UNIVERSITÉ DE LIMOGES

ÉCOLE DOCTORALE SISMI

FACULTÉ DES SCIENCES ET TECHNIQUES

Année : 2018 Thèse N° X

Thèse

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE LIMOGES

Discipline : Informatique Graphique

présentée et soutenue par

Công-Tâm TRAN

le 3 mai 2018

Simulations de fluides complexes à l'échelle mésoscopique sur GPU

Thèse dirigée par Benoît Crespin, Arnaud Videcoq et Manuella Cerbelaud

JURY :

Phillipe MESEURE	Professeur, Université de Poitiers	Président
Marie JARDAT	Professeur, Université de la Sorbonne	Rapporteur
Fabrice JAILLET	Maître de Conférences HDR, Université de Lyon	Rapporteur
Loïc BARTHE	Professeur, Université de Toulouse	Examinateur
Riccardo FERRANDO	Professeur associé, Université de Gênes	Examinateur
Benoît CRESPIN	Maître de Conférence HDR, Université de Limoges	Examinateur
Manuella CERBELLAUD	Chargés de recherche, CNRS	Examinateur
Arnaud VIDECOQ	Professeur, Université de Limoges	Examinateur

« I don't try to get close to people who don't interest me. Being relieved to know you're the same as someone else is stupid. I want to resist to get along with all those ***perfect*** *people.*

I want to become ***special****. I don't want to become the same as the others »*

Kousaka Reina

à Oumae Kumiko,

pour tout ce qu'elle représente à mes yeux.

Remerciements

Cette thèse est une collaboration entre l'Institut de Recherche sur les Céramiques (IRCER) et le laboratoire XLim de Limoges. Un travail de deux mois au sein du Département de Physique de l'Université de Gênes en Italie a également été effectué.
Je tiens tout d'abord à remercier mes directeurs de thèse Benoît Crespin, Arnaud Videcoq et Manuella Cerbelaud pour avoir assuré la direction mes travaux, ainsi que m'avoir guidé par leurs conseils et leurs réflexions tout au long de ma thèse.
Je remercie également Riccardo Ferrando, Professeur à l'Université de Gênes, de m'avoir accueilli pendant deux mois, et avoir pris soin de diriger mes travaux.
Je remercie également Σ-Lim Labex, sans qui cette thèse n'aurait pas pu voir le jour.
Je remercie aussi Marie Jardat et Fabrice Jaillet pour avoir accepté de relire ce manuscrit.
Et enfin, et non des moindres, je remercie très chaleureusement tous mes collègues doctorants, avec qui j'ai eu le plaisir de pouvoir partager de nombreux moments à Limoges, ainsi que ma famille que j'aime.

Table des matières

Introduction générale

Cette thèse est une collaboration entre deux laboratoires, financée par le Labex. L'IRCER (Institut de Recherche sur les Céramiques) s'inscrit dans le domaine des sciences des matériaux, dont l'une des équipes travaille dans les simulations numériques visant à reproduire virtuellement des expérimentations. XLIM est un institut de recherche dont l'un des domaines est l'informatique graphique, et plus particulièrement l'animation physique de fluides à une échelle macroscopique (vagues déferlantes, phénomènes d'érosion). Bien que travaillant dans des domaines très différents, ces deux laboratoires utilisent tous deux ce que l'on nomme des simulations de fluide. Par commodité de langage, les simulations employées par l'IRCER seront appelées simulations de type dynamique moléculaire (MD), tandis que le terme simulations de fluide désignera celles employées dans le domaine de l'informatique graphique. Ces simulations sont utilisées dans de nombreux domaines industriels et trouvent des applications variées pouvant être classées en deux catégories. D'un côté, celles issues de l'imagerie de synthèse ont pour but de produire des rendus graphiques physiquement vraisemblables, avec des applications dans des milieux tels que le cinéma, les jeux vidéos, l'infographie. D'un autre côté, celles dont le but est de reproduire rigoureusement le comportement des fluides afin d'étudier leur comportement, trouvent leurs applications en ingénierie pour aider à la conception d'objets réels. Dans les deux cas, les contraintes diffèrent. En informatique graphique, l'objectif est de pouvoir obtenir des simulations de fluide en temps réel et de grande échelle par l'utilisation de modèles physiques simplifiés, permettant tout de même d'obtenir un comportement visuellement crédible. Au contraire, dans le domaine des études numériques scientifiques, une précision rigoureuse des modèles physiques employés est primordiale. Ici, le temps de calcul se doit uniquement d'être raisonnable avec des simulations d'une durée de l'ordre de la semaine, et la taille du système d'être simplement assez importante pour étudier un échantillon statistique représentatif. Malgré ces contraintes différentes, les problématiques fondamentales qui se posent sont les mêmes. Les simulations se basent sur la résolution d'équations du mouvement des particules ayant pour étape principale la recherche de voisinage. De plus, des modèles issus de domaines physiques divers, tels que le modèle Smoothed Particle Hydrodynamics (SPH) provenant de l'astrophysique, ont pu être adaptés pour être employés en informatique graphique.
Les travaux d'optimisation en informatique graphique, exploitant au maximum des architectures parallèles GPU, peuvent aider à augmenter les performances des simulations de type MD. Inversement, les simulations de type MD utilisent des modèles physiques qui peuvent apporter un plus grand réalisme aux simulations de fluide en informatique graphique. Ainsi cette thèse s'est déroulée en deux étapes. Dans un premier temps, différents modèles physiques dans le cadre de suspensions colloïdales, utilisés dans des simulations de type MD par l'IRCER, ont été étudiés. Ils ont été implémentés sur une architecture GPU afin d'améliorer la performance des simulations et de permettre de

simuler des systèmes de plus grande taille. Puis dans un second temps, nous nous sommes intéressés aux applications possibles de ces modèles pour des problématiques liées au domaine de l'informatique graphique afin de pouvoir améliorer le réalisme ou simuler de nouveaux phénomènes.
Le Chapitre 1 introduit le contexte des suspensions colloïdales, qui sont beaucoup utilisées dans la conception des matériaux céramiques et dont le comportement est étudié par simulation numérique par l'IRCER. Il donne les différentes notions théoriques fondamentales nécessaires à toutes les simulations de type MD de suspensions colloïdales. Le modèle de dynamique brownienne (BD) y est décrit, un modèle simple permettant de modéliser le déplacement des particules au sein de suspensions colloïdales. Il présente également les principales problématiques et la littérature qui leur est associée, dont la recherche de voisinage nécessaire pour ces simulations. De plus, la seconde partie de ce chapitre est une introduction à la programmation GPU via des technologies telles que OpenCL ou CUDA, qui sont employées dans les Chapitres suivants.
Le Chapitre 2 présente diverses études sur des simulations de BD. La première étude propose une optimisation dans la méthode de recherche de voisinage, dans le cas d'une simulation de BD, à partir d'un système simple de suspensions colloïdales avec un seul type de colloïdes et une distribution en taille monomodale. Cette optimisation permet un gain en temps pour ce type de simulations de suspensions colloïdales. Les parties suivantes présentent des systèmes plus complexes dans lesquels notre nouvelle approche a pu être adaptée. La deuxième partie est consacrée à une étude d'hétéroagrégation avec deux types de colloïdes de même taille, mais de charge opposée. La troisième présente également une hétéroagrégation, mais cette fois-ci avec deux types de particules avec une grande différence de taille, ce qui impose de modifier la dynamique des colloïdes, ainsi qu'adapter notre méthode de recherche de voisinage. Enfin la dernière partie décrit une étude en dehors des suspensions colloïdales, issue d'une collaboration avec l'Université de Gênes. Cette étude portant sur des simulations de nanoalliages, montre que notre approche s'adapte aussi dans le cadre de simulations de MD à l'échelle atomique.
Le Chapitre 3 étudie un modèle plus complexe que la BD, prenant mieux en compte l'effet hydrodynamique dans les suspensions colloïdales. Il existe plusieurs simulations qui sont plus ou moins coûteuses en calculs (comme la *Dissipative Particle Dynamics* (DPD) qui modélise explicitement toutes les interactions entre particules de fluide). Pour la thèse, le choix a été porté sur le modèle hybride de Stochastic Rotation Dynamics - Molecular Dynamics (SRD-MD). Ce modèle a la particularité de décrire la dynamique des colloïdes précisément grâce à la MD et donne une description plus grossière de la dynamique des particules de fluide grâce à la SRD. Différents types de SRD-MD existent dans la littérature selon leur manière de coupler la partie SRD du modèle avec la partie MD. Le couplage étudié au cours de cette thèse nécessite de calculer des forces de répulsion

entre les colloïdes et les particules de fluide, empêchant les interpénétrations entre ces deux types de particules. Cette méthode, par la suite appelée SRD-MD avec force de couplage, permet de modéliser les interactions hydrodynamiques (HIs) correctement sans augmenter considérablement le temps de calcul. Des simulations de SRD-MD sur GPU ont déjà été développées par d'autres groupes [2, 3]. Cependant ces simulations utilisent un couplage simple qui ne permet pas de décrire tous les effets des HIs. Notre étude a donc consisté à paralléliser, pour des architectures GPU, les codes séquentiels des études de simulations de SRD-MD avec des forces de couplage, sur des systèmes avec un seul type de particule et une distribution en taille monomodale. Pour cela, nous avons adapté les travaux déjà existants pour la partie SRD, tout en reprenant notre nouvelle approche de recherche de voisinage pour la partie MD. Une nouvelle optimisation de cette recherche de voisinage est présentée dans ce chapitre, adaptée à ce type de système qui possède un très grand ratio entre le nombre de particules de fluide et le nombre de colloïdes.
Le Chapitre 4 est consacré à l'utilisation de ce modèle de SRD issu du contexte des suspensions colloïdales. Tout d'abord, le contexte des simulations de fluide dans le domaine de l'animation physique est introduit en décrivant les nouvelles problématiques qu'il pose. Puis les différents modèles utilisés dans la littérature de l'informatique graphique pour modéliser le fluide sont présentés. Enfin, dans une dernière partie sont décrites les modifications apportées au modèle de SRD pour être adapté aux systèmes étudiés en informatique graphique et les premiers résultats obtenus sont présentés.
Pour une question d'uniformisation avec la littérature, tous les acronymes utilisés sont en version anglaise dans toute cette thèse. De plus une liste des acronymes et termes techniques utilisés est donnée à la fin du document (page 121).

Chapitre I :

Introduction aux simulations de suspensions colloïdales sur GPU

Ce chapitre permet d'introduire les concepts fondamentaux afin de pouvoir réaliser des simulations de suspensions colloïdales sur GPU. Il est divisé en deux parties. La première décrit ce qu'est une suspension colloïdale et donne un exemple permettant de la modéliser avec un modèle simple nommé dynamique brownienne (BD), qui est ensuite utilisé dans le chapitre 2. Cette partie présente également les principales problématiques à résoudre pour simuler une suspension colloïdale et les différents travaux de la littérature sur ce sujet. La seconde partie est consacrée à la programmation sur GPU en présentant les notions primordiales liées à l'architecture GPU à prendre en compte pour obtenir de bonnes performances, en prenant comme exemple les différentes problématiques énoncées dans la première partie.

I.1 Introduction

Les suspensions colloïdales étudiées à l'IRCER sont des suspensions céramiques. La science des procédés céramiques est une sous-branche de la science des matériaux, dont l'un des objectifs est la mise en œuvre de nouveaux matériaux céramiques (des matériaux inorganiques et non métalliques). Les céramiques étant des matériaux non malléables, d'une grande dureté et dont la température de fusion est très élevée, les méthodes de fabrication par usinage ou fonderie ne sont pas adaptées. Ainsi, les procédés céramiques se basent essentiellement sur la consolidation d'une poudre céramique. Plusieurs voies de mise en forme sont possibles, dont la voie liquide qui nous intéresse ici. Elle consiste en la mise en suspension de particules céramiques, appelées colloïdes, qui est une dispersion homogène de particules à une échelle mésoscopique (de 10 nm à 1 μm), dans un liquide appelé solvant, qui peut être de l'eau par exemple. Les colloïdes interagissent généralement avec des forces à courte portée. Deux types de forces sont souvent pris en compte : des forces répulsives ou attractives dues à la charge se trouvant à la surface des colloïdes, et dans tous les cas, une force attractive due aux interactions de van der Waals. Ce sont ces forces qui entraînent le processus d'agrégation des colloïdes en suspension au cours du temps. La figure I.2 montre la structuration des agrégats obtenus pour des suspensions avec deux types de particules, observées par SEM-FEG (Scanning Electron Microscopy - Field Emission Gun) équipé d'une cellule cryogénique.

La manière dont les colloïdes s'arrangent dans les suspensions influence grandement les propriétés finales du matériau obtenu (électriques, thermiques, mécaniques, optiques). Ainsi la compréhension et la maîtrise du processus d'agrégation sont primordiales pour contrôler l'arrangement des colloïdes et obtenir les propriétés souhaitées. Or le nombre très important de paramètres tels que le pH, la force ionique ou toute autre force externe qui influencent leur comportement, rend l'étude expérimentale complexe. Les simulations

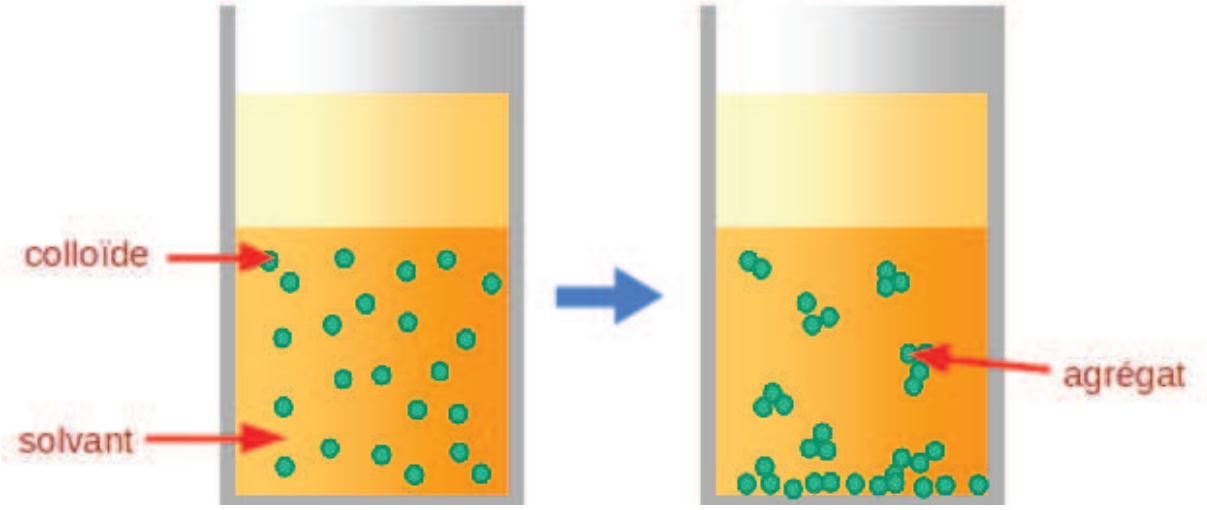


FIGURE I.1 – Schéma représentant une suspension colloïdale

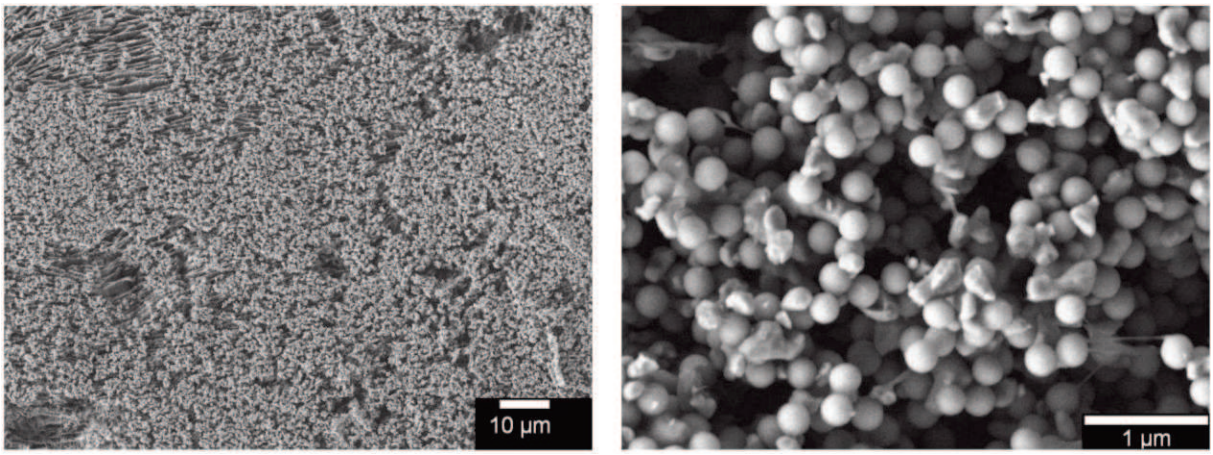


FIGURE I.2 – Images de suspensions expérimentales obtenues au SEM-FEG équipé d'une cellule cryogénique (Figure extraite de la référence [4])

numériques de ces suspensions colloïdales, qui permettent de calculer l'évolution de la position des colloïdes en suspension en fonction des paramètres, peuvent aider à la compréhension des procédés expérimentaux. Ces simulations constituent un outil très utile pour discriminer et comprendre le rôle de chaque paramètre.

Afin de reproduire les suspensions colloïdales virtuellement, la simulation doit calculer les forces d'interaction entre toutes les particules du système, ce qui est le principe de la dynamique moléculaire (MD). En effet, ce sont ces interactions qui sont la cause de l'agrégation des colloïdes et de la structure qu'ils forment. La difficulté dans la simulation des suspensions colloïdales réside dans la différence d'échelles de temps et d'espace entre les molécules de fluide et les colloïdes, ce qui est appelé le problème de séparation d'échelle. Le nombre de molécules de fluide à considérer est beaucoup trop important comparé aux capacités de mémoire et de calcul d'un ordinateur. Par exemple, le déplacement

d'un colloïde d'un diamètre 10^{-6}m peut déplacer de l'ordre de 10^{10} molécules de solvant. D'autre part, le pas de temps nécessaire pour décrire le mouvement des particules de fluide correctement est très faible comparé à celui adapté aux colloïdes, ce qui impose de devoir calculer un nombre d'itérations bien trop important pour terminer une simulation dans un temps raisonnable. Il est donc impossible de décrire à la fois les trajectoires des molécules de fluide et celles des colloïdes d'une suspension colloïdale. Or, seuls les déplacements des colloïdes nous intéressent. Ceux-ci ne peuvent cependant pas être calculés précisément sans prendre en compte la présence des molécules de fluide. Une partie de leur dynamique trouve en effet son origine dans leurs interactions avec les particules de fluide, notamment les HIs qui représentent le transfert de quantité de mouvement (soit le produit de la masse par la vélocité) entre colloïdes transmis via les particules de fluide. Pour répondre à ce problème, les modèles employés pour décrire le fluide sont des représentations dites "coarse-grained", c'est-à-dire qu'ils permettent de décrire plus ou moins précisément les HIs sans avoir besoin pour autant de modéliser toutes les particules de fluide. Dans le paragraphe suivant, nous allons présenter la dynamique brownienne qui est un modèle de MD représentant le fluide par un terme de friction et un terme aléatoire dans l'équation du mouvement des colloïdes.

I.2 Les suspensions colloïdales

I.2.1 Le modèle de dynamique brownienne

La dynamique brownienne (BD) est un modèle qui, comme son nom l'indique, est basé sur le mouvement brownien, un phénomène largement observé dans la nature. Il permet de décrire le mouvement erratique des particules suspendues dans un fluide, qui peut être aussi bien des particules de fumée dans de l'air que des colloïdes dans un liquide. La méthode de simulation la plus simple pour étudier les suspensions colloïdales est la BD [5], qui a commencé à se développer dans les années 70. Dans cette technique, le fluide est représenté comme un milieu continu et son effet sur les colloïdes est représenté par une combinaison entre des forces de friction ($\Xi_i(t)$) opposées au mouvement et une force stochastique (Γ_i), entraînant un mouvement brownien. Chaque colloïde subit de la part des autres colloïdes du système une force qui dépend de la distance r_{ij} qui les sépare. Cette force est représentée par le terme $\sum_j F_{ij}(r_{ij}(t))$. Ainsi, le mouvement d'un colloïde i est décrit par l'équation suivante, dite équation de Langevin :

$$m_i \frac{dv_i(t)}{dt} = \sum_j F_{ij}(r_{ij}(t)) + \Xi_i(t) + \Gamma_i(t) \tag{I.1}$$

où v_i représente la vélocité de la particule i, m_i sa masse, t le temps, r_{ij} la distance entre le colloïde i et un de ses voisins j et F_{ij} la force d'interaction exercée par le colloïde j sur le colloïde i. Les forces d'interaction entre colloïdes $\sum_j F_{ij}(r_{ij}(t))$ découlent du potentiel d'interaction V_{ij} :

$$F_{ij}(r_{ij}(t)) = -\nabla V_{ij}(r_{ij}(t)) \tag{I.2}$$

Le potentiel V varie en fonction de la simulation étudiée. La force de friction $\Xi_i(t)$ quant à elle s'exprime par la loi de Stokes :

$$\Xi_i(t) = -\zeta_i v_i(t) \tag{I.3}$$

avec ζ_i, le coefficient de friction, exprimé par la relation : $\zeta_i = 6\pi\eta a_i$ où a_i est le rayon de la particule i et η la viscosité du solvant.
Enfin la force aléatoire $\Gamma_i(t)$ doit respecter deux conditions :

— cette force ne privilégie aucune direction, elle est équitablement répartie. Cela implique que la moyenne statistique de cette force $< \Gamma_i(t) >$ est nulle, donc

$$< \Gamma_i(t) >= 0$$

— cette force est décorrélée dans le temps et entre les particules. Cela signifie que

$$< \Gamma_i(t)\Gamma_j(t') >= C\delta(t - t')\Delta_{ij}$$

où $< \Gamma_i(t)\Gamma_j(t') >$ est la moyenne statistique du produit entre la force fluctuante appliquée à deux particules pour deux instants donnés et où δ représente la distribution de Dirac, Δ_{ij} le symbole de Kronecker et C une constante. Cette constante C doit être égale à $2\zeta_i k_B T$ afin de respecter le théorème de l'équipartition de l'énergie :

$$\sum_{i=1}^{N} \frac{1}{2} m_i v_i^2 = \frac{3}{2} N k_B T \tag{I.4}$$

où N est le nombre de particules, k_B est la constante de Boltzmann et T est la température. La résolution de l'équation de Langevin sous la forme de l'équation (I.1) est un problème complexe. Un des algorithmes les plus connus dans les simulations numériques de BD, est celui proposé par Ermak [6] se basant sur une simplification de l'équation précédente. L'intégration de l'équation (I.1), fait apparaître un terme en $exp(-\frac{t}{\tau_v})$ dans l'expression de la vélocité, où $\tau_v = \frac{m_i}{\zeta_i}$ représente le temps de relaxation de la vitesse des particules browniennes. En prenant un pas de temps Δt très supérieur à τ_v, le terme $\frac{t}{\tau_v}$ devient très grand, d'où $exp(-\frac{t}{\tau_v}) \approx 0$. Ainsi, avec un pas de temps $\Delta t >> \tau_i$, le terme d'accélération

devient négligeable, ce qui nous permet de simplifier l'équation de Langevin (I.1) et obtenir l'équation de Langevin simplifiée suivante :

$$\frac{dr_i(t)}{dt} = \frac{1}{\zeta_i}\sum_j F_{ij}(r_{ij}(t)) + \frac{1}{\zeta_i}\Gamma_i(t), \tag{I.5}$$

Par ailleurs, le pas de temps ne doit pas non plus être trop grand. En effet, l'interaction entre les particules a une influence très importante sur leur mouvement. Pour que le mouvement calculé soit le plus proche du modèle théorique de la BD, il faut donc que la variation de cette interaction ne soit pas trop importante entre deux pas de temps successifs. Le temps à ne pas dépasser dépend du potentiel utilisé. Il doit être inférieur à la période d'oscillation des particules afin de pouvoir reproduire ces oscillations. L'équation de Langevin simplifiée (I.5) a été intégrée en utilisant la méthode dite du "schéma d'intégration du bruit blanc", qui s'appuie sur des développements limités de l'équation stochastique considérée. En utilisant les expressions établies par Mannella et al. [7, 8] à l'ordre 1, l'intégration de l'équation (I.5) permet d'obtenir l'évolution des positions des particules au cours du temps suivant l'équation :

$$r_i(t+\Delta t) = r_i(t) + \sqrt{\frac{2k_BT}{\zeta_i}}(\Delta t)^{1/2}Y_i + \frac{1}{\zeta_i}\sum_j F_{ij}(r_{ij}(t))\Delta t \tag{I.6}$$

où les variables Y_i sont des nombres aléatoires non corrélés, distribués selon une gaussienne de moyenne nulle et de variance égale à 1.
Cependant, lorsque l'équation de Langevin ne peut pas être simplifiée, il faut alors intégrer l'équation complète de Langevin (équation I.1). Les vélocités sont alors obtenues par :

$$v_i(t+\Delta t) = v_i(t) + \frac{\sqrt{2k_BT\zeta_i}}{m_i}(\Delta t)^{1/2}Y_i + \frac{1}{m_i}\left(-\zeta_i v_i(t) + \sum_j F_{ij}(r_{ij}(t))\right)\Delta t, \tag{I.7}$$

Et les positions par :

$$r_i(t+\Delta t) = r_i(t) + v_i(t)\Delta t. \tag{I.8}$$

Ces équations (I.7 et I.8) sont employées dans les simulations dites de dynamique de Langevin. La BD ne prenant pas en compte les HIs, elle est utilisée pour des suspensions diluées et sans écoulement, où, malgré les approximations, le modèle permet d'obtenir des résultats valides. Des simulations de BD ont déjà été implémentées et utilisées avec succès dans le cadre de suspensions colloïdales dans la littérature [7, 9, 10] en intégrant l'équation (I.6).

