients. The regularity property P3 imposes $M_s(r+1)$ interface conditions at integer values, and $[(r+1)/2]$ conditions to express that derivatives of odd order vanish at zero. The properties P4 and P5 impose $p+1+M_s$ conditions. One reasonable constraint under which one can expect to find kernels satisfying these conditions is therefore

$$M_s(M+1) \geq (r+1)M_s + [(r+1)/2] + p + 1 + M_s.$$

Table 1 lists several kernels that have been obtained through this approach by symbolic calculations. In this table, the kernels have been labelled by 2 indices that refer to the regularity and the order to which moment conditions are satisfied, as we will see that these are the parameters which control the order of accuracy of the RPM: $\Lambda_{p,r}$ is a kernel in $W^{r+1,\infty}(\mathbb{R})$ which satisfies (2.5). $\Lambda_{2,1}$ corresponds to the kernel M_4' already mentioned. The kernel $\Lambda_{4,2}$ was derived with this approach and used for the first time in [2] under the name of M_6'. For a sake of completeness, we have provided in the appendix the analytical formulas for the kernels which are considered in the numerical experiments of Section 4: $\Lambda_{2,1}$, $\Lambda_{2,2}$, $\Lambda_{4,2}$, $\Lambda_{4,4}$, $\Lambda_{6,4}$, $\Lambda_{6,6}$ and $\Lambda_{8,4}$.

3. Numerical analysis

We consider in this section RPM with kernels satisfying the moment properties (2.5) and the following regularity conditions:

$$\Gamma \in W^{r+1,\infty}(\mathbb{R}) \text{ and } \Gamma \in \mathcal{C}^\infty\left(]l, l+1[\right), \, l \in \mathbb{Z}. \tag{3.1}$$

The RPM is defined by the formulas (2.2), (2.3) and we will denote by $\mathcal{T}_i u(\cdot, t_n)$ the result of the scheme (2.3), at the grid point x_i, starting from grid values $u(x_j, t_n)$.

In this section we are interested by the stability and spatial accuracy of the method. For a sake of simplicity we will therefore assume that a does not depend on time and that particles advance with an explicit first-order Euler scheme. In this case we simply have $\tilde{a}_j^n = a(x_j)$.

A striking feature of RPM, common with all semi-lagrangian methods, is that their stability does not rely on CFL conditions. In this section we prove stability and consistency results under the condition

$$\Delta t \leq \frac{M}{\|a'\|_{L^\infty}} \tag{3.2}$$

for a given constant $M < 1$. This condition in particular ensures that particle trajectories cannot intersect. The constant M is often called Lagrangian CFL number.

3.1. Consistency

We will prove the following consistency result

Proposition 3.1. *Assume that the condition (3.2) is satisfied, and that the moment and regularity conditions (2.5), (3.1) hold for some $r, p > 1$. Let $T > 0$ and assume further that a and the solution u to equation (2.1) belong to $L^\infty\left(0, T; W^{r+1,\infty}(\mathbb{R})\right)$. Then, if we set $\beta = \inf(r, p)$, the following estimate holds*

$$u\left(x_i, t^{n+1}\right) = \mathcal{T}_i\left(u\left(\cdot, t^n\right)\right) + O\left(\Delta t^2\right) + O\left(\Delta t \Delta x^\beta + \Delta x^{\beta+1} + \Delta t^{\beta+1}\right). \tag{3.3}$$

Moreover if every cell of size Δx contains exactly one particle after an advection step, then $\beta = p$.

Proof. If we use the notation u_j^n for $u(x_j, t_n)$ and set $j = i + k$ we can write the following expansion

$$u_j^n = \sum_{l=0}^{\beta} \frac{k^l \Delta x^l}{l!} u^{(l)}(x_i) + O\left(\Delta x^{\beta+1}\right), \tag{3.4}$$

where $u^{(l)}(x_i)$ stands for $\partial^l u / \partial x^l(x_i, t_n)$. Next, one can write $x_j^{n+1} = x_i + k\Delta x + a_{i+k}\Delta t$ and therefore

$$\Gamma\left(\frac{x_j^{n+1} - x_i}{\Delta x}\right) = \Gamma(k + \lambda_{i+k}),$$

where λ_j is the local CFL number: $\lambda_j = a_j \Delta t / \Delta x$. We then write $\lambda_j = \lambda_i + (a_j - a_i)\Delta t / \Delta x$ and use the following formula

$$(f \circ g)^{(m)}(x) = \sum_{|q|=1}^{m} c_q f^{q_1 + \ldots + q_m}(g(x)) \prod_{s=1}^{m} \left(g^{(s)}(x)\right)^{q_s}$$

where $q = (q_1, \ldots, q_m)$ is a multi-index, $|q| = q_1 + 2q_2 + \ldots + mq_m$ and c_q are positive coefficients, to obtain

$$\begin{aligned}\Gamma\left(\frac{x_j^{n+1} - x_i}{\Delta x}\right) &= \sum_{m=0}^{\beta} (k\Delta x)^m \sum_{|q|=1}^{m} c_q \Gamma^{(q_1+\ldots+q_m)}(k + \lambda_i)\, \nu^{q_1+\ldots+q_m} \prod_{s=1}^{m} \left(a^{(s)}(x)\right)^{q_s} \\ &\quad + O\left((k\Delta x)^{\beta+1} \|\Gamma\|_{\beta+1,\infty} \|a\|_{\beta+1,\infty} \sum_{|q|=\beta+1} \nu^{q_1+\ldots+q_{\beta+1}}\right).\end{aligned} \tag{3.5}$$

In the above equation we have denoted by ν the ratio $\Delta t / \Delta x$. Since $q_1 + \ldots + q_{\beta+1} \leq |q|$ we have

$$\sum_{|q|=\beta+1} \nu^{q_1+\ldots+q_{\beta+1}} \leq C\left(\nu + \nu^{\beta+1}\right).$$

The remainder in (3.5) can thus be bounded by $O\left(\Delta x^{\beta+1} + \Delta x^{\beta}\Delta t + \Delta t^{\beta+1}\right)$. Combining (3.4) and (3.5) we obtain

$$\begin{aligned}\mathcal{T}_i(u(\cdot,t_n)) &= \sum_j u_j^n \Gamma\left(\frac{x_j^{n+1}-x_i}{\Delta x}\right) \qquad (3.6)\\ &= \sum_k \sum_{0\le l+m\le\beta} (k\Delta x)^{l+m}\frac{u^{(l)}(x_i)}{l!}\sum_{|q|=1}^{m} c_q \Gamma^{(q_1+\ldots+q_m)}(k+\lambda_i)\,\nu^{q_1+\ldots+q_m}\prod_{s=1}^{m}\left(a^{(s)}(x_i)\right)^{q_s}\\ &\qquad + O\left(\Delta x^{\beta+1}+\Delta x^{\beta}\Delta t+\Delta t^{\beta+1}\right)\\ &= \sum_k E_k(x_i) + O\left(\Delta x^{\beta+1}+\Delta x^{\beta}\Delta t+\Delta t^{\beta+1}\right).\end{aligned}$$

By differentiation, the moment conditions (2.5) yield

$$\sum_k k^{q'}\Gamma^{(q)}(k+x) = (-x)^{q'-q}q'(q'-1)\ldots(q'-q+1), \text{ for } 0\le q\le q'\le\beta.$$

Using these identities with $x=\lambda_i$ and observing that $\lambda_i^{l+m-(q_1+\ldots+q_m)}\nu^{q_1+\ldots+q_m} = d_i(\Delta t/\Delta x)^{l+m}$ where d_i are coefficients only depending on a, we get

$$\sum_k E_k(x_i) = \sum_{0\le l+m\le\beta} O\left(\Delta t^{l+m}\right) + O\left(\Delta x^{\beta+1}+\Delta x^{\beta}\Delta t+\Delta t^{\beta+1}\right). \qquad (3.7)$$

One can easily check that the zero and first order terms in the above expansion, corresponding to $0\le l+m\le 1$, result in

$$u(x_i,t_n) - \Delta t\, a(x_i)\partial u/\partial x(x_i,t_n) - \Delta t\, a'(x_i)u(x_i,t_n) = u(x_i,t_n) - \Delta t\,\partial(a\,u)/\partial x(x_i,t_n).$$

We therefore finally obtain

$$\begin{aligned}\mathcal{T}_i(u(\cdot,t_n)) &= u_i^n - \Delta t\frac{\partial(au)}{\partial x}(x_i,t_n) + O\left(\Delta t^2\right) + O\left(\Delta x^{\beta+1}\right) + O\left(\Delta x^{\beta}\Delta t\right) \qquad (3.8)\\ &= u(x_i,t^{n+1}) + O\left(\Delta t^2\right) + O\left(\Delta x^{\beta+1}+\Delta x^{\beta}\Delta t+\Delta t^{\beta+1}\right). \qquad (3.9)\end{aligned}$$

To prove the second assertion of our proposition we observe that if, at the end of an advection step, every cell within the stencil centered at the grid point x_i contains exactly one particle, since particles cannot cross this means that λ_j and λ_i lie in the same interval between successive integers. The kernel Γ is $\mathcal{C}^\infty$ in such intervals and thus the expansion 3.5 is valid to any order. The coefficient β can therefore be taken equal to p. □

3.2. Stability

We start with the linear stability analysis and assume that the velocity field a is constant.

Linear stability

In this paragraph, we will show unconditional stability (with respect to $\lambda = a\,\Delta t/\Delta x$) for a certain class of remeshing kernels. We set, for $k\in\mathbb{Z}$,

$$\alpha_k(\lambda) = \Gamma(k+\lambda)\ ,\ A_k(\lambda) = \sum_i \alpha_i(\lambda)\alpha_{i+k}(\lambda). \qquad (3.10)$$

In all the calculations that follow we will write α_k and A_k for $\alpha_k(\lambda)$ and $A_k(\lambda)$, respectively, when there is no ambiguity on the value of λ. We start with the following result

Lemma 3.2. *If the kernel Γ satisfies (2.5) then*

$$\sum_{k,l} (k-l)^q \, \alpha_k \alpha_l = 0 \ , \ \sum_{k\geq 1} k^q \, A_k = 0, \textit{ if } 1 \leq q \leq p, \ q \textit{ even.} \tag{3.11}$$

Proof. We have

$$\sum_{k,l} (k-l)^q \, \alpha_k \alpha_l = \sum_{k,l} \sum_{m=0}^{q} C_q^m k^m (-l)^{q-m} \, \alpha_k \alpha_l.$$

By the moment properties (2.5) $\sum_{k,l} k^m (-l)^{q-m} \, \alpha_k \alpha_l = (-\lambda)^m \lambda^{q-m}$ and thus

$$\sum_{k,l} (k-l)^q \, \alpha_k \alpha_l = \sum_{m=0}^{q} C_q^m \, (-\lambda)^m \lambda^{q-m} = 0.$$

The second identity readily follows, since $A_{-k} = A_k$. □

We consider the case of kernels involving stencils with an even number of points. We further restrict our analysis to second order kernels involving 4 points (which means that the support of Γ is the interval $[-2, +2]$) and fourth order kernels involving 6 points (the support of Γ is the interval $[-3, +3]$). The key properties of the remeshing kernel that are needed to ensure stability are the moment conditions and a sign and a decay properties for the kernel values. More precisely we will prove the following result

Proposition 3.3. *Assume that the velocity field a is constant, and that the kernel Γ satisfies one of the following conditions:*

(i) *Γ satisfies (2.5) with $p = 2$, Γ has support in $[-2, +2]$ and satisfies for all values of $\lambda \in [0, 1]$*

$$A_1 \geq 0 \ , \ A_2 \leq 0 \ , \ A_3 \geq 0 \ , \ -A_2 \geq 6A_3. \tag{3.12}$$

(ii) *Γ satisfies (2.5) with $p = 4$, Γ has support in $[-3, +3]$ and satisfies for all values of $\lambda \in [0, 1]$*

$$A_1 \geq 0 \ , \ A_2 \leq 0 \ , \ A_3 \geq 0 \ , \ A_4 \leq 0 \ , \ A_5 \geq 0 \ , \ A_3 \geq -8A_4. \tag{3.13}$$

Then the remeshed particle method is unconditionally stable.

Before proceeding with the proof of this result, some comments are in order. For all the kernels that are used in practice one has $\Gamma(x) \geq 0$ for $x \in [-1, 1]$ and Γ alternates sign in successive integer intervals. The conditions (3.12), (3.13) therefore mean that the kernel decays fast enough. In the case *i*, $A_2 = \alpha_{-2}\alpha_0 + \alpha_{-1}\alpha_1$ and $A_3 = \alpha_{-1}\alpha_1$. The condition $-A_2 \geq 6A_3$ is thus satisfied as soon has $\alpha_0 \geq 6\alpha_1$. For the $\Lambda_{2,1}$ kernel, this condition can easily be checked: one has, for $\lambda \in [0, 1]$

$$|\alpha_0/\alpha_1| = \frac{2 + 2\lambda - 3\lambda^2}{(1-\lambda)\lambda} = 2 + \frac{2 - \lambda^2}{(1-\lambda)\lambda} \geq 2 + \frac{1}{(1-\lambda)\lambda} \geq 6.$$

Similar calculations show that the other 2nd order kernels in the Table 1 satisfy this decay condition. For 4th order, 6 points kernels, the analytic calculations are more involved, but it is possible to check by symbolic calculations that the conditions (3.13) hold for the kernels that are indicated in Table 1. Our result only covers 2nd and 4th order kernels. However one can conjecture from the proof that will be given below that it extends to higher order kernels, with larger stencils, under an appropriate decay condition for the coefficients A_k.

We now proceed with the proof of Proposition 3.3 and begin with the case *i*.

We define by I the integer such that $\lambda = a\Delta t/\Delta x \in [I, I+1[$ and we set $\mu = \lambda - I \in [0, 1[$. We write $j = i - I + k$ and set $v_i = u_{i-I}^n$. The scheme (2.3) now reads

$$u_i^{n+1} = \sum_k v_{i+k} \Gamma(k + \mu). \tag{3.14}$$

Since the support of Γ is in $[-2,+2]$, the coefficients $\alpha_k(\mu)$ vanish if $k \le -3$ or $k \ge 2$. Using (3.10) we obtain

$$\sum_i |u_i^{n+1}|^2 = \sum_i \sum_{-2\le k,l\le 1} v_{i+k}\, v_{i+l}\, \alpha_k(\mu)\, \alpha_l(\mu) = \sum_i v_i^2 \sum_k \alpha_k^2 + 2\sum_i \sum_{-2\le k<l\le 1} v_{i+k}\, v_{i+l}\, \alpha_k\, \alpha_l. \tag{3.15}$$

We then write

$$2v_{i+k}\, v_{i+l} = v_{i+k}^2 + v_{i+l}^2 - |v_{i+k} - v_{i+l}|^2.$$

The conservation of the first moment gives

$$1 = \left(\sum_k \alpha_k\right)^2 = \sum_k \alpha_k^2 + 2 \sum_{-2\le k<l\le 1} \alpha_k\, \alpha_l,$$

and therefore

$$\sum_i \left|u_i^{n+1}\right|^2 = \sum_i v_i^2 - \sum_i S_i \tag{3.16}$$

where $S_i = \sum_{-2\le k<l\le 1} \alpha_k\, \alpha_l |v_{i+k} - v_{i+l}|^2$.

Since obviously $\sum_i v_i^2 = \sum_i |u_i^n|^2$, it remains to prove that $\sum_i S_i \ge 0$. Using the change of index $l = k+m$ for $l > k$ and the notations in (3.10) allows to write

$$\sum_i S_i = \sum_k \sum_{1\le m\le 3} \alpha_k\, \alpha_{k+m} \sum_i |v_{i+m} - v_i|^2 = \sum_{1\le m\le 3} A_m \sum_i |v_{i+m} - v_i|^2.$$

We set $\delta_i = v_{i+1} - v_i$ to rewrite the above expression as

$$\sum_i S_i = A_1 \sum_i \delta_i^2 + A_2 \sum_i (\delta_i + \delta_{i+1})^2 + A_3 \sum_i (\delta_i + \delta_{i+1} + \delta_{i+2})^2. \tag{3.17}$$

Expanding the above sums we obtain

$$\sum_i S_i = (A_1 + 2A_2 + 3A_3) \sum_i \delta_i^2 + (2A_2 + 4A_3) \sum_i \delta_i\delta_{i+1} + 2A_3 \sum_i \delta_i\delta_{i+2}.$$

We again write $2\delta_i\delta_{i+1} = \delta_i^2 + \delta_{i+1}^2 - |\delta_i - \delta_{i+1}|^2$ and a similar identity for $2\delta_i\delta_{i+2}$ to obtain

$$\sum_i S_i = (A_1 + 4A_2 + 9A_3) \sum_i \delta_i^2 - (A_2 + 2A_3) \sum_i |\delta_i - \delta_{i+1}|^2 - A_3 \sum_i |\delta_i - \delta_{i+2}|^2.$$

By (3.11) with $q = 2$ we get $A_1 + 4A_2 + 9A_3 = 0$. If we set $\eta_i = \delta_i - \delta_{i+1}$ we thus have

$$\sum_i S_i = -(A_2 + 2A_3) \sum_i |\eta_i|^2 - A_3 \sum_i |\eta_i + \eta_{i+1}|^2.$$

Since $A_3 > 0$ and $|\eta_i + \eta_{i+1}|^2 \le 4(\eta_i^2 + \eta_{i+1}^2)$ we can write

$$\sum_i S_i = -(A_2 + 6A_3) \sum_i |\eta_i|^2 \ge 0,$$

provided the decay property (3.12) is satisfied.