I.2.2 Les conditions périodiques

Un volume macroscopique de suspension colloïdale étant composé d'un nombre trop important de colloïdes pour être modélisé informatiquement, les simulations sont contraintes à ne représenter qu'une petite fraction du système total de l'expérimentation réelle au sein de la boîte de simulation. Cette boîte est la plupart du temps cubique, est sa longueur L_{box} est calculée pour que le système simulé corresponde à une fraction volumique colloïdale donné (ϕ). Ainsi, pour un système de N colloïdes de rayon a, la taille L_{box} de la boîte de simulation est donnée par l'équation :

$$L_{box} = \sqrt[3]{\frac{4\pi N a^3}{3\phi}} \tag{I.9}$$

Pour pouvoir représenter la totalité du système, cette boîte de simulation est dupliquée

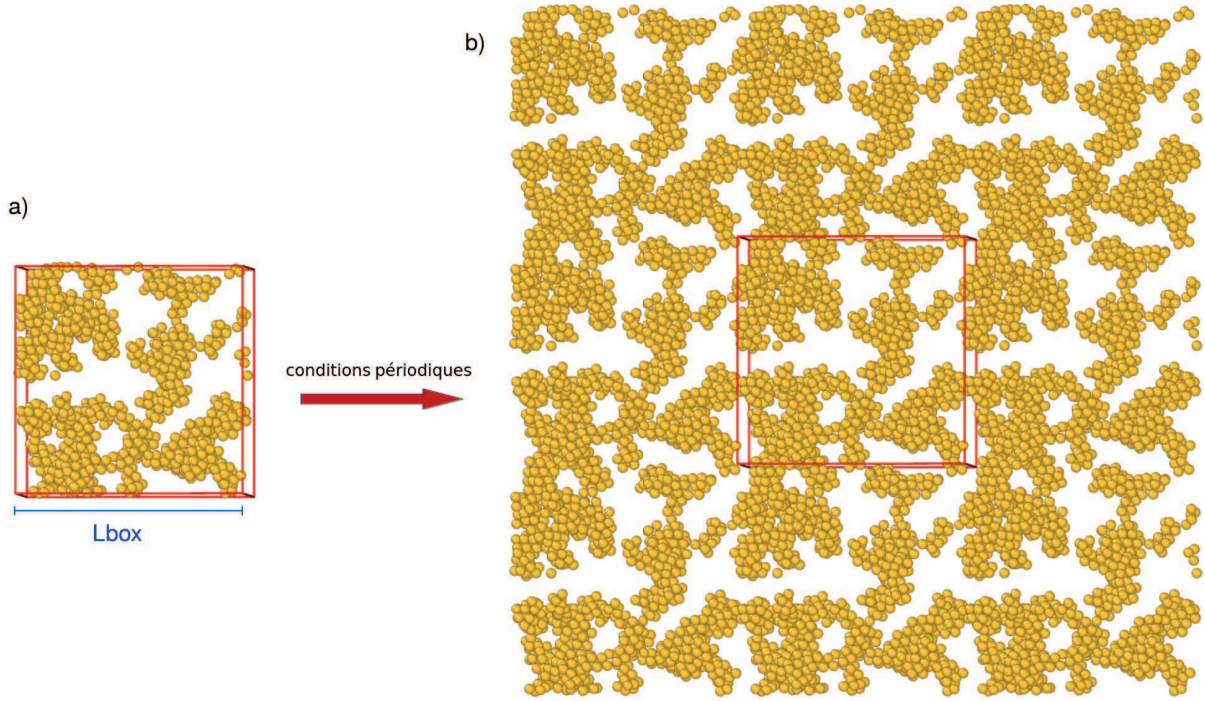


FIGURE I.3 – Illustration de l'application des conditions périodiques. La figure a) est une capture d'écran d'une simulation de suspension colloïdale, la boîte de simulation étant représentée par le cube en rouge. La figure b) reprend la figure a) en lui appliquant des conditions périodiques pour les directions X et Y.

à l'infini dans toutes les directions de l'espace, tout autour d'elle, comme le montre la figure I.3 (pour les directions X et Y uniquement pour une question de lisibilité), ce qui permet d'obtenir une approximation du système complet. Ainsi, lorsqu'une particule sort de la boîte de simulation, elle réapparaît dans la face opposée. Pour une question de simplicité, dans les simulations, cela se traduit par le déplacement de la particule

sortante, et non par la suppression d'une particule dans le système et l'ajout d'une nouvelle particule. Ainsi, dans les algorithmes des simulations, chaque déplacement de particule est suivi d'une application des conditions périodiques modifiant les positions des particules selon l'équation suivante :

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x + \lfloor x/L_{box} \rfloor L_{box} \\ y + \lfloor y/L_{box} \rfloor L_{box} \\ z + \lfloor z/L_{box} \rfloor L_{box} \end{pmatrix} \tag{I.10}$$

De même, les conditions périodiques jouent un rôle lors du calcul des interactions entre les particules qui se base sur leur distance. Avec les conditions périodiques, deux particules possèdent neuf distances périodiques. Dans toutes les simulations étudiées dans cette thèse, où les forces d'interaction sont à courte portée, lorsqu'il faut calculer la distance r_{ij} entre la particule i et j, nous avons utilisé la convention d'image minimum qui consiste à considérer la plus courte des distances périodiques entre i et j :

$$\begin{pmatrix} r_{ij}.x \\ r_{ij}.y \\ r_{ij}.z \end{pmatrix} = \begin{pmatrix} x_i - x_j + \lfloor (x_i - x_j + 0.5L_{box})/L_{box} \rfloor L_{box} \\ y_i - y_j + \lfloor (y_i - y_j + 0.5L_{box})/L_{box} \rfloor L_{box} \\ z_i - z_j + \lfloor (z_i - z_j + 0.5L_{box})/L_{box} \rfloor L_{box} \end{pmatrix} \tag{I.11}$$

Comme le montre la figure I.4, les conditions périodiques, en dupliquant la boîte de simulation, créent une régularité dans la structure de l'agrégat obtenu qui n'est pas présente dans la réalité. C'est ce qu'on appelle le problème des effets de taille finie. Avec une boîte de simulation trop petite où l'échantillon statistique représenté est faible, cette régularité peut engendrer des effets non réalistes. C'est pourquoi, il est primordial de pouvoir simuler un système de taille importante pour diminuer le plus possible ces effets liés aux conditions périodiques. Pour que la simulation puisse prédire correctement le type de structure formée, il faut que la boîte de simulation soit suffisamment grande pour représenter un échantillon statistique représentatif du système total.

I.2.3 Les problématiques associées aux simulations de type MD

L'implémentation de simulations de type MD présente plusieurs difficultés. Tout d'abord, le support informatique impose certaines limites empêchant de pouvoir appliquer parfaitement les modèles physiques. Ainsi, la capacité mémoire, les temps de calculs et leur exactitude sont des problématiques liées au domaine des simulations numériques au sens large. Ensuite, la plupart des problèmes rencontrés dans le domaine des simulations de type MD, résident dans la recherche de voisinage. Cette étape est très coûteuse en termes de temps et doit être effectuée à chaque itération de la simulation, ce qui en fait généralement le goulot d'étranglement de ces simulations. Cette problématique se retrouve

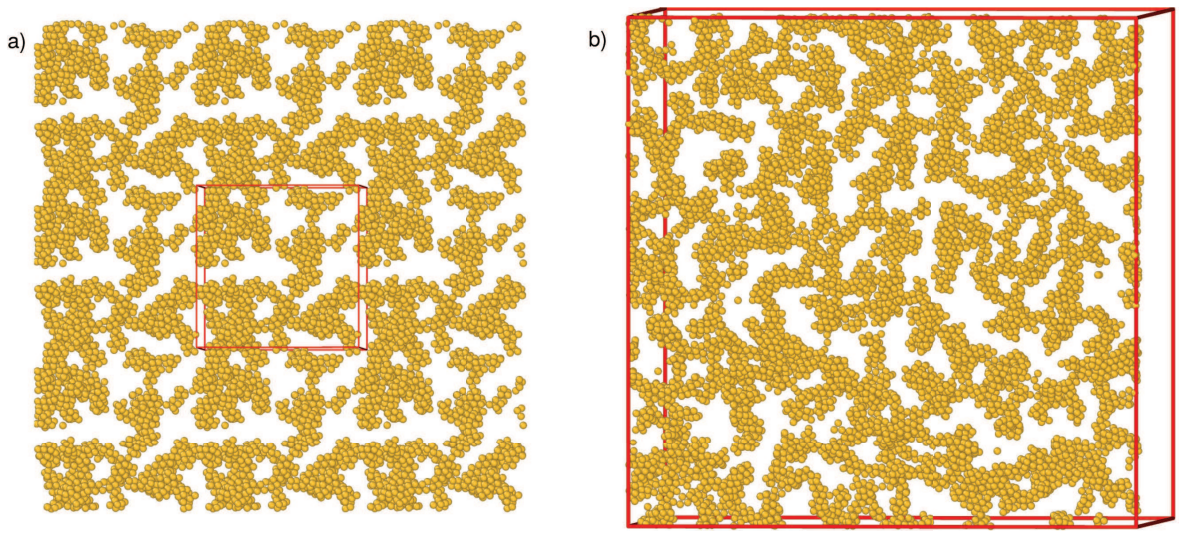


FIGURE I.4 – Comparaison entre un petit système de 2000 colloïdes sur lequel des conditions périodiques sont appliquées (a) et un système plus grand de 60000 colloïdes (b). L'application des conditions périodiques amène une régularité dans la structure formée. Ainsi, la boîte de simulation doit représenter une fraction suffisamment importante pour réduire au maximum l'approximation qu'entraînent les conditions périodiques.

dans les autres simulations basées sur des systèmes de particules, et plus généralement dans le domaine des animations physiques interactives. Les systèmes de particules sont en effet devenus populaires dans le domaine de l'informatique graphique depuis les premiers travaux de Reeves [11], et les représentations par systèmes de particules, sont aujourd'hui largement utilisées dans de nombreux et différents domaines allant des agents autonomes aux réseaux de neurones [12]. Un bon exemple est le domaine traité dans cette thèse, celui des simulations de fluide, où les particules peuvent être utilisées pour représenter des volumes de fluide indépendants dont la masse est constante, et ces particules interagissent entre elles de telle sorte à pouvoir calculer un mouvement réaliste [13]. De telles simulations doivent résoudre des problèmes similaires à ceux utilisant la MD, notamment le problème de recherche de voisinage.

I.2.3.1 La recherche de voisinage

Dans notre système, comme il a déjà été mentionné, les forces d'interaction entre colloïdes sont à courte portée, ce qui signifie qu'à longue distance, elles peuvent être négligées et qu'un rayon de coupure peut être utilisé dans les simulations. Ainsi, pour résoudre l'équation (I.6), il est nécessaire de déterminer pour chaque colloïde i, l'ensemble des colloïdes j étant en interaction avec i, c'est-à-dire dont la distance r_{ij} est inférieure au rayon de coupure R_C qui est la distance maximale pour laquelle l'interaction entre deux colloïdes est considérée comme non nulle (voir figure I.5). Déterminer cet ensemble revient à résoudre un problème de recherche de voisinage. Différentes approches ont été étudiées

dans la littérature, utilisant divers types de structures. Deux approches sont à distinguer : les méthodes conservatives issues de la littérature de MD, et les méthodes d'accélération provenant principalement du domaine de la simulation de fluide. Les premières utilisent des structures appelées listes de Verlet (ou listes de voisinage), qui sont des listes associées à chaque particule i stockant un ensemble de particules j pouvant possiblement être en interaction avec i durant les prochaines itérations [5,14]. Ces listes qui conservent un large voisinage, ne sont mises à jour que tous les n_i-pas de temps, et ainsi diminue le nombre de recherches de voisinage. Les méthodes d'accélération quant à elles, utilisent une grille régulière (représentée par une liste de cellules) qui subdivisent le domaine de simulation pour répartir les particules dans les cellules de cette grille qui les contiennent [13, 15]. Ceci a pour but de diminuer le nombre de particules parcourues lors d'une recherche de voisinage. Dans le cadre de simulations de type MD, il semble y avoir un consensus pour combiner ces deux structures afin de résoudre ce problème : l'emploi d'une liste de Verlet comme structure conservative pour garder en mémoire un voisinage assez large pour l'utiliser pendant plusieurs itérations, et d'une grille régulière comme structure accélératrice permettant de créer ces listes plus rapidement en diminuant le nombre de paires de particules à tester [16–18]. L'utilisation de ces deux structures est décrite dans les paragraphes suivants. Dans le paragraphe I.3.5, nous allons présenter les techniques d'utilisation de ces mêmes structures, dans le cadre d'une implémentation GPU et la manière de les combiner.

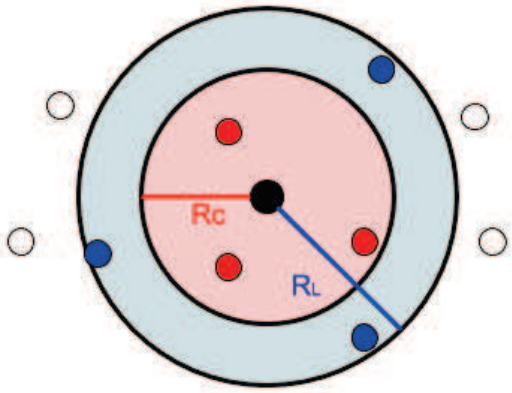


FIGURE I.5 – Illustration du rayon de coupure R_C et du rayon de conservation R_L. Les particules rouges dans la zone d'interaction définie par R_C exercent une force d'interaction non nulle sur la particule noire. Les particules bleues dans zone de conservation définie par R_L sont conservées dans la liste de Verlet de la particule noire.

Les listes de Verlet Verlet a introduit le concept de structure conservative pour les simulations de type MD [19]. L'idée est de stocker dans une liste, appelée liste de Verlet, pour chaque colloïde i, un voisinage plus large N_L que le voisinage N_C réellement en

interaction avec i à l'instant t, afin de pouvoir utiliser N_L durant n_i itérations. Ainsi, avec cette méthode, il est nécessaire de rechercher le voisinage uniquement toutes les n_i itérations plutôt qu'à chaque itération. Le voisinage N_L est défini par un rayon de voisinage R_L (avec $R_L > R_C$), qui représente la distance maximale pour laquelle un colloïde i est stocké dans la liste de Verlet de i (voir figure I.5). Théoriquement, il est uniquement nécessaire de mettre à jour la liste de Verlet du colloïde i quand un nouveau colloïde arrive dans sa zone d'interaction. Cependant, vérifier cette condition revient à résoudre un problème de recherche de voisinage. C'est pourquoi en pratique, on utilise un test simple qui détermine quand réactualiser les listes de Verlet. Le choix de ce test qui affecte les performances, est un compromis entre un test très simple et peu coûteux, mais réactualisant les listes bien souvent plus que nécessaire, et un test plus précis mais coûteux. Les différents tests qui existent seront discutés dans le chapitre II.

La grille régulière Plusieurs structures de subdivision de l'espace ont été étudiées dans la littérature afin de répondre au problème de recherche de voisinage. Des structures hiérarchiques telles que des arbres-KD ou -BSP sont utilisées dans des implémentations basées CPU, comme c'est le cas pour bien d'autres problématiques comme pour les détections de collisions [20]. Cependant, la grille régulière est la structure principalement utilisée pour les simulations basées sur des architectures parallèles telles que le GPU ou un CPU multicœurs [15]. La thèse étant dans le contexte d'implémentation GPU, seule la méthode de la grille régulière est présentée ici.
La majorité des simulations de fluide repose sur les grilles régulières comme structure accélératrice pour la recherche de voisinage, particulièrement dans le domaine de l'informatique graphique où les méthodes conservatives sont moins adaptées. La grille régulière est une structure de données qui partitionne l'espace de simulation en cellules régulières cubiques. Les particules sont réparties dans des cellules de taille L_{cell}. Avec $L_{cell} = R_C$, chercher le voisinage N_C de la particule i nécessite uniquement de parcourir les particules contenues par la cellule contenant i et ses 26 cellules adjacentes, réduisant ainsi drastiquement le nombre de particules à tester et donc le temps de calcul de cette étape. Cependant, l'utilisation d'une grille régulière nécessite de la reconstruire à chaque itération car le contenu des cellules risque d'être obsolète à chaque déplacement des particules. La façon d'implémenter cette grille régulière dépend du type d'architecture. L'implémentation GPU de cette structure dans la littérature est présenté dans le paragraphe I.3.5.

I.2.3.2 La génération de nombres aléatoires

Les simulations de BD font intervenir des nombres aléatoires distribués selon une gaussienne, représentés dans les équations (I.6) et (I.7) par le terme Y_i. En informatique, la génération de nombres aléatoires est un problème complexe à cause de la nature de

l'ordinateur, dont le fonctionnement est déterministe. En réalité, les seuls moyens de réellement générer des suites aléatoires sont les méthodes basées sur des phénomènes physiques quantiques. Cependant, cette solution est très coûteuse que ce soit en terme matériel ou de temps. Il existe également des méthodes purement algorithmiques permettant de générer des suites dites pseudo-aléatoires à partir de graines, dont les propriétés aléatoires peuvent être validées par différents tests. On parle alors de générateurs de nombres pseudos aléatoires ou Pseudo Random Number Generators (PRNG). Pour les algorithmes les plus complexes, les tests ne permettent pas de différencier une suite générée d'une suite purement aléatoire, mais la génération d'un tel nombre aléatoire est alors coûteuse en temps. Le choix d'un algorithme aléatoire est donc un compromis entre la qualité de l'aléatoire obtenu et le coût en temps. Dans le cas de nos simulations de suspensions colloïdales, le choix a été de reprendre l'algorithme développé par Marsaglia et Zaman [21], qui a été déjà employé dans des études précédentes. Comme la plupart des PRNG, ce générateur se base sur une graine fixée par l'utilisateur pour initialiser l'état du générateur qui va ensuite pouvoir permettre de générer les nombres aléatoires. La méthode qui génère ces nombres modifie l'état du générateur, ce qui va permettre de générer par la suite un nouveau nombre à partir de ce nouvel état. Ces nombres aléatoires suivent une loi uniforme. Dans le cadre de nos simulations de suspensions colloïdales, il est nécessaire d'obtenir des nombres aléatoires de distribution gaussienne. Nous avons choisi d'utiliser la méthode de Box-Müller pour cela [22], qui se base sur une fonction permettant de générer deux nombres aléatoires suivant une distribution gaussienne, à partir de deux nombres aléatoires de distribution uniforme. Il existe deux formes à cette méthode. La forme basique est à proscrire car elle utilise les fonctions trigonométriques *cos* et *sin* dont le calcul sur ordinateur est trop imprécis. C'est pourquoi la forme polaire est plus communément utilisée notamment dans le *Numerical Recipes* [22]. Cette forme polaire permet de générer deux nombres aléatoires de distribution gaussienne selon l'équation :

$$z_0 = \sqrt{-2\ln s}\left(\frac{u}{\sqrt{s}}\right) = u\cdot\sqrt{\frac{-2\ln s}{s}}, z_1 = \sqrt{-2\ln s}\left(\frac{v}{\sqrt{s}}\right) = v\cdot\sqrt{\frac{-2\ln s}{s}} \qquad \text{(I.12)}$$

où u et v sont des nombres aléatoires de distribution uniforme dans l'intervalle $[-1\,;1]$ et $s = u^2+v^2$ doit être compris dans l'intervalle $s \in]0\,;1]$. Si ce n'est pas le cas, il faut retirer des nouvelles valeur de u et v jusqu'à ce que cette dernière condition soit valide. Cette méthode de génération de nombres aléatoires a été utilisé durant de nombreuses années à l'IRCER, pour divers types de simulations de suspensions colloïdales, c'est pourquoi elle a été privilégiée malgré d'autres méthodes pouvant être plus performantes.

I.2.4 Les méthodes d'analyse des résultats de simulations MD

La visualisation moléculaire Afin de pouvoir analyser le processus d'agrégation, les positions des particules sont sauvegardées au cours de la simulation dans des fichiers XYZ, un format utilisé dans plusieurs logiciels de visualisation moléculaire, tels que VMD [23], Ovito [24] et Jmol [25]. Ce format de fichier contient le nombre de particules du système, puis pour chaque particule, son type et ses trois coordonnées spatiales. Les logiciels de visualisation permettent principalement de représenter le système à l'état des sauvegardes en 3 dimensions, et ainsi d'observer le processus d'agrégation. Les différentes captures des simulations présentées dans cette thèse sont issues du logiciel VMD. Outre la visualisation, les fichiers de sauvegarde permettent également de calculer des propriétés caractéristiques de la suspension colloïdale.

La cinétique d'agrégation Une des caractéristiques des suspensions colloïdales est la vitesse à laquelle les colloïdes s'agrègent. La cinétique d'agrégation d'une suspension colloïdale peut être analysée à la fin de la simulation, en déterminant l'évolution du nombre d'agrégats dans le système au cours du temps, à partir des fichiers de sauvegarde XYZ. Lorsque deux colloïdes se trouvent à une distance inférieure à une valeur fixe r_{agreg}, ils sont considérés comme s'étant agrégés. Dans cette thèse, un agrégat est composé d'au moins deux colloïdes. Le calcul du nombre d'agrégats dans un système s'effectue par une boucle parcourant toutes les particules qui n'ont pas encore été marquées comme faisant partie d'un agrégat. Pour chaque particule i parcourue, il faut chercher les particules j non marquées se trouvant à une distance inférieure à r_{agreg} et marquer ces particules, puis effectuer ce même processus avec les particules j. Si des particules voisines de i ont été marquées, la particule i parcourue est également marquée, et le compteur du nombre d'agrégats est incrémenté. La répétition de ce processus sur tous les fichiers de sauvegarde d'une simulation permet de générer une courbe du nombre d'agrégats en fonction du temps de la simulation.

La température Dans les simulations où la vitesse des particules est utilisée, telles que par exemple les simulations de dynamique de Langevin basées sur les équations (I.7 et I.8), un test sur la température peut être facilement effectué. D'après l'équation (I.4), la température d'un système peut être calculée à partir de la vitesse des colloïdes, selon l'équation suivante :

$$T(t) = \frac{1}{3Nk_B} \sum_{i=1}^{N} m_i v_i(t)^2 \tag{I.13}$$

où m_i est la masse de la particule i, N est le nombre de particules total dans le système, v_i est la vitesse de la particule i et k_B est la constante de Boltzmann. Cette température

doit être calculée à chaque itération, et moyennée sur des intervalles de temps afin de suivre son évolution. Cette température doit rester à peu près constante au cours de la simulation.

Afin de pouvoir simuler des systèmes de suspensions colloïdales de taille importante, dans des temps de calculs raisonnable, il est nécessaire d'employer des technologies de calculs parallèles, telles que le GPU. La partie suivante est consacrée aux fondamentaux à connaître sur la programmation GPU.

I.3 La programmation GPU

Tout d'abord essentiellement utilisés dans l'industrie du jeu vidéo et de l'infographie, les GPUs (Graphical Processing Unit) ont commencé à être également exploités dans les domaines scientifiques nécessitant de lourds calculs, à partir des années 2000. En effet, les GPUs sont peu à peu devenus une alternative pour effectuer des calculs parallèles, offrant ainsi une grande puissance de calcul pour un prix relativement faible, comparé à celui de clusters de CPUs (Central Processing Unit). Cependant, les GPUs étant à la base uniquement conçus pour le rendu 3D, leur utilisation pour d'autres types de calcul était alors très fastidieuse, car elle nécessitait à la fois une connaissance profonde de l'architecture du GPU et de détourner les outils de programmation sur GPU de leur fonctionnalité première. Pour remédier à cela, en 2007, les deux principaux fabricants NVIDIA et ATI/AMD développent respectivement les technologies CUDA (Compute Unified Device Architecture) [26] et ATI Stream, qui permettent d'utiliser les GPUs pour tout type de calcul. On parle alors de GPGPU (General-Purpose Graphics Processing Units). Cependant, ces deux technologies sont limitées à l'exploitation des GPUs de leurs constructeurs respectifs. Par la suite, en 2008, Khronos Group développe une API (ou interface de programmation) nommée OpenCL, permettant de programmer sur des systèmes parallèles sans contraintes d'architecture. En proposant un paradigme de programmation abstrait à l'architecture des systèmes, cette API peut exploiter des architectures hétérogènes comme des GPUs de différents constructeurs ou même des CPU multicœurs [27]. Ainsi, les technologies CUDA ou OpenCL sont des extensions du langage C et C++ qui permettent de simplifier grandement l'implémentation d'applications exploitant le GPU. Néanmoins, l'architecture des GPUs impose des contraintes dans le paradigme de programmation. Dans la suite sont détaillés les concepts fondamentaux de la programmation sur GPU nécessaires pour obtenir des performances correctes, en prenant en exemple le modèle OpenCL, la technologie principalement utilisée au cours de la thèse.

I.3.1 L'architecture GPU

Les GPUs ont une architecture très différente des CPUs, ce qui nécessite de changer le paradigme de programmation habituel pour prendre en compte leurs spécificités. La figure III.1 illustre la différence d'architecture entre le CPU et le GPU. Le CPU est doté d'un faible nombre d'unités de calculs ou cœurs, principalement conçus pour exécuter des codes séquentiels complexes. Il est très efficace et polyvalent, dispose de plusieurs niveaux de mémoire cache en quantité relativement importante, d'une unité de contrôle par cœur et d'une unité arithmétique et logique (UAL) très performante, et peut accéder à une mémoire commune mais très lente, appelée RAM (Random Acces Memory). Au contraire, le GPU est composé d'un nombre très important de cœurs qui sont conçus pour fonctionner exclusivement de façon parallèle afin d'exécuter une tâche globale. Ces cœurs sont spécialisés dans les calculs matriciels utilisés dans la 3D, disposent d'une capacité de mémoire cache limitée, et ne peuvent s'exécuter de façon indépendante contrairement à ceux du CPU. Les cœurs sont liés physiquement par groupes de 32, appelés *warps* [26]. Chaque cœur d'un warp possède sa propre mémoire cache d'une capacité faible et son UAL, mais partage la même unité d'instruction et un type de mémoire partagée par l'ensemble du warp. De ce fait, l'architecture GPU est conçue pour être programmée selon le paradigme SIMD (Single Instruction on Multiple Data), un patron de conception où tous les fils d'exécution ou *threads* lancés par le GPU exécutent la même série d'instructions en parallèle, mais l'appliquent à des données différentes. Dans ce modèle, le nombre de threads exécutables est considéré comme étant infini, car le lien entre les coeurs et les threads à exécuter n'est pas pris en charge par le programmeur. Tout comme pour le CPU, la mémoire est hiérarchisée selon la rapidité d'accès et la capacité maximale [26]. La mémoire *globale* est équivalente à la RAM en CPU. Elle peut être accédée par tous les threads et a largement la plus grande capacité. Néanmoins, puisqu'elle est hors puce, elle est bien plus lente d'accès que les autres types de mémoire [27]. C'est également la mémoire *globale* qui permet les transferts de données entre le CPU et le GPU. La mémoire *partagée* est accessible par les threads d'un même bloc, un bloc étant composé d'un nombre de warps liés logiquement, fixé par le programmeur lors de l'exécution. Elle a une capacité très réduite par rapport à la mémoire globale, mais est bien plus rapide. Elle est utilisée dans les cas où les threads d'un même warp sont amenés à s'échanger des données, ou lorsque la mémoire *privée* des threads est saturée. La mémoire privée correspond à la mémoire cache du CPU. C'est une mémoire de très petite capacité, mais elle est très rapide d'accès et n'est accessible que par son thread. Dans le paragraphe suivant, nous allons voir comment OpenCL permet d'exploiter l'architecture GPU.

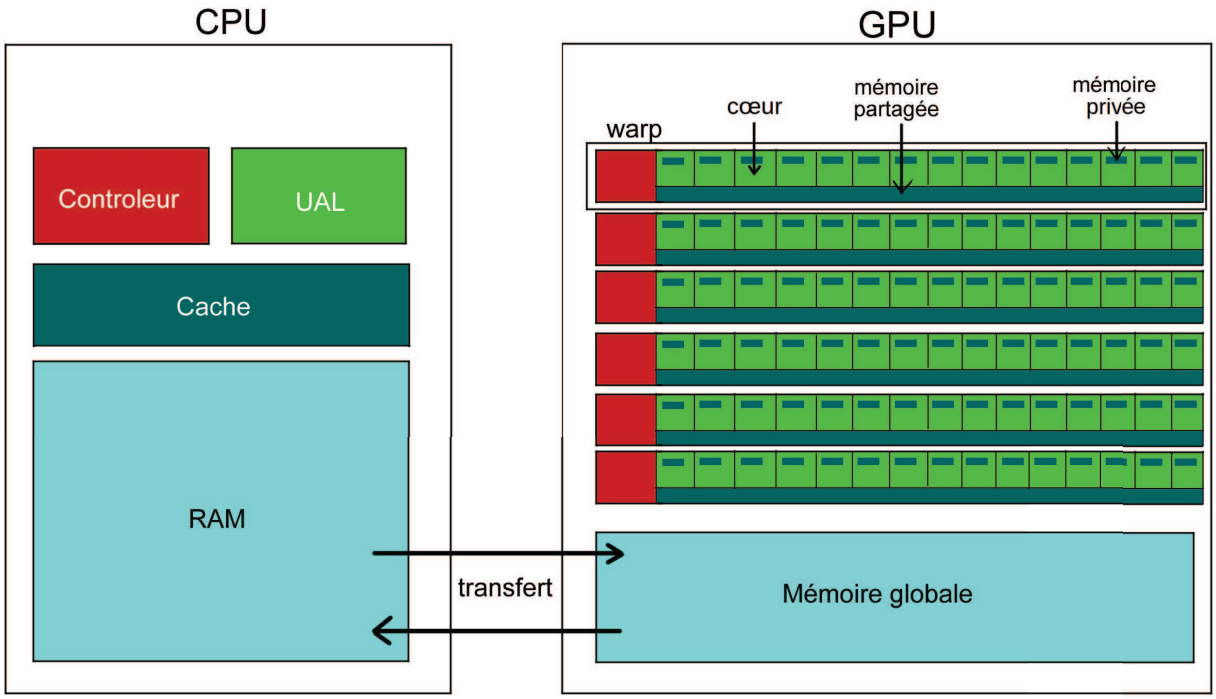


FIGURE I.6 – Comparaison entre une architecture CPU et GPU.

I.3.2 Les bases d'OpenCL

OpenCL distingue deux parties dans une application GPU : celle concernant le processus hôte s'exécutant sur le CPU de façon séquentielle comme dans un programme classique et celle concernant les périphériques de calculs qui exécutent des petites portions de programme appelées noyaux, qui sont l'équivalent de fonctions C ou C++ s'exécutant de façon parallèle [27]. Outre l'exécution des parties du programme non parallélisables telles que des appels systèmes, la partie hôte est également chargée de réaliser l'interfaçage entre la partie hôte et la partie périphérique et l'initialisation du périphérique. La partie périphérique est composée quant à elle de fonctions exécutables sur le périphérique de manière parallèle, toutes programmées selon le paradigme SIMD. Ainsi, un portage d'une fonction sur GPU est limité à celle qui peut être décomposée en plusieurs sous tâches identiques et ne dépendant pas du résultat des autres sous tâches. Le paragraphe suivant décrit plus précisément le rôle que tient la partie hôte en rapport avec l'utilisation du GPU.

I.3.3 La partie hôte d'OpenCL

La partie hôte est chargée de toute la partie d'initialisation d'OpenCL. Elle consiste dans un premier temps à charger OpenCL, puis à créer le contexte du système qui permet notamment de déterminer les informations du périphérique telles que la mémoire totale du GPU et son nombre de cœurs physiques. Ensuite, elle doit charger tous les noyaux,

les compiler et créer la file de commandes qui permettra d'enfiler les noyaux à exécuter. Chaque noyau inséré dans la file est exécuté de façon asynchrone au CPU hôte, de façon séquentielle, c'est-à-dire qu'un noyau attend que les précédents noyaux soient terminés avant de s'exécuter. Enfin le CPU hôte alloue et initialise la mémoire du périphérique. Dans un second temps, le CPU hôte est chargé d'exécuter les noyaux en les insérant dans la file de commande après leur avoir associé leurs arguments et indique le nombre total de threads et le nombre de threads par bloc du GPU. Il s'occupe aussi des transferts mémoire entre le GPU et le CPU qui permettent notamment de récupérer les résultats des calculs effectués sur GPU dans la partie hôte. Enfin, à la fin du programme, il libère les ressources mémoire utilisées par le GPU.

I.3.4 La partie périphérique d'OpenCL

La partie périphérique est uniquement constituée de noyaux qui sont des fonctions s'exécutant sur le GPU de façon parallèle. Des fonctions OpenCL permettent de récupérer leur *id*, c'est-à-dire le numéro du thread courant, ce qui permet d'affecter chacun des threads à des éléments différents du tableau que le noyau traite. Par exemple, pour additionner deux tableaux de 1000 éléments, tous les threads dont l'*id* est inférieur à 1000 sont chargés de réaliser l'addition des *id*èmes éléments des deux tableaux, tandis que les autres threads, eux, restent inactifs. Dans des cas de noyaux plus complexes, les threads ont besoin de communiquer entre eux et de se synchroniser. De plus, le respect de certains paradigmes de programmation est nécessaire pour obtenir de bonnes performances [27]. Les sous-paragraphes suivants traitent les concepts principaux pour répondre à ces deux points.

I.3.4.1 Les mémoires du GPU

Le périphérique (qui est dans notre cas, un GPU) ne peut accéder qu'à sa mémoire interne. Afin de communiquer, l'hôte et le périphérique peuvent effectuer des transferts de données via les fonctions de base de l'API. Ces transferts de données s'effectuent vers la mémoire globale du GPU, qui est permanente, c'est-à-dire qui subsiste entre deux exécutions de noyaux. Les autres types de mémoire plus rapides d'accès que sont la mémoire partagée et privée ne se conservent pas entre deux exécutions de noyaux.
Pour obtenir les meilleurs performances en GPU, il est crucial d'utiliser le moins possible l'accès à la mémoire globale, que ce soit en lecture ou en écriture. Si un noyau nécessite un accès à une même donnée à plusieurs reprises, que ce soit en écriture ou en lecture, il est nécessaire de transférer cette donnée dans une mémoire plus rapide, telle que la mémoire privée du thread ou la mémoire partagée du bloc qui a besoin d'y accéder. La mémoire partagée sert à faire communiquer les threads d'un même bloc, ou pour stocker des données

trop volumineuses pour être stockées dans la mémoire privée. Lorsque l'utilisation de la mémoire globale est indispensable, son accès peut être optimisé en organisant les données dans cette mémoire afin de pouvoir faire des lectures ou écritures coalescentes. Un accès est dit coalescent lorsque les threads adjacents accèdent à des données adjacentes, ce qui est le cas par exemple dans la figure I.7a. Les cœurs des GPU étant groupés par warp, les lectures et écritures ne se font jamais de manière individuelle, mais s'effectuent par paquets. Ainsi, pour que chaque thread du warp récupère sa donnée à traiter, il est possible d'effectuer entre 1 et 32 transactions selon la configuration des données [26]. La figure I.7 montre différentes configurations, dont l'optimale où la mémoire est coalescente, c'est-à-dire où les données à accéder sont contiguës et peuvent être récupérées en une seule transaction.

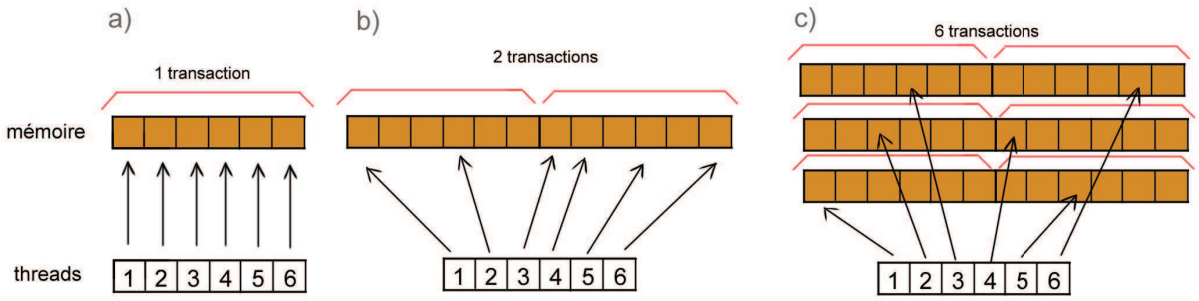


FIGURE I.7 – Accès à la mémoire globale en fonction de la configuration des données à récupérer. Le cas (a) est la configuration optimale où les données sont organisées de façon coalescente. Cela signifie que les threads d'un même warp accèdent à des données contiguës, ce qui leur permet de récupérer ces données en une seule transaction. Le cas (b) est la configuration où les données en mémoire sont décalées. Selon la taille du décalage, le nombre de transactions nécessaires pour récupérer les données est plus ou moins important. Enfin, le cas (c) représente le cas où la mémoire est organisée de façon aléatoire. Dans cette configuration, le nombre de transactions nécessaires est généralement égal au nombre de threads, soit 6 dans cet exemple.

I.3.4.2 La synchronisation

Certains algorithmes nécessitent de pouvoir synchroniser des threads afin de garantir la validé des résultats. Par exemple, pour calculer la moyenne des vélocités des particules présentes dans une cellule, il est nécessaire que la somme des vélocités soit terminée avant d'effectuer la division. Pour répondre à ce problème, OpenCL propose un système de barrières qui permet de faire attendre les threads devant une barrière jusqu'à ce que tous les threads du même bloc l'aient atteinte. Cette barrière ne doit pas être placée dans une branche conditionnelle, mais être atteinte par tous les threads pour éviter une attente infinie. Elle ne permet cependant que de réaliser la synchronisation des threads d'un même

bloc. La synchronisation des threads du même warp est déjà assurée physiquement, les threads n'étant pas indépendants les uns des autres. La synchronisation de tous les threads du GPU se fait par l'utilisation de différents noyaux. En effet, les noyaux s'exécutent séquentiellement, cela signifie qu'il faut attendre qu'un noyau soit terminé avant d'en lancer un autre. Il est donc nécessaire de créer plusieurs noyaux pour effectuer une tâche qui nécessite des synchronisations globales.

I.3.4.3 Les opérations atomiques

Une autre fonctionnalité nécessaire pour certains algorithmes est la possibilité de gérer le problème des accès concurrents. En effet, l'ordonnancement des threads n'étant pas prédictible, rien ne permet d'empêcher qu'une donnée ne soit pas simultanément en accès en lecture ou écriture par différents threads. Si ce cas se produit, la validité des opérations n'est pas prédictible. Ainsi, lorsque plusieurs threads peuvent modifier ou lire une même donnée, il est nécessaire d'employer des opérations atomiques. Ce sont des fonctions d'OpenCL qui rendent inaccessible l'accès à une donnée aux autres threads pendant l'exécution de l'opération, s'assurant ainsi le monopole des données qu'elle traite. Avant d'effectuer cette opération, le thread teste si la donnée à traiter n'a pas été réservée par un autre thread jusqu'à ce qu'elle soit libre. Ensuite, il réserve l'accès à cette donnée, empêchant tout autre thread d'y accéder. Enfin, lorsque l'opération est terminée, le thread libère la donnée. Les opérations atomiques sont plus coûteuses en temps, du fait des tests qu'elles requièrent et du temps d'attente des threads lorsque la donnée à traiter n'est pas disponible. Néanmoins, ce problème tend à diminuer avec les architectures GPU les plus récentes, permettant d'avoir des performances pour des opérations atomiques de plus en plus proches de celles obtenues avec des opérations classiques [26].

I.3.4.4 Le problème des branches divergentes

Le programmeur a la possibilité de regrouper les threads dans des blocs, qui sont logiquement regroupés eux mêmes en une grille de blocs. Les threads sont quant à eux groupés en warps par groupe de 32, car dans les GPUs actuels, les cœurs sont physiquement liés par groupe de 32. Une des limitations du GPU liée aux warps vient du fait que leurs threads doivent tous exécuter la même instruction au même moment. Cette limitation pose problème lorsque les threads d'un même warp ont des chemins d'exécution différents, à cause de branches conditionnelles différentes telles que des branches IF ou WHILE. Les threads devraient alors exécuter des instructions différentes, ce qui n'est pas possible. En réalité, les threads du warp qui sont dans des branches différentes de l'instruction en cours, deviennent inactifs et le warp exécute les instructions de toutes les branches de façon séquentielle. Ainsi, ce problème de branches divergentes nuit au parallélisme et

peut entraîner des mauvaises performances [26]. Il est donc nécessaire de les éviter au maximum dans les noyaux.

I.3.5 La recherche de voisinage sur GPU

La liste de Verlet Sur GPU, l'implémentation des listes de Verlet, structure décrite dans le paragraphe I.2.3.1, conduit à une utilisation mémoire intensive. En effet, les listes sont implémentées en GPU comme des tableaux à taille bornée en mémoire globale, la borne étant ici le nombre maximal de voisins qui doit être fixé arbitrairement, et généralement très au-dessus du nombre effectif de particules dans la liste. Pour réduire ce surcoût mémoire, seuls les index des particules voisines sont en réalité stockés dans ces listes.
De plus, le test souvent utilisé pour mettre à jour ces listes, consiste à tester si la somme des deux plus grands déplacements depuis la dernière réactualisation des listes de Verlet est supérieure à une valeur limite. Ce test nécessite de déterminer deux maxima, ce qui nécessite en GPU d'utiliser deux fois un patron de programmation nommé réduction, dont le coût n'est pas négligeable. Ce patron de programmation consiste à traiter le problème de manière récursive dans chaque bloc, c'est-à-dire que chaque bloc est chargé de trouver et d'écrire la valeur maximale des données qui lui sont associées, dans un buffer résultat en mémoire globale, à l'index correspondant à son numéro de bloc. Ce processus est renouvelé sur le buffer résultat, jusqu'à ce que la taille du buffer résultat ne contienne plus qu'un élément, qui équivaut au maximum de tous les déplacements. Ce processus nommé réduction, permet de traiter ce genre de problème en une complexité de $O(\log N)$. La recherche d'un maximum dans un bloc, quant à elle, s'effectue également par une réduction réalisée au niveau des warps, en synchronisant les différents warps par des barrières et en écrivant le résultat dans la mémoire partagée.

La grille régulière L'implémentation d'une grille régulière (décrite dans le paragraphe I.2.3.1) sur GPU n'est pas triviale, car la solution évidente de stocker les particules d'une cellule dans une liste est mal adaptée au GPU. En effet, comme vu précédement avec les listes de Verlet, l'utilisation de listes est très coûteuse en mémoire sur GPU. De plus, le principal problème vient de la manière d'insérer des éléments dans cette liste : la méthode intuitive de parcourir toutes les particules en parallèle et d'insérer cette particule dans la liste de sa cellule, pose des problèmes de concurrence coûteux à résoudre, car il faut alors utiliser des opérations atomiques. C'est pourquoi, des méthodes ont été développées afin d'implémenter une grille régulière, sans utiliser de liste. Dans la littérature, la construction de grilles régulières en parallèle sur GPU repose sur un tri en parallèle des particules selon leurs coordonnées spatiales [28]. À chaque particule

est associée une clé de hachage liée à la cellule qui la contient, de telle manière que toutes les particules d'une même cellule possèdent la même clé de hachage. Les particules sont alors triées selon leur clé de hachage, et ainsi, les particules d'une même cellule deviennent adjacentes en mémoire. Pour parcourir les particules d'une cellule, il suffit alors de connaître le premier et le dernier index des particules de la cellule dans le tableau contenant toutes les particules, données qui sont stockées dans un autre tableau. De plus, le fait de parcourir des particules proches en mémoire est très adapté au GPU, car il permet d'utiliser la bande passante de façon très efficace. Cette cohérence mémoire peut être également améliorée par un choix judicieux de l'ordre des cellules pour le tri c'est-à-dire dans l'association clé-cellule, afin que les cellules adjacentes soit proches en mémoire. Goswami *et al.* [29] propose d'utiliser l'ordre de Morton, aussi connu sous le nom de Z-curve, une fonction préservant la localité spatiale avec une structure récursive par blocs [30]. Plutôt que de trier les cellules comme un tableau à n dimensions transposé dans un tableau à une dimension (index order), la Z-curve organise l'espace par 2^n blocs de n dimensions. Les clés sont alors calculées en entrelaçant les bits des coordonnées spatiales de la cellule (voir figure I.8), ce qui peut être également obtenu pour une question d'efficacité par une table de consultation [29]. Combiner les listes de Verlet avec une grille régulière revient donc à utiliser la stratégie de conservation du voisinage par les listes de Verlet en accélérant la construction du voisinage à l'aide d'une grille régulière. De plus, l'utilisation du Z-index permet de créer des listes de Verlet avec une mémoire plus cohérente, dans lesquelles les particules proches spatialement sont également proches dans leur liste. Cela permet d'obtenir des listes de Verlet optimisées pour l'étape de calcul des forces.

I.3.6 La génération de nombres aléatoires sur GPU

Dans le cadre de la simulation de suspensions colloïdales, des nombres aléatoires sont utilisés généralement pour calculer les trajectoires des particules. Pour résoudre par exemple l'équation (I.6), il faut générer un nombre aléatoire gaussien pour chaque coordonnée spatiale, ce qui implique avec la méthode de Box-Müller de générer deux nombres aléatoires uniformes. Dans les algorithmes séquentiels vus dans le paragraphe I.2.3.2, cela consiste à générer les différents nombres aléatoires de la première particule, puis de réeffectuer ce même traitement pour les suivantes de manière itérative. L'utilisation du GPU a donc pour but de réaliser ce même processus, mais cette fois-ci en traitant chaque particule en parallèle. Chaque thread va devoir générer des nombres aléatoires décorrélés des autres nombres générés par les autres threads de manière indépendante. Cependant, il n'est pas possible avec le générateur aléatoire de Marsaglia et Zaman, de générer plusieurs nombres aléatoires en parallèle. En effet, pour pouvoir obtenir le nombre i de la suite aléatoire générée par le PRNG, il faut calculer auparavant

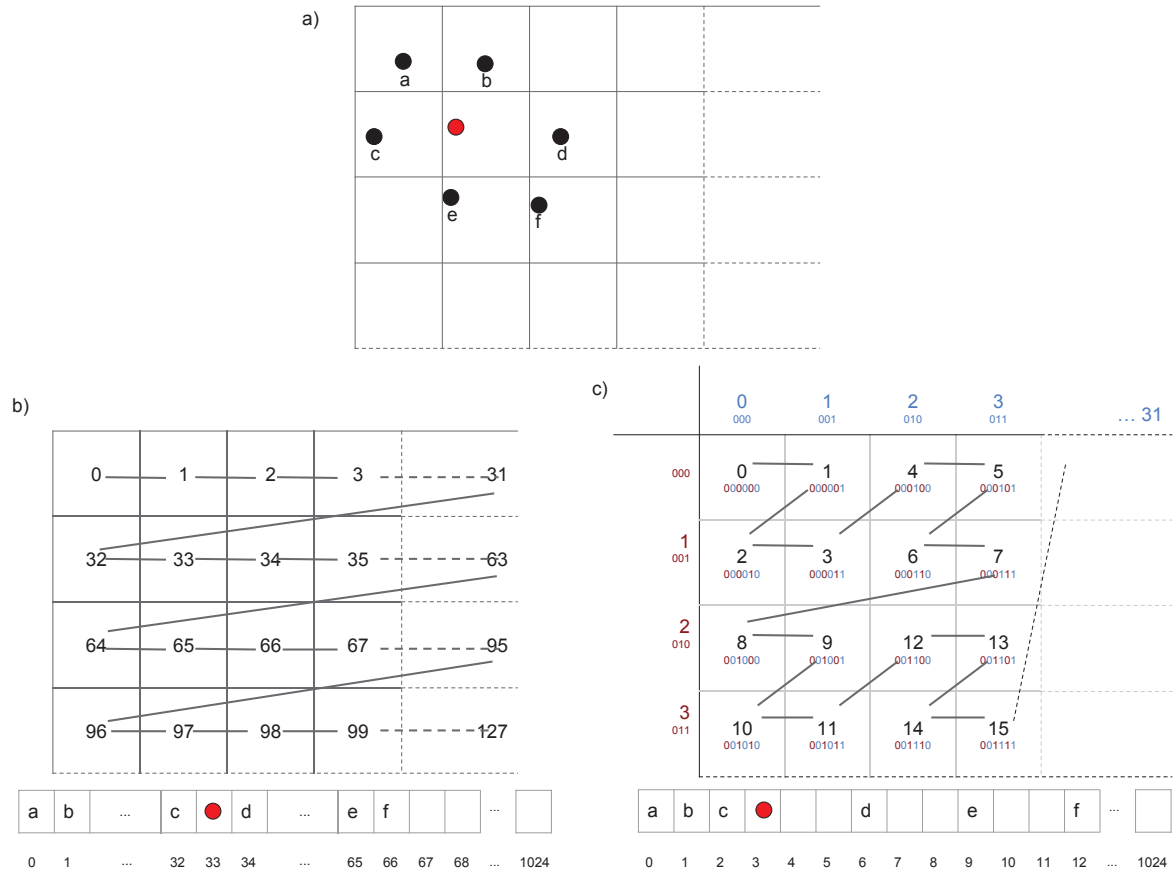


FIGURE I.8 – (a) Configuration spatiale des particules dans une boîte de simulation en 2D. Comparaison entre (b) l'ordre index standard et (c) l'ordre du Z-index. Avec le Z-index, les particules proches spatialement sont plus proches en mémoire qu'avec l'ordre standard.

le nombre $i-1$ afin de basculer le PRNG à son état i. Ce générateur est donc purement séquentiel. Une alternative pour répondre au problème est d'associer un PRNG à chaque thread. Néanmoins, cette méthode pose des problèmes en termes de performances en temps et de coût mémoire très important. En effet, stocker l'état du PRNG de Marsaglia et Zaman nécessite une centaine de nombres flottants, soit 800 octets au total, et ce pour chaque particule. Ainsi, pour des simulations avec 1 million de particules, le coût mémoire des PRNG est de 800 Mo tandis que la position des particules coûte 32 Mo à titre de comparaison, et donc la majorité de la mémoire utilisée par la simulation serait prise par l'état des PRNG. De plus, l'état des PRNG se trouvant dans la mémoire globale, l'accès et les modifications de cet état sont également coûteux en temps. Une alternative est de réduire le nombre de PRNG en employant le modèle du thread persistant. Ce modèle se base sur le fait que le GPU ne peut pas lancer un nombre infini de threads concurrents. Au lieu d'associer un thread à chaque particule, ce modèle associe un thread à un ensemble de particule, ce qui correspond à ré-associer manuellement aux threads

une nouvelle particule, une fois sa série d'instructions terminée. Ce modèle permet de considérablement réduire le nombre de PRNG à stocker en mémoire. De plus, chaque générateur va pouvoir générer un grand nombre de nombres aléatoires, ce qui permet de copier l'état des PRNG dans la mémoire privée des threads actifs, en compensant des transferts entre la mémoire globale et privée par une réduction importante des accès en mémoire globale. Cette méthode de thread persistant pour générer des nombres aléatoires a été implémentée dans toutes les simulations de suspensions colloïdales de cette thèse, et les résultats en terme de temps montrent une accélération d'un facteur de 4 à 5 par rapport à la méthode associant un PRNG par particule.