We now turn to the second assertion of the proposition. We start from (3.15) and obtain an estimate similar to (3.17), with $m = 5$ instead of 3 since the kernel has now a support of size 6

$$\sum_i S_i = \sum_k \sum_{1\le m\le 3} \alpha_k\, \alpha_{k+m} \sum_i |v_{i+m} - v_i|^2 = \sum_{1\le m\le 5} A_m \sum_i |v_{i+m} - v_i|^2. \tag{3.18}$$

We set $\delta_i = v_{i+1} - v_i$ and obtain:

$$\begin{aligned}\sum_i S_i &= A_1 \sum_i \delta_i^2 + A_2 \sum_i (\delta_i + \delta_{i+1})^2 + A_3 \sum_i (\delta_i + \delta_{i+1} + \delta_{i+2})^2 \\ &\quad + A_4 \sum_i (\delta_i + \delta_{i+1} + \delta_{i+2} + \delta_{i+3})^2 + A_5 \sum_i (\delta_i + \delta_{i+1} + \delta_{i+2} + \delta_{i+3} + \delta_{i+4})^2 \\ &= (A_1 + 2A_2 + 3A_3 + 4A_4 + 5A_5) \sum_i \delta_i^2 + (2A_2 + 4A_3 + 6A_4 + 8A_5) \sum_i \delta_i \delta_{i+1} \\ &\quad + (2A_3 + 4A_4 + 6A_5) \sum_i \delta_i \delta_{i+2} + (2A_4 + 4A_5) \sum_i \delta_i \delta_{i+3} + 2A_5 \sum_i \delta_i \delta_{i+4}.\end{aligned}$$

Rewriting double products as above and introducing $\eta_i = \delta_i - \delta_{i+1}$, we obtain

$$\begin{aligned}\sum_i S_i &= (A_1 + 4A_2 + 9A_3 + 16A_4 + 25A_5) \sum_i \delta_i^2 \\ &\quad - (A_2 + 2A_3 + 3A_4 + 4A_5) \sum_i \eta_i^2 - (A_3 + 2A_4 + 3A_5) \sum_i (\eta_i + \eta_{i+1})^2 \\ &\quad - (A_4 + 2A_5) \sum_i (\eta_i + \eta_{i+1} + \eta_{i+2})^2 - A_5 \sum_i (\eta_i + \eta_{i+1} + \eta_{i+2} + \eta_{i+3})^2.\end{aligned}$$

By (3.11) with $q = 2$ we have $A_1 + 4A_2 + 9A_3 + 16A_4 + 25A_5 = 0$. Expanding the remaining squares in the above identity we get

$$\begin{aligned}\sum_i S_i &= -(A_2 + 4A_3 + 10A_4 + 20A_5) \sum_i \eta_i^2 - 2(A_3 + 4A_4 + 10A_5) \sum_i \eta_i \eta_{i+1} \\ &\quad - 2(A_4 + 2A_5) \sum \eta_i \eta_{i+2} - 2A_5 \sum \eta_i \eta_{i+3}.\end{aligned}$$

We again write $2\eta_i\eta_{i+1} = \eta_i^2 + \eta_{i+1}^2 - |\eta_i - \eta_{i+1}|^2$ and similar identities for the other double products to obtain

$$\begin{aligned}\sum_i S_i &= -(A_2 + 6A_3 + 20A_4 + 50A_5) \sum_i \eta_i^2 \\ &\quad + (A_3 + 4A_4 + 10A_5) \sum_i (\eta_i - \eta_{i+1})^2 + (A_4 + 4A_5) \sum_i (\eta_i - \eta_{i+2})^2 + A_5 \sum_i (\eta_i - \eta_{i+3})^2.\end{aligned}$$

Subtracting (3.11) with $q = 4$ from the corresponding identity with $q = 2$ yields $A_2 + 6A_3 + 20A_4 + 50A_5 = 0$. Moreover, since $A_4 \le 0$ and $A_5 \ge 0$ we can write

$$(A_4 + 4A_5) \sum_i (\eta_i - \eta_{i+2})^2 \ge A_4 \sum_i (\eta_i - \eta_{i+2})^2 \ge 4A_4 \sum_i (\eta_i - \eta_{i+i})^2.$$

Therefore

$$\sum_i S_i \ge (A_3 + 8A_4) \sum_i (\eta_i - \eta_{i+i})^2 \ge 0,$$

provided the decay property (3.13) is satisfied. This concludes our proof in the case of a constant velocity field.

General case

We now consider the general case of a smooth, non-constant, velocity field a.

Proposition 3.4. *Assume that the velocity field a is in $W^{1,\infty}(\mathbb{R})$ and that the assumptions of Proposition 3.3 are satisfied. Assume in addition that the kernel Γ satisfies the interpolation property (2.10). Then there exists a constant C, independent of Δt and Δx, such that*

$$\sum_i \left|u_i^{n+1}\right|^2 \le (1 + C\Delta t) \sum_i \left|u_i^n\right|^2. \tag{3.19}$$

Proof. We will only give the proof in the case of a second order method corresponding to the case i of Proposition 3.3. The proof in the case of ii follows along the same lines.

The local Courant number λ can now vary from one particle to the next. To start with we will consider the case when this number remains in the same integer interval, that is we make the assumption that

$$\exists\, I \in \mathbb{Z} \text{ such that, } \forall i \in \mathbb{Z},\ \lambda_i = a(x_i, t_n)\Delta t/\Delta x \in [I, I+1].$$

We proceed like in the proof of Proposition 3.3. We set $\mu_j = \lambda_j - I \in [0,1[$, $v_i = u^n_{i-I}$ and $j = i - I + k$, and the scheme (3.14) becomes

$$u_i^{n+1} = \sum_k v_{i+k}\Gamma(k+\mu_j) = \sum_k v_{i+k}\alpha_k(\mu_j). \tag{3.20}$$

We first observe that, by the regularity of the velocity field a,

$$|\lambda_j - \lambda_{j'}| = O\left(|j-j'|\Delta t\right)$$

and, since $\Gamma \in W^{1,\infty}(\mathbb{R})$, for $-2 \le k < l \le 1$, $j = i - I + k$, $j' = i - I + l$

$$\alpha_k(\mu_j) = \alpha_k(\mu_{j'}) + O(\Delta t). \tag{3.21}$$

In particular $\alpha_k(\mu_j) = \alpha_k(\mu_{i-I}) + O(\Delta t)$ if $j = i - I + k$ with $k \in [-2, 1]$. This allows to write

$$\sum_i |u_i^{n+1}|^2 = \sum_i \left[\sum_{-2\le k\le 1} v_{i+k}\alpha_k(\mu_{i-I})\right]^2 + O\left(\Delta t|v|^2\right),$$

where we have used the notation $|v|^2 = \sum_i |v_i|^2$. Proceeding like in estimates (3.16), (3.17) we can write:

$$\sum_i |u_i^{n+1}|^2 = O\left(\Delta t|v|^2\right) + \sum_i \left[\sum_k |v_{i+k}|^2\alpha_k^2(\mu_{i-I}) + \sum_{k<l}\left(|v_{i+k}|^2 + |v_{i+l}|^2 - |v_{i+k} - v_{i+l}|^2\right)\alpha_k(\mu_{i-I})\alpha_l(\mu_{i-I})\right]. \tag{3.22}$$

Using again (3.21) and setting $j = i + k$ we have

$$\begin{aligned}
\sum_i\sum_k |v_{i+k}|^2\alpha_k^2(\mu_{i-I}) &= \sum_j\sum_k |v_j|^2\alpha_k^2(\mu_{j-I}) + O\left(\Delta t|v|^2\right),\\
\sum_i\sum_{k<l} |v_{i+k}|^2\alpha_k(\mu_{i-I})\alpha_l(\mu_{i-I}) &= \sum_j\sum_{k<l} |v_j|^2\alpha_k(\mu_{j-I})\alpha_l(\mu_{j-I}) + O\left(\Delta t|v|^2\right),\\
\sum_i\sum_{k<l} |v_{i+l}|^2\alpha_k(\mu_{i-I})\alpha_l(\mu_{i-I}) &= \sum_j\sum_{k<l} |v_j|^2\alpha_k(\mu_{j-I})\alpha_l(\mu_{j-I}) + O\left(\Delta t|v|^2\right),\\
\sum_i\sum_{k<l} |v_{i+k} - v_{i+l}|^2\alpha_k(\mu_{i-I})\alpha_l(\mu_{i-I}) &= \sum_j\sum_m A_m(\mu_{j-I})|v_j - v_{j+m}|^2 + O\left(\Delta t|v|^2\right).
\end{aligned}$$

This allows us to rewrite the summation in the right hand side of (3.22) as

$$\sum_j |v_j|^2\left[\sum_k \alpha_k^2(\mu_{j-I}) + 2\sum_{k<l}\alpha_k(\mu_{j-I})\alpha_l(\mu_{j-I})\right] - \sum_j\sum_m A_m(\mu_{j-I})|v_j - v_{j+m}|^2 + O\left(\Delta t|v|^2\right).$$

By the conservation of the first moment we have

$$\sum_k \alpha_k^2(\mu_{j-I}) + 2\sum_{k<l}\alpha_k(\mu_{j-I})\alpha_l(\mu_{j-I}) = 1$$

and, like for the constant velocity case, it only remains to check the sign of $S = \sum_j \sum_m A_m(\mu_{j-I})|v_j - v_{j+m}|^2$. We proceed like in the constant velocity case and set $\delta_i = v_{i+1} - v_i$ and $\eta_i = \delta_{i+1} - \delta_i$. We expand the squares in S using the fact that

$$\delta_{i+1}^2 A_2(\mu_{i-I}) = \delta_{i+1}^2 A_2(\mu_{i+1-I}) + O(\Delta t)\left(|v_{i+2}|^2 + |v_{i+1}|^2\right)$$

and similar expressions for $\delta_{i+1}^2 A_3(\mu_{i-I})$ and $\delta_{i+2}^2 A_3(\mu_{i-I})$. This leads to

$$\begin{aligned} S =& |v|^2 O(\Delta t) + \sum_i \delta_i^2 \left[A_1(\mu_{i-I}) + 4A_2(\mu_{i-I}) + 16A_3(\mu_{i-I})\right] - \sum_i \eta_i^2 \left[A_2(\mu_{i-I}) + 2A_3(\mu_{i-I})\right] \\ &- \sum_i (\eta_i + \eta_{i+1})^2 A_3(\mu_{i-I}). \end{aligned} \tag{3.23}$$

We next write

$$(\eta_i + \eta_{i+1})^2 A_3(\mu_{i-I}) \le 2\left(\eta_i^2 A_3(\mu_{i-I}) + \eta_{i+1}^2 A_3(\mu_{i+1-I})\right) + C\Delta t\left(|v_{i+3}|^2 + |v_{i+2}|^2 + |v_{i+1}|^2\right).$$

Due to (3.11), under the condition (3.12), (3.23) finally yields

$$S \ge -C\Delta t|v|^2$$

and therefore

$$\sum_i |u_i^{n+1}|^2 \le \sum_i |u_i^n|^2 + C\Delta t.$$

We finally turn to the general case, when neighboring particles can have their local CFL numbers in different integer intervals. We can always group particles in subsets where the CFL numbers are in the same integer interval. For $I \in \mathbb{Z}$, we set

$$J_I = \{\, j \in \mathbb{Z} \text{ such that } \lambda_j \in [I, I+1[\ \}.$$

We can rewrite the remeshed particle method as

$$u_i^{n+1} = \sum_I \sum_{j \in J_I} u_j^n \Gamma(\lambda_j + j - i) = \sum_I S_I(i). \tag{3.24}$$

The next step is to take the square of the above identity. The key point is to observe that, by the interpolation property (2.10), double products of the form $S_I(i)S_{I'}(i)$ with $I \neq I'$ are of order Δt. We indeed observe that, due to the regularity of a, we have

$$|\lambda_j - \lambda_{j'}| \le C\Delta t|j - j'|$$

and therefore, if the kernel Γ has a support of size $2M_s$ and if Δt is small enough

$$\Gamma(\lambda_j + j - i)\Gamma(\lambda_{j'} + j' - i) \neq 0 \Rightarrow |j - j'| \le 2M_s + 1. \tag{3.25}$$

Next, again in view of the regularity of a, if two neighboring particles have indices which belong to a different set J_I this implies that their local CFL number is close to an integer value. More precisely

$$\begin{aligned} &[j \in J_I, j' \in J_{I'}, I \neq I', |j - j'| \le 2M_s + 1] \Rightarrow \\ &\quad \exists (K, K') \in \mathbb{Z}^2, K \neq K' \text{such that } j + \lambda_j = K + O(\Delta t),\ j' + \lambda_{j'} = K' + O(\Delta t). \end{aligned}$$

By the interpolation property (2.10) and the regularity of Γ this implies

$$[j \in J_I, j' \in J_{I'}, I \neq I', |j - j'| \le 2M_s + 1] \Rightarrow \Gamma(\lambda_j + j - i)\Gamma(\lambda_{j'} + j' - i) = O(\Delta t).$$

Combined with (3.25) this shows that, for $I \neq I'$,

$$S_I(i)S_{I'}(i) \leq C\Delta t \sum_{j \in J_I \bigcup J_{I'}} |u_j^n|^2. \tag{3.26}$$

It remains now to sum over i the above identities. To identify the indices j which, for a given index i, contribute to the sum $S_I(i)$, we denote by ϕ the application such that

$$\phi(x) = x + a(x)\Delta t.$$

In view of (3.2) we have $0 < 1 - M \leq \phi'(x) \leq 1 + M$. ϕ is thus one-to-one and its inverse ϕ^{-1} is strictly increasing. If we set $\psi(i) = \phi^{-1}(i\Delta x)/\Delta x$, the indices j which contribute to $S_I(i)$ must satisfy

$$i\Delta x - M_s\Delta x \leq j\Delta x + a_j\Delta t \leq i\Delta x + M_s\Delta x$$

which yields

$$\psi(i - M_s) \leq j \leq \psi(i + M_s).$$

If a particle j contributes to two different terms $S_I(i)$ and $S_I(i')$ with, say, $i < i'$ this means that

$$\psi(i' - M_s) \leq j \leq \psi(i + M_s)$$

and therefore

$$|i' - i| \leq M_s.$$

As a result, a given particle of index j contributes to at most $2M_s$ sums $S_I(i)$. One can thus deduce from (3.26) that, for a certain constant C independent of Δt and Δx

$$\sum_i \sum_{I \neq I'} S_I(i)S_{I'}(i) \leq C\Delta t|u^n|^2,$$

and, from (3.24),

$$\sum_i |u_i^{n+1}|^2 \leq \sum_i \sum_I S_I^2(i) + C\Delta t|u^n|^2. \tag{3.27}$$

Finally, if we set

$$w_j^I = \begin{cases} u_j^n & \text{if } j \in J_I \\ 0 & \text{otherwise,} \end{cases}$$

by the preceding proof in the case when I takes a constant value, we have

$$\sum_i S_I^2(i) \leq (1 + C\Delta t)|w^I|^2$$

and

$$\sum_I \sum_i S_I^2(i) \leq (1 + C\Delta t) \sum_I |w^I|^2.$$

In view of (3.27) this leads to

$$\sum_i |u_i^{n+1}|^2 \leq |u^n|^2 + C\Delta t|u^n|^2$$

and our proof is completed. □

Remark 3.5. For $CFL \leq 1$, Remeshed Particle Methods reduce to finite-difference schemes [8]. The convergence analysis above contains thus as a by-product a convergence proof of a class of conservative second and fourth order finite-difference schemes.