Les différentes méthodes d'implémentation GPU présentées ici sont utilisées dans les chapitres qui suivent, en utilisant OpenCL. Le chapitre suivant est consacrée à la parallélisation de simulations de BD pour des suspensions colloïdales sur GPU.

Chapitre II :

Les simulations de dynamique brownienne sur GPU

Dans le chapitre précédent, nous avons décrit le modèle de dynamique brownienne dans la partie I.2.1 et présenté la littérature traitant du problème de recherche de voisinage dans la partie I.2.3.1. Dans ce chapitre, nous allons dans un premier temps présenter un algorithme optimisé de recherche de voisinage dans le cas d'une simulation de BD simple, qui a donné lieu à une publication dans la conférence VRIPHYS 2015 [18]. Par la suite, cet algorithme sera adapté dans le cas de simulations de BD avec des particules de tailles différentes et dans un cadre de simulation de nanoalliage qui s'applique sur des atomes.

II.1 Les simulations de dynamique brownienne avec un type de particule

Dans cette section, nous allons nous concentrer sur les simulations de dynamique brownienne avec un type de particule, ce qui signifie que toutes les particules du système possèdent les mêmes caractéristiques (masse, rayon, etc). L'approche que nous avons développée combine une grille régulière avec des listes de voisinage comme les travaux cités précédemment, avec une différence notable : au lieu d'utiliser un nombre fixe d'itérations pour réactualiser les listes de voisinages, nous nous basons sur un test simple appliqué à chaque itération. Nos simulations GPU de BD dans cette partie ont été implémentées pour des systèmes de particules d'alumine en suspension dans un solvant tel que l'eau [1]. Les particules d'alumine ont un diamètre $d_A = 600$ nm. Les forces d'interaction entre les particules sont modélisées par le potentiel généralisé de Lennard Jones, un potentiel attractif de courte portée défini par :

$$\begin{cases} V_{ij}(r_{ij}) = 4\epsilon_p k_B T \left[\left(\frac{d_A}{r_{ij}}\right)^{36} - \left(\frac{d_A}{r_{ij}}\right)^{18} \right] \text{ si } r_{ij} \leqslant R_C, \\ V_{ij}(r_{ij}) = 0 \text{ sinon,} \end{cases} \tag{II.1}$$

où d_A est le diamètre d'une particule, R_C est le rayon de coupure représentant la distance maximale pour laquelle deux particules interagissent, et ϵ_p le puits de potentiel fixé à $\epsilon_p = 14$. Le système évolue dans de l'eau à une température de $T = 293\,\text{K}$ et de viscosité $\eta = 10^{-3}\,\text{Pa.s}$. Le temps de relaxation de la vitesse d'une particule d'alumine est de $\tau = 8.5 \times 10^{-8}\,\text{s}$. Les particules d'alumine se trouvent dans une boîte cubique de simulation soumise à des conditions périodiques. La taille de la boîte de simulation est fixée en fonction du nombre de particules (n) et de la fraction volumique de la simulation (ϕ) par $L_{box} = \sqrt[3]{\frac{4\pi n a^3}{3\phi}}$.
La fraction volumique ϕ de la simulation est le rapport du volume total représenté par les particules d'alumine sur le volume de la boîte de simulation. La simulation est basée sur l'équation (I.6) avec un pas de temps de $\Delta t = 3.685 \times 10^{-7}$ s ([1]), ce qui implique de

calculer environ 2.5M d'itérations par seconde. Lorsqu'on utilise une grille régulière pour la recherche de voisinage, la taille des cellules doit être supérieure ou égale à R_C pour que la recherche des particules en interaction avec une particule i consiste uniquement à parcourir les cellules adjacentes à celle-ci. En effet, l'ensemble des particules voisines N_i ainsi obtenue, contient alors toutes les particules qui interagissent avec la particule i. Une taille de cellule égale à R_C est optimale pour la recherche de voisinage. En effet, elle permet d'obtenir un ensemble N_i contenant moins de particules qui ne sont pas en interaction avec la particule i et ainsi, évite au mieux des calculs inutiles.
Si la recherche de voisinage s'effectue à partir de listes de Verlet, un autre paramètre est nécessaire, le rayon de conservation R_L illustré dans la figure II.1. Le rayon R_L doit être supérieur à R_C. Il représente la distance maximale pour laquelle une particule j est considérée comme étant dans la zone de conservation de la particule i.

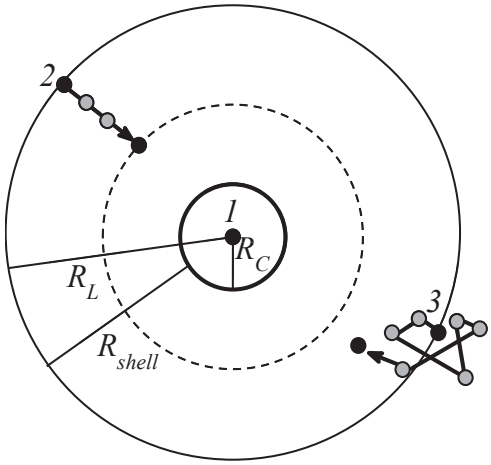


FIGURE II.1 – Une particule (1) avec à l'état initial, une liste de voisinage vide, définie par le rayon de conservation R_L. Si les particules 1 et 2 parcourent une distance de $R_{shell}/2$ en se déplaçant l'une vers l'autre, alors leurs listes de Verlet doivent être recalculées car les deux particules rentrent en interaction, étant à une distance de R_C l'une de l'autre. Contrairement à cet exemple, dans une simulation de BD, où les particules ont un mouvement erratique, les particules mettent la majorité du temps un grand nombre d'itérations avant de parcourir cette distance, comme le montre la particule 3.

La valeur de R_L a un fort impact sur les performances de la simulation [5]. Si $R_L \simeq R_C$, le nombre de voisins que contiennent les listes de voisinage est alors faible et donc la recherche des particules en interaction avec une particule i pour le calcul des forces est rapide. Cependant, ces listes deviennent obsolètes en très peu d'itérations, ce qui implique des réactualisations des listes très régulières et très coûteuses. Inversement, si $R_L >> R_C$, les listes de voisinages peuvent être utilisées pendant de très nombreuses itérations, mais le calcul des forces devient bien plus coûteux. Une valeur optimale de R_L se trouve entre ces deux extrémités : il faut avoir une zone de conservation suffisamment large pour pouvoir conserver la liste de voisinage pendant un grand nombre d'itérations tout en ayant

des listes relativement petites. Cette valeur optimale varie en fonction du type de force d'interaction. Ces trois types de cas sont représentés dans la figure II.2. Afin de trouver la

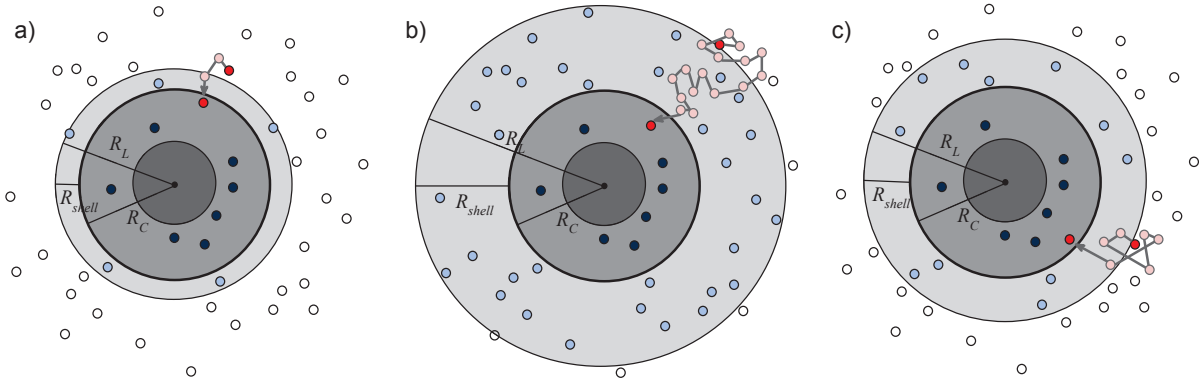


FIGURE II.2 – Influence de la taille de R_L : dans le cas a) R_L est proche de R_C. Cela implique que le nombre de voisins est faible et que le temps pour parcourir les listes de Verlet est court. Cependant, cela implique également que ces listes de Verlet se réactualisent très fréquemment (après deux itérations dans notre exemple). Dans le cas b), R_L est très grand comparé à R_C. Les listes de Verlet peuvent donc être réutilisées pendant un grand nombre d'itérations (18 dans l'exemple), mais en contrepartie, ces listes contiennent un nombre de voisins très important également. Ce surplus de voisins rend le nombre de paires de particules à tester très important, ce qui rend l'étape de calcul des forces coûteuse alors qu'elle s'exécute à chaque itération. Le cas c) représente le cas intermédiaire où R_L est suffisamment grand pour pouvoir conserver le voisinage durant un nombre conséquent d'itérations, tout en ayant un nombre de voisins modéré permettant que le surcoût engendré par le surplus de voisins dans l'étape de calcul de forces soit faible.

valeur optimale de R_L, il faut déterminer quand les listes de voisinage sont réactualisées. En théorie, il suffirait de réactualiser une liste de voisinage uniquement lorsque celle-ci est devenue obsolète, c'est-à-dire qu'une particule qu'elle ne contient pas vient de rentrer en interaction avec sa particule associée. Cependant, le test permettant de vérifier cela, revient à faire une recherche de voisinage. Il faut donc trouver un moyen simplifié, pour déterminer uniquement si une liste est possiblement obsolète, quitte à la réactualiser plus souvent que nécessaire. Afin d'avoir un test simple, les méthodes proposées dans la littérature reposent toutes sur un test global, indiquant au programme s'il doit mettre à jour toutes les listes de Verlet, et non sur un ensemble de tests locaux à chaque liste. Elles se basent toutes sur la configuration où deux particules sont juste en dehors de leur zone de conservation, et détecte s'il existe une possibilité pour que ces deux particules soit entrées en interaction, à partir des distances parcourues par toutes les particules.
Soit *dist* la somme des deux déplacements maximaux de particules du système depuis la dernière mise à jour. Allen et Tildesley proposent de vérifier si $dist \geqslant R_{shell}$ ($R_{shell} = R_L - R_C$) [5]. En effet, comme le montre la figure II.1 dans le cas 3, si ces déplacements maximaux sont ceux de ces deux particules, et dans la direction l'une de

l'autre, elles entrent alors en interaction. Réciproquement, il est certain qu'aucune liste de voisinage n'est obsolète tant que $dist < R_{shell}$. Ce test est utilisé dans les implémentations non parallèles mais sur une implémentation GPU, la recherche des deux déplacements maximum doit s'effectuer par deux réductions, ce qui rend le test plus coûteux. D'autres tests ont donc été développés pour ces architectures. Wang utilise une mise à jour à partir d'un nombre d'itérations fixe n_i [14] basé sur :

$$R_L = R_C + 2v_{max}n_i\Delta t \ , \ v_{max} \approx v_{moy} = \sqrt{\frac{3k_BT}{m_i}} \tag{II.2}$$

où n_i est un nombre fixe d'itérations, Δt le pas de temps et v_{max} est la vitesse maximale de toutes les particules du système. Cette équation correspond à la figure II.1 dans le cas 2, où les deux particules se déplaceraient à leur vitesse maximale l'une vers l'autre et entreraient en interaction. Il est alors possible de déduire qu'avant n_i itérations, il est impossible que ces deux particules entrent en interaction. Et donc de n'avoir à réactualiser les listes que toutes les n_i itérations, sans avoir à effectuer de test. De manière pratique, cette équation ne peut pas être employée dans la plupart des cas, car la vitesse maximale n'est pas souvent déterminable. Il est par contre en pratique possible de l'utiliser, en remplaçant la vitesse maximale par la vitesse moyenne v_{moy}, et en prenant une valeur pour n_i suffisamment élevée pour pouvoir supposer que la distance $R_{shell}/2$ ne soit jamais franchie. Le problème de cette méthode est qu'elle repose sur un scénario peu vraisemblable dans le cas de suspension colloïdale, car les colloïdes ont tendance à avoir un mouvement erratique. C'est pourquoi, cette méthode réactualise excessivement les listes de Verlet pour nos types de simulation.

II.1.1 L'approche hybride pour la recherche de voisinage : grille régulière / liste de Verlet

Notre méthode utilise une combinaison entre des listes de Verlet et une grille régulière comme dans la littérature présentée dans le paragraphe précédent. Elle utilise également un test dynamique pour déterminer quand il faut recalculer les voisinages de toutes les particules comme le test d'Allen utilisé pour les simulations CPU. Cependant, le test choisi a été simplifié pour ne vérifier qu'une seule distance au lieu d'une paire de distance. En effet, les listes de Verlet sont valides tant qu'aucune particule ne s'est déplacée de plus de $R_{shell}/2$ depuis la dernière mise à jour. Ce test simplifié ne nécessite pas de réduction et n'utilise que des accès mémoires parfaitement coalescents. Il est donc parfaitement adapté au GPU. L'algorithme 1 décrit le déroulement de notre simulation de BD utilisant notre test simplifié. La première étape d'une itération est de calculer la somme des forces d'interaction appliquées à chaque particule. Pour cela, il faut avoir au préalable une liste

Algorithm 1 Simulation de dynamique brownienne avec un type de particule

```
1: actualiserListesDeVoisinage = vrai
2: for all iteration do
3:     if actualiserListesDeVoisinage = vrai then
4:         tri des particules selon leur clé de hachage
5:         for all particule i do        // en parallèle
6:             PosPrecedente_i = position_i
7:             réactualiser ListeDeVoisinage_i
8:     actualiserListesDeVoisinage = faux
9:     for all particule i do        // en parallèle
10:        sommeDesForces_i = 0
11:        for all particule j ∈ ListeDeVoisinage_i do
12:            if distance(position_i, position_j) < R_C then
13:                calculer sommeDesForces_i
14:        calculer position_i
15:        if dist(position_i, PosPrecedente_i) > R_shell/2 then
16:            actualiserListesDeVoisinage = vrai
```

de Verlet valide pour chaque particule. Si les listes de voisinage sont obsolètes, il faut exécuter une étape intermédiaire pour les mettre à jour. Pour cela les particules sont tout d'abord regroupées selon les cellules de la grille régulière auxquelles elles appartiennent, en utilisant la méthode du tri par z-index. A chaque particule est associée une clé de hachage issu du z-index, puis les particules sont triées selon leur clé à l'aide d'un tri par base, le tri le plus efficace avec une architecture GPU [13]. Ainsi, toutes les particules d'une même cellule sont adjacentes en mémoire. Il suffit alors de calculer l'index de la première et dernière particules d'une cellule, afin de pouvoir accéder rapidement à toutes les particules qu'elle contient. Les index des particules qui se trouvent à une distance inférieure à R_L à la particule i sont stockés dans les listes de voisinage. Une fois que ces listes sont valides, le calcul des sommes des forces s'effectue en lançant un thread par particule i. Chaque thread va parcourir la liste de Verlet associée à i, afin de déterminer les particules se trouvant à une distance inférieure à R_C de la particule i, et de calculer la force qu'elles exercent sur i grâce à l'équation (II.1). Dans le cas où les listes de voisinage viennent tout juste d'être mises à jour, cette étape de calcul des forces est réalisée en même temps que la reconstruction des listes, afin de ne pas reparcourir et recalculer les distances. La deuxième étape consiste à calculer la nouvelle position à partir des forces d'interaction précédemment calculées, en y ajoutant une force liée à la viscosité et une force erratique, suivant l'équation (I.6). De plus, le déplacement de toutes les particules depuis la dernière mise à jour est également calculé et stocké afin de pouvoir réaliser l'étape suivante. La dernière étape est l'élément principal qui diffère avec les travaux précédents de la littérature. Le but est de déterminer si les listes de voisinage peuvent être potentiellement obsolètes durant la prochaine itération. Notre test consiste à vérifier

si une des distances parcourues depuis la dernière réactualisation des listes de voisinage a dépassé le seuil limite de $R_{shell}/2$. Cette étape est très rapide à réaliser sur GPU : il suffit de lancer un thread par particule afin de regarder si son déplacement a dépassé la limite. Si c'est le cas, il va modifier le marqueur indiquant si les listes doivent être mises à jour. Ce marqueur est une variable globale à tous les threads, cependant, il n'y a pas de problème de concurrence. Même si deux threads écrivent dans cette variable en même temps, la seule chose nécessaire de vérifier est uniquement si ce marqueur global a été modifié au cours du test. Ainsi, ce test ne représente qu'une petite portion du temps moyen total pour calculer une itération comme le montre le tableau II.1. Comparé au test de la littérature, notre test va mettre à jour les listes de Verlet légèrement plus régulièrement, mais la réactualisation reste tout de même bien moins excessive qu'avec la méthode avec un nombre d'itérations fixe. Ainsi, le goulot d'étranglement qui sans méthode de conservation était la recherche de voisinage (réactualisation des voisins, tri et test de réactualisation), ne représente plus que 13% du temps d'exécution de la simulation, contre près de 60% pour le calcul de la somme des forces. Sur GPU, les listes de voisinage correspondent à un seul tableau en

	Temps moyen	Itérations	Ratio
Calcul somme forces	7.15 ms	1	57.7%
Calcul positions	3.6 ms	1	29.2%
Test reactualisation	0.4 ms	1	3.2%
Tri par base	29.5 ms	51	4.6%
Reactualisation des voisins	33.5 ms	51	5.3%

Tableau II.1 – Temps moyen pour exécuter les différentes étapes de l'algorithme 1 et le pourcentage sur le temps total de calcul sur toute la simulation (1M de particules, ϕ =15%). La 3e colonne montre le nombre moyen d'itérations entre deux exécutions de cette étape.

mémoire global, dont chaque sous-partie d'une taille fixe de $N_{L_{max}}$ cases, correspond à une liste de voisinage. Cette valeur de $N_{L_{max}}$ doit être suffisamment grande afin que chaque particule ait assez d'espace pour stocker tous ses voisins, mais d'une grandeur raisonnable pour ne pas consommer excessivement de mémoire. Un maximum théorique peut être calculé à partir du théorème de Lagrange, démontrant que l'agencement compact le plus dense d'un ensemble de sphères de rayon 1 est de $\frac{\pi}{3\sqrt{2}}$. Ainsi, il est garanti d'avoir une valeur de $N_{L_{max}}$ suffisamment grande en employant l'équation suivante :

$$N_{L_{max}} = \left\lceil \frac{\frac{4\pi(R_L+r)^3}{3}}{\frac{\pi}{3\sqrt{2}} * \frac{4\pi r^3}{3}} \right\rceil \tag{II.3}$$

où r est le rayon d'une particule et R_L est le rayon de conservation associé aux listes de Verlet. Cependant, cet agencement compact n'apparaît jamais dans nos simulations

de BD, ainsi l'utilisateur peut affecter une valeur plus petite que le $N_{L_{max}}$ théorique à ce paramètre. La valeur optimale de R_L n'est pas déterminable automatiquement également. Le moyen par lequel nous l'avons déterminée est en testant des valeurs de R_L empiriquement à l'aide de l'équation (II.2) avec la vitesse moyenne.

II.1.2 Les détails d'implémentation

La structure de données globale de notre implémentation repose sur de grands tableaux situés dans la mémoire globale du GPU pour les attributs :

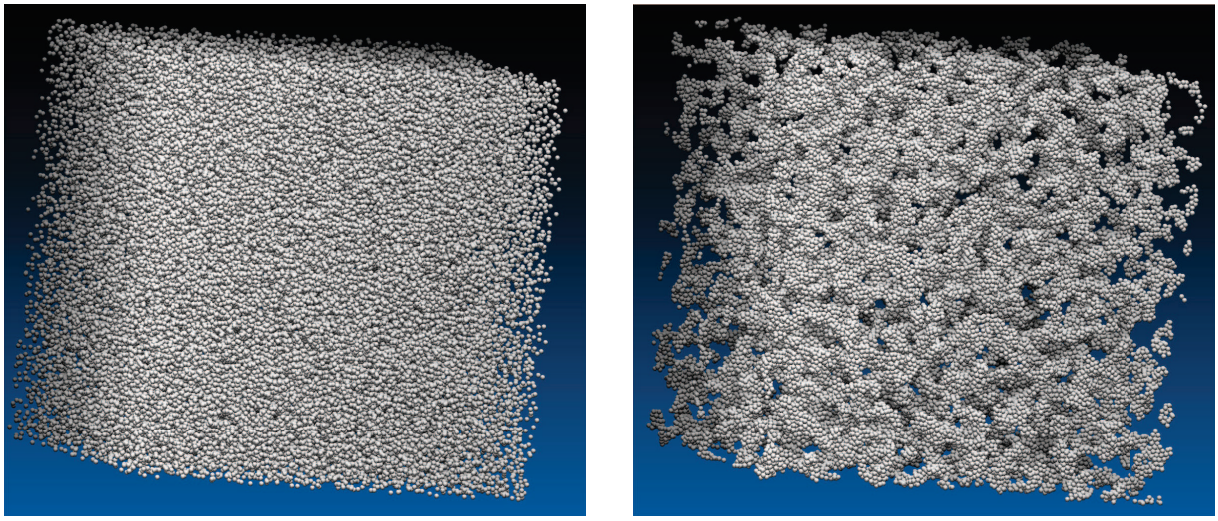
- les positions, les positions triées, les déplacements depuis la dernière mise à jour des listes de voisinage, les vecteurs de force (4 `double4` par particule)
- les clés de hachage et les index pour lier les positions triées et non triées (2 `uint` par particule)
- des listes de voisinage ($N_{L_{max}}$ `uint` par particule)
- des états de générateurs aléatoires (100 `float` par thread lancé)
- la particule de début, de fin de cellule (2 `uint` par cellule)

Le coût mémoire total pour notre simulation de BD avec 1M de particules est d'environ 300 Mo. Ceci peut être diminué en utilisant uniquement la simple précision pour les attributs servant à calculer la somme de forces d'interaction, comme c'est le cas dans quelques logiciels de MD [31]. Cependant notre choix a été de conserver le maximum de précision possible.

La génération des nombres aléatoires s'effectue à partir des techniques décrites dans la partie I.3.6 du chapitre précédent. Malgré ces optimisations, cette étape représente une partie conséquente du temps de calcul d'une itération. Le tableau II.1 montre que l'étape de *calcul des positions*, dont la partie la plus complexe est la génération de nombres aléatoires gaussiens, représente 29.2% de la simulation.

II.1.3 Résultats

Des simulations ont été effectuées pour différentes concentrations volumiques de particules. Une vidéo montrant l'évolution de 60k particules avec ϕ =15% de la figure II.3a à II.3b est visible sur : http://vimeo.com/133039225. Elle correspond aux courbes de la figure II.4 qui décrivent la cinétique d'agrégation des colloïdes en fonction de la fraction volumique [1]. La simulation se déroule comme attendu, en deux phases. Le nombre d'agrégats augmente rapidement en première partie de simulation pendant laquelle la plupart des particules isolées s'agrègent sous forme de dimères. Puis le nombre d'agrégats chute jusqu'à ce qu'il n'en reste plus qu'un seul. La fraction volumique joue un rôle significatif dans le processus : plus les particules sont regroupées, ce qui est le cas



(a) (b)

FIGURE II.3 – (a) État initial de la simulation de BD avec un système de 60k alumines en suspension, (b) État final de la simulation

pour les concentrations les plus fortes, plus les agrégations se font rapidement.

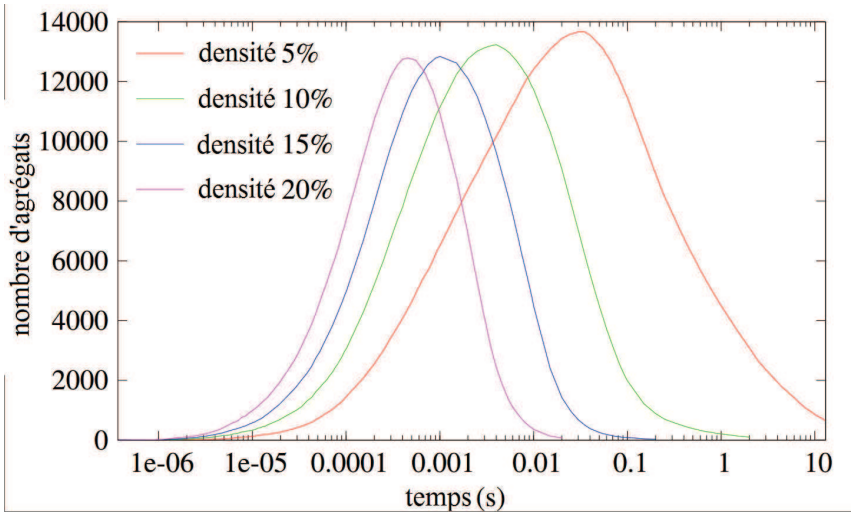


FIGURE II.4 – Evolution du nombre d'agrégats au cours du temps, pendant 10s de simulation de BD, sur une échelle logarithmique, avec 60k particules pour différentes fractions volumiques. Un agrégat est composé d'au moins deux particules.

Notre méthode a été implémentée avec OpenCL et testée sur deux différentes cartes graphiques NVIDIA : la Tesla K20m qui possède 13 cœurs physiques et 5Go de mémoire, et une carte graphique de moyenne gamme, la GTX690, avec 8 cœurs et 2Go de mémoire. La figure II.5 montre que la complexité de notre méthode est linéaire, indépendamment de la fraction volumique du système simulé, permettant de lancer des simulations avec de grands systèmes, uniquement limitées par la capacité mémoire des GPU.

La première étape a été de déterminer la valeur optimale de R_L pour notre type de

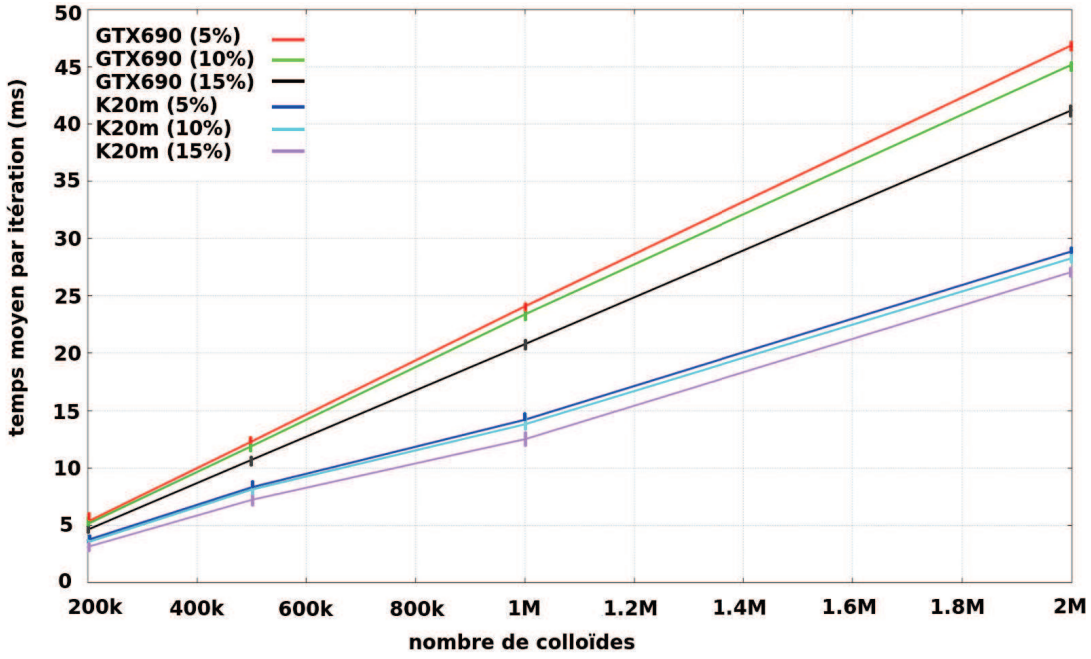


FIGURE II.5 – Temps moyens pour calculer une itération obtenue à partir de notre méthode en fonction du nombre de particules du système et différentes fractions volumiques. Ces tests ont été effectués sur les deux cartes NVIDIA, la Tesla K20m et la GTX690

simulation. Pour cela différents simulations ont été exécutées, en variant ce paramètre. Le nombre d'itérations pendant lequel un voisinage est conservé à l'aide de notre test, a été calculé à chaque itération. Ceci permet de mettre à jour deux variables représentant le nombre d'itérations minimal et maximal durant lequel le voisinage a été conservé sans réactualisation depuis le début de la simulation. Le tableau II.2 présente les résultats, indiquant les deux valeurs trouvées en fonction de la taille de R_L définie par n_i de l'équation (II.2), ainsi que les performances associées. Ces résultats montrent comme prévu, que l'utilisation de la vitesse moyenne n'est pas possible dans le cas d'un n_i trop petit ($n_i \leqslant 8$ dans ce cas-là), car notre test a au moins une fois réactualisé le voisinage en moins de n_i itérations durant toute la simulation. Cependant, la moyenne du nombre d'itération par réactualisation reste au-dessus. Pour un n_i suffisamment élevé ($n_i \geqslant 11$), le nombre minimal d'itérations est supérieur à n_i avec notre test, et le nombre moyen d'itérations par réactualisation est largement supérieur à n_i ce qui montre une mise à jour trop importante des listes de Verlet avec la méthode à itération fixe. Les meilleures performances sont atteintes avec $n_i = 12$ et $n_i = 13$. Par la suite, les simulations ont donc été réalisées avec $n_i = 12$ pour avoir les performances optimales tout en ayant la consommation mémoire la moins importante.

Les seules variations liées à la fraction volumique sont le nombre de voisins un peu plus

		Nombre d'itérations $R_{shell}/2$ avant d'atteindre $R_{shell}/2$			
n_i	R_L	Min	Max	Moyenne	temps moyen par itération
5	9.81×10^{-7} m	0	9	7.22	22.23 ms
8	10.06×10^{-7} m	2	25	17.8	14.9 ms
11	10.06×10^{-7} m	16	52	41.5	12.7 ms
12	10.11×10^{-7} m	18	66	51.0	12.5 ms
13	10.16×10^{-7} m	37	77	61.3	12.5 ms
20	10.57×10^{-7} m	107	209	161	12.8 ms
30	11.09×10^{-7} m	187	384	323.9	13.3 ms

Tableau II.2 – Résumé de l'influence de R_L dans nos simulations de dynamique brownienne à 1 type de particule, en comparant le nombre d'itérations fixe n_i et sa valeur "réelle" pour atteindre la distance limite $R_{shell}/2$ (1M de particules, ϕ =15%). La dernière colonne indique le temps moyen pour calculer une itération, afin de déterminer la valeur optimale du paramètre n_i (qui est 12 dans ce cas précis).

important pour les fractions volumiques les plus grandes et le nombre total d'itérations pour obtenir un seul agrégat dans le système pour les fractions volumiques les plus basses. Ainsi, même si le temps moyen pour calculer une itération est un peu plus important dans le cas de systèmes à forte fraction volumique, le temps total de la simulation reste tout de même très largement plus court.

Les différentes méthodes de recherche de voisinage de la littérature ont été implémentées dans le but de les comparer avec notre approche. Le tableau II.3 récapitule le temps moyen pour calculer une itération selon chaque méthode, pour notre simulation de BD, avec un système de 1M de particules. Les moins bons résultats sont obtenus avec l'utilisation d'une grille régulière seule sans conservation, à cause de l'étape de tri très coûteuse qui doit s'effectuer à chaque itération. La seconde approche utilise une stratégie de conservation du voisinage, avec une réactualisation à nombre d'itérations fixe et en utilisant une grille régulière élargie. Le problème de cette méthode est que le nombre de voisins obtenus par une grille régulière comparé à une liste de voisinage, est en moyenne 8 fois supérieur. Les gains obtenus en n'ayant pas à construire de listes tout en alignant la mémoire, ne compensent pas le surplus de calculs liés à un plus grand nombre de voisins. L'apport d'une stratégie de conservation permet tout de même de diviser le temps d'exécution par 2 par rapport à l'utilisation d'une grille sans conservation. La troisième approche combine une liste de voisinage avec la grille régulière, permettant de diviser en moyenne par 8 le nombre de voisins à parcourir pendant les étapes de calculs des forces. Cela permet de diviser par 2 le temps d'exécution d'une itération, cependant, la mise à jour des listes de voisinage reste bien trop fréquente par rapport au nombre de voisins qui y sont stockés. Notre approche ajoute un test dynamique rapide et adapté

au GPU, pour déterminer lorsqu'il faut réactualiser les listes de voisins, ce qui permet de diminuer considérablement la fréquence de mise à jour de ces listes. Ainsi ce test permet un gain de 20% par rapport à la méthode précédente. Nous avons comparé nos

Méthode de recherche de voisinage	Temps moyen par itération
Grille régulière	58.5 ms
Grille régulière avec conservation	33.6 ms
Grille régulière + liste de voisins (mise à jour statique)	16.5 ms
Notre approche	12.5 ms

Tableau II.3 – Temps moyen de calcul pour une itération avec différentes stratégies de recherche de voisinage, dans le cas d'une simulation de BD à un type de particules (1M de particules, $\phi = 15\%$)

résultats avec le logiciel de dynamique moléculaire Gromacs [31] (v5.02). Gromacs ne proposant que des simulations sur des architectures CPU multicœurs et non GPU, le comparatif a été fait avec des simulations sur deux Intel Xeon E5-2650-v2 de 8 cœurs, ayant chacun 8 Go de mémoire, soit un total de 16 cœurs et 16 Go de mémoire. De plus seule la simple précision est disponible sur cette version de Gromacs. Bien que d'une architecture différente, nous avons comparé les performances en termes de temps avec la carte K20m qui possède 3 cœurs de moins et surtout beaucoup moins de mémoire, mais est équivalente en termes de prix d'achat. Les simulations sur Gromacs ont été réalisées pour une fraction volumique $\phi = 10\%$, en simple précision uniquement et avec des potentiels de Lennard Jones précalculés qui permettent d'accélérer le calcul des forces. Même avec ces paramètres rendant ces simulations sur Gromacs plus performantes, notre approche reste approximativement 15% plus rapide (*e.g.* le temps moyen d'exécution a été de 14.83 ms avec Gromacs).

Les optimisations GPU qui viennent d'être décrites ont pu être utilisées dans diverses études menées par la suite. La deuxième partie est consacrée à l'étude d'une hétéroagrégation de deux types de particules de même taille, la troisième à une hétéroagrégation avec deux types de particules de taille très différente. Enfin la dernière partie présente des travaux en dehors du contexte des suspensions colloïdales, en collaboration avec l'Université de Gênes, sur des simulations de nanoalliages.

II.2 L'étude de l'hétéroagrégation à l'interface de deux suspensions colloïdales

II.2.1 La description du système

Cette partie se consacre à l'étude de l'hétéroagrégation à l'interface de deux suspensions colloïdales qui a donné lieu à une publication dans le journal PCCP [32].
Le système est composé de deux suspensions colloïdales de types différents, mais de taille similaire (a = 300 nm). A l'état initial, les populations ne sont pas mélangées, avec au départ l'ensemble des colloïdes d'un des types se trouvant d'un côté et l'ensemble des colloïdes de l'autre type de l'autre côté (voir figure II.6). Les deux types de particules ont des charges opposées et elles sont présentes en même quantité dans chaque partie de la boîte de simulation. Au cours de la simulation, les particules de charge opposée s'agrègent au centre de la boîte, créant ainsi une couche à l'interface. L'évolution et la structure de cette couche ont été étudiées en faisant varier les potentiels d'interactions entre particules par BD sur GPU.

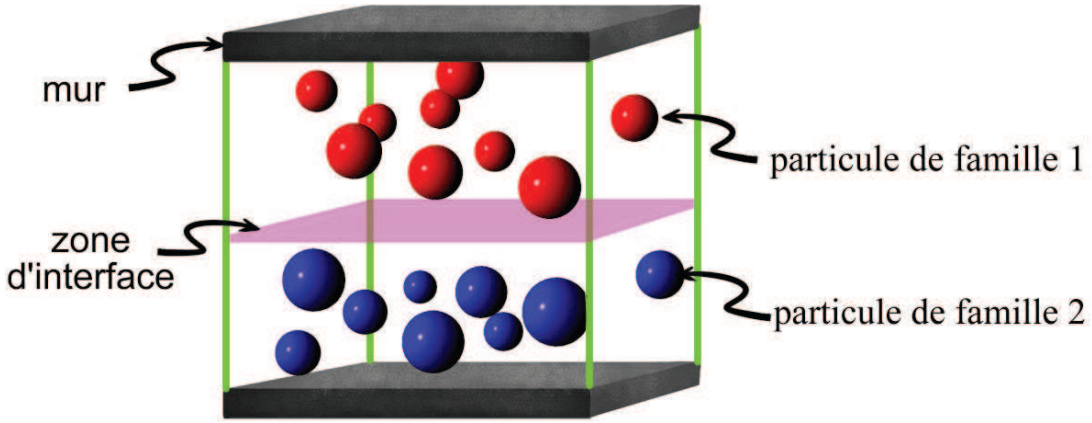


FIGURE II.6 – Schéma du système d'hétéroagrégation à l'état initial. Les deux familles de colloïdes sont placées aléatoirement dans la boîte de simulation, chacune d'un côté de la zone d'interface. La boîte de simulation a des murs sur ses bords pour l'axe z et des conditions périodiques pour les bords en x et y.

II.2.2 Les simulations GPU

Les conditions périodiques ne sont appliquées qu'aux axes x et y, tandis que l'axe z, des murs sont imposés sur les bords de la boîte de simulation, ce qui permet d'obtenir un seul plan d'interface entre les deux suspensions. Ces murs sont modélisés par un potentiel

répulsif qui permet de repousser les colloïdes, selon les formules suivantes :

$$U_{w,bottom} = 36k_BT\left(\frac{2a}{z_i}\right)^{18} \tag{II.4}$$

et

$$U_{w,top} = 36k_BT\left(\frac{2a}{L_{box} - z_i}\right)^{18} \tag{II.5}$$

pour les murs inférieurs et supérieurs, respectivement, avec L_{box} la taille de la boîte de simulation. À l'état initial, les deux populations de colloïdes sont placées initialement dans la partie inférieure ($z_i < L_{box}/2$) et supérieure ($z_i > L_{box}/2$) respectivement dans une configuration aléatoire. Les forces d'interaction entre particules sont représentées par le potentiel de Yukawa :

$$U_{ij}(r_{ij}) = U^* q_i q_j \frac{2a}{r_{ij}} e^{-\kappa(r_{ij}-2a)} \tag{II.6}$$

avec U^* la profondeur du puits de potentiel, $q_i = +1$ pour les particules chargées positivement et $q_i = -1$ pour les particules chargées négativement (ainsi, $q_1q_2 = -1$, $q_1q_1 = 1$ et $q_2q_2 = 1$). κ est l'inverse de la longueur de Debye, et le pas de temps vaut $\Delta t = 3.7 * 10^{-7}s$. À la fin de la simulation, le but est d'analyser l'interdiffusion entre les deux populations de colloïdes ainsi que la structure de l'agrégat au niveau de l'interface, afin de pouvoir déterminer les paramètres favorisant la cristallisation de l'agrégat obtenu. Dans un premier temps, l'étude a porté sur les effets de la profondeur du puits de potentiel et la portée du potentiel d'interaction en gardant une fraction volumique fixe. Dans un second temps, l'influence de la fraction volumique a été étudiée.

II.2.3 Résultats

Plusieurs simulations ont été réalisées en faisant varier les différents paramètres de l'équation (II.6). Le but est au final de déterminer les valeurs optimales de ces paramètres pour la cristallisation, c'est-à-dire un ordonnancement régulier des colloïdes des deux types s'agençant en quinconce. Les simulations ont tout d'abord été effectuées en ne faisant varier qu'un seul paramètre, les autres ayant été fixés à une valeur arbitraire, afin de déterminer et comprendre son influence selon sa valeur en observant l'agrégat final obtenu, puis d'établir la valeur favorisant le plus la cristallisation. Une fois cette valeur établie, le même processus est réalisé, pour un autre paramètre, en prenant cette fois-ci les valeurs favorables déterminées pour les paramètres fixes. C'est ainsi que nous avons pu obtenir les résultats montrés dans la figure II.7.

Les simulations initiales n'atteignaient que 10 s. Ces premiers résultats montraient une cristallisation avancée pour $\kappa = 10$ et 30 tandis que pour $\kappa = 5$ et surtout 2.5, le processus de cristallisation n'a pas encore débuté. En poursuivant les simulations jusqu'à 300 s, nous

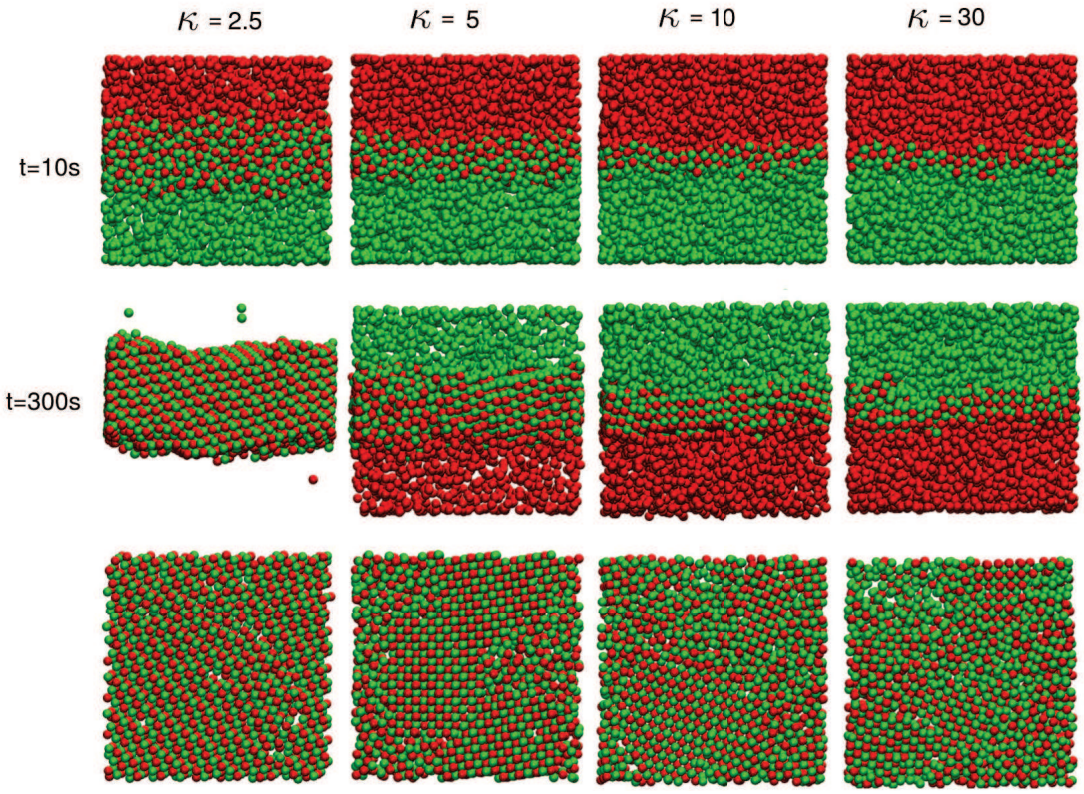


FIGURE II.7 – Captures d'écran de simulations à avec $U^* = 9$ et $\phi = 0.2$ et différentes valeurs de κ. En haut et au milieu, la capture d'écran est prise de face à respectivement $t = 10\,\mathrm{s}$ et $t = 300\,\mathrm{s}$, tandis qu'en bas, seule une coupe d'une épaisseur de 6a de hauteur prise au milieu de la boîte est montrée à $t = 300\,\mathrm{s}$.

nous sommes rendu compte que la cristallisation progresse rapidement entre 150 et 200 s pour $\kappa = 2.5$ tandis que pour les κ les plus élevés, ce processus progresse lentement. Ainsi, il était primordial de poursuivre sur une très longue durée les simulations afin de comprendre réellement l'influence des paramètres, ce qui est impossible sans un algorithme parallèle optimisé. Ces travaux permettent donc d'illustrer l'intérêt fondamental de l'utilisation du GPU pour l'étude de suspensions colloïdales. D'une part, le nombre de particules total du système a pu être augmenté, permettant d'avoir des échantillons statistiques plus importants qu'avec les simulations CPU. D'autre part, les dernières simulations montrées dans la figure II.7, ont été prolongées jusqu'à atteindre 300 s, ce qui représente une semaine de calcul sur GPU, tandis qu'avec des codes séquentiels, le temps total des simulations aurait été de l'ordre de plusieurs mois.

II.3 La simulations de dynamique brownienne avec deux types de particules

Comme l'indique la présentation du modèle de BD dans le chapitre I.2.1, le modèle de BD présente des limitations dues à ses approximations qui ne permettent pas de reproduire certaines suspensions de manière réaliste. Un des systèmes ne pouvant pas être reproduit correctement est celui des suspensions colloïdales avec des colloïdes de tailles très différentes, comme ceux montrés dans la figure II.9. Dans ce cas, avec la BD, la diffusion de l'ensemble formé par un grand colloïde entouré d'un nombre important de petits colloïdes est sous évaluée [8]. Cela entraîne une faible mobilité de ces grands colloïdes et une évolution très lente du système entier qui n'est pas réaliste. Dans ce chapitre, nous allons présenter une modification du modèle de BD afin de mieux reproduire ce type de système, qui a été présentée dans la conférence VRIPHYS 2015 [18]. Dans cette partie, nous allons étudier une simulation plus complexe en poursuivant l'étude [7, 8], pour des suspensions composées de deux types de particules, alumine et silice qui s'hétéroagrègent. La difficulté de ces simulations repose sur le fait que les deux types de particules ont des caractéristiques très différentes, notamment leur rapport de taille qui est très important (1/16). En effet, les particules d'alumine ont un rayon $a_A = 200\,\text{nm}$ et sont chargées positivement, tandis que les particules de silice ont un rayon $a_S = 12.5\,\text{nm}$ et sont chargées négativement. Ainsi, du fait de leur charge électrique, les particules d'alumine se repoussent entre elles, les particules de silice se repoussent entre elles également, mais les particules de silice et d'alumine s'attirent mutuellement. Au cours de la simulation, chaque particule d'alumine attire donc plusieurs particules de silice, les adsorbant et les piégeant ainsi à sa surface. L'ensemble forme alors ce que nous avons nommé des Complexes Silice-Alumine (CSA) (voir figure II.8). Lorsqu'un CSA contient suffisamment de silice, il peut

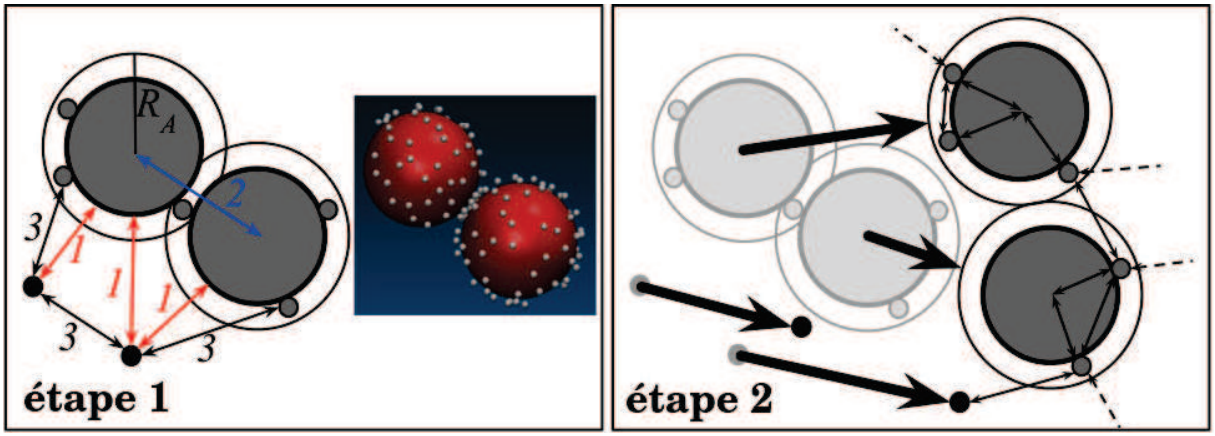
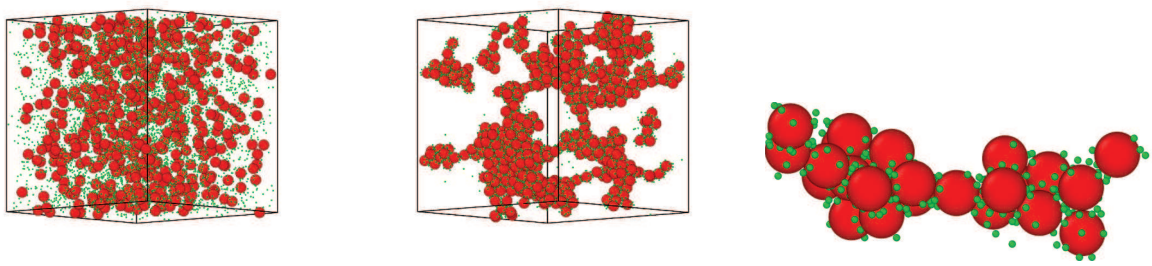


FIGURE II.8 – Schéma illustrant les forces d'interaction entre les deux types de particules du système, une très grande particule d'alumine relativement à une petite particule de silice. Ces interactions sont calculées en deux étapes distinctes. *A gauche :* Force d'interaction entre (1) une silice isolée et une grande alumine (rouge), (2) entre deux alumines (bleu) et (3) deux silices (noir). *A droite :* Après avoir déplacé toutes les particules de silice isolées et les *Complexes Silice-Alumine* (CSA), toutes les forces d'interaction appliquées sur les silices adsorbées sont calculées et ces dernières sont déplacées.

attirer des grosses particules d'alumine ou encore d'autres CSAs. Finalement, ce type de suspension amène à former des hétéroagrégats de CSAs comme le montre la figure II.9. Le modèle utilisé pour représenter les interactions est le potentiel DLVO (nommé à partir



(a) (b) (c)

FIGURE II.9 – (a) Etat initial de la simulation de BD à 2 types de particules, (b) Etat final de la simulation, (c) Formation de CSA

des scientifiques Derjaguin, Landau, Verwey et Overbeek) [33] :

$$V^{DLVO}(r_{ij}) = V^{vdW}(r_{ij}) + V^{el}(r_{ij}), \tag{II.7}$$

où V^{vdW} représente le potentiel d'interaction issu des forces de van der Waals et V^{el} est le potentiel issu des interactions électrostatiques. L'attraction de van der Waals est

Compositions	R=0.2%	R=1.1%
Nombre de silices pour une alumine	15	82
ψ_A (mV)	45	30
ψ_S (mV)	-20	-26

Tableau II.4 – Paramètres utilisés pour les simulations de BD avec des particules de silice et d'alumine.

représentée par l'équation suivante :

$$V^{vdW}(r_{ij}) = -\frac{A_{ij}}{6}\left[\frac{2a_ia_j}{r_{ij}^2-(a_i+a_j)^2} + \frac{2a_ia_j}{r_{ij}^2-(a_i-a_j)^2} + \ln\left(\frac{r_{ij}^2-(a_i+a_j)^2}{r_{ij}^2-(a_i-a_j)^2}\right)\right] \quad \text{(II.8)}$$

où a_i et a_j sont respectivement les rayons des particules i et j et A_{ij} est la constante d'Hamaker dépendant de la nature des particules et du solvant (ce qui donne dans notre cas, pour des particules d'alumine et silice dans de l'eau : $A_{AA} = 4.76 \times 10^{-20}$ J, $A_{SS} = 4.6 \times 10^{-21}$ J and $A_{AS} = 1.48 \times 10^{-20}$ J).