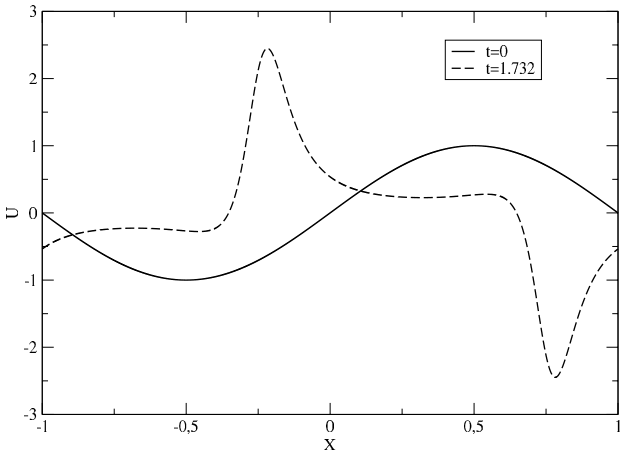


FIGURE 1. Advection equation with data (4.1). Initial condition (solid curve) and solution at $t = \sqrt{3}$ (dashed curve).

4. Numerical illustrations

In this section we compare the accuracy of the various kernels considered above in 1D, 2D and 3D smooth flows. The reference [15] deals with remeshed particle methods using the $\Lambda_{4,2}$ kernel in the context of the transport of passive scalar in turbulent flows. It in particular emphasizes the efficiency of this method in the case of high Schmidt numbers, that is when the diffusivity of the scalar is small compared to the viscosity of the flow. The extension of this work to higher order kernels introduced in the present paper will be reported in a future work.

We first consider a 1D advection equation in the interval $[-1, +1]$ with periodic boundary conditions. The velocity a and the initial condition u_0 are given by

$$a(x) = 1 + \sin(\pi x)/2\ ,\ u_0(x) = \sin(\pi x). \tag{4.1}$$

This flow induces compression and dilatation which successively increase and decrease the value of the solution. The exact solution is given by the following formula :

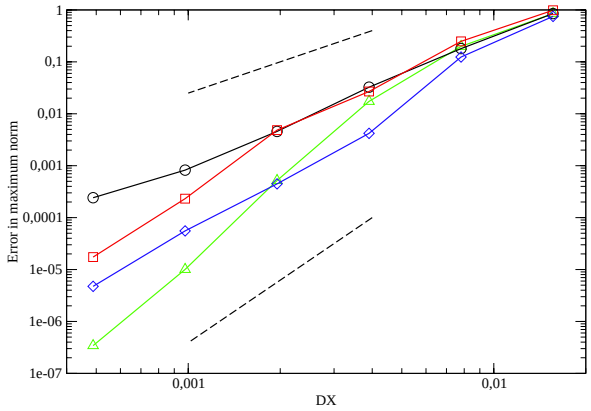
$$u(x(t), t) = u_0(x_0)\frac{2 + \sin 2\pi x_0}{2 + \sin 2\pi x(t)}$$

where $x(t)$ is the trajectory with origin x_0 associated to the velocity field a, given by

$$\arctan[\tan(\pi x(t)) + 1/2] = \arctan[\tan(\pi x_0) + 1/2] + t\pi\sqrt{3}/2.$$

The solutions are periodic in time with period $T = 4/\sqrt{3}$.

Figure 1 shows the solution at initial time and $t = \sqrt{3}$. For this case we compare in Figure 2 the results obtained with the kernels $\Lambda_{2,1}$, $\Lambda_{4,2}$, $\Lambda_{2,2}$ and $\Lambda_{4,4}$. In all cases, and throughout this section, particles where pushed with a fourth order Runge−Kutta time-stepping. The number of grid points ranged from 128 to 4096. For the purpose of this refinement study, we had to choose a constant value for the ratio $\Delta t/\Delta x$. We fixed a CFL value of 12. This value corresponds, for the coarsest resolution considered in this test, to a constant M in (3.2) equal to 0.7. The error is measured in the maximum norm. The numerical analysis above indicates that the two last kernels should deliver respectively second and fourth order methods, while the two first kernels,



Kernel	Order of convergence
$\Lambda_{2,1}$	2.35
$\Lambda_{2,2}$	3.15
$\Lambda_{4,2}$	3.45
$\Lambda_{4,4}$	4.25

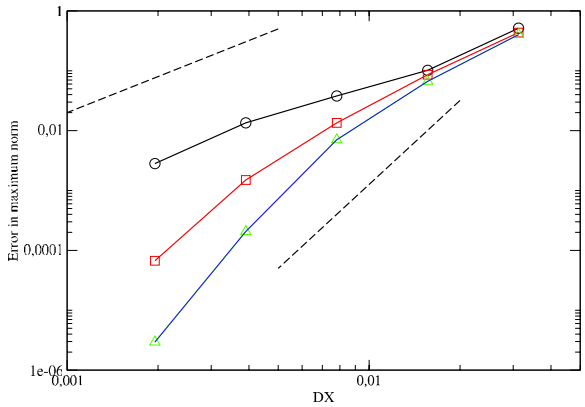
FIGURE 2. Refinement study for the 1D advection equation (4.1) and several first to fourth order RPM and CFL value equal to 12. *Left picture*: *black-circle curve*: kernel $\Lambda_{2,1}$; *red-square*: kernel $\Lambda_{2,2}$; *blue-diamond*: kernel $\Lambda_{4,2}$; *green-triangle*: $\Lambda_{4,4}$; dashed lines indicate slopes corresponding to second and fourth order convergence. Right table: average order of convergence for these RPM.

because of their lack of regularity, should be limited to first and second order, respectively. One can indeed observe that, for a given number of conserved moments, increasing the regularity does improve the accuracy of the RPM. However it is interesting to note that the kernel $\Lambda_{2,1}$ already gives a second order method. The reason is that, in this example, zones where the solution undergoes large variations, and thus where numerical errors are likely to be larger, only rarely correspond to grid points where the local CFL numbers cross integer values.

To illustrate next the directional splitting which we use to address multidimensional problems, we first consider the classical two-dimensional case of an off-centered circle strained in an incompressible rotational flow. More precisely, the computational box is the square $[0,1]^2$. The velocity field is given by

$$\mathbf{a}(x_1,x_2,t) = f(t)\left(-\sin^2(\pi x_1)\sin(2\pi x_2), \sin(2\pi x_1)\sin^2(\pi x_2)\right). \tag{4.2}$$

The initial condition is a sign function whose zero level set is a circle of radius 0.15 and centered at the point $(0.5, 0.15)$. The function $f(t)$ is introduced to allow the solution to eventually return to the initial condition. We take $f(t) = \cos(\pi t/12)$, and compare the computed solution at time $t = 12$ with the initial condition. In this experiment, like in the 3D case below, we use the classical second order Strang formula for the dimensional splitting, that is we successively push and remesh particles in the x_1, x_2, x_2 and x_1 directions for $\Delta t/2$. It is indeed possible to derive and use higher order splitting methods, but our choice of a second order splitting was made for a practical reason – fourth order splitting methods are more demanding in terms of CPU and memory requirement. Furthermore, although second order splitting limits the theoretical order of convergence to 2, one can still see the benefit of using higher order kernels. Using a 6th order kernel in this example indeed yields an effective order of convergence close to 6. In this experiment the number of grid points in each direction ranged form 32 to 512. Like in the previous experiment, for the purpose of this refinement study, the CFL number was kept constant, equal to 12. The average order of convergence is shown on the right picture of Figure 3. In this experiment the kernels $\Lambda_{2,2}$, $\Lambda_{4,4}$ and $\Lambda_{6,6}$ would give similar results to the kernels $\Lambda_{2,1}$, $\Lambda_{4,2}$ and $\Lambda_{6,4}$ respectively. Figure 4 shows the solution obtained with $N = 256$ grid points in each direction and the kernels $\Lambda_{2,1}$ and $\Lambda_{6,4}$, at $t = 6$ and $t = 12$, compared to the exact solution. At $t = 6$, which is the time of maximal stretching for this experiment, the exact solution was obtained by a front tracking method using 10 000 markers.



Kernel	Order of convergence
$\Lambda_{2,1}$	1.87
$\Lambda_{4,2}$	3.17
$\Lambda_{6,4}$	5.92

FIGURE 3. Refinement study for the 2D advection field (4.2). CFL value is equal to 12. Left picture: *black-circle curve*: kernel $\Lambda_{2,1}$; *red-square*: kernel $\Lambda_{4,2}$; *blue-triangle*: kernel $\Lambda_{6,4}$; dashed lines indicate slopes corresponding to second and fourth order convergence. Right table: average order of convergence for these RPM.

This study clearly shows the gain obtained in this case by using high order kernels, even though the theoretical order of convergence is limited by that of the dimensional splitting.

As already stressed, one distinctive feature of Semi Lagrangian Particles are that their stability and the spatial order of convergence is not constrained by a CFL condition. To illustrate this property, we repeated the above experiment with 256 points in each direction and a CFL number equal to 30. These grid resolution and time-step give a value of the constant M in (3.2) equal to 0.35. Figure 5 shows the zero level set at times 6 and 12. Although one cannot expect the same overall accuracy as in the previous experiment with such a large time-step, the interface appears to remain rather well captured.

We now turn to a 2D example where the kernel regularity seems to be more important than in the previous cases. We consider an expansion field given by

$$\mathbf{a}(x_1, x_2) = (x_1/r, x_2/r), r = \sqrt{x_1^2 + x_2^2}, \tag{4.3}$$

and the initial value of u is an axisymmetric function supported in an annulus:

$$u_0(r) = \begin{cases} C[(r - r_a)(r - r_b)]^4 \text{ if } r_a \leq r \leq r_b \ , \\ 0 \text{ otherwise.} \end{cases}$$

with $r_a = 0.1, r_b = 0.25$ and $C = [2/(r_a - r_b)]^8$.

The exact solution at time t is given by $u(r,t) = u_0(r - t)(r - t)/r$. Figure 6 shows the result of a refinement study for the kernels $\Lambda_{2,1}$, $\Lambda_{2,2}$, $\Lambda_{4,2}$ and $\Lambda_{4,4}$. In that case areas where the local CFL number cross integer values are aligned with the flow directions. As a result the kernel $\Lambda_{2,1}$ only gives first order convergence, while $\Lambda_{2,2}$ is close to the second order accuracy predicted by our analysis. Note that increasing the order and regularity of the kernel improves the convergence but does not allow to reach fourth order. In this example the second order time-splitting limits the observed overall accuracy.

We close this section with a 3D version case suggested in [17] and often used to validate level set methods. In this example the computational box is the unit cube and the initial condition is a sphere of radius 0.15 centered

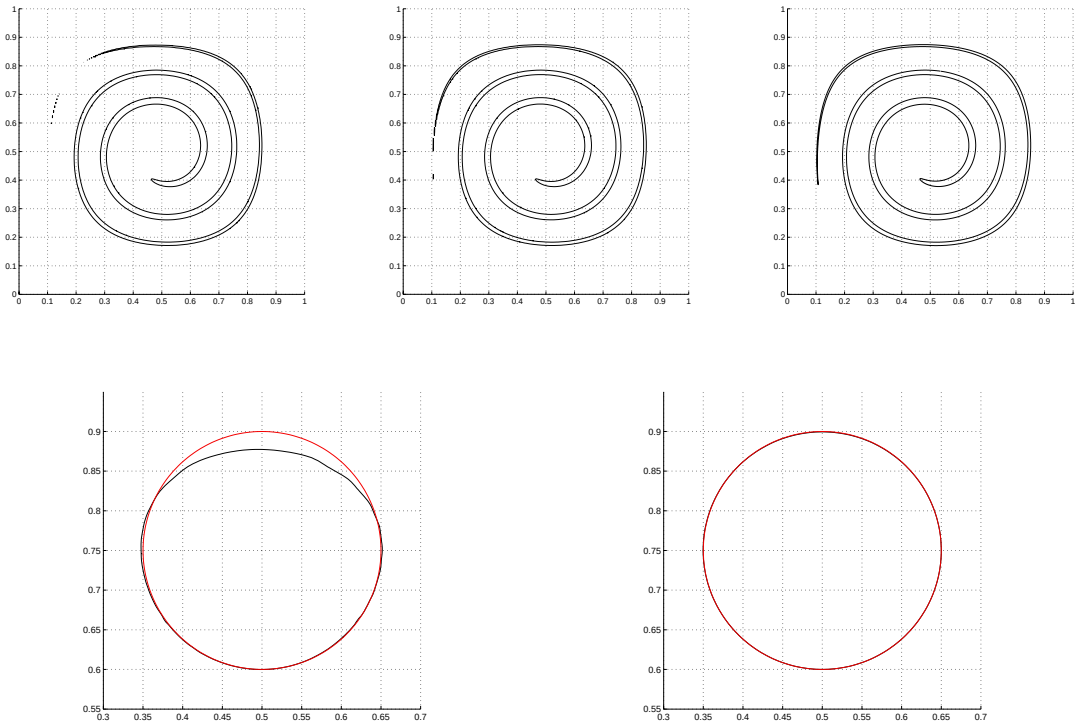


FIGURE 4. Interface $u = 0$ for the advection field (4.2) with $N_x = N_y = 256$. Top-left: $t = 6$, kernel $\Lambda_{2,1}$; top-middle: $t = 6$, kernel $\Lambda_{6,4}$; top-right: exact front-tracking solution; bottom-left: $t = 12$, kernel $\Lambda_{2,1}$; bottom-right: $t = 12$, kernel $\Lambda_{6,4}$. On the bottom pictures the red curve represents the exact (initial) interface at $t = 12$.

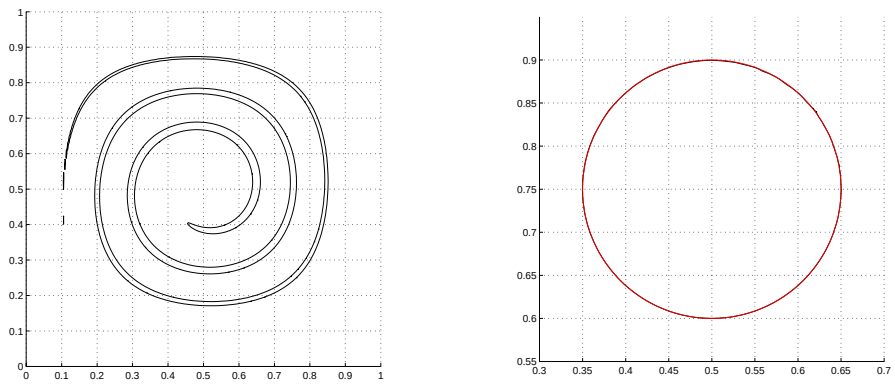
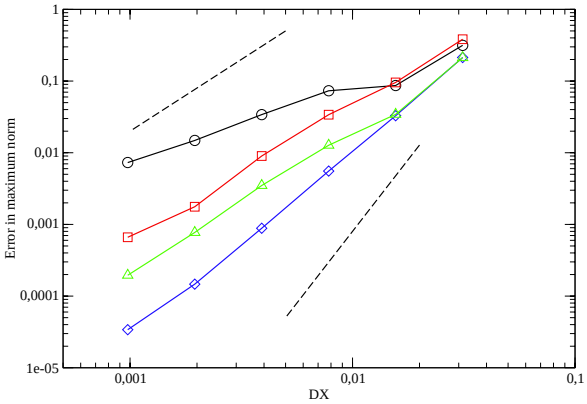


FIGURE 5. Same experiment as in Figure 4 with $Nx = Ny = 256$, the kernel $\Lambda_{6,4}$ and $CFL = 30$. Left picture: interface at $t = 6$; right picture: interface at $t = 12$ (red curve is the exact solution).



Kernel	Order of convergence
$\Lambda_{2,1}$	1.08
$\Lambda_{2,2}$	1.83
$\Lambda_{4,2}$	2.01
$\Lambda_{4,4}$	2.52

FIGURE 6. Refinement study for the radial test case (4.3). Left picture: *black-circle curve*: kernel $\Lambda_{2,1}$; *red-square*: kernel $\Lambda_{2,2}$; *blue-diamond*: kernel $\Lambda_{4,2}$; *green-triangle*: $\Lambda_{4,4}$; dashed lines indicate slopes corresponding to second and fourth order convergence. Right table: average order of convergence for these kernels. The CFL number is equal to 4.

around the point with coordinates $(0.35, 0.35, 0.35)$. This sphere is advected with the velocity field

$$a_1(x_1, x_2, x_3) = 2\,\sin^2(\pi x_1)\sin(2\pi x_2)\sin(2\pi x_3), \tag{4.4}$$

$$a_2(x_1, x_2, x_3) = -\,\sin(2\pi x_1)\sin^2(\pi x_2)\sin(2\pi x_3), \tag{4.5}$$

$$a_3(x_1, x_2, x_3) = -\,\sin(2\pi x_1)\sin(2\pi x_2)\sin^2(\pi x_3). \tag{4.6}$$

This motion wraps the sphere into thinner and thinner surfaces difficult to capture. In this case we want also to illustrate the fact that RPM only need particles in the support of the advected quantity. To capture the sphere we use a color function (1 inside the sphere, 0 outside). At the end of the remeshing step, particles are created only when their strength is above 0.001. In this experiment, we use 256 grid points in each direction. The CFL number is equal to 30, which corresponds to the same value of M as in the 2D case (4.2). The number of particles roughly varies between 1.5%, at $t = 0$, and 10%, at $t = 4$, of the total number of grid points inside the computational box. Due to the cut-off used in the remeshing step, the total mass was not algebraically conserved. However, in the present experiments the deviation did not exceed 0.1% of the total mass. Figure 7 shows the evolution of the volume inside the interface (this volume should remain constant at all times) obtained with the kernels $\Lambda_{2,1}$, $\Lambda_{4,2}$ and $\Lambda_{8,4}$. Figure 8 show the interface $u = 0.5$ at time $t = 4$, with the kernels $\Lambda_{2,1}$ and $\Lambda_{8,4}$. These results confirm the gain in accuracy that can be obtained by using high order kernels. The next section will discuss the computational complexity associated with these kernels.

5. Remeshed particle methods and software engineering

One challenge in High Performance Computing is to develop algorithms that are able to take advantage of clusters made of emerging exascale hardware. These clusters are hybrid architectures which combine many heterogeneous distributed processors and accelerators. A natural idea is to match models, algorithms and physical scales resolved by these algorithms to the specific features of the processors involved in these architectures.

A typical example is the case of turbulent transport in incompressible flows. Hybrid algorithms can be designed to resolve in an optimal fashion the minimal scales present in the flow and in the advected scalar [15]. Moreover in these hybrid approaches the different algorithms can have different parallel scalability and it may be

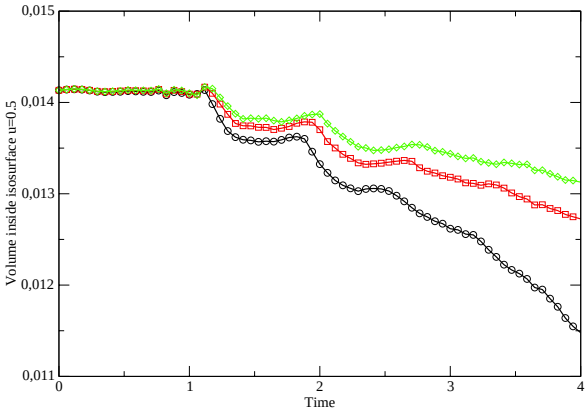


FIGURE 7. Volumes inside the interface $u = 0.5$ for the test case (4.4)−(4.6) with $CFL = 30$ and 256 grid points in each direction. *Green-circle curve*: kernel $\Lambda_{2,1}$; *red-square curve*: kernel $\Lambda_{4,2}$; green-diamond curve: kernel $\Lambda_{8,4}$.

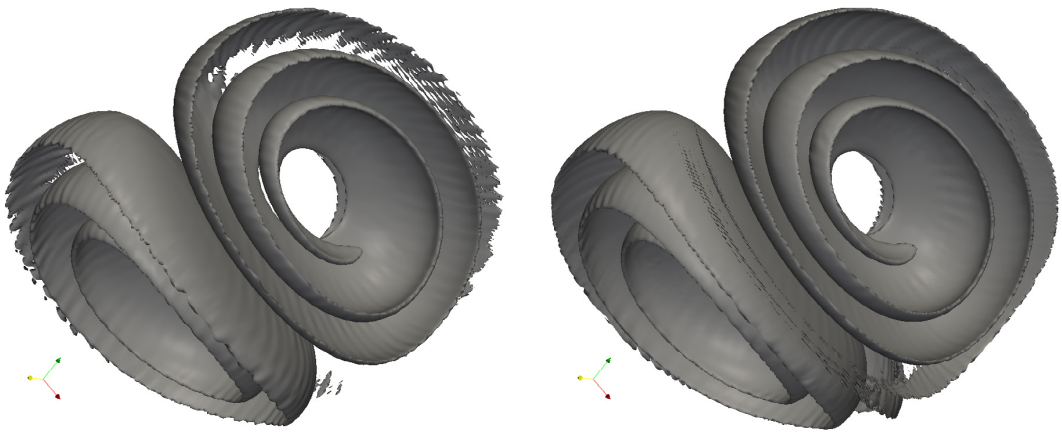
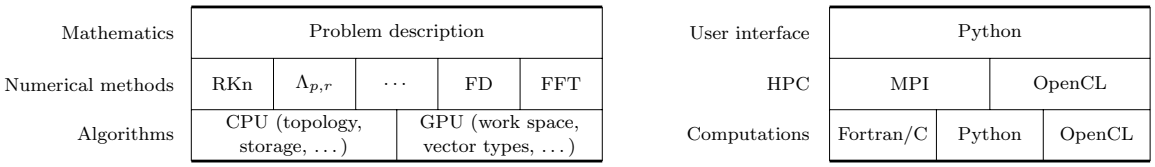


FIGURE 8. Isosurface $u = 0.5$ at $t = 4.0$ for the test case of the sphere in the flow (4.4)−(4.6) with $N = 256$ and $CFL = 30$. Left picture: kernel $\Lambda_{2,1}$; right picture: kernel $\Lambda_{8,4}$.

advantageous to distribute these algorithms to different type of processors. The local nature of RPM make these methods well suited for highly parallel processors, like GPUs, and one may envision simulations of transport at high Schmidt numbers in turbulent flows (that is when the scalar smallest scales extend much beyond those of the flows) at very high resolution at a cost that does not exceed that of the flow solver at a much lower resolution.

To be able to implement hybrid algorithms on hybrid architectures, one needs to develop high level frameworks and libraries with a high level description which allows to distribute different solvers and grids to different parts of the clusters in a seamless fashion.



(a) Functional view (b) Programming view

FIGURE 9. Application layout.

In this section we first outline the general approach followed for this framework. We then go into more details for a specific branch of this framework, namely the implementation on GPU of RPM. We detail the performance of the different kernels associated to the different parts of the algorithm and the overall performance of the GPU implementation of RPM. We in particular focus on the complexity of the RPM using various high order kernels that have been derived and analyzed in the previous section.

5.1. Library description

In this library we use object oriented programming techniques to reach a high level of modularity, with a strong focus on usability and flexibility. The goal is to enable the user to launch indifferently sequential, parallel or hybrid numerical simulations. We use Python as an abstraction framework.

Our library currently provides two levels of parallelism, MPI and OpenCL. However, thanks to the modularity it would also be possible to implement other parallelism paradigms such as task parallelism.

In order to achieve good portability, the computational frameworks are written using OpenCL C whenever it will enable good performances on the target architecture. OpenCL is an open standard for parallel programming of heterogeneous systems [21]. It provides application programming interfaces to address hybrid platforms containing many CPUs and GPUs and a programming language based on C99 to write instructions executed concurrently on the OpenCL devices. This framework is used through the PyOpenCL python interface to OpenCL [11].

In OpenCL applications, the program, that executes on the host system, must send OpenCL kernels to available devices in a command queue. The execution model of the application defines the devices and queues management. A given kernel contains an executable code that concurrently runs on device compute units which are called work-items. A memory model needs to be explicitly defined to manage data layout in the memory hierarchy. Details of these models, such as the number of work-items in a work-group or the memory access patterns, have a very strong impact on the program efficiency.

The high level of abstraction of Python allows to conceal from the user the parallel paradigms and the low level implementations of the numerical algorithms. It will participate in our goal to develop a library that is portable to various kind of hybrid architectures of modern computers.

Figure 9a shows the global design of the library from a functional point of view using concepts from [14]. In this work, a methodology is proposed for the design of object oriented codes. It relies on a distinction of components in different fields: abstract mathematical concepts are independent of their implementation and of the numerical methods. In order to use the library, the user only needs to describe his problem with high level components – mathematical operators, problem variables and physical domain description. The resolution is performed using default options *(Mathematics level)*. The user chooses the available numerical methods *(Numerical methods level)* and algorithms *(Algorithms level)*, or develop his own modules, depending on his needs. Thanks to the flexibility and modularity of Python, new modules are readily available in the library *(User interface level)* through its programming layers, illustrated on Figure 9b. In order to add his modules written in a common programming language *(Computations level)*, the user needs to develop a small Python interface. The module

may use a parallel paradigm *(HPC level)*. This design of the library allowed us to easily verify and optimize all the kernels presented below.

5.2. Implementation on GPU

The literature contains several GPU implementations of mesh-free particle methods. In the context of Smoothed Particles Hydrodynamics (SPH) for gas dynamics, [30] presents an implementation on multi-GPUs cluster for 32 millions of particles distributed on 4 GPUs. A mesh-free solver for incompressible flows using a Fast Multipole Method accelerated on a large number GPU is presented in [32]. These authors report reference a 0.5 petaflop/s computation for 2048^3 particles distributed on 2048 cards for homogeneous isotropic turbulence simulations.

The advection equation is a key component in the simulation of gas dynamics. In [31], the authors show that porting the resolution of Knudsen gases by particle methods to the GPGPU results in an important speedup in the simulations. Remeshed vortex methods have also been ported on GPU within the Parallel Particle Mesh library [28]. The implementation presented in [26] is devoted to two-dimensional incompressible flows in vorticity formulation, using a multigrid finite-difference solver for the Poisson equation and the $\Lambda_{2,1}$ remeshing formula for the advection. This reference reports performances in single precision floating point from 25 frames/iterations per seconds (FPS) with 1024^2 particles – for their *fastest solver*, Euler time stepping and coarse multigrid – to 3 FPS – for their *most accurate*, fourth order Runge−Kutta time stepping and accurate multigrid. In [24], penalization was added to deal with complex boundaries and a FFT solver was used instead of the multi-grid solver. The hand coded interpolation between the Cartesian grid and the particles was dropped in favor of OpenGL shaders with improved performances. For 1024^2 particles the full Navier–Stokes solver reached 22 FPS in single precision, which represents a speedup against a similar CPU solver of about 30.

As particle-mesh interpolations are core functions in RPM, some works have been devoted to these operations. In [3, 25] an implementation based on OpenCL Images objects, allowed to obtain, for a 4096×2048 problem size, a maximum speedup of 45 against a single-threaded CPU implementation, in double precision, and 150 in single precision. The speedup drops to 2.8 when comparing to a multi-threaded CPU version (9.4 in single precision). In [3], an efficient data structure enables interesting performances on GPU for tensorial particle-mesh interpolations in 2D and 3D. This reference reports that linear interpolation exhibits lower speedups than higher order schemes and that 3D interpolations are more efficient than their 2D counterparts.

The method presented in this article differs from the previous works by the fact that we use a dimensional splitting and higher order remeshing formulas. The dimensional splitting reduces 2D and 3D problems to a collection of 1D problems. The complexity will therefore remain linear with the stencil width, but this strategy has some implications that we will see below in the memory management.

In this work, we intend to produce an efficient GPU implementation of the particle advection and remeshing steps that will be used, in future work, as part of a fluid solver. The following results will demonstrate that high order remeshing formulas can be implemented in an efficient way. As in the previous section, particles are advanced with a fourth order Runge−Kutta method. The velocity field is given at grid points at the beginning of each time-step and is linearly interpolated at particle locations corresponding to the intermediate Runge−Kutta sub-steps. We use a second order time-splitting which for each time-step alternates advection successively along the directions x, y, z, y, x. The time-step of the advection is $\Delta t/2$ in the x and y directions and Δt in the z direction.

5.2.1. From method to algorithm

In order to better understand the library performances and behavior with respect to the number of particles, we analyze the computational complexity of the algorithm. The complexity C is given for one advection and one remeshing step for N^d particles, where N is the number of particles in one direction and d is the number of directions. C depends on several parameters: c, the number of scalar quantities or vector components carried by particles; the order of the dimensional splitting (second order in our case); N_s, the remeshing stencil width;

and p, the polynomial degree of the formula:

$$C(N_s, p, c) = (C_A + C_R(N_s, p, c))(2d-1)N^d = O\left(N^d\right) \text{ [Operations]}, \tag{5.1}$$

where C_A and C_R represent the advection and remeshing complexity for one particle. Advection with a fourth order Runge−Kutta scheme and linear velocity interpolation leads to the following estimate

$$C_A = \underbrace{4 \times 2 + 3}_{\text{positions computation}} + \underbrace{3 \times 9}_{\text{linear interpolations}} + \underbrace{5}_{\text{weights combination}} = 43 \text{ [Operations]}. \tag{5.2}$$

The complexity of remeshing algorithms takes into account the polynomial evaluations, performed with the Horner's rule rearrangements. The total amount of operations corresponds to N_s polynomials of degree p, c linear combinations of the N_s elements and the N_s grid point indices.

$$C_R(N_s, p, c) = \underbrace{N_s(2p+1)}_{\text{Horner's rule}} + \underbrace{2N_s c}_{\text{linear combinations}} + \underbrace{3 + N_s}_{\text{grid indices}} = 2N_s(p+c+1) + 3 \text{ [Operations]}. \tag{5.3}$$

Similarly, we study the memory access complexity. We obtain the following evaluations

$$M_A = (\underbrace{3 \times 2 + 1}_{\text{read velocity}} + \underbrace{1}_{\text{write particle position}})P = 8P \text{ [Bytes]}, \tag{5.4}$$

$$M_R(N_s, c) = (\underbrace{1}_{\text{read particle position}} + \underbrace{(1 + N_s)c}_{\text{read and write particle quantity}})P = ((1 + N_s)c + 1)P \text{ [Bytes]}, \tag{5.5}$$

$$D(c) = \underbrace{2cP}_{\text{read and write}} \text{ [Bytes]}, \tag{5.6}$$

$$M(N_s, c) = (D(c) + M_A + M_R(N_s, c))(2d-1)N^d = P((1 + N_s)c + 5)(2d-1)N^d \text{ [Bytes]}, \tag{5.7}$$

where P equals to the size of a floating point number in bytes (4 bytes for single precision or 8 bytes for double precision). $D(c)$ denotes the memory complexity of a copy or a transposition. In this expression, reading and writing access are supposed to be equivalent.

These complexities do not take into account the algorithms optimization for targeted platforms. For instance, vectorized access or *fma* operations are not considered here.

As already mentioned semi-Lagrangian particle methods can be viewed as an alternative to forward semi-Lagrangian methods developped in [9]. The use of explicit remeshing kernels significantly increase the parallel efficiency of the method. In the method described in [9], one needs to solve for each 1D problem a periodic tridiagonal system. [33] compares various implementations for solving tridiagonal systems on GPU. They show that a cyclic reduction algorithm requires at least $\log_2 N - 2$ algorithmic steps and $5N$ accesses to global memory. For the $\Lambda_{4,2}$ formula, with stencils using 6 points, the remeshing algorithm only requires 1 algorithmic step, and $2N$ accesses to global memory. For problems with at least 64 particles in one direction, the semi-Lagrangian particle methods proposed in this paper require minimal access to global and shared memories.

5.2.2. From algorithm to GPU implementation

The choice of the OpenCL execution model constrained us to an in-order OpenCL kernels call sequence. The main bottleneck of this model is the host-device data transfer rate. The data should stay on the device as long as possible. However, host-device transfers are mandatory for regular checkpointing and shared computations between CPU and GPUs or multiple GPUs. The index space for calling OpenCL kernels follows the intrinsic dimension decomposition of the method: each work-group is in charge of a 1D problem. The kernels performances are strongly connected to the layout of work-items in each work-groups. Section 5.3 will detail the different strategies for the work-items layout.

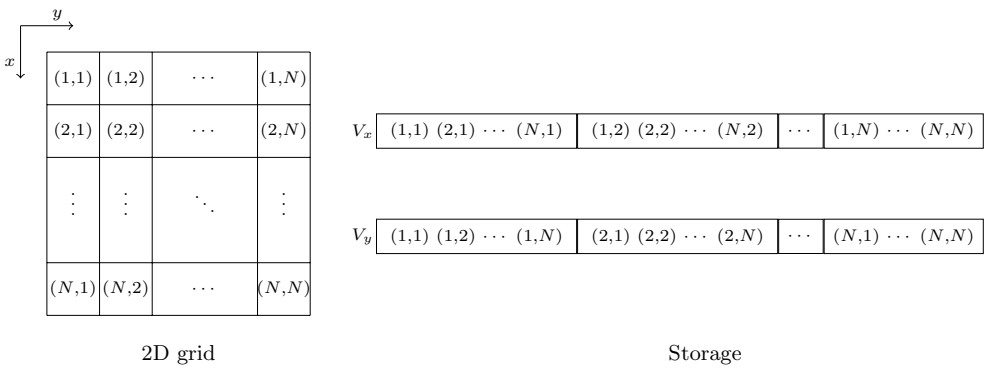


FIGURE 10. Velocity field storage.

The computation of one splitting step requires only the velocity component and the 1D coordinate of the particles in the corresponding direction. The velocity field needs to be stored as structured arrays. The maximum bandwidth for accessing global memory is reached for contiguous data. Arrays are thus stored contiguously in the direction of use. Figure 10 illustrates the storage of the velocity components with respect to the grid layout for a 2D problem. This setup imposes the advected quantities to be transposed when alternating splitting directions.