Le potentiel HHF (nommé d'après les scientifiques Hogg, Healy et Fuerstenau) a été utilisé pour modéliser les interactions électrostatiques [34] :

$$\begin{gathered} V^{el}(r_{ij}) = \pi\epsilon\frac{a_ia_j}{a_i+a_j}\left(\psi_i^2+\psi_j^2\right)\times \\ \left[\frac{2\psi_i\psi_j}{\psi_i^2+\psi_j^2}\ln\left(\frac{1+\exp(-\kappa h_{ij})}{1-\exp(-\kappa h_{ij})}\right)+\ln(1-\exp(-2\kappa h_{ij}))\right] \end{gathered} \quad \text{(II.9)}$$

avec κ l'inverse de la longueur Debye ($\kappa = 10^8\,\mathrm{m}^{-1}$), ϵ la constante diélectrique de l'eau, ψ_i le potentiel de surface de i et h_{ij} la distance de séparation entre les surfaces des particules ($h_{ij} = r_{ij} - (a_i + a_j)$). La composition de la suspension est décrite par le rapport massique $R = \frac{m_s}{m_s+m_a}$, ont été utilisées dans nos simulations. Les valeurs des paramètres utilisés dans les simulations sont présentées dans le tableau II.4. Les simulations ont été exécutées avec une fraction volumique $\phi = 0.03$.

II.3.1 L'adaptation de la méthode de conservation avec test dynamique

Le temps de relaxation de la vitesse des particules de silice est de $\tau_v = 7.63 \times 10^{-11}$ s tandis que pour les particules d'alumine, il est de $\tau_v = 3.53 \times 10^{-8}$ s. Comme rappelé dans le chapitre I.1, le modèle de BD utilisé précédemment impose que le pas de temps de la simulation soit supérieur à ces deux valeurs. Le problème est qu'avec un pas de temps aussi élevé, il est impossible de décrire correctement les trajectoires des particules

de silice, car leurs forces d'interaction changent bien trop significativement durant ce laps de temps. Donc il est impossible d'utiliser comme précédemment l'équation (I.6) pour calculer leur mouvement. Pour résoudre ce problème, on peut utiliser l'équation (I.6) pour les particules de silice avec un pas de temps adapté, soit $\delta t = 5 \times 10^{-10}$ s. Et ainsi, il n'est pas possible d'utiliser la même équation pour les particules d'alumine. Il faut donc se servir de l'équation de Langevin complète (équations I.7 et I.8) pour ces particules.
Les trois types d'interactions alumine-alumine, alumine-silice et silice-silice, doivent être traitées séparément, car les distances sur lesquelles elles s'appliquent sont différentes. Pour chaque type d'interaction, il faut donc définir un rayon de coupure R_C et un rayon de conservation R_L. Pour adapter notre méthode de recherche de voisinage de la partie II.1.1, il faut utiliser quatre grilles régulières et stocker quatre types de voisinage, silice-silice, alumine-silice, silice-alumine et alumine-alumine dans trois listes de voisinage séparées. Une méthode alternative avec seulement trois grilles et trois listes de voisinage est également possible, en calculant les interactions alumine-silice et silice-alumine avec la même grille et liste de voisinage. En effet, d'après la 3e loi de Newton, lorsque nous avons calculé la force exercée par une alumine sur une silice, nous pouvons en déduire celle exercée par la silice sur l'alumine. Ainsi, nous pouvons utiliser la 3e grille+liste de voisinage pour connaître les silices voisines aux alumines ou inversement. De même, un test par liste de voisinage doit être effectué pour vérifier leur validité.
L'algorithme 2 résume la façon d'adapter notre méthode avec un système à deux particules, dans le cas d'une stratégie avec uniquement 3 grilles et 3 listes de voisinage, avec la 3e permettant de détecter les silices voisines des alumines. La première étape consiste à tester la validité de chaque liste de Verlet, ce qui peut être appliqué exactement comme précédemment. Le thread vérifie si le déplacement des particules n'a pas dépassé la distance limite $R_{shell}/2$ correspondant au type d'interaction.
Dans le cas où une liste de voisinage est invalide, il faut la recalculer. C'est ici, que notre

Algorithm 2 Simulation de dynamique Brownienne/Langevin avec deux types de particules

```
for all iteration do
    for all type de force d'interaction do
        mise à jour des listes de Verlet si nécessaire
    for all type de force d'interaction do
        calcul des sumForces
    calcul de la nouvelle vélocité des alumines
    calcul de la nouvelle position des alumines
    calcul de la nouvelle position des silices
```

stratégie doit être adaptée. Précédemment, la liste était construite à partir d'une grille régulière, qui se traduisait concrètement par un tri des particules selon leur cellule. Ici,

le problème se situe dans le tri des particules de silice, qui doivent à la fois correspondre au tri selon la grille silice-silice et silice-alumine. Une solution peut être de créer deux tableaux de positions de silice triées, un en fonction du tri pour silice-silice et un autre pour silice-alumine. Le problème de cette méthode est que pour chaque modification des positions et des forces, il faudrait alors modifier les deux tableaux, doublant ainsi toutes les écritures avec une mémoire non alignée.
Un autre choix pourrait être de ne trier réellement les particules de silice que selon l'une des interactions, par exemple silice-alumine, et d'utiliser un tableau d'indexation pour accéder aux silices selon le tri de l'interaction silice-alumine. Le tableau d'indexation permet de savoir où se trouve la silice i selon l'ordre silice-silice, dans le tableau trié dans l'ordre de silice-alumine. Le désavantage de cette méthode est qu'elle lie les deux listes de voisinage silice-silice et silice-alumine. Cela signifie que si les listes de voisinages silice-alumine viennent à être modifiées, les positions triées silice-alumine sont réordonnées, invalidant dans le même temps la liste silice-silice. À chaque mise à jour des listes silice-alumine, il faut donc remettre à jour celle de silice-silice. Cependant, dans le cas de ce type de simulation, ce problème n'a que peu d'impact, car l'interaction silice-silice ayant une distance seuil $R_{shell}/2$ bien plus courte que celle de l'interaction silice-alumine, la réactualisation de son voisinage se produit bien plus fréquemment que celui de l'interaction silice-alumine. La double actualisation n'a donc qu'un impact marginal sur les performances.
La seconde étape est le calcul des forces selon chaque type d'interaction. Les interactions alumine-alumine et silice-silice se traitent exactement comme dans le système précédent (voir partie II.1). Pour l'interaction alumine-silice et silice-alumine, elle s'effectue en revanche avec une seule liste de voisinage alumine-silice. Le calcul de ces forces s'effectue comme précédemment en ce qui concerne la somme des forces des alumines, c'est-à-dire en lançant un thread par alumine. Dans le même temps, chaque alumine va ajouter aux sommes de forces des silices avec lesquelles elle est en interaction, la force opposée qui lui était appliquée. Cela doit se faire par opération atomique, car rien ne garantit qu'un autre thread associé à une autre alumine, ne va pas lui aussi modifier la somme des forces de la même silice. Les opérations atomiques sont coûteuses, mais cela évite d'avoir à maintenir une 4e liste supplémentaire, la mise à jour des listes étant de plus en plus complexe selon le nombre de listes. De plus, ces opérations atomiques tendent à être de moins en moins coûteuses dans les GPUs modernes, c'est pourquoi cette méthode a été privilégiée.
Le déplacement des alumines étant calculé avec l'équation de Langevin complète (équations (I.7) et (I.8)), il faut également ajouter une étape de calcul des nouvelles vitesses. Enfin, les nouvelles positions des silices sont calculées comme dans le système précédent (voir partie II.1), tandis que celles des alumines sont calculées à partir de leur

vitesse selon l'intégration Verlet vitesse issue des équations :

$$r_i(t + \Delta t) = r_i(t) + v_i(t)\Delta t + \frac{1}{2}\frac{v_i(t)}{dt}(\Delta t)^2 \tag{II.10}$$

et

$$v_i(t + \Delta t) = v_i(t) + \frac{1}{2}\left[\frac{v_i(t)}{dt} - \frac{v_i(t + \Delta t)}{dt}\right]\Delta t \tag{II.11}$$

Cependant, en utilisant le modèle de BD et Langevin pur, comme attendu, les particules d'alumine, une fois couvertes de silices, se diffusent beaucoup moins rapidement. Dans la partie suivante, nous allons présenter une extension du modèle pour résoudre ce problème.

II.3.2 La modification du modèle de BD

L'idée de notre méthode est de modifier le mécanisme le comportement des CSAs afin qu'ils puissent se déplacer comme s'il s'agissait d'une seule et même particule. Cela permet d'éviter l'accumulation des forces de friction qui entraîne leur quasi-immobilisation. Avec cette méthode, le comportement d'un CSA se rapproche de celui d'une particule de taille légèrement supérieure à celle d'une alumine. Notre méthode se base sur une décomposition des mouvements des particules du système. Dans une première étape, les particules isolées et les CSAs sont déplacés. Le mouvement des CSAs se calcule seulement à partir des forces exercées par des particules qui lui sont externes. Cela signifie que les forces entre les particules appartenant au CSA ne sont pas prises en compte. Grâce à cette somme des forces, un vecteur de déplacement est calculé à partir duquel la particule d'alumine et toutes ses silices adsorbées formant le CSA sont translatées. Cette étape est illustrée par le schéma II.8 à droite. La deuxième étape consiste à réorganiser les silices adsorbées à la surface de leur alumine associée. Toutes les forces qui s'appliquent à elles sont calculées pour déterminer leur nouvelle position, c'est-à-dire leur interaction avec toutes les alumines à proximité, y compris celle qui les a adsorbées, et toutes les silices à proximité (voir figure II.8 à gauche).

Cette décomposition du mouvement permet de supprimer les effets du cumul de friction sur le déplacement libre, sans avoir à appliquer des calculs complexes pour modéliser des effets de l'hydrodynamique. Pour pouvoir effectuer cette décomposition, il faut dans un premier temps pouvoir déterminer et définir les CSAs. Pour cela, il faut se baser sur R_A, le rayon d'adsorption fixé à $R_A = 1.05(a_A + a_S)$ tel que si une particule de silice se trouve à une distance inférieure à R_A d'une particule d'alumine, alors elle est considérée comme adsorbée par celle-ci. Son mouvement est alors calculé d'après la décomposition décrite ci-dessus. Il faut ajouter à cela que dans notre méthode, une silice ne peut appartenir qu'à un seul CSA. Lorsqu'une silice est à une distance inférieure à R_A de plusieurs alumines, une alumine lui est alors aléatoirement assignée. L'algorithme 3 présente les modifications

de la BD présentée dans la partie II.1 pour simuler notre système à deux particules. Il ajoute une étape pour déterminer les CSAs et la décomposition de leur mouvement en deux étapes : tout d'abord le déplacement de l'ensemble des particules composant chaque CSA comme étant une simple particule, puis la réorganisation de leurs particules de silices à la surface de leur alumine. La détermination des CSAs se fait en associant une nouvelle donnée aux silices, qui indique l'index de l'alumine qui l'adsorbe, ou -1 si c'est une silice isolée. Cette opération se fait à chaque itération à partir des listes de Verlet. Chaque silice parcourt ses alumines voisines et lorsque l'alumine est à une distance inférieure à R_A, elle affecte dans le 4e élément du double4 contenant sa position, l'index de cette alumine. Si aucune alumine adsorbante n'est détectée, alors le 4e élément est affecté à -1. Une seconde méthode pour déterminer les CSAs aurait été de parcourir les silices, à partir des alumines et de leur liste de Verlet. Cependant cette méthode est moins parallélisée, car les alumines étant peu nombreuses par rapport aux silices, peu de threads seraient utilisés et devraient effectuer des calculs plus longs. C'est pourquoi la stratégie basée silice a été choisie.

Ensuite vient l'étape de la première décomposition du mouvement, où seules les forces exercées sur les silices isolées, les alumines isolées et les CSAs sont prises en compte pour leurs déplacements. Ainsi, dans l'étape de calcul des forces pour chaque interaction, le premier calcul se charge de toutes les interactions alumine-alumine. Le second calcule les interactions entre particules de silices qui n'appartiennent pas au même CSA. Le troisième calcule les interactions alumine-silice et silice-alumine dans le même temps par opérations atomiques, en évitant les interactions de particules du même CSA. Le déplacement des alumines étant calculé avec l'équation de Langevin complète (équations (I.7) et (I.8)), il faut également ajouter une étape de calcul des nouvelles vitesses. La nouvelle position des particules peut alors être calculée comme précédemment, excepté pour les CSAs. Pour une question d'efficacité, les étapes *calcul des sumForces des CSAs par l'addition des sumForces de ses particules* et *calcul de la nouvelle position des CSAs et des alumines isolées* sont regroupées dans le même noyau GPU. Ce noyau lance un thread par alumine. Si l'alumine est isolée, il la traite comme auparavant, en calculant sa nouvelle position à partir de sa vitesse précédemment calculée. Si cette alumine forme un CSA, dans un premier temps, il va calculer la somme des forces appliquées aux CSAs. Pour cela, il faut additionner la force appliquée à l'alumine et celles des silices qui lui sont associées. Il faut donc parcourir toutes les cellules voisines afin de détecter celles qui sont marquées comme étant associées à l'alumine, puis accéder à leurs forces associées pour faire la somme. Une fois la force du CSA obtenue, un vecteur de translation est calculé, et est appliqué à l'alumine, ainsi qu'à toutes ses silices associées. L'étape de *calcul de la nouvelle position des silices isolées* s'exécute de la même manière, en évitant les silices adsorbées, déjà déplacées dans le noyau précédent. Pour finir, vient l'étape de réorganisation des silices adsorbées.

Avant de débuter cette étape, il faut à nouveau vérifier que les listes de voisinage silice-alumine et silice-silice sont toujours valides, à cause du premier déplacement. Ensuite l'étape *calcul sumForces pour les silices adsorbées uniquement*, se fait uniquement pour les interactions silice-silice et silice-alumine. Toutes les interactions sont prises en comptes, mais seules les silices adsorbées ont leur somme des forces calculées. Enfin, à partir de cette somme des forces, la nouvelle position des silices adsorbées est calculée.

Algorithm 3 Simulation de BD avec 2 types de particules

```
1: for all iteration do
2:     // étape 1
3:     for all type de force d'interaction do
4:         mise à jour des listes de Verlet si nécessaire
5:     marquage des silices vers leur alumine associée
6:     for all type de force d'interaction do
7:         calcul des sumForces en évitant les interactions entre particules de même CSA
8:     calcul de la nouvelle vélocité des alumines
9:     calcul des sumForces des CSAs par l'addition des sumForces de leurs particules
10:    calcul de la nouvelle position des CSAs et des alumines isolées
11:    calcul de la nouvelle position des silices isolées
12:    // étape 2
13:    for all type de force d'interaction do
14:        mise à jour des listes de Verlet si nécessaire
15:    for all type de force d'interaction (hormis alumine-alumine) do
16:        calcul sumForces pour les silices adsorbés uniquement
17:    calcul des nouvelles positions des silices adsorbées
```

II.3.3 Les résultats et discussion

Le pas de temps de $\Delta t = 5 \times 10^{-10}$ s fait que pour une seconde de simulation, il faut calculer près de 2 milliards d'itérations. Dans ces conditions, même avec un temps moyen par itération de 12.5 ms obtenu dans la simulation BD avec un seul type de particule pour 1M de particules, il nous faudrait plus de 289 jours de calculs. Ainsi, il n'est pas possible d'obtenir des systèmes de même taille que précédemment et la question de la complexité en temps prédomine bien plus que celle du coût mémoire pour ce type de simulation. C'était à cause de ce problème, couplé aux problèmes liés au modèle BD, que la précédente étude [7, 8], n'avait pas pu aboutir lorsque le rapport massique R était trop important. Les améliorations apportées par notre méthode qui résident dans le changement de modèle et l'utilisation d'une architecture parallèle avec une stratégie de conservation grille régulière/listes de Verlet, réactualisées à l'aide d'un test dynamique, permet cependant d'augmenter de façon conséquente la taille des systèmes précédemment

étudiés, dans des temps raisonnables, comme le montre le Tableau II.5.

Alumine	Silice	Temps moyen/iteration (ms)	Temps total (jours)
200	2969	0.85	1.96
2000	29699	2.92	6.76
3000	44549	3.87	8.95

Tableau II.5 – Temps moyen pour calculer une iteration et pour calculer 0.1s de simulation de dynamique Brownienne-Langevin modifiée, sur une carte Tesla K20m

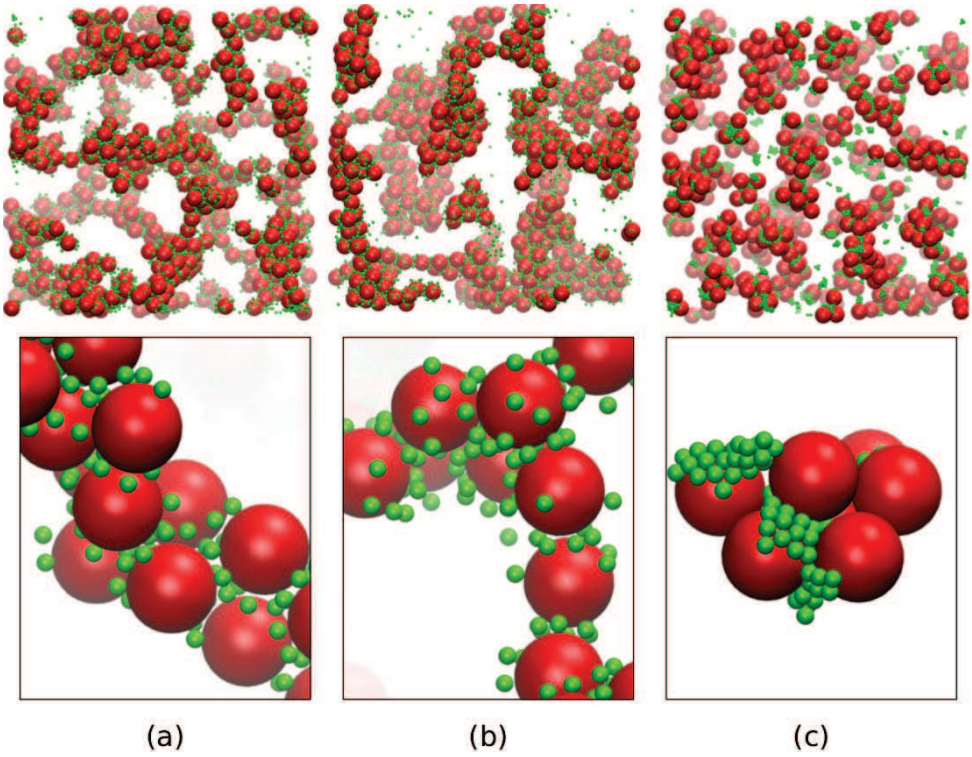


FIGURE II.10 – Capture d'écrans de différentes simulations avec ψ_{Al} = 25 mV, ψ_{Si} = -25 mV et une augmentation progressive de l'inverse de la longueur de Debye, κ, $\kappa = 7 \times 10^7\ m^{-1}$ (a), $\kappa = 10^8\ m^{-1}$ (b), $\kappa = 5 \times 10^9\ m^{-1}$ (c). Une augmentation de κ traduit une augmentation de la concentration en sel dans la suspension [35].

Par la suite, l'agrégation du point de vue des particules d'alumine a été étudié. Un agrégat est donc composé d'au moins deux alumines. En regardant l'évolution du nombre d'agrégats au cours du temps, on trouve un comportement similaire à celui obtenu lors de l'étude précédente pour des homoagrégations. Dans un premier temps, on observe une augmentation du nombre d'agrégats, puis une diminution due à leur coalescence. Le nouveau modèle proposé permet au coefficient de diffusion des alumines de ne plus être autant affecté par l'adsorption de silice, supprimant donc tout effet de cage.
Ces simulations ont ensuite été utilisées pour déterminer la forme des agrégats et le seuil

de percolation dans des suspensions avec un rapport de taille de 5, en fonction de par exemple les potentiels de surface des deux types de particules ou encore l'inverse de la longueur de Debye. Ces résultats ont fait l'objet d'une publication au journal JCIS [35]. La figure II.10 permet par exemple d'illustrer les différents types d'agrégats obtenus, plus ou moins fins et allongés lorsque l'on change la teneur en sel.

II.4 Les simulations de nanoalliages

Au cours de la thèse, un séjour de deux mois a été effectué à l'Université de Gênes afin d'entreprendre un projet collaboratif. Un de leur domaine de recherche est la nanoscience. C'est un domaine interdisciplinaire entre la physique, la chimie et l'ingénierie, dont le but est de comprendre, contrôler et manipuler des objets nanométriques (1 à 100 nm) composés d'au moins deux atomes. Ces nano-objets sont constitués d'une centaine d'atomes pour les plus petits, à plusieurs millions d'atomes pour les plus gros. Ils possèdent des propriétés particulières très différentes des matériaux macroscopiques ou des atomes qui les constituent, variant grandement en fonction de leur taille. Selon leurs propriétés, ils peuvent être utilisés en électronique, en ingénierie, ou dans les processus de catalyse. Dans le cadre de notre collaboration, nous nous sommes intéressés à des systèmes de l'ordre de 2 à 10 nm, une taille intéressante pour des applications en catalyse, et qui correspond pour les plus gros objets à quelques milliers d'atomes. Tout comme pour les suspensions colloïdales, le contrôle du processus de leur formation permet de contrôler leurs propriétés. Les simulations numériques sont également utilisées ici afin de reproduire les expérimentations, se basant sur la MD pour calculer le déplacement des atomes. La problématique principale est l'échelle de temps. Afin de décrire correctement les mouvements des atomes, le pas de temps utilisé pour appliquer la MD doit être de l'ordre de la femtoseconde, tandis que la durée d'une simulation peut atteindre une microseconde. Cela signifie qu'une simulation complète nécessite entre 100 millions et un milliard d'itérations. Il est donc primordial d'utiliser des architectures de calcul parallèles telles que le GPU, afin de pouvoir simuler les plus grands systèmes dans des temps raisonnables. Ces simulations de MD ont beaucoup de points communs avec les autres simulations de suspension colloïdale vue précédemment, en dehors de la différence d'échelle et des forces employées. L'objectif de cette collaboration est donc d'adapter les optimisations vues dans la partie II.1 pour ce nouveau contexte. La simulation a été implémentée en CUDA.

II.4.1 La description de la simulation

Les nanoalliages peuvent être conçus en utilisant diverses techniques. L'une des possibilités est de les produire à l'état gazeux par la condensation d'une vapeur d'atomes de métal de différents types. Dans ce cas, les nanoalliages se forment à haute température puis ils sont refroidis par des collisions avec des gaz inertes de basse température injectés dans la chambre de condensation [36]. Au cours de l'expérimentation, les atomes du nanoalliage se réorganisent vers une structure plus stable, à cause des forces d'interaction de courte portée que les atomes exercent entre eux. Cette réorganisation des atomes est visible sur la figure II.11 qui montre le nanoalliage dans ses états initial et final. Les forces à courte portée impliquent qu'il est donc nécessaire d'effectuer une recherche de voisinage avec un rayon de coupure R_C. Dans le cas des simulations présentées dans cette partie, l'agrégat de départ est constitué de deux types d'atomes, argent et cuivre, disposés dans une structure *cœur-coquille*, constituée par un cœur de cuivre entouré par une coquille d'argent. Les forces d'interaction entre les atomes sont modélisées par le potentiel Gupta, défini par :

$$E_i = A \sum_{j=1}^{n_v} e^{-p(r_{ij}/r_0-1)} - \xi \sqrt{\sum_{j=1}^{n_v} e^{-2q(r_{ij}/r_0-1)}} \qquad \text{(II.12)}$$

où n_v est le nombre d'atomes dans la zone d'interaction à une distance inférieure à R_C, et A, ξ, p et q sont des paramètres de l'agrégat. Ce potentiel est complexe à calculer : il n'est pas possible de l'exprimer sous forme d'une simple somme de termes d'interaction de paires.