Algorithm 1 summarizes the GPU implementation, where each part of the inner loop is performed using an OpenCL kernel.

Algorithm 1: Particle method.

while $t \leqslant T_{end}$ **do**
  Computation of velocity field a_G {*on grid G*} ;
  {*Solve 1D problems in each direction*};
  foreach *splitting direction* **do**
    Initialize particles ;
    Compute particle positions {*from corresponding velocity component*} ;
    Compute particle remeshing {*in corresponding direction to grid G*};
  $t \leftarrow t + dt$

5.3. Description of the OpenCL kernels

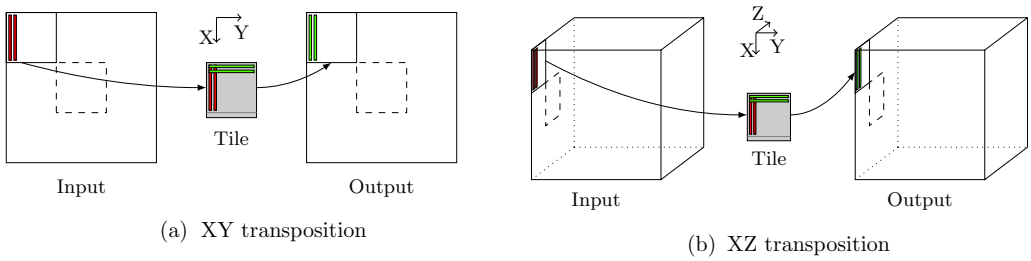
All our tests were performed on a NVIDIA Tesla K20m. The main specifications of this device are given in Table 2. Obviously, the global and local memory sizes and work-group size set some limits in our performance analysis. Each kernel used in Algorithm 1 must be studied separately to obtain the most efficient execution model characteristics such as *e.g.* work-group size, work-item workload and memory levels usage. We considered single and double precision implementations of the various remeshing kernels considered in the previous sections. Depending on the type of kernels, different optimizations – concerning OpenCL vector types, built-in functions, temporary private variables, bank conflicts avoidance, ... – were tested.

5.3.1. Memory management related to method and hardware

From the data on the grid, we update the advected quantities on particles. Data initialization depends on the alternating splitting direction in which advection and remeshing are performed. Computational efficiency requires the data to be contiguous in the working direction. Therefore, one needs to change the data layout in memory for each direction. Data initialization consists in transpositions except in the cases of unchanged

TABLE 2. NVIDIA Tesla K20m main features.

Device	OpenCL version	1.1	Global memory	Size	4 GB
	Compute units	13		Bandwidth	208 GB/s
	Maximum work-items	[1024,1024,64]		Bus width	320 bit
	Work-group size	1024		Max. allocation	1 GB
	Clock rate	705 MHz	Local memory	Size	48 KB
Host	Connection type	PCIe 2.1 ×16	Processing power	Double Precision	3.52 TFlops
	Bandwidth	8 GB/s		Single Precision	1.17 TFlops



(a) XY transposition

(b) XZ transposition

FIGURE 11. Transposition kernels.

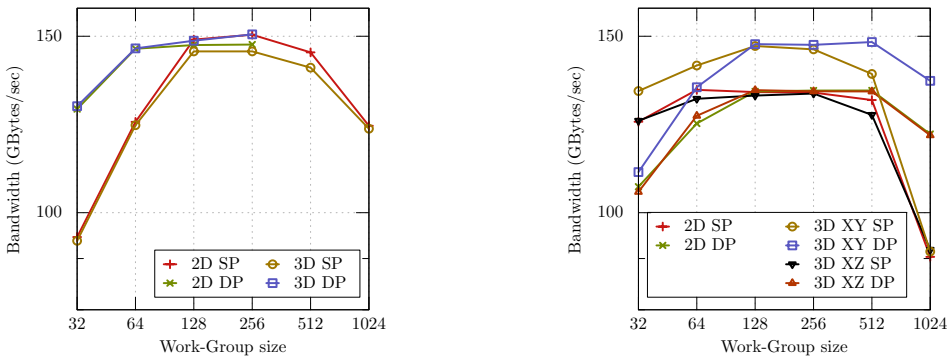
successive directions where a simple copy is sufficient. For a 2D problem, efficient transposition algorithms on GPU have been designed in [27]. This algorithm reaches 96 percent performance of the simple copy bandwidth for a 2048^2 square matrix in single precision. We implemented this algorithm and extended it to 3D. In order to achieve good performances, global memory accesses should be contiguous, fit the bus width and avoid memory camping. Memory camping occurs when successive work-groups operate on the same global memory page. It affects global efficiency of the algorithm and should be avoided as much as possible. Each work-group is handling a subset of the data, called a tile, which is stored in local memory. The tile is transposed locally, requiring bank conflict free access.

We denote by XY transposition the transposition operating on the first and second directions in which the data are contiguous. Similarly, XZ transposition operates on the first and third directions.

The XY transposition follows [27] except that, in 3D, it is performed on every slices along Z direction. Figure 11a illustrates this transformation. Each work-group loads a tile (grey square) into local memory (red). Each work-item transposes its own subset of data by writing data to their correct positions (green). Contiguous access in global memory is ensured by use of the tile and the bus width is filled up by wrapping elements into OpenCL vector types. Memory camping is avoided by roaming the global memory arrays diagonally – in dashed lines in the Figure. Finally a one row padding is used in tiles to avoid bank conflicts.

Similarly, XZ transpositions are performed along slices in Y direction. The work of [27] is extended to deal with 3D data. Using the same conventions as previously, Figure 11b sketches this transformation. To avoid memory camping, the global memory is roamed in XZ planes. Because of the larger strides when going along Z direction than in Y, we used a cubic tile to reduce the number of large strides.

Performances of the kernels associated to copy and transposition are presented on Figures 12a and 12b. These results concern fully optimized kernels and the best configurations. From Figure 12a we conclude that using 256 work-items in work-groups lead to performances close to 72 percent of the theoretical bandwidth. The remaining gap can be explained by the presence of the NVIDIA Error Correction Code (ECC) that checks and eventually corrects bit errors in data storage and transmissions. The performances should increase when disabling ECC



(a) Copy kernels (b) Transposition kernels

FIGURE 12. Performances of Copy, XY and XZ transpositions kernels for 2D and 3D, single (SP) and double precision (DP).

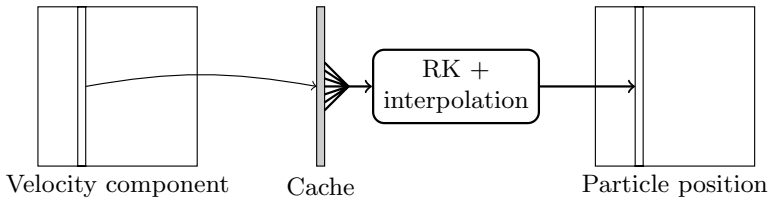


FIGURE 13. Advection algorithm.

because it uses a significant part of the bandwidth and 12.5 percent of the memory[2] to work. One can notice that the performance of the XY transposition is better in 3D than in 2D, very close to the performance of the copies for work-group size up to 512. This is due to smaller stride when moving work-group diagonally in 3D. In the optimal configuration, with respect to the work-group size, the XZ transposition exhibits performances similar to the 2D XY transposition.

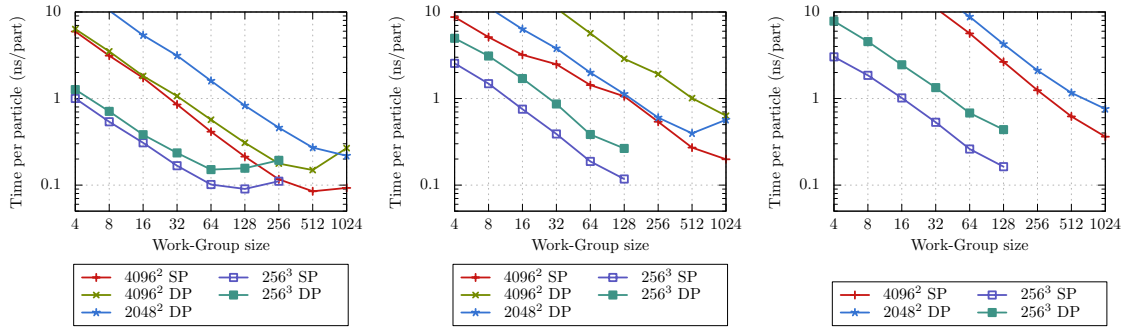
5.3.2. Advection

This kernel computes the particles positions by means of a Runge−Kutta scheme and a 2D or 3D linear interpolation of velocities on particles from known grid values. As suggested by (5.4), the fourth order Runge−Kutta method requires 8 reads in the velocity field. We bring it to a single read using local memory as a cache as illustrated on Figure 13. After filling up the cache, each work-group performs the computations from this local buffer. Results are directly stored contiguously in global memory. Using local memory, (5.4) becomes:

$$M_{A,\mathrm{GPU}} = 2P \text{ [Bytes]}. \tag{5.8}$$

Benchmarks for this kernel are shown on Figure 14a. The maximum work-group size allowed on the device is 1024, as given in Table 2. Performances increase with the work-group size until they reach a plateau around 128 work-items in 3D and 512 in 2D. Note that constraints on the global memory size and the bus width limit 3D cases to small work-group sizes.

[2]From http://www.nvidia.com/object/tesla-servers.html



(a) Advection kernels (RK4) (b) Remeshing kernels ($\Lambda_{2,1}$) (c) Monolithic kernels (RK4 + $\Lambda_{2,1}$)

FIGURE 14. Benchmarks for computational kernels in single (SP) and double precision (DP).

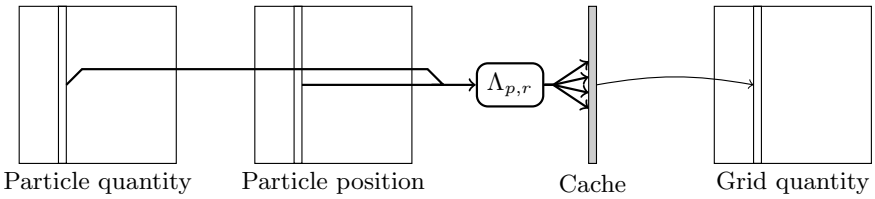


FIGURE 15. Remeshing algorithm.

5.3.3. Remeshing

Remeshing particles on the grid involves particle positions and strengths that are read once in global memory. Each work-group computes new quantities on the grid in a local buffer to avoid multiple writes in global memory, as shown on Figure 15. Again, the number of global memory accesses decreases and (5.5) becomes:

$$M_{R,\mathrm{GPU}} = (2c+1)P \text{ [Bytes]}. \tag{5.9}$$

In case of large velocity gradients, it may happen that several particles have the same remeshing points. To avoid concurrent writings, computational load is organized so that any pair of work-items do not remesh contiguous particles.

As for the advection benchmarks, Figure 14b shows that performances increase with the work-group size until it reaches a plateau.

5.3.4. Monolithic advection and remeshing

This kernel combines the advection and the remeshing computations in a single kernel. The particle positions are computed directly by the kernel, which reduces global memory occupation. The number of lunched OpenCL kernels is also reduced. However, it requires more local memory, as illustrated on Figure 16. For this kernel, the global memory complexity, given by (5.7) is reduced to:

$$M_{\mathrm{GPU,Monolithic}}(N_s, c) = P(2c+1)(2d-1)N^d \text{ [Bytes]} \tag{5.10}$$

instead of:

$$M_{\mathrm{GPU}}(N_s, c) = P(2c+3)(2d-1)N^d \text{ [Bytes]} \tag{5.11}$$

for separate kernels.

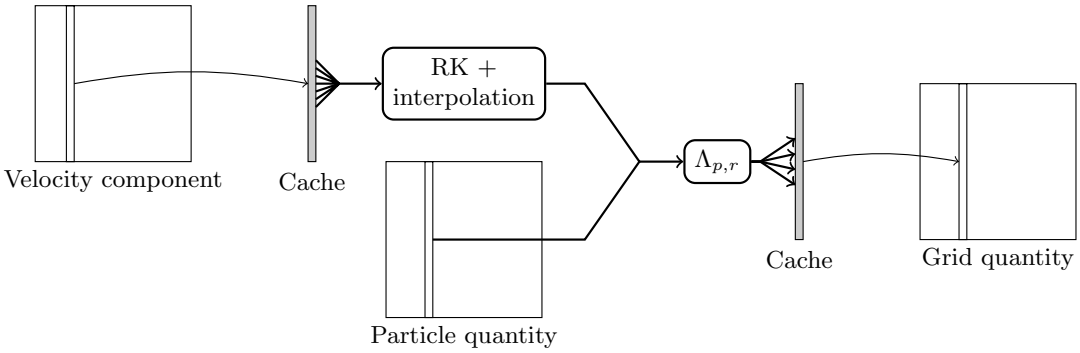


FIGURE 16. Monolithic advection and remeshing algorithm.

Again similar performances are obtained for monolithic kernel where results are close to the sum of the separate ones. Because of local memory limitations to 48 kBytes, we were not able to compute the 4096^2 double precision case (which would require 64 kBytes).

5.4. Performances

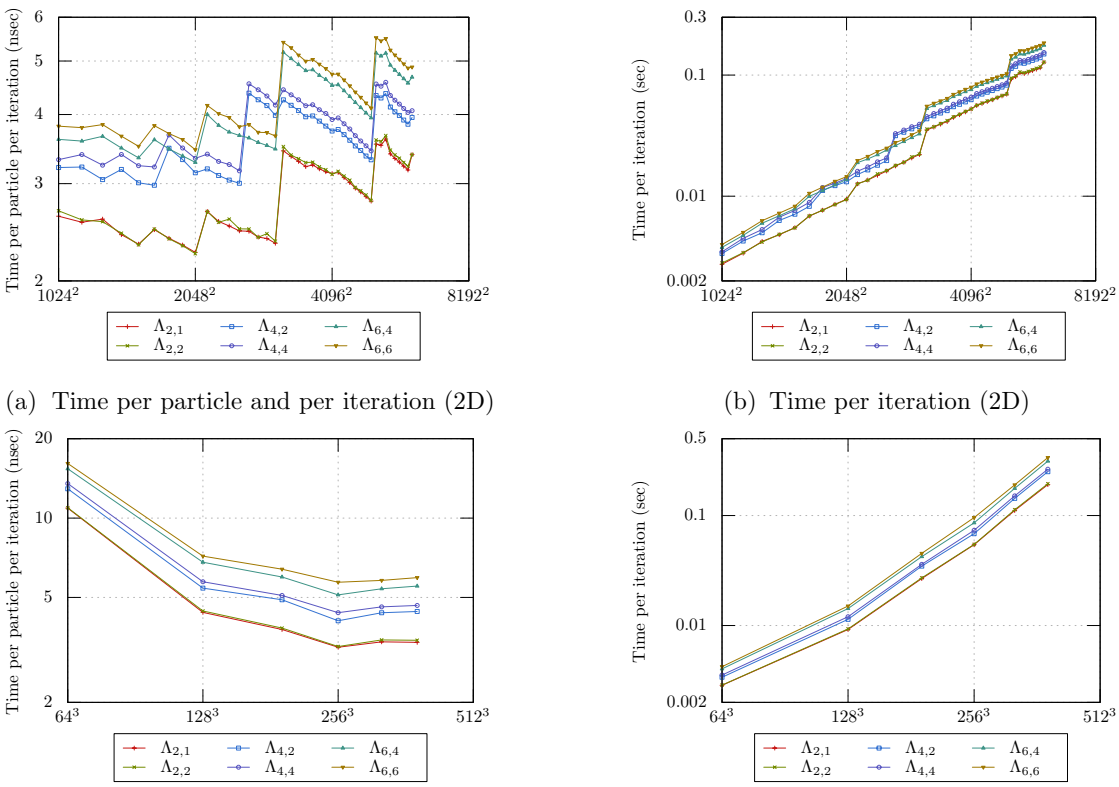
The whole code is profiled with the best configurations determined by the previous benchmarks. In this section, for a sake of simplicity we have restricted the discussion to calculations with double precision arithmetic.

Figure 17 gives the global performances for 2D and 3D problems and different remeshing formulas. The measured time includes kernel execution and OpenCL host code such as queue management, kernels launching, profiling events, ... Figure 17a shows that the time needed to compute one step for one particle is nearly constant, especially for small problems. This is consistent with (5.1), since the complexity per particle and per iteration, $(C_A + C_R(N_s, p, c))(2d - 1)$, does not depend on the problem size. The variations are due to built-in mechanisms of the device depending on the memory occupancy. In 3D, Figure 17a shows that, except for the 64^3 resolution, the computational time is also constant.