Contrairement aux simulations de suspensions colloïdales, la simulation ne se déroule pas dans une boîte de simulation avec des conditions périodiques, mais l'espace de simulation peut être considéré comme infini. Selon les types de simulation, la température augmente ou diminue de façon progressive, ou peut rester constante. Cela dépend du fichier de paramétrage dans lequel sont fixés une valeur de température initiale, de température finale, un incrément de température et un nombre d'itérations entre deux incréments.

II.4.2 L'adaptation des optimisations GPU

Une simulation de nanoalliage se déroule comme les autres simulations de type MD vues dans les parties précédentes. L'algorithme consiste tout d'abord à calculer les forces d'interaction, puis la vélocité de chaque atome à partir des forces qui leur sont appliquées et enfin la nouvelle position des atomes. Le calcul des forces d'interaction nécessite également une recherche de voisinage, qui peut être résolue en adaptant notre méthode optimisée pour les simulations de BD. La principale différence avec les systèmes de suspensions colloïdales est que l'espace de simulation peut être infini. L'utilisation d'une

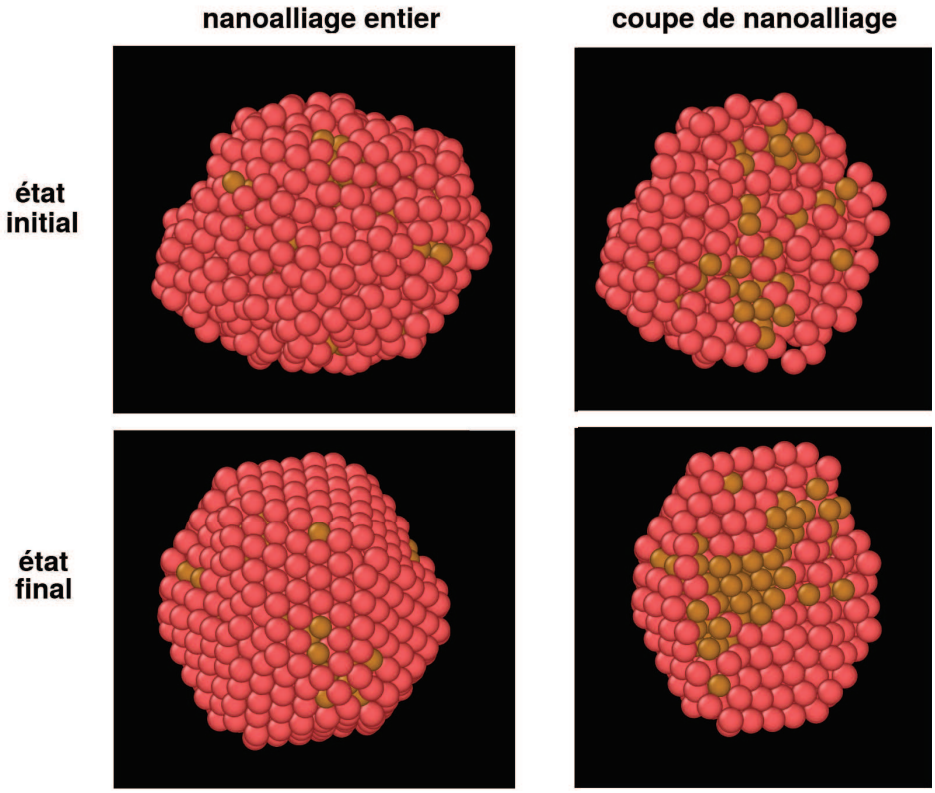


FIGURE II.11 – Captures d'écran d'une simulation avec un système Ag-Cu avec 1289 atomes à l'état initial (haut) et l'état final (bas) avec un affichage du nanoalliage entier (gauche) et d'une coupe transversale (droite).

grille pour la recherche de voisinage doit alors être modifiée, car il n'est plus possible d'associer un index à une cellule avec un nombre de cellules infini. Pour pallier à ce problème, l'idée est d'associer un index à un ensemble de cellules :

$$index_i = p_z nbCells_y nbCells_x + p_y nbCells_x + p_x \tag{II.13}$$

avec

$$\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} \frac{r_x}{L_{box}.x} & \mod nbCell_x \\ \frac{r_y}{L_{box}.y} & \mod nbCell_y \\ \frac{r_z}{L_{box}.z} & \mod nbCell_z \end{pmatrix} \tag{II.14}$$

Ainsi, un index ne permet plus d'accéder uniquement aux atomes d'une cellule, mais tous les atomes d'une même cellule sont regroupés dans le même index.
L'autre différence par rapport aux simulations de suspensions colloïdales est la taille du système, allant de 100 à 15700 atomes pour les simulations de nanoalliages. Ainsi les capacités du GPU ne sont pas toujours totalement exploitées pour les plus petits systèmes. Les algorithmes de tri GPU étant surtout efficaces à partir d'une taille importante de système, plusieurs méthodes de calculs du voisinage ont été implémentées. La première, *Verlet seule*, est celle n'utilisant que des listes de Verlet construites en parcourant tous les atomes du système. Cette méthode est plus adaptée aux plus petits systèmes, où le coût de la création d'une grille et l'application du tri n'est pas compensé par le temps gagné en parcourant moins d'atomes. Au contraire, la deuxième méthode, *grille et Verlet*, qui est l'approche décrite dans la partie II.1 est plus efficace pour les plus grands systèmes. Enfin, la troisième méthode intermédiaire *Verlet sort*, utilise une liste de Verlet seule, en triant également les atomes afin d'organiser de manière plus efficace la mémoire. Elle évite cependant l'utilisation d'une grille qui pourrait être plus coûteuse sur des petits systèmes. Cette méthode pourrait être plus efficace pour des systèmes de taille intermédiaire.
Enfin, la dernière différence avec les suspensions colloïdales est que le système débute directement avec des atomes agrégés. Ainsi, le voisinage des atomes varie peu au cours des itérations comparé aux suspensions colloïdales, la réorganisation des atomes étant lente. Cependant avec nos méthodes, les voisinages ne se conservent pas plus longtemps, à cause du système de réactualisation des voisinages, basé sur la distance parcourue des atomes. En effet, bien que le voisinage évolue peu au cours des itérations, les atomes se déplacent tout de même car l'agrégat global dérive dans l'espace. Cependant, au vu du faible nombre d'atomes dans le système qui rend la recherche de voisinage peu coûteuse, et le nombre déjà important durant lequel les voisinages sont conservés, ajouter des étapes pour ne prendre que les déplacements relatifs à l'agrégat dans la réactualisation du voisinage est plus coûteux.

II.4.3 Résultats

Plusieurs simulations avec différentes tailles de systèmes ont été exécutées afin de pouvoir comparer les performances en termes de temps, entre la version séquentielle Fortran et les différentes approches GPU que nous avons développées. Afin de valider les résultats obtenus, ces simulations ont été exécutées avec des paramètres faisant évoluer le système sur une très longue durée, en diminuant ou en augmentant progressivement la température. A l'état initial, $T = 800\,\mathrm{K}$ et la simulation se termine lorsque le système atteint $T = 300\,\mathrm{K}$, ce qui correspond à une simulation de 100M d'itérations.

Nombre d'atomes	CPU (ms)	GPU Verlet seule(ms)	GPU Verlet triée(ms)	GPU grille et Verlet(ms)
147	0.2	0.63	0.76	0.76
1289	5.2	0.63	0.76	0.76
15700	2318	3.9	2.9	2

Tableau II.6 – Temps moyen pour calculer une itération. La moyenne a été réalisée sur des simulations de 100M d'itérations

Les résultats en terme de performance, affichés dans le tableau II.6, montrent que la version CPU reste plus performante pour les systèmes les plus petits. Cela s'explique par le fait que pour ces systèmes, la majorité de la puissance de calcul du GPU n'est pas exploitée. C'est pour cela qu'en augmentant la taille du système, de 147 à 1289 atomes, le temps moyen d'une itération ne varie pas pour la version GPU. Nos tests nous ont permis de déterminer qu'il reste plus intéressant d'utiliser la version séquentielle pour des nanoalliages avec un nombre d'atomes inférieur à 300 atomes. En outre, la méthode n'utilisant qu'une liste de Verlet s'avère la plus efficace pour les systèmes intermédiaires de 300 atomes à l'ordre d'un millier d'atomes. Pour la simulation Ag-Cu de 1289 atomes, cette version est huit fois plus rapide que la version CPU. Bien que les versions GPU n'utilisent toujours qu'une faible partie de leur capacité, le calcul des forces entre atomes en séquentiel devient trop coûteux. Comparée aux versions *grille et Verlet* ou *Verlet triée*, les meilleures performances de l'approche *Verlet seule* s'expliquent par le coût du tri, qui n'est compensé que très partiellement par une accélération de la recherche de voisinage par rapport à la force brute lorsque le nombre d'atomes est faible. Enfin, pour le système le plus grand de 15700 atomes, la méthode utilisant une grille avec des listes de Verlet est la plus efficace, tandis que la complexité du problème devient trop importante pour envisager d'utiliser une version séquentielle. Celle-ci devient 1000 fois plus lente, et nécessiterait des années de calculs pour atteindre la fin de la simulation. Il faut noter que l'approche *Verlet triée* est plus efficace que la version *Verlet seule* pour le dernier cas, ce qui montre l'importance de la cohérence mémoire dans les calculs sur GPU. Des

essais avec des systèmes d'atomes entre 15700 atomes et 1289 atomes sont nécessaires pour pouvoir déterminer si l'approche *Verlet triée* peut être la plus performante. Ces tests n'ont pu être effectués au cours de la thèse, mais sont envisagés pour la suite.

Ainsi, l'utilisation du GPU permet des gains en temps considérables pour les systèmes de tailles intermédiaires et les plus grands systèmes. Pour le plus gros système de 15700 atomes, une simulation complète s'effectue en environ 55 heures au lieu de plus de 7 ans pour la version séquentielle. L'implémentation d'une fonction *growth* ajoutant des atomes au cours de la simulation et le calcul de l'optimisation globale sur GPU sont les projets à étudier dans une poursuite de cette collaboration.

II.5 Conclusion

Dans ce chapitre, nous avons présenté notre nouvelle approche pour résoudre le problème de recherche de voisinage adaptée dans le cadre de simulations de MD sur GPU. Basée sur la combinaison entre une grille régulière et des listes de Verlet, avec une stratégie de réactualisation du voisinage basée sur un test dynamique adapté au GPU, notre méthode permet de surpasser les précédentes approches. Cela nous a permis de simuler des systèmes de suspensions colloïdales simples composées d'un seul type de particules, avec une taille de système allant jusqu'à 2M de particules sur un seul GPU, en obtenant des performances équivalentes à celles des logiciels de MD tels que Gromacs.

Cette stratégie a été adaptée à des systèmes plus complexes avec des particules de charges électriques différentes, pour des systèmes d'hétéroagrégation, qui ont permis de réaliser des simulations d'une durée inatteignable avec les versions séquentielles. Puis cette stratégie a été adapté pour des systèmes a deux types de particules aux caractéristiques et à la taille très différentes. Il a fallu pour cela utiliser trois grilles et trois listes de Verlet afin de traiter les trois types d'interactions. De plus, il a fallu modifier le modèle de BD classique, en décomposant le mouvement des particules formant un complexe silice-alumine en deux. Cela a permis de supprimer en partie les effets de cage qui apparaissent en utilisant le modèle de BD classique.

De plus, cette approche a pu aussi être adaptée pour des simulations en dehors du contexte des suspensions colloïdales, pour des simulations de MD sur des nanoalliages. Ainsi, ces mêmes optimisations GPU ont pu être implémentées pour diverses études aux cours de la thèse, dans différents domaines, permettant d'étudier des systèmes plus importants sur des temps plus longs, améliorant ainsi les possibilités de recherche. Dans le prochain chapitre, nous allons étudier des modèles plus complexes que la BD, prenant mieux en compte les forces hydrodynamiques exercées par le solvant sur les colloïdes.

Chapitre III :

Simulation de Stochastic Rotation Dynamics - Molecular Dynamics sur GPU

Dans le chapitre précédent, nous avons présenté le modèle de BD où le fluide était simplement représenté par des forces de friction et des forces aléatoires modélisant les collisions des molécules de fluide sur les colloïdes. Dans ce chapitre, nous allons présenter un autre modèle, nommé Stochastic Rotation Dynamics (SRD) ou autrement connu sous le nom de *Multi Particle Collision Dynamics*(MPCD), qui a été introduit pour la première fois par Malevanets et Kapral en 1999 [37, 38]. Contrairement à la BD, la SRD modélise directement le fluide de manière simplifiée. Il est représenté par des particules ponctuelles dont les positions et vélocités sont continues, et sont distribuées dans des cellules représentant une petite portion de l'espace. Une particule de fluide dans la SRD ne représente pas une molécule de fluide dans la réalité, mais un ensemble de molécules de fluide. C'est pourquoi, contrairement à la BD, ce modèle est dit à gros grains et permet de mieux représenter les effets de l'hydrodynamique. La dynamique du fluide se décompose en deux étapes : une étape d'écoulement libre et une étape de collision. C'est une dynamique très simple qui ne nécessite aucun calcul explicite de forces d'interaction entre particules de fluide. Pour simuler des suspensions colloïdales, il faut

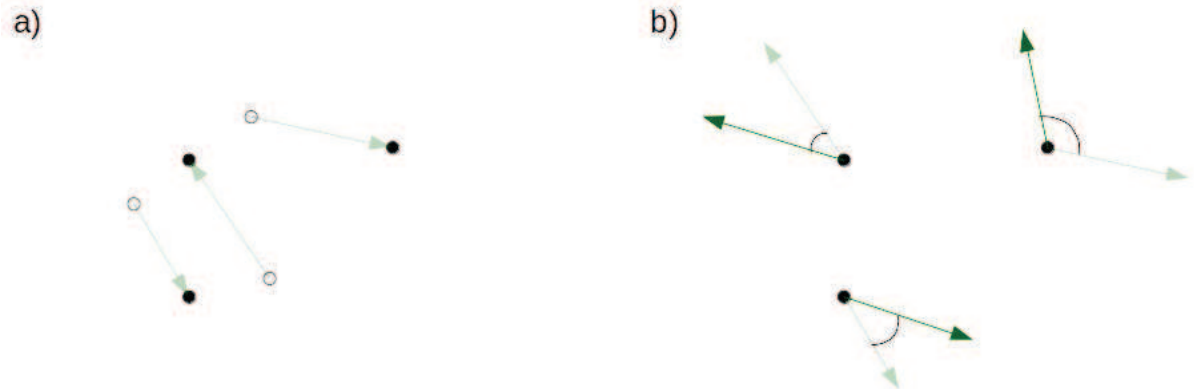


FIGURE III.1 – Étapes d'une itération de SRD qui se décompose d'une phase d'écoulement où les particules se déplacent (a) et d'une phase de collision où la vélocité est tournée (b)

intégrer des colloïdes dans le système dont la dynamique est modélisée par MD. Afin de faire interagir les particules de fluide avec les colloïdes, la dynamique des particules de fluide suivant le modèle SRD doit être couplée avec la dynamique des colloïdes suivant le modèle MD. Le nouveau modèle obtenu est appelé SRD-MD. Différentes variantes de SRD-MD existent dans la littérature suivant la méthode de couplage utilisée. La plus commune et la plus simple consiste uniquement à introduire les colloïdes dans l'étape de collision des particules de fluide. De cette manière, dans la partie MD du modèle, seules les interactions colloïde-colloïde sont explicitement calculées. L'interaction entre les particules de fluide et les colloïdes a lieu uniquement pendant les étapes de collision de la SRD. Ce couplage a été largement utilisé et donne de bons résultats lorsque la fraction volumique des colloïdes est modérée [39]. Elle est également très adaptée à la

parallélisation et une version GPU de cette simulation de SRD-MD a été développée récemment par Westphal *et al.* [3]. Cependant, la simplicité de ce couplage implique également une représentation des effets de l'hydrodynamique très simplifiée également. En effet, dans cette méthode, les particules de fluide pénètrent à l'intérieur des colloïdes et donc, ce couplage ne permet pas de décrire correctement l'hydrodynamique. Dans ce chapitre, nous allons présenter un couplage plus complexe permettant de mieux décrire les effets hydrodynamiques. Il consiste à ajouter explicitement des forces d'interaction répulsives entre les colloïdes et les particules de fluide, ce qui a pour effet d'éviter que les particules de fluide ne se retrouvent dans les colloïdes [40]. Dans ce modèle appelé dans la suite du chapitre "SRD-MD avec force de couplage", les particules de fluide ne sont donc plus distribuées de façon homogène dans l'espace de simulation comme avec le couplage précédent. Le contrecoup de cette méthode est qu'elle nécessite de calculer les interactions fluides-colloïdes à chaque étape de MD, ce qui rend la simulation très coûteuse en termes de calculs. Pour les simulations de SRD-MD de ce type, l'utilisation d'architectures parallèles, telles que le GPU, est donc primordiale pour obtenir des performances correctes. Ce chapitre propose une implémentation GPU avec une approche adaptée pour le système de SRD-MD avec force de couplage, basée sur une association des colloïdes par bloc de threads. Cette stratégie permet une meilleure répartition de la charge de calcul sur le GPU que les méthodes d'association standard employées dans les simulations classiques de MD, et donc d'obtenir de meilleurs performances. Ce chapitre est organisé comme suit. Premièrement, dans la partie III.1, le modèle SRD-MD est présenté plus précisément. L'implémentation GPU est détaillée dans la partie III.2, en se focalisant sur les différentes méthodes d'association possibles. Enfin, la partie III.3 montre les résultats obtenus par notre approche.

III.1 Le modèle hybride SRD-MD avec force de couplage

III.1.1 La dynamique du fluide : le modèle de SRD

Dans le modèle de SRD-MD, la SRD décrit la dynamique des particules de fluide seule. Le fluide est représenté par N_f particules ponctuelles de masse m_f, réparties dans des petites cellules, appelées cellules de SRD. Ces cellules cubiques contiennent en moyenne γ particules de fluide et ont une taille a_0, qui influe sur la précision du modèle. Dans la SRD seule, la dynamique des particules de fluide est décomposée en deux parties : une phase d'écoulement libre suivie d'une phase de collision [41]. Pendant l'étape d'écoulement libre, les particules de fluide se déplacent uniquement en fonction de leur vélocité v_i, selon

l'équation suivante :

$$r_i(t + \Delta t_{SRD}) = r_i(t) + \Delta t_{SRD} v_i(t) \tag{III.1}$$

où Δt_{SRD} est le pas de temps qui s'écoule entre deux étapes de collision. L'étape de collision consiste quant à elle, à appliquer une rotation sur les vélocités de chaque particule de fluide, relatives à la vélocité du centre de masse v_{cm} des particules de fluide que contient la cellule de SRD dans laquelle elle se trouve. Les rotations s'effectuent à partir d'un angle α fixe et d'un axe de rotation aléatoire, mais toutes les particules de fluide qui se trouvent dans la même cellule de SRD partagent le même axe de rotation. La nouvelle vélocité des particules de fluide est calculée à partir de l'équation (III.2) :

$$v_i(t + \Delta_{SRD}) = v_{cm}(t) + R(\alpha)[v_i(t) - v_{cm}(t)], \tag{III.2}$$

avec R une matrice de rotation aléatoire, α l'angle de rotation constant et v_{cm} la vélocité du centre de masse des particules de fluide appartenant à la cellule de SRD contenant la particule i. De plus, avant chaque étape de collision, une translation aléatoire est appliquée temporairement sur l'ensemble des particules de fluide en fonction d'un seul vecteur pour toutes les particules, dans l'intervalle $[-a_0/2,\ a_0/2]$. Ceci a pour but de ne pas faire interagir toujours les mêmes particules de fluide lors du calcul de la rotation des vélocités, ce qui permet de restaurer l'invariance Galliléenne [42] qui n'est pas toujours garantie sans l'étape de translation. Après l'application de la rotation, les particules de fluide retrouvent leur position d'origine, c'est-à-dire celle avant la translation. Ainsi, l'étape de collision modifie la vélocité des particules de fluide et non leur position.

III.1.2 Le modèle hybride SRD-MD avec une 'force de couplage'

Dans le cadre d'un couplage entre la SRD et la MD avec force de couplage, la MD permet de décrire les interactions colloïde-colloïde et colloïde-fluide, ce qui représente la dynamique des colloïdes et également leurs interactions avec les particules de fluide. Dans notre modèle hybride de SRD-MD, l'étape d'écoulement libre du modèle SRD est remplacée par des étapes de MD : au lieu de se déplacer de manière rectiligne selon leur vélocité, les particules de fluide se déplacent également en fonction des interactions colloïde-fluide qu'elles subissent. Durant l'étape de MD, les positions et vélocités de chaque particule de fluide et de chaque colloïde sont calculées selon l'équation de Newton suivante :

$$v_i = \frac{dr_i(t)}{dt},\ \textbf{et}\ m_i \frac{dv_i(t)}{dt} = \sum_j F_{ij}, \tag{III.3}$$

où F_{ij} représente la force appliquée par une particule j sur une particule i. Pour résumer, durant l'étape de MD, pour les particules de fluide, seules les forces d'interaction fluide-

colloïde sont prises en compte et non les interactions fluide-fluide car ce type d'interaction est déjà représenté de manière simplifiée par l'étape de collision de la SRD. Pour ce qui est des colloïdes, les forces qui leur sont appliquées viennent à la fois de leur interactions avec les particules de fluide et avec les autres colloïdes. Les étapes de MD s'appliquent à chaque pas de temps Δt_{MD}, tandis que les étapes de collision s'effectuent à chaque pas de temps Δt_{SRD}. Δt_{MD} est déterminé selon les potentiels d'interaction et les caractéristiques de colloïdes, tandis que Δt_{SRD} détermine les propriétés du fluide simulé. En prenant un pas de temps $\Delta t_{MD} = n\Delta t_{SRD}, n \in \mathbb{N}$, une itération de notre simulation est donc composée de n étapes MD successives suivies d'une étape de collision (voir figure III.2).

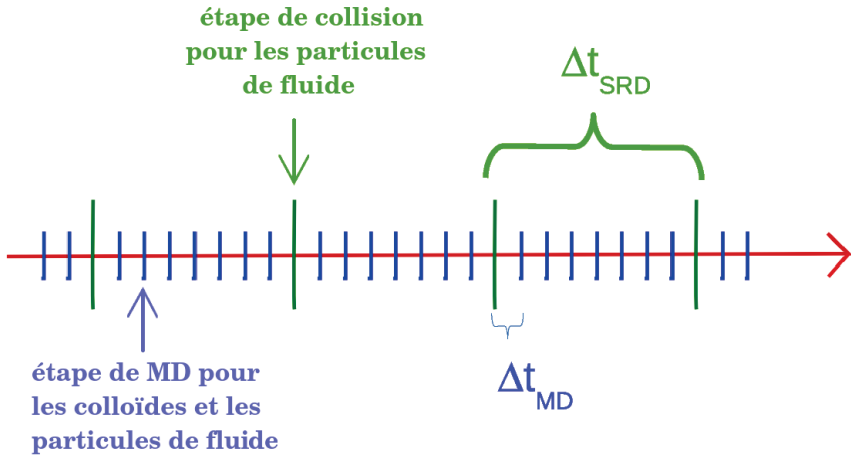


FIGURE III.2 – Schéma global d'une itération de modèle hybride SRD-MD avec "force de couplage"

III.1.3 L'étude expérimentale de référence

Le système présenté dans la thèse de A. Tomilov [1] a été utilisé comme référence pour notre étude. Il est composé de colloïdes de rayon $a_c = 300\,\text{nm}$ interagissant entre eux selon le potentiel attractif de Lennard-Jones généralisé :

$$V_{\text{cc}}(r_{ij}) = \begin{cases} 4\varepsilon_{cc}\left[\left(\frac{\sigma_{cc}}{r_{ij}}\right)^{36} - \left(\frac{\sigma_{cc}}{r_{ij}}\right)^{18}\right] & (r_{ij} \leq R_C \equiv 1.6\sigma_{cc}), \\ 0 & (r_{ij} > R_C), \end{cases} \tag{III.4}$$

avec $\sigma_{cc} = 2a_c$ et $\epsilon_{cc} = 14k_BT$. Les interactions colloïde-fluide sont quant à elles décrites par le potentiel répulsif de courte portée :

$$V_{\text{cf}}(r_{ij}) = \begin{cases} \varepsilon_{cf}\left(\frac{\sigma_{cf}}{r_{ij}}\right)^{12} & (r_{ij} \leq R_C \equiv 2.5\sigma_{cf}), \\ 0 & (r_{ij} > R_C), \end{cases} \tag{III.5}$$

où $\sigma_{cf} = 0.8a_c$ et $\varepsilon_{cf} = 2.5k_{\mathrm{B}}T$. Les forces utilisées dans cette étude sont exclusivement dérivées de potentiels d'interaction de courte portée, ce qui permet de couper le potentiel en le considérant comme étant nul sur des longues distances ($r_{ij} > R_C$). Les paramètres de la SRD déterminent les propriétés du fluide modélisé. La taille des cellules de SRD est fixée à $a_0 = a_c/2$, le libre parcours moyen sans dimension des particules de fluide est égal à $\lambda = 0.1$, le nombre moyen de particules de fluide par cellule de SRD vaut $\gamma = 5$ et l'angle de rotation pour l'étape de collision est $\alpha = 90°$. Les masses des particules de fluide et des colloïdes sont respectivement fixées à $m_f = 6.75 \times 10^{-19}\,\mathrm{kg}$ et $m_C = 2.49 \times 10^{-16}\,\mathrm{kg}$. Le pas de temps de SRD vaut $\Delta t_{SRD} = 7.37 \times 10^{-5}\,\mathrm{s}$ et 8 étapes de MD sont effectuées par itération. Ce système a été utilisé afin d'étudier la cinétique d'agrégation et la structure des agrégats formés par les colloïdes dans des suspensions colloïdales. Pour cela, la fraction volumique des colloïdes Φ_C varie dans le but de déterminer l'influence de ce paramètre sur l'agrégation des colloïdes. Cela implique qu'à nombre de colloïdes constant, la taille de la boîte de simulation varie également ainsi que le nombre total de particules de fluide. Le Tableau III.1 montre le ratio entre le nombre de particules de fluide et le nombre de colloïdes selon les différentes fractions volumiques étudiées. Dans toutes ces simulations, le nombre de particules de fluide est largement supérieur au nombre des colloïdes.

Φ_C	n_f/n_C
0.05	3241
0.1	1578
0.15	1011
0.2	737

Tableau III.1 – Ratio entre le nombre de particules de fluide et le nombre de colloïdes n_f/n_C pour différentes fractions volumiques.

Pour une fraction volumique en colloïde de $\Phi_c = 0.1$, un système de 4000 colloïdes compte plus de 6M de particules de fluide. Le modèle de SRD-MD est ainsi très coûteux en mémoire pour simuler des systèmes avec de nombreux colloïdes.

III.2 L'implémentation GPU

III.2.1 L'implémentation GPU de l'étape de collision

L'Algorithme 4 montre les différentes étapes requises pour réaliser l'étape de collision de la SRD. Les deux principales sont le calcul des v_{cm} et la rotation des vélocités des particules de fluide selon un axe de rotation aléatoire associé à leur cellule de SRD.

Algorithm 4 étape de collision de SRD

1: *//étape de translation globale*
2: calcule un vecteur de translation aléatoire v_{Trans}
3: **for all** particule p_i **do** en parallèle
4: calcule la position de la particule virtuelle p_{Trans_i}, issue de la translation de la particule i selon le vecteur v_{Trans}
5: *//étape du calcul de la vélocité du centre de masse*
6: **for all** cellule $cell_i$ **do** in parallel
7: calcul de v_{cm_i} à partir des particules virtuelles p_{Trans_i} contenues par $cell_i$
8: *//étape de rotation des vélocités*
9: **for all** cellule $cell_i$ **do** en parallèle
10: calcul d'un axe de rotation aléatoire $axeRotation_i$ associé à $cell_i$
11: **for all** particule p_i **do** en parallèle
12: applique la rotation sur la vélocité $veloc_i$ de p_i, à partir de $rotationAxis_i$ et la valeur v_{cm_i} associée à la cellule contenant p_i

III.2.1.1 Le calcul des vélocités des centres de masse

Une translation globale de toutes les particules de fluide est appliquée avant l'étape de calcul des vélocités des centres de masse selon un vecteur de translation aléatoire global. Ce vecteur est généré par le CPU et diffusé à toutes les particules par le GPU. Le GPU peut alors calculer les positions des particules de fluide issues de la translation selon le vecteur global. Les nouvelles positions obtenues sont stockées dans une variable temporaire, et les conditions périodiques leur sont appliquées afin que les positions ne se retrouvent pas à l'extérieur de la boîte. Enfin, la v_{cm} de chaque cellule de SRD peut être calculée en parcourant les particules de fluide précédemment déplacées qu'elle contient. La translation des particules ainsi que l'application des conditions périodiques sont les parties les plus adaptées à une implémentation GPU, car l'accès aux positions est coalescente et les calculs sont indépendants. En revanche, le calcul des v_{cm} pour chaque cellule de SRD nécessite d'établir une stratégie particulière afin d'être efficace pour le GPU. Cette étape se décompose en deux sous étapes : d'abord pour chaque cellule de SRD, il faut faire la somme des vélocités des particules de fluide qu'elle contient, puis diviser cette somme par leur nombre. Le problème vient de la sous-étape de la somme des vélocités qui nécessite de soit rechercher les particules de fluide contenues dans chaque cellule de SRD, soit faire la somme des vélocités de manière non indépendante. Récemment, quelques travaux ont proposé des solutions pour résoudre ce problème. Chen *et al.* utilise une stratégie basée particule [2], lançant un thread par particule. La somme des vélocités se fait alors en calculant pour chaque thread, la cellule de SRD associée à leur particule de fluide à partir de ses coordonnées spatiales, puis en ajoutant la vélocité de leur particule à la somme des vélocités de cette cellule de SRD, et en incrémentant également son compteur indiquant le

nombre de particules de fluide qu'elle contient. Ces deux dernières opérations nécessitent d'écrire dans des zones mémoires qui peuvent être utilisées par les autres threads associés à des particules de fluide appartenant à la même cellule de SRD, ce qui amène des problèmes de concurrence d'écriture mémoire car des threads n'est pas prévisible. Pour palier à ce problème, l'utilisation d'opérations atomiques est donc nécessaire pour l'incrémentation du compteur et l'ajout de la vélocité à la somme. Les opérations atomiques entraînent un surcoût, mais permettent à cette méthode de pouvoir effectuer la translation des particules de fluide et la somme des vélocités dans un unique noyau GPU. Ceci permet d'éviter des opérations sur la mémoire globale contenant les positions des particules en stockant les positions issues de la translation dans la mémoire privée du thread. Cette stratégie est précisément décrite par l'Algorithme 5.

Westphal *et al.* utilise une stratégie basée sur les cellules de SRD [3]. Afin de parcourir les

Algorithm 5 calcul des vélocités des centres de masse à l'aide d'opérations atomiques

1: **Input** : v_{Trans}
2: *//étape de la somme des vélocités*
3: **for all** particule p_i **do** en parallèle
4: *//translation des particules*
5: calcul de la particule virtuelle p_{Trans_i}, issue de la translation de la particule p_i avec le vecteur v_{Trans}
6: *//ajoute la vélocité et incrémente le compteur avec des opérations atomiques*
7: détermine $cell_i$, la cellule de SRD contenant p_{Shift_i}
8: incrémente avec une opération atomique $counter_i$ associé à $cell_i$
9: ajoute avec des opérations atomiques la vélocité de p_{Shift} à la somme $sumVeloc_{cell_i}$ associée à $cell_i$
10: *//application de la division*
11: **for all** cellule de SRD $cell_i$ **do** en parallèle
12: calcule v_{cm_i} de $cell_i$, à partir de $sumVeloc_{cell_i}$ et $counter_i$

particules contenues par chaque cellule de SRD, une recherche de voisinage est effectuée de la même manière que pour le calcul des interactions entre particules, c'est-à-dire en triant les positions de particules de fluide selon les cellules auxquelles elles appartiennent. Le tri des particules est coûteux et doit être recalculé à chaque étape de SRD à cause du processus de translation. De plus, il n'est pas possible d'utiliser la mémoire privée pour stocker les positions issues de la translation comme avec la méthode de Chen et *al.*. Cependant, cela est compensé par un calcul de la somme des vélocités plus efficace, car il ne nécessite plus d'opération atomique. Dans le cas d'une configuration homogène des particules de fluide dans la boîte de simulation, la répartition de la charge de travail entre les threads est équilibrée. Cela permet d'éviter d'avoir des threads inactifs dus à des cellules de SRD vides ou à des variations importantes du nombre de particules de fluide à traiter entre différents threads d'un même warp. Cependant, avec notre couplage

de SRD-MD, la distribution du fluide dans l'espace est très hétérogène, à cause de la présence des colloïdes et des forces de répulsion empêchant la présence du fluide dans les colloïdes. Ainsi, de nombreuses cellules de SRD ne contiennent aucune particule de fluides, car elles sont incluses dans un colloïde (dont la taille est bien plus grande). La variation par rapport à la moyenne de γ particules de fluide par cellule de SRD est donc très importante. Cela signifie que les stratégies spatiales telles que celles basées cellules ne permettent pas une répartition de la charge de calculs équilibrée, et que de nombreux threads sont inactifs car associés à des cellules ne contenant pas de particules de fluide. Pour ces raisons, l'approche utilisée par la suite est la stratégie basée particule de Chen *et al.* [2].

III.2.1.2 La rotation de la vélocité

Durant l'étape de rotation des vélocités, la vélocité des particules de fluide est mise à jour selon l'équation (III.2). Un premier noyau GPU est chargé de calculer pour chaque cellule, un axe de rotation aléatoire à partir de deux nombres aléatoires uniformes à générer. Ensuite, dans un second noyau GPU, les nouvelles vélocités des particules sont obtenues à partir des matrices de rotation générées avec l'angle de rotation constant et les axes de rotation précédemment calculés. Deux types d'association de threads peuvent être considérés pour le deuxième noyau [3]. Avec une stratégie basée particule, les threads doivent déterminer la cellule de SRD contenant leur particule, et l'accès à l'axe de rotation de cette cellule n'est pas coalescent. Cependant, l'accès mémoire en lecture et en écriture de la vélocité des particules est quant à lui coalescent. Avec une stratégie basée cellule, le problème est inversé, l'accès mémoire en lecture des axes de rotation est coalescent, mais pas pour les vélocités. L'accès aux vélocités étant plus utilisé dans ce noyau GPU, la solution basée particule est plus efficace. De plus, dans le cas de notre couplage de SRD-MD, pour les mêmes raisons que celles énoncées précédemment dans le paragraphe III.2.1.1, les stratégies spatiales telles que celles basées cellule sont inefficaces, à cause de la distribution non uniforme des particules de fluide dans la boîte de simulation.

III.2.2 Implémentation GPU de la partie MD

Tout comme pour la BD vue dans le chapitre II.3, la MD est réalisée en utilisant l'algorithme de Verlet vitesse. L'Algorithme 6 en décrit les étapes principales. Dans un premier temps, les nouvelles positions des particules de fluide et des colloïdes sont calculées à partir de leur accélération et de leur vélocité, en appliquant les conditions périodiques ensuite pour qu'aucune particule ne sorte de la boîte de simulation. Dans un second temps, l'accélération des colloïdes est calculée à partir des forces d'interaction colloïde-colloïde et colloïde-fluide qui lui sont appliquées, tandis que l'accélération des

particules de fluide provient uniquement des interactions fluide-colloïde. Enfin, la vélocité des particules de fluide et des colloïdes se calcule à partir de leur accélération courante et de leur accélération durant l'itération précédente. La plupart des étapes de cet algorithme sont adaptées de manière optimale pour une parallélisation GPU. Les noyaux *updatePos*, *updateVeloc* n'effectuent que des accès mémoire totalement coalescents. Les données sont lues et réécrites dans l'ordre, et tous les calculs se font de manière indépendante des autres threads, ce qui entraîne une utilisation de la bande passante efficace. Le goulot d'étranglement de notre couplage MD est le noyau *updateAcc*, car le calcul de l'accélération nécessite au préalable de calculer les interactions entre les particules. En réalité, cette étape est divisée en deux noyaux distincts : le noyau *updateAccCC* qui a pour but de calculer les interactions colloïde-colloïde et le noyau *updateAccCF* qui calcule les interactions colloïde-fluide et fluide-colloïde. Dans notre système, comme nous l'avons déjà mentionné, les forces d'interaction s'annulent pour des distances $r \geqslant R_C$ (voir les équations (III.4) et (III.5)). Cela amène à la résolution du problème de recherche de voisinage, tout comme c'était le cas pour la BD. Les différents moyens de résoudre ce problème ont déjà été discutés dans le chapitre précédent dans la partie II.1. Comme dans ce dernier, la solution retenue ici est notre approche hybride combinant grille régulière avec des listes de Verlet, et une réactualisation déterminée par un test dynamique sur les distances parcourues par les particules. Dans la suite sont décrits les différents schémas de décomposition proposés dans

Algorithm 6 Étape MD (algorithme de Verlet vitesse)

1: **for all** particule i **do** en parallèle
2: mettre à jour la position r_i à partir de v_i et $accel_i$
3: mettre à jour le déplacement $depl_i$ depuis la dernière mise à jour des listes de Verlet
4: stocker acc_i dans $acc_{precedent_i}$
5: **if** un des $depl_i > R_{shell}/2$ **then**
6: **for all** particule i **do** en parallèle
7: associer la particule i à la grille régulière
8: mettre à jour sa liste de Verlet avec la grille
9: remettre à zéro $depl_i$
10: **for all** particule i **do** en parallèle
11: calculer acc_i à partir de sa liste de Verlet
12: mettre à jour v_i à partir de acc_i et $acc_{precedent_i}$

la littérature pour les noyaux *updateAccCF*, c'est-à-dire la répartition entre les threads et leur ensemble des forces d'interaction à calculer.

III.2.2.1 Schémas de décomposition

Avec la SRD-MD avec force de couplage, les interactions colloïde-colloïde et colloïde-fluide doivent être calculées de manière indépendante. Dans la littérature, différentes techniques de parallélisation ont été développées pour calculer les interactions de paires. Chaque approche est plus ou moins coûteuse en termes de communication entre threads, plus ou moins équilibrée en termes de répartition de la charge de calcul. Leur efficacité dépend du type de système simulé. Du point de vue du schéma de décomposition thread-donnée adopté, elles peuvent être classifiées en quatre groupes [43], qui sont décrits dans les paragraphes suivants. Nous allons discuter de l'efficacité de ces schémas de décomposition dans le cas de systèmes SRD-MD avec force de couplage, en se focalisant sur le calcul des interactions colloïde-fluide qui est l'étape la plus coûteuse à cause du nombre très important de particules de fluide. Dans la dernière partie de la discussion, nous proposons un nouveau schéma de décomposition mieux adapté aux interactions colloïde-fluide spécifiques à la SRD-MD avec force de couplage.

Schéma de décomposition par particule Dans ce schéma de décomposition, chaque thread est associé à un colloïde et doit calculer les forces qui s'exercent sur lui, issues de ses interactions avec les particules de fluide voisines. La majorité des implémentations GPU utilisent ce schéma de décomposition [43–45], en associant à chaque thread le colloïde correspondant à son *global id*. Ainsi, pour un système composé de N colloïdes, le GPU lance N threads en parallèle et chaque thread calcule et additionne indépendamment des autres threads, les forces d'interaction F_{ij} appliquées à son colloïde i. Cette stratégie a l'avantage d'être très facilement parallélisable avec une répartition de la charge de calcul équilibrée entre les threads si le nombre de voisins de chaque colloïde est à peu près le même. Cependant, dans notre système, le nombre de colloïdes est très petit par rapport au nombre de particules de fluide (voir Table III.1), ce qui implique que la variation du nombre de voisins entre les colloïdes peut être très importante. Outre un déséquilibre important dans la répartition des charges, cette forte variation nuit également aux performances en augmentant le temps d'inactivité des threads lié aux branches divergentes entre threads d'un même warp. Le problème est illustré dans la figure III.3. Lorsqu'un thread n'a plus aucun voisin à parcourir, il doit alors être associé à un nouveau colloïde pour poursuivre les calculs d'interaction. Le problème de divergence de branches fait que chaque thread doit attendre que tous les threads de leur warp quittent la branche de parcours des voisins. Le temps de calcul des interactions des particules dépend donc du colloïde étant en interaction avec le plus grand nombre de particules de fluide. De plus, la décomposition par particule doit se baser sur les colloïdes et non sur les particules de fluide pour être efficace, car une grande partie des particules de fluide n'interagissent

pas avec les colloïdes. Ainsi, pour des systèmes avec un nombre de colloïdes faible, cette approche ne permet pas de saturer la capacité de calcul offert par le GPU, malgré le nombre important de particules de fluide.

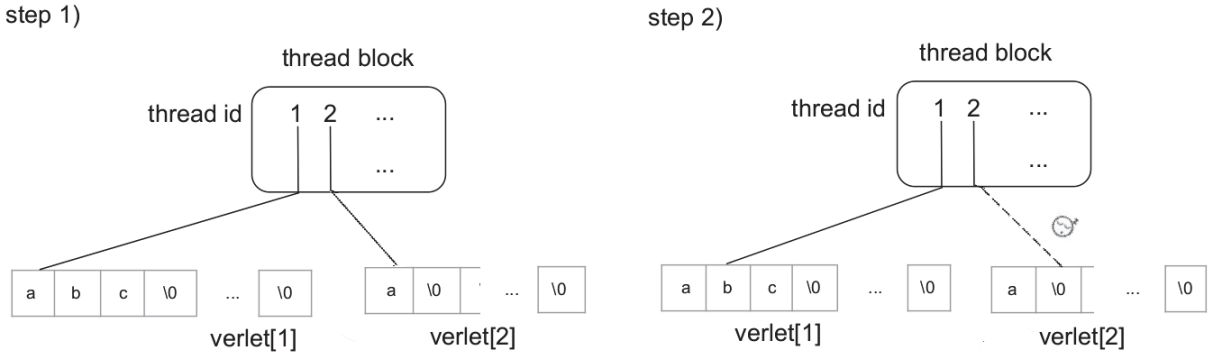


FIGURE III.3 – Illustration du schéma de décomposition par particule. Dans cet exemple, les threads 1 et 2 sont respectivement associés aux listes de Verlet 1 et 2. Dans la première étape, le thread 1 calcule l'interaction 1-a (entre la particule 1 et son voisin a) et le thread 2 calcule l'interaction 2-a. Dans la seconde étape, le thread 1 calcule l'interaction 1-b, mais le thread 2 n'a aucune interaction à calculer. Ces deux threads appartenant au même warp, le thread 2 doit attendre que le thread 1 finisse le calcul de toutes ses interactions avant de pouvoir poursuivre ses calculs avec une nouvelle particule. Ainsi, cette décomposition entraîne des temps d'inactivité des threads importants lorsque la taille des listes de Verlet varie de façon importante.

Schéma de décomposition par force Le schéma de décomposition par force associe chaque thread à une force d'interaction à calculer, c'est-à-dire à une paire de particules voisines. Cette association assure une répartition des charges parfaitement équilibrée : chaque thread doit calculer une seule interaction. De plus, contrairement à la décomposition par particule, cette méthode permet d'utiliser la troisième loi de Newton ($\vec{F_{ij}} = -\vec{F_{ji}}$) ce qui permet d'éviter de calculer deux fois l'interaction entre i et j [**?**]. Le principal désavantage est qu'associer une interaction à chaque thread, dans le cas de forces à courte portée, nécessite au préalable de réorganiser les données afin d'éliminer toutes les paires de particules n'étant pas en interaction. Cette réorganisation consiste à compresser la liste de Verlet en supprimant les cases vides qui sont dues à la stratégie fixant un nombre maximum de voisins. C'est une tâche complexe et coûteuse à paralléliser, mais elle n'est effectuée que lors des mises à jour des listes de Verlet. Le surcoût en temps de la réorganisation des données doit donc être compensé par le gain sur le calcul des forces afin de rendre cette décomposition avantageuse. Une manière de compresser la liste de Verlet est d'utiliser des opérations atomiques, mais cette méthode coûteuse et la liste

de Verlet compacte obtenue est non ordonnée, ce qui entraîne des calculs d'interactions moins efficaces à cause d'accès mémoire non coalescents. Zhmurov *et al.* [46] propose d'effectuer un tri sur la liste de Verlet en marquant les cellules avec une valeur, et les cellules contenant un index de voisin d'une autre valeur, afin de séparer les deux types de cases. Cependant, cette méthode requiert d'utiliser des buffers supplémentaires afin de marquer et trier les cellules. Un autre problème est d'effectuer la somme des forces appliquées à une particule, car elle ne peut se faire de manière indépendante : elle nécessite d'additionner les forces d'interactions calculées par plusieurs threads différents, menant à une situation de compétition. Un moyen de régler ce problème est d'utiliser des opérations atomiques, mais leur coût contrebalance le gain obtenu par une meilleure répartition de la charge de calcul. Un autre moyen est de stocker les forces partielles appliquées à chaque particule dans la mémoire globale, pour appliquer une réduction afin d'obtenir la somme des forces complète de chaque particule [46]. Cependant, là encore, cette méthode implique un coût mémoire supplémentaire important.

Schéma de décomposition spatiale

Le schéma de décomposition spatiale associe chaque thread à un sous-espace de la boîte de simulation. Chaque thread doit calculer les interactions des particules se trouvant dans ce sous espace. Afin de ne pas à avoir à faire de réorganisation de données, dans le cas d'une simulation de SRD-MD, les sous espaces peuvent être les cellules de la grille régulière permettant d'accélérer la recherche de voisinage. Cette stratégie permet d'avoir une répartition équilibrée de la charge de calcul entre les threads si la répartition des particules est homogène [47]. Cependant, dans notre système, pour les fractions volumiques colloïdales étudiées (5% à 20%), une large partie des cellules ne contiennent aucun colloïde, ce qui signifie qu'une partie des threads sont inactifs. De plus la répartition des particules est très hétérogène, c'est pourquoi, cette stratégie n'est pas adaptée pour nos simulations de SRD-MD avec force de couplage.

Schéma de décomposition par bloc Afin de correspondre à notre cas spécifique de SRD-MD avec force de couplage, nous avons proposé un nouveau type de schéma de décomposition. Cette décomposition est un mélange entre celle par force et celle par particule : chaque colloïde est associé à un bloc de threads, ce qui permet à chaque thread de ce bloc d'être associé à un sous-ensemble des interactions appliquées à ce colloïde. Comparée à la décomposition par particule, notre *décomposition par bloc* permet d'exploiter totalement les capacités de calcul du GPU, même dans le cas de petits systèmes, et la répartition de la charge de calcul entre les threads est plus équilibrée pour les systèmes qui, comme le nôtre, ont un nombre très petit de colloïdes comparé au nombre moyen de voisins. De plus, cette méthode permet d'éviter le problème des

branches divergentes lié à la variation du nombre de voisins. En effet, les threads d'un même warp étant tous liés à la même particule, lorsqu'un thread n'a plus d'interaction à calculer, tous les autres threads du warp n'ont plus d'interaction à calculer également au tour suivant, comme le montre la figure III.4.

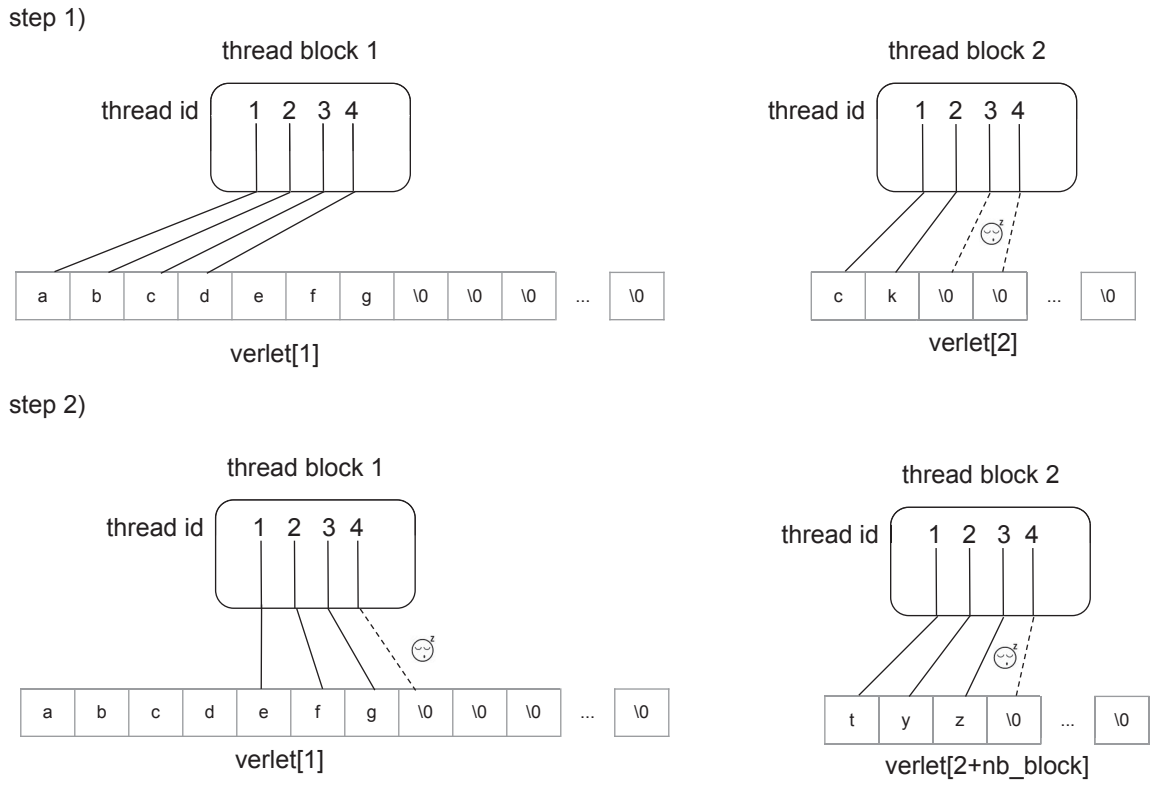


FIGURE III.4 – Illustration du schéma de décomposition par bloc. Dans cet exemple, les blocs 1 et 2 sont respectivement associés aux listes de Verlet 1 et 2. Dans la première étape, les threads du bloc 1 calculent les premières interactions de la particule 1, sans qu'il n'y ait de thread inactif. D'un autre côté, certains threads du bloc 2 sont inactifs, à cause du faible nombre de voisins de la particule 2. Dans la seconde étape, le bloc 2 est associé à une autre particule : les threads n'ont pas eu à attendre