Similarly, the global complexity, $C(N_s, p, c)$, in (5.1) grows linearly with the problem size, N^d. This linearity is observed for the time per iteration on Figures 17b and 17d. For the 256^3 and 4096^2 resolutions, performances range from 10 to 20 FPS.

We provide on Figure 18 profiling details for the largest cases, corresponding to the 256^3 and 4096^2 particles. As expected, most of the time is devoted to particle remeshing. Remeshing times are grouped with respect to the stencil width. Table 3 exhibits the remeshing computational time per particle averaged – from data on Figures 17a and 17c – on different problem sizes together with the arithmetic complexity – from (5.3) – and the stencil width for the different remeshing formulas. In the *Ratio* columns, we compute the costs using the $\Lambda_{2,1}$ formula as a reference. We observe that the cost of remeshing depends mostly on the stencil width and only slightly on the polynomial computational complexity. The reason is that memory accesses are a bottleneck for remeshing kernels and this confirms the efficiency of GPU to accelerate algorithms with high computational intensity. The number of arithmetic operations plays only a marginal role in performances. Note that if the traditional tensor-product formulas were used in 3D, instead of the directional splitting used here, the ratio between the kernels $\Lambda_{6,6}$ and $\Lambda_{2,1}$ would be of order 8 instead of 2.

The dimensional splitting is also advantageous from the point of view of memory access. The number of operations required for a tensorial formula is of order N_s^3 instead of $5N_s$ for the splitting. In a tensor-product remeshing algorithm, $2N_s^3$ particles would need to be transferred to and from the global memory. Cache optimization allows to reduce the number of transfers per particle, but due to the limited size of the shared memory, this reduction is limited. For instance, with the GPU described in Table 2, one can only load 12^3 particles in



(a) Time per particle and per iteration (2D)

(b) Time per iteration (2D)

(c) Time per particle and per iteration (3D)

(d) Time per iteration (3D)

FIGURE 17. Overall performances for different problem sizes in double precision. (lower is better). Times are averaged over 20 independent executions.

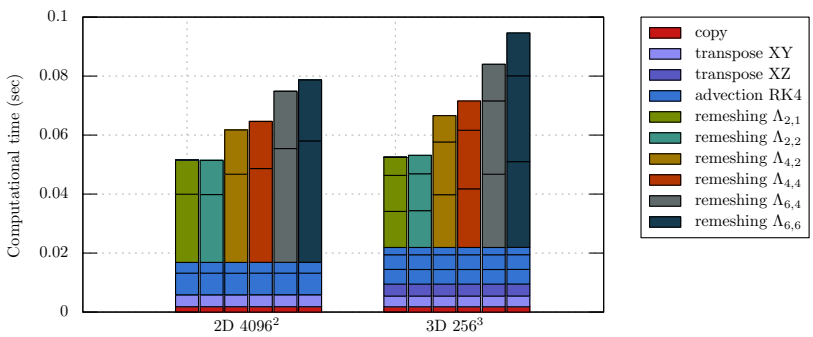


FIGURE 18. Code profiling of one time step for different problem sizes in double precision.

TABLE 3. Influence of remeshing formulas on performances. Times are averaged among the different runs reported in Figure 17b for the 2D cases and Figure 17c for the 3D cases.

	Arithmetics		Stencil		2D Time		3D Time	
Formula	$C_R(N_s, p, c)$	Ratio	Points	Ratio	Average	Ratio	Average	Ratio
$\Lambda_{2,1}$	43	1	4	1	3.13	1.00	3.88	1.00
$\Lambda_{2,2}$	59	1.37	4	1	3.17	1.01	3.99	1.03
$\Lambda_{4,2}$	87	2.02	6	1.5	4.22	1.35	5.81	1.50
$\Lambda_{4,4}$	135	3.14	6	1.5	4.49	1.43	6.28	1.62
$\Lambda_{6,4}$	179	4.16	8	2	5.22	1.67	7.88	2.03
$\Lambda_{6,6}$	243	5.65	8	2	5.62	1.79	8.78	2.26

shared memory. This limitation would cause the number of transfers with global memory performed at each particle location to increase. This would significantly impact the overall performance of the remeshing step.

Our benchmarks showed that the computational time remains linear with the stencil width. Increasing the order of the numerical method enables us to leverage the processing power of the GPU. From an applicative point of view, the accuracy of the results is improved either by increasing the problem size or increasing the numerical order of the method while keeping the problem size constant, which allows to relax the constraints due to the global memory size on the GPU. Note that the remeshing formulas involve algebraic fractions for the polynomials coefficients whose numerator and denominator are increasing with the order. In the present implementation, we kept this coefficients under their rational form but this could introduce numerical errors if high order kernels are used.

Unlike in [3], for an equivalent number of particles our 3D and 2D implementations exhibit similar performances, as they both rely on 1D problems. The global efficiency of our implementations could be further improved by gathering 1D problems to better fit the device characteristics. To conclude this section and give an idea of the compared efficiency of GPU and multi-threaded CPU implementations of the particle method, let us mention that the algorithm just described yields speedups ranging between 20 and 25, depending on the remeshing formula, against the optimized Fortran MPI code used in [15] running on 8 `Xeon E5-2640` cores.

6. Conclusion

In this paper we have discussed and analyzed remeshed particle methods, from the point of view of forward semi-Lagrangian methods. We have given a systematic consistency analysis of these methods, based on the regularity and moment properties of the remeshing kernels. These consistency results are supplemented by a stability analysis valid for a class of second order and fourth order methods, under a Lagrangian CFL condition which is independent of the grid size.

Remeshed particle methods are designed to deal with advection dominated problems and we have outlined a general computational framework in which they can be combined with other numerical methods for hybrid calculations. In this context, GPU are ideally suited to the local nature of RPM. We have presented a strategy to optimize the efficiency of RPM on GPU and discussed the complexity of theses algorithms in function of the kernel accuracy. Our results show in particular that, for a given number of conserved moments, increasing the smoothness of the kernels can improve the accuracy of the RPM without added computational complexity. The performance of the algorithms for a linear transport equation in double precision ranges between 10 and 20 FPS for grid resolutions of 4096^2 in 2D or 256^3 in 3D resolutions and kernels with order of accuracy varying between 1 and 6.

Although the focus in this paper was the implementation on GPU of particle methods, the framework we have described is designed with hybrid architectures in mind. Multi-CPUs and multi-GPUs implementations of semi-Lagrangian particle methods are the topic of ongoing work.

Appendix A

For a sake of completeness, we give here the analytical formulas for the kernels used in this work.

$$\Lambda_{2,1}(x) = \begin{cases} 1 - \frac{5}{2}|x|^2 + \frac{3}{2}|x|^3 & 0 \leqslant |x| < 1 \\ 2 - 4|x| + \frac{5}{2}|x|^2 - \frac{1}{2}|x|^3 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases}$$

$$\Lambda_{2,2}(x) = \begin{cases} 1 - |x|^2 - \frac{9}{2}|x|^3 + \frac{15}{2}|x|^4 - 3|x|^5 & 0 \leqslant |x| < 1 \\ -4 + 18|x| - 29|x|^2 + \frac{43}{2}|x|^3 - \frac{15}{2}|x|^4 + |x|^5 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases}$$

$$\Lambda_{4,2}(x) = \begin{cases} 1 - \frac{5}{4}|x|^2 - \frac{35}{12}|x|^3 + \frac{21}{4}|x|^4 - \frac{25}{12}|x|^5 & 0 \leqslant |x| < 1 \\ -4 + \frac{75}{4}|x| - \frac{245}{8}|x|^2 + \frac{545}{24}|x|^3 - \frac{63}{8}|x|^4 + \frac{25}{24}|x|^5 & 1 \leqslant |x| < 2 \\ 18 - \frac{153}{4}|x| + \frac{255}{8}|x|^2 - \frac{313}{24}|x|^3 + \frac{21}{8}|x|^4 - \frac{5}{24}|x|^5 & 2 \leqslant |x| < 3 \\ 0 & 3 \leqslant |x| \end{cases}$$

$$\Lambda_{4,4}(x) = \begin{cases} 1 - \frac{5}{4}|x|^2 + \frac{1}{4}|x|^4 - \frac{100}{3}|x|^5 + \frac{455}{4}|x|^6 - \frac{295}{2}|x|^7 + \frac{345}{4}|x|^8 - \frac{115}{6}|x|^9 & 0 \leqslant |x| < 1 \\ -199 + \frac{5485}{4}|x| - \frac{32975}{8}|x|^2 + \frac{28425}{4}|x|^3 - \frac{61953}{8}|x|^4 + \frac{33175}{6}|x|^5 \\ \qquad - \frac{20685}{8}|x|^6 + \frac{3055}{4}|x|^7 - \frac{1035}{8}|x|^8 + \frac{115}{12}|x|^9 & 1 \leqslant |x| < 2 \\ 5913 - \frac{89235}{4}|x| + \frac{297585}{8}|x|^2 - \frac{143895}{4}|x|^3 + \frac{177871}{8}|x|^4 - \frac{54641}{6}|x|^5 \\ \qquad + \frac{19775}{8}|x|^6 - \frac{1715}{4}|x|^7 + \frac{345}{8}|x|^8 - \frac{23}{12}|x|^9 & 2 \leqslant |x| < 3 \\ 0 & 3 \leqslant |x| \end{cases}$$

$$\Lambda_{6,4}(x) = \begin{cases} 1 - \frac{49}{36}|x|^2 + \frac{7}{18}|x|^4 - \frac{3521}{144}|x|^5 + \frac{12029}{144}|x|^6 - \frac{15617}{144}|x|^7 + \frac{1015}{16}|x|^8 - \frac{1015}{72}|x|^9 & 0 \leqslant |x| < 1 \\ -\frac{877}{5} + \frac{72583}{60}|x| - \frac{145467}{40}|x|^2 + \frac{18809}{3}|x|^3 - \frac{54663}{8}|x|^4 + \frac{390327}{80}|x|^5 \\ \qquad - \frac{182549}{80}|x|^6 + \frac{161777}{240}|x|^7 - \frac{1827}{16}|x|^8 + \frac{203}{24}|x|^9 & 1 \leqslant |x| < 2 \\ 8695 - \frac{656131}{20}|x| + \frac{3938809}{72}|x|^2 - \frac{158725}{3}|x|^3 + \frac{2354569}{72}|x|^4 - \frac{9644621}{720}|x|^5 \\ \qquad + \frac{523589}{144}|x|^6 - \frac{454097}{720}|x|^7 + \frac{1015}{16}|x|^8 - \frac{203}{72}|x|^9 & 2 \leqslant |x| < 3 \\ -\frac{142528}{5} + \frac{375344}{5}|x| - \frac{3942344}{45}|x|^2 + \frac{178394}{3}|x|^3 - \frac{931315}{36}|x|^4 + \frac{5385983}{720}|x|^5 \\ \qquad - \frac{1035149}{720}|x|^6 + \frac{127511}{720}|x|^7 - \frac{203}{16}|x|^8 + \frac{29}{72}|x|^9 & 3 \leqslant |x| < 4 \\ 0 & 4 \leqslant |x| \end{cases}$$

$$\Lambda_{6,6}(x) = \begin{cases} 1 - \frac{49}{36}|x|^2 + \frac{7}{18}|x|^4 - \frac{1}{36}|x|^6 - \frac{46109}{144}|x|^7 + \frac{81361}{48}|x|^8 - \frac{544705}{144}|x|^9 + \frac{655039}{144}|x|^{10} \\ \qquad - \frac{223531}{72}|x|^{11} + \frac{81991}{72}|x|^{12} - \frac{6307}{36}|x|^{13}, & 0 \leqslant |x| < 1 \\ -\frac{44291}{5} + \frac{1745121}{20}|x| - \frac{15711339}{40}|x|^2 + \frac{32087377}{30}|x|^3 - \frac{7860503}{4}|x|^4 + \frac{38576524}{15}|x|^5 \\ \qquad - \frac{24659323}{10}|x|^6 + \frac{84181657}{48}|x|^7 - \frac{74009313}{80}|x|^8 + \frac{17159513}{48}|x|^9 \\ \qquad - \frac{7870247}{80}|x|^{10} + \frac{438263}{24}|x|^{11} - \frac{81991}{40}|x|^{12} + \frac{6307}{60}|x|^{13}, & 1 \leqslant |x| < 2 \\ 3905497 - \frac{424679647}{20}|x| + \frac{3822627865}{72}|x|^2 - \frac{2424839767}{30}|x|^3 + \frac{3009271097}{36}|x|^4 \\ \qquad - \frac{930168127}{15}|x|^5 + \frac{305535494}{9}|x|^6 - \frac{9998313437}{720}|x|^7 + \frac{203720335}{48}|x|^8 - \frac{137843153}{144}|x|^9 \\ \qquad + \frac{22300663}{144}|x|^{10} - \frac{6126883}{360}|x|^{11} + \frac{81991}{72}|x|^{12} - \frac{6307}{180}|x|^{13}, & 2 \leqslant |x| < 3 \\ -\frac{255622144}{5} + \frac{971097344}{5}|x| - \frac{15295867328}{45}|x|^2 + \frac{5442932656}{15}|x|^3 - \frac{2372571796}{9}|x|^4 \\ \qquad + \frac{2064517469}{15}|x|^5 - \frac{9563054381}{180}|x|^6 + \frac{2210666335}{144}|x|^7 - \frac{796980541}{240}|x|^8 \\ \qquad + \frac{76474979}{144}|x|^9 - \frac{43946287}{720}|x|^{10} + \frac{343721}{72}|x|^{11} - \frac{81991}{360}|x|^{12} + \frac{901}{180}|x|^{13} & 3 \leqslant |x| < 4 \\ 0 & 4 \leqslant |x| \end{cases}$$

$$\Lambda_{8,4}(x) = \begin{cases} 1 - \frac{205}{144}x^2 + \frac{91}{192}x^4 - \frac{6181}{320}x^5 + \frac{6337}{96}x^6 - \frac{2745}{32}x^7 + \frac{28909}{576}x^8 - \frac{3569}{320}x^9 & 0 \leqslant |x| < 1 \\ -154 + \frac{12757}{12}x - \frac{230123}{72}x^2 + \frac{264481}{48}x^3 - \frac{576499}{96}x^4 + \frac{686147}{160}x^5 & \\ \qquad - \frac{96277}{48}x^6 + \frac{14221}{24}x^7 - \frac{28909}{288}x^8 + \frac{3569}{480}x^9 & 1 \leqslant |x| < 2 \\ \frac{68776}{7} - \frac{1038011}{28}x + \frac{31157515}{504}x^2 - \frac{956669}{16}x^3 + \frac{3548009}{96}x^4 - \frac{2422263}{160}x^5 & \\ \qquad + \frac{197255}{48}x^6 - \frac{19959}{28}x^7 + \frac{144545}{2016}x^8 - \frac{3569}{1120}x^9 & 2 \leqslant |x| < 3 \\ -56375 + \frac{8314091}{56}x - \frac{49901303}{288}x^2 + \frac{3763529}{32}x^3 - \frac{19648027}{384}x^4 + \frac{9469163}{640}x^5 & \\ \qquad - \frac{545977}{192}x^6 + \frac{156927}{448}x^7 - \frac{28909}{1152}x^8 + \frac{3569}{4480}x^9 & 3 \leqslant |x| < 4 \\ \frac{439375}{7} - \frac{64188125}{504}x + \frac{231125375}{2016}x^2 - \frac{17306975}{288}x^3 + \frac{7761805}{384}x^4 - \frac{2895587}{640}x^5 & \\ \qquad + \frac{129391}{192}x^6 - \frac{259715}{4032}x^7 + \frac{28909}{8064}x^8 - \frac{3569}{40320}x^9 & 4 \leqslant |x| < 5 \\ 0 & 5 \leqslant |x| \end{cases}$$

Acknowledgements. The authors are grateful to M. Bergdorf, D. Rossinnelli and P. Koumoutsakos for enlightening discussions on GPU implementation and the derivation of high order kernels. The first author gratefully acknowledges support from Institut Universitaire de France. This research has been partially supported by the Agence Nationale pour la Recherche (ANR) under Contracts No. ANR-2010-COSI-0009 and ANR-2010-JCJC-091601.

References

[1] M. Bergdorf, G.-H. Cottet and P. Koumoutsakos, Multilevel adaptive particle methods for convection-diffusion equations. *SIAM Multiscale Model. Simul.* **4** (2005) 328–357.

[2] M. Bergdorf and P. Koumoutsakos, A lagrangian particle-wavelet method. *SIAM Multiscale Model. Simul.* **5** (2006) 980–995.

[3] F. Büyükkeçeci, O. Awile and I. Sbalzarini, A portable opencl implementation of generic particle-mesh and mesh-particle interpolation in 2d and 3d. *Parallel Comput.* **39** (2013) 94–111.

[4] A. Chorin, Numerical study of slightly viscous flow. *J. Fluid Mech.* **57** (1973) 785–796.

[5] C. Cocle, G. Winckelmans and G. Daeninck, Combining the vortex-in-cell and parallel fast multipole methods for efficient domain decomposition simulations. *J. Comput. Phys.* **227** (2008) 9091–9120.

[6] C. Cotter, J. Frank and S. Reich, The remapped particle-mesh semi-lagrangian advection scheme. *Q. J. Meteorol. Soc.* **133** (2007) 251–260.

[7] G.-H. Cottet and P. Koumoutsakos, *Vortex methods*. Cambridge University Press (2000).

[8] G.-H. Cottet and L. Weynans, Particle methods revisited: a class of high order finite-difference methods. *C.R. Math.* **343** (2006) 51–56.

[9] N. Crouseilles, T. Respaud and E. Sonnendrücker, A forward semi-lagrangian method for the numerical solution of the vlasov equation. *Comput. Phys. Commun.* **180** (2009) 1730–1745.

[10] R. Hockney and J. Eastwood, *Simulation Using Particles*. Inst. Phys. Publ. (1988).

[11] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov and A. Fasih, PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Comput.* **38** (2012) 157–174.

[12] P. Koumoutsakos, Inviscid axisymmetrization of an elliptical vortex. *J. Comput. Phys.* **138** (1997) 821–857.

[13] P. Koumoutsakos and A. Leonard, High resolution simulation of the flow around an impulsively started cylinder using vortex methods. *J. Fluid Mech.* **296** (1995) 1–38.

[14] S. Labbé, J. Laminie and V. Louvet, Méthodologie et environnement de développement orientés objets: de l'analyse mathématique à la programmation. *MATAPLI* **70** (2003) 79–92.

[15] J.-B. Lagaert, G Balarac and G.-H. Cottet, Hybrid spectral particle method for turbulent transport of passive scalar. *J. Comput. Phys.* **260** (2014) 127–142.

[16] A. Leonard. Computing three-dimensional incompressible flows with vortex elements. *Annu. Rev. Fluid Mech.* **17** (1985) 523–559.

[17] R.J. LeVeque, High-resolution conservative algorithms for advection in incompressible flow. *SIAM J. Numer. Anal.* **33** (1996) 627–665.

[18] A. Magni and G.-H. Cottet, Accurate, non-oscillatory, remeshing schemes for particle methods. *J. Comput. Phys.* **231** (2012) 152–172.

[19] J. Monaghan, Extrapolating B splines for interpolation. *J. Comput. Phys.* **60** (1985) 253–262.

[20] J. Monaghan, An introduction to SPH. *Comput. Phys. Commun.* **48** (1988) 89–96.

[21] A. Munshi, The OpenCL Specification. *Khronos OpenCL Working Group* (2011).

[22] M. Ould-Salihi, G.-H. Cottet and M. El Hamraoui, Blending finite-difference and vortex methods for incompressible flow computations. *SIAM J. Sci. Comput.* **22** (2000) 1655–1674.

[23] T. Respaud and E. Sonnendruücker, Analysis of a new class of forward semi-lagrangian schemes for the 1d Vlasov-Poisson equations. *Numer. Math.* **118** (2011) 329–366.

[24] D. Rossinelli, M. Bergdorf, G.H. Cottet and P. Koumoutsakos, GPU accelerated simulations of bluff body flows using vortex methods. *J. Comput. Phys.* **229** (2010) 3316–3333.

[25] D. Rossinelli, C. Conti and P. Koumoutsakos, Mesh-particle interpolations on graphics processing units and multicorecentral processing units. *Philosophical Transactions of the Royal Society A: Mathematical, Phys. Engrg. Sci.* **369** (2011) 2164–2175.

[26] D. Rossinelli and P. Koumoutsakos, Vortex methods for incompressible flow simulations on the GPU. *Visual Comput.* **24** (2008) 699–708.

[27] G. Ruetsch and P. Micikevicius, Optimizing matrix transpose in cuda. *NVIDIA CUDA SDK Application Note* (2009).

[28] I. Sbalzarini, J. Walther, M. Bergdorf, S. Hieber, E. Kotsalis and P. Koumoutsakos, PPM–a highly efficient parallel particle–mesh library for the simulation of continuum systems. *J. Comput. Phys.* **215** (2006) 566–588.

[29] I. Schoenberg, Contribution to the problem of approximation of equidistant data by analytic functions. *Q. Appl. Math.* **4** (1946) 45–99.

[30] D. Valdez-Balderas, J. Dominguez, B. Rogers and A. Crespo, Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-gpu clusters. *J. Parallel Distrib. Comput.* **73** (2012) 1483–1493.

[31] F. De Vuyst and F. Salvarani, GPU-accelerated numerical simulations of the knudsen gas on time- dependent domains. *Comput. Phys. Commun.* **184** (2013) 532–536.

[32] R. Yokota, L. Barba, T. Narumi and K. Yasuoka, Petascale turbulence simulation using a highly parallel fast multipole method. *Comput. Phys. Commun.* **184** (2013) 445–455.

[33] Y. Zhang, J. Cohen and J.D. Owens, Fast tridiagonal solvers on the GPU. *SIGPLAN Not.* **45** (2010) 127–136.

Multi-scale problems, high performance computing and hybrid numerical methods

G. Balarac, G.-H. Cottet, J.-M. Etancelin, J.-B. Lagaert, F. Perignon and C. Picard

Abstract The turbulent transport of a passive scalar is an important and challenging problem in many applications in fluid mechanics. It involves different range of scales in the fluid and in the scalar and requires important computational resources. In this work we show how hybrid numerical methods, combining Eulerian and Lagrangian schemes, are natural tools to address this mutli-scale problem. One in particular shows that in homogeneous turbulence experiments at various Schmidt numbers these methods allow to recover the theoretical predictions of universal scaling at a minimal cost. We also outline how hybrid methods can take advantage of heterogeneous platforms combining CPU and GPU processors.

1 Introduction

Numerical simulations have become a routine tool to develop, prototype and/or validate products and processes in industry. Applications encompass virtually all sectors of activity from Aeronautics, Automotive industry and Oil exploration to Circuit design, Biomechanics and Animations studios, to name a few. With the need to perform more and more realistic simulations and the advent of supercomputers, available in national or regional centers, the field of High Performance Computing

G. Balarac
LEGI, CNRS and Université de Grenoble, BP 53, 38041 Grenoble Cedex 9, France, guillaume.balarac@grenoble-inp.fr

G.-H. Cottet, J.-M. Etancelin, F. Perignon and C. Picard
Laboratoire Jean Kuntzmann, CNRS and Université de Grenoble, BP 53, 38041 Grenoble Cedex 9, France, georges-henri.cottet@imag.fr

J.-B. Lagaert
Laboratoire de Mathématiques, Université Paris 11, 91405 Orsay Cedex, France, jean-baptiste.lagaert@math.u-psud.fr

(HPC) is not anymore restricted to academia and scientific grand challenges but starts to reach SMEs.

HPC requires easy and flexible access to HPC facilities, obviously, and to master the appropriate programming language, but also to question the numerical methods and algorithms that are used in the simulations. These methods and algorithms should be adapted both to the physics of the problems to solve and to the architecture of the simulation platforms. Moreover since these platforms are often of an hybrid nature, that is combine different type of processors, typically CPU and GPU processors, one may also wish to develop or use methods which couple different types of algorithms that can be optimally distributed to different types of processors.

This is particularly desirable if the problem to solve is multi-level by nature. In that case the different scales that are to be represented can also be resolved on different types of processors. This is hybrid computing. In some sense the nature of the problem and of the hardware inspires the type of mathematical and numerical models that should be used for optimal efficiency.

The purpose of this paper is to describe ongoing work in our group towards hybrid computing for applications in turbulent transport of passive scalar. In the next section we briefly describe the physical context of this work. In section 3 we describe a hybrid method coupling a semi-Lagrangian method for the scalar transport and a spectral method for incompressible flows and we show some results obtained with this method. Section 4 is devoted to the implementation of scalar transport on GPU processors.

2 Universal scaling in turbulent transport

The prediction of the dynamics of a scalar advected by a turbulent flow is an important challenge in many applications. Some of these applications are illustrated in Figures 1 to 3. Figure 1 shows the dynamics of a pollutant ejected by a sewer in the Los Angeles bay at two different times. Figure 2 shows the atomization of a jet. In that case the transported quantity is the interface water-air interface [1]. Figure 3 shows a similar experiment but in the context of combustion. In this case the transported quantities are concentrations of chemical species [2]. All these illustrations share a common feature, namely that very small scales spontaneously appear and need to be captured if accurate predictions are needed for the location of the pollutant, the size of the droplets or the combustion efficiency, respectively, are sought.

The production of small scales in an advected scalar indeed reflects some fundamental turbulence properties and is driven by the value of the Schmidt number, Sc, the ratio between the viscosity of the fluid and the diffusivity of the scalar. if $Sc > 1$, the so-called Batchelor scale η_B which measures the size of the smallest scalar fluctuations is smaller than the smallest length scales of the turbulent flow (the Kolmogorov scale η_K). These scales are related by $\eta_B = \eta_K/\sqrt{Sc}$. More precisely, for $Sc > 1$, Batchelor [3] reports that the classical Corrsin-Obukhov cascade

associated with a $k^{-5/3}$ law (where k is the wave number) for the scalar variance spectrum [4, 5] is followed by a viscous-convective range with a k^{-1} power law. This viscous-convective range is followed by the dissipation range, where various theoretical scalings have been proposed for the spectrum [3, 6]. A direct consequence of this fact is that, for $Sc > 1$, in numerical simulations the scalar is more demanding, in terms of grid resolution, than the flow itself. It is therefore natural to envision numerical approaches which use different grid resolutions for the scalar and the momentum.

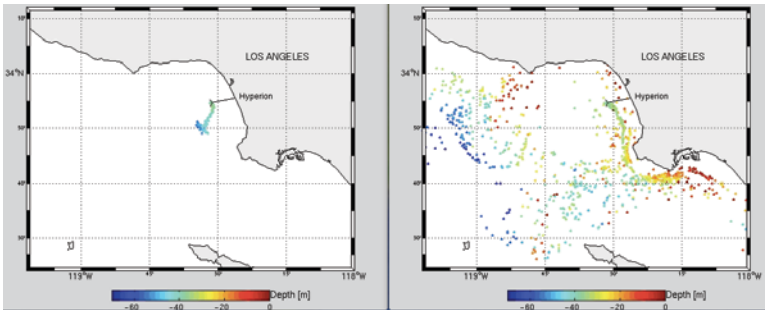


Fig. 1 Transport of a pollutant in the bay of Los Angeles at two different times. Courtesy of E. Blayo (Université Joseph Fourier, Grenoble)

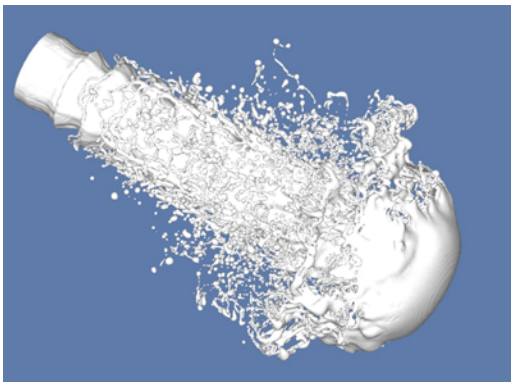


Fig. 2 Atomization of a jet (Courtesy of S. Zaleski, Université Pierre et Marie Curie, Paris).

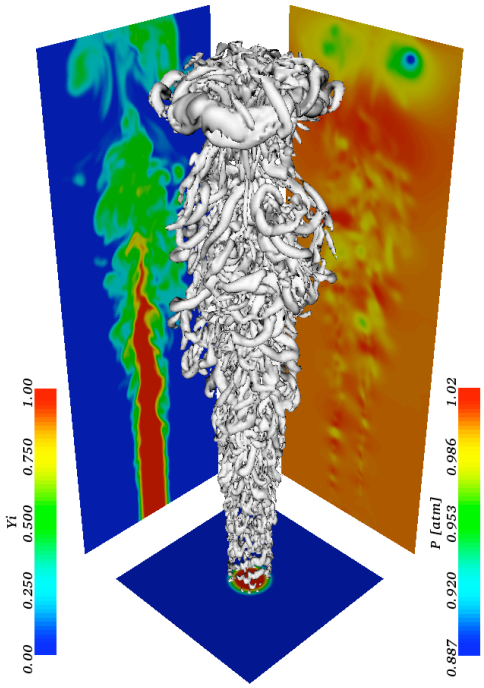


Fig. 3 Reacting jet . Courtesy of L. Vervisch, INSA Rouen.

3 Hybrid particle-spectral method

We consider in the following scalar equation

$$\frac{\partial \theta}{\partial t} + \mathbf{u} \cdot \nabla \theta = \nabla \cdot (\kappa \nabla \theta) \tag{1}$$

coupled with the incompressible Navier-Stokes equation

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \nabla \cdot (\nu \nabla \mathbf{u}) - \nabla p \, , \; \nabla \cdot \mathbf{u} = 0, \tag{2}$$

in a periodic box. κ is the molecular scalar diffusivity, ν the flow viscosity and $\mathbf{u}$ the flow velocity field. Using different grid resolutions for the scalar and the flow has already been considered for instance in [7, 8]. In the latter reference a compact finite-difference method was used for the scalar and a pseudo-spectral method for the flow. A significant speed-up over a pure spectral solver with high resolution for

both the momentum and the scalar was obtained. Our choice here is to combine a particle method for the scalar and a spectral method for the flow.

Our motivation to choose a particle method for the scalar advection comes form the fact that, for large Schmidt numbers, the scalar dynamics is essentially driven by advection, a regime for which Lagrangian or semi-Lagrangian methods are well suited. Moreover in such method, the stability limits for the time-step are governed by the amount of strain in the flow, and not by the grid size. In the present context where high resolutions of the scalar are desired, this is definitely a feature that is expected lead to an important speed up.

More precisely, the method we use for the scalar is a semi-Lagrangian (or remeshed) particle method, where at every time step particles carrying the scalar values are moved along the streamlines of the velocity then remeshed on a regular cartesian grid. Remeshing particles on a regular grid is a way to guarantee the accuracy of particle methods. This approach has been systematically used and validated in a number of simulation of vortex flows [9, 10, 11, 12, 13] or in combination with level set methods for interface capturing [14, 15, 16]. Remeshing particles at every time-step also allows to easily couple the method to grid based methods, in particular when velocity values are computed on a grid. These methods can be summarized by the following formula

$$\theta_i^{n+1} = \sum_j \theta_j^n \Gamma\left(\frac{x_j^{n+1} - x_i}{\Delta x^\theta}\right). \tag{3}$$

where θ_j^n denotes the value of the scalar at the grid point x_j and at time $t_n = n\Delta t^\theta$, x_j^{n+1} is the location after one advection step of the particle initialized at time t_n on the grid point x_j, and Δx^θ and Δt^θ denote the grid size and the time-step. In the above formula Γ is an interpolating kernel, the smoothness and the moment properties of which govern the spatial overall accuracy of the method [17]. In this work we chose the following kernel second order kernel

$$\Gamma(x) = \begin{cases} \frac{1}{12}(1-|x|)\left(25|x|^4 - 38|x|^3 - 3|x|^2 + 12|x| + 12\right) & \text{if } 0 \le |x| < 1 \\ \frac{1}{24}(|x|-1)(|x|-2)\left(25|x|^3 - 114|x|^2 + 153|x| - 48\right) & \text{if } 1 \le |x| < 2 \\ \frac{1}{24}(3-|x|)^3(5|x|-8)(|x|-2) & \text{if } 2 \le |x| < 3 \\ 0 & \text{if } 3 \le |x|. \end{cases}$$

The scalar time-step is given by $\Delta t^\theta = (|\nabla \mathbf{u}|_{max})^{-1}$. As already mentioned this value does not depend on the scalar grid size.

For the momentum equation we use a classical pseudo-spectral method, with the 3/2 rule to de-alias inertial terms and a second-order Runge-Kutta scheme is used both for the time-stepping of the spectral method and to advect particles. Precise descriptions of the methods and of the experimental set up are given in [18, 17].

Figure 4 shows a comparison of the scalar spectra obtained by the present coupling method and pure spectral method in an experiment of decaying homogeneous

turbulence. In the hybrid method two different resolutions where used for the scalar. This experiment shows that, provided the particle method is used with slightly more grid points than needed by the spectral method, the scalar values are well recovered all the way to the dissipation scale. In this Direct Numerical Simulation, the Schmidt number was equal to 50 and the momentum equation was solved with 256 modes in each direction.

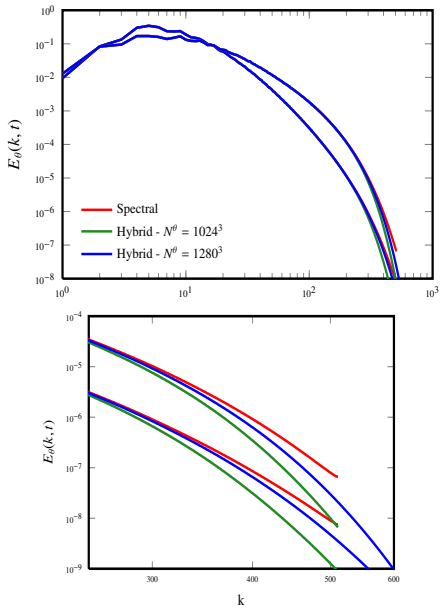


Fig. 4 Spectra of the scalar variance $E_\theta(k,t)$ at two two different times for $Sc = 50$. Right picture is a zoom of left picture on the smallest scales.

To evaluate the efficiency of the hybrid method, we show in Table 1 CPU times for the full spectral method and the hybrid method for $Sc = 50$. All runs correspond to fully resolved simulations for the Navier-Stokes equations. One can see that, because it can use much larger time-steps, the hybrid method, even when it uses slightly more points to accurately resolve the finest scales, leads to significant savings over the pure spectral method. Additional validation and diagnostics are give in [18].

The computational efficiency of the hybrid method allows to address more challenging cases and to investigate in a systematic fashion the universal scaling laws in the case of forced homogeneous turbulence. Table 2 summarizes the simulation set up corresponding to two values of the Reynolds number and several Schmidt numbers. For the highest Schmidt number, $Sc = 128$, the simulation used 3064^3 compu-

Method	N^u	N^θ	Δt^u ($\times 10^{-4}$)	Δt^θ ($\times 10^{-4}$)	total CPU time
Spectral	1024^3	1024^3	2.5	2.5	43 590s
Spectral	256^3	1024^3	2.5	2.5	16 671s
Hybrid	256^3	1024^3	10	100	1 139s
Hybrid	256^3	1280^3	10	100	1 328s

Table 1 Numerical efficiency of the different methods on a decaying turbulence experiment- Runs are performed on 2048 cores of a Blue Gene Q. N^u, N^θ denotes the spatial resolution for velocity and scalar and $\Delta t^u, \Delta t^\theta$ are the numerical time steps for momentum and scalar equations. CPU times correspond to the simulation time $t = 6$.

tational elements for the scalar equation, on a IBM Blue Gene supercomputer. The ratio between the time-step used in the present simulation and that which would have been required in a comparable spectral simulation is almost equal to 100. Figure 5 shows the compensated spectra of the scalar for a Reynolds number based on the Taylor mico-scale R_λ [21] equal to 130. These spectra do exhibit a k^{-1} decay on a range which increases with the Schmidt number. Beyond this viscous-convective range, the spectra follow an exponential decay coinciding with the scaling law proposed by Kraichnan [6].

R_λ	N^u	Δt^u	Sc	N^θ	Δt^θ	$\Delta t^\theta_{\text{spec}}$
			0.7	512^3		$6e^{-3}$
			4	1024^3		$3e^{-3}$
			8	1024^3		$3e^{-3}$
130	256^3	$1.2e^{-2}$	16	1536^3	$8.6e^{-2}$	$2e^{-3}$
			32	1536^3		$2e^{-3}$
			64	2048^3		$1.5e^{-3}$
			128	3064^3		$1e^{-3}$
210	512^3	$3e^{-3}$	0.7	770^3	$2e^{-2}$	$2e^{-3}$
			4	1024^3		$1.5e^{-3}$

Table 2 Setup of simulations performed in forced homogeneous turbulence. Δt^u is the time step used to solve the Navier-Stokes equation with a pseudo-spectral solver. Δt^θ is the time step used to solve the scalar transport equation with the particle method. $\Delta t^\theta_{\text{spec}}$ is the time step which would be needed if a pseudo-spectral method was used for the same number of scalar grid points [18]

Figure 6 shows vorticity and scalar contours in a cross section of the computational box for the simulation corresponding $R_\lambda \approx 130$ and $Sc = 128$. It illustrates the scale separation between the flow and the scalar for these parameters. The extension of the hybrid method to the coupling of particle methods with finite-volume methods to address engineering configurations is under way.

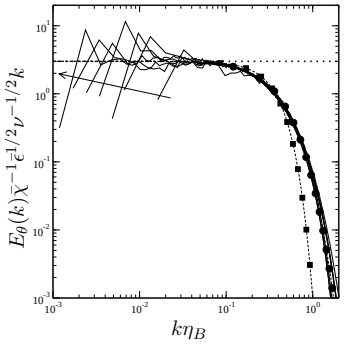


Fig. 5 Compensated spectra for the scalar variance at $R_\lambda \approx 130$. The arrow shows the direction of increasing Schmidt numbers. In the dissipative region, the circles show the law proposed by Kraichnan and the squares show the law proposed by Batchelor in the dissipative scales. The vertical axis shows the spectra compensated by the Batchelor law predicting a k^{-1} decay in the intermediate scale.

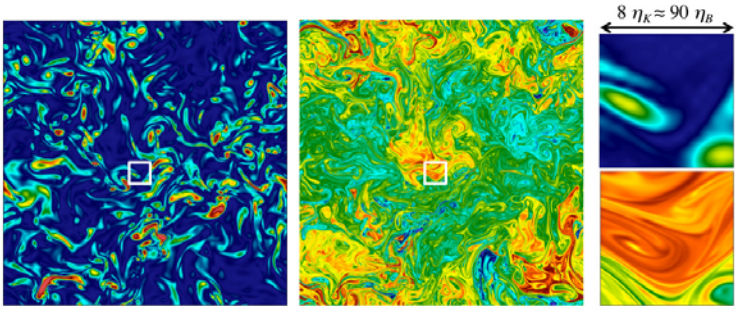


Fig. 6 Cross-section colored by the vorticity magnitude (left, blue regions are for the lowest vorticity values and red regions are for the highest vorticity values) and by the passive scalar (middle, blue regions are for the lowest scalar values and red regions are for the highest scalar values) for $R_\lambda \approx 130$ and $Sc = 128$. The zooms (right) for the vorticity magnitude (top) and the scalar (bottom) correspond to the white box with a length approximately equal to the Kolmogorov scale.

4 Towards hybrid computing

As already mentioned the multi-scale nature of turbulent transport makes natural the idea of hybrid computing methodologies where different part of the problems are distributed to different types of hardware. To be able to implement hybrid algorithms on hybrid architectures, one needs to develop frameworks and libraries with a high level description which allows to distribute different solvers and grids

to different parts of the clusters in a seamless fashion. Both particle advection and particle remeshing are local operations. This limits the communications between computational elements and makes semi-Lagrangian particle methods well suited to parallel implementations [20]. Such an implementation is described in [19] for the 2D Navier-Stokes equations and in [17] for 3D linear transport equations. In [17], to achieve good portability, the computational frameworks are written using OpenCL.

The efficiency of GPU algorithms is very much conditioned by memory access strategies. To minimize the resulting computational overhead, we use a directional splitting where particles are pushed and remeshed successively along each direction. This allows to send a given number of independent particle lines on a single workgroup. This strategy requires to transpose data after each direction has been processes. However on modern GPU cards, transpositions can be achieved at a cost close to that of a simple copy operation. Figure 5 shows the computational cost of our GPU implementations [17] in double precision arithmetics for different remeshing kernels, for 2D and 3D experiments using about 16 million points. In these experiments a second order splitting was used to alternate one dimensional particle advection and remeshing. The number of points in the kernel stencils in each direction varied from 4 to 8 [17]. These calculations were done on a NVIDIA Tesla K20m. These performances reached between 20 and 50 % of the peak performance of the GPU, depending on the size of the stencil and represented a speed up of about 25 over a multi-threaded MPI implementation running on 8 `Xeon E5-2640` cores

Hybrid computing would consist of combining the above implementation of scalar transport at high resolution with flow calculations on CPU processors. Based on timing obtained in our CPU and GPU implementations, in the case when the full scalar grid fits on a single GPU, up to resolutions of 512^3, a target toy configuration where computational times on CPU and GPU would be similar, consist of a 128^3 flow resolution running on 8 CPUs together with a 512^3 scalar resolution running on the GPU, for an overall computational time of about 1s per iteration. To obtain such performance it is essential that communications between velocity data processed on the CPUs and the GPU are processed in an optimal way. This is the object of ongoing research.

5 Conclusion

Combining high order semi-Lagrangian and Eulerian methods is an efficient strategy to address turbulent transport problems. It allows to describe accurately the viscous-convective range and dissipation scales of the scalar at a minimal cost. This is due to the fact that semi-Lagrangian methods are not subject to CFL conditions. The local nature of particle methods naturally opens the way to hybrid computations using heterogeneous hardware.

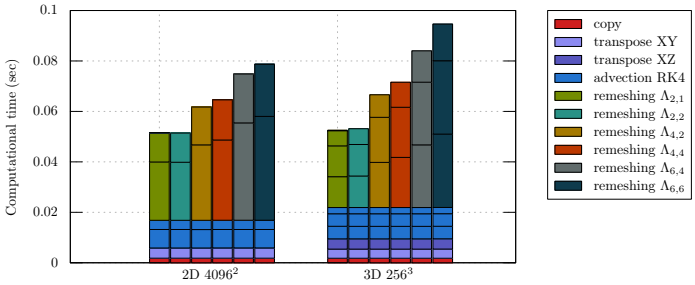


Fig. 7 Code profiling of one time step for different problem sizes in double precision.

Acknowledgments

This work was partially supported by the Agence Nationale pour la Recherche (ANR) under Contracts No. ANR-2010-JCJC-091601 and ANR-2010-COSI-0009. G.-H. C. is also grateful for the support from Institut Universitaire de France. Computations reported in section 3 were performed using HPC resources from GENCI-IDRIS (Grant 2012-020611).

References

1. G. Tryggvason, R. Scardovelli, S. Zaleski, Direct Numerical Simulations of Gas-Liquid Multiphase Flows, Cambridge University Press, 2011.
2. G. Lodato, P. Domingo, L. Vervisch, Three-dimensional boundary conditions for direct and large-eddy simulation of compressible viscous flows, Journal of Computational Physics 227 (1) (2008) 5105–5143.
3. G. K. Batchelor, Small-scale variation of convected quantities like temperature in turbulent fluid part 1. general discussion and the case of small conductivity, Journal of Fluid Mechanics 5 (01) (1959) 113–133.
4. S. Corrsin, On the spectrum of isotropic temperature fluctuations in an isotropic turbulence, Journal of Applied Physics 22 (4) (1951) 469–473.
5. A. M. Obukhov, The structure of the temperature field in a turbulent flow, Dokl. Akad. Navk. SSSR 39 (1949) 391.
6. R. Kraichnan, Small-scale structure of a scalar field convected by turbulence, Phys. Fluids 11 (1968) 945–953.
7. G.-H. Cottet, G. Balarac, M. Coquerelle, Subgrid particle resolution for the turbulent transport of a passive scalar, in: Advances in Turbulence XII, Proceedings of the 12th EUROMECH European Turbulence Conference, September, Vol. 132, 2009, pp. 779–782.
8. T. Gotoh, S. Hatanaka, H. Miura, Spectral compact difference hybrid computation of passive scalar in isotropic turbulence, Journal of Computational Physics 231 (21) (2012) 7398–7414.
9. P. Koumoutsakos, A. Leonard, High-resolution simulations of the flow around an impulsively started cylinder using vortex methods, Journal of Fluid Mechanics 296 (1995) 1–38.

10. G.-H. Cottet, B. Michaux, S. Ossia, G. Vanderlinden, A comparison of spectral and vortex methods in three-dimensional incompressible flows, J. Comput. Phys. 175 (2) (2002) 702–712.
11. G.-H. Cottet, P. Poncet, Advances in direct numerical simulations of 3d wall-bounded flows by vortex-in-cell methods, Journal of Computational Physics 193 (1) (2004) 136–158.
12. P. Ploumhans, G. Winckelmans, J. Salmon, A. Leonard, M. Warren, Vortex methods for direct numerical simulation of three-dimensional bluff body flows: Application to the sphere at Re=300, 500 and 1000, Journal of Computational Physics 178 (2) (2002) 427–463.
13. P. Poncet, Topological aspects of the three-dimensional wake behind rotary oscillating circular cylinder, J. Fluid Mech. 517 (2004) 27–53.
14. S. E. Hieber, P. Koumoutsakos, A lagrangian particle level set method, Journal of Computational Physics 210 (1) (2005) 342–367.
15. M. Bergdorf, P. K. Koumoutsakos, A Lagrangian particle-wavelet method, Multiscale Modeling and Simulation: A SIAM Interdisciplinary Journal 5 (2006) 980–995.
16. A. Magni, G. Cottet, Accurate, non-oscillatory, remeshing schemes for particle methods, Journal of Computational Physics 231 (1) (2012) 152–172.
17. G.-H. Cottet, J.-M. Etancelin, F. Perignon, C. Picard, High-order semi-lagrangian particle methods for transport equation: numerical analysis and implementation issues, to appear in ESAIM: Mathematical Modelling and Numerical Analysis.
18. J.-B. Lagaert, G. Balarac, G.-H. Cottet, Hybrid spectral-particle method for the turbulent transport of a passive scalar, Journal of Computational Physics 260 (1) (2014) 127–142.
19. D. Rossinelli, M. Bergdorf, G.-H. Cottet and P. Koumoutsakos, GPU accelerated simulations of bluff body flows using vortex methods, Journal of Computational Physics, 229 (9), 33163333, 2010.
20. I.F Sbalzarini, J.H. Walther, M. Bergdorf, S.E. Hieber, E.M. Kotsalis, P. Koumoutsakos, PPM - A highly efficient parallel particle-mesh library for the simulation of continuum systems. J. Computational Physics, 215:566-588, 2006
21. M. Lesieur, Turbulence in fluids, Fluid mechanics and its applications, Springer, Dordrecht, 2008.

Couplage de modèles, algorithmes multi-échelles et calcul hybride

Résumé :

Dans cette thèse nous explorons les possibilités offertes par l'implémentation de méthodes hybrides sur des machines de calcul hétérogènes dans le but de réaliser des simulations numériques de problèmes multiéchelles. La méthode hybride consiste à coupler des méthodes de diverses natures pour résoudre les différents aspects physiques et numériques des problèmes considérés. Elle repose sur une méthode particulaire avec remaillage qui combine les avantages des méthodes Lagrangiennes et Eulériennes. Les particules sont déplacées selon le champ de vitesse puis remaillées à chaque itération sur une grille en utilisant des formules de remaillage d'ordre élevés. Cette méthode semi-Lagrangienne bénéficie des avantages du maillage régulier mais n'est pas contrainte par une condition de CFL.

Nous construisons une classe de méthodes d'ordre élevé pour lesquelles les preuves de convergence sont obtenues sous la seule contrainte de stabilité telle que les trajectoires des particules ne se croisent pas.

Dans un contexte de calcul à haute performance, le développement du code de calcul a été axé sur la portabilité afin de supporter l'évolution rapide des architectures et leur nature hétérogène. Une étude des performances numériques de l'implémentation GPU de la méthode pour la résolution d'équations de transport est réalisée puis étendue au cas multi-GPU. La méthode hybride est appliquée à la simulation du transport d'un scalaire passif dans un écoulement turbulent 3D. Les deux sous-problèmes que sont l'écoulement turbulent et le transport du scalaire sont résolus simultanément sur des architectures multi-CPU et multi-GPU.

Mots clés :

Méthodes pariculaires ; couplage de modèles ; calcul hybride ; écoulements turbulents.

Model coupling, multi-scale algorithms and hybrid computing

Abstract :

In this work, we investigate the implementation of hybrid methods on heterogeneous computers in order to achieve numerical simulations of multi-scale problems. The hybrid numerical method consists of coupling methods of different natures to solve the physical and numerical characteristics of the problem. It is based on a remeshed particle method that combines the advantages of Lagrangian and Eulerian methods. Particles are pushed by local velocities and remeshed at every time-step on a grid using high order interpolation formulas. This forward semi-lagrangian method takes advantage of the regular mesh on which particles are reinitialized but is not limited by CFL conditions.

We derive a class of high order methods for which we are able to prove convergence results under the sole stability constraint that particle trajectories do not intersect.

In the context of high performance computing, a strong portability constraint is applied to the code development in order to handle the rapid evolution of architectures and their heterogeneous nature. An analysis of the numerical efficiency of the GPU implementation of the method is performed and extended to multi-GPU platforms. The hybrid method is applied to the simulation of the transport of a passive scalar in a 3D turbulent flow. The two sub-problems of the flow and the scalar calculations are solved simultaneously on multi-CPU and multi-GPU architectures.

Keywords :

Particle methods; model coupling; hybrid computing; turbulent flows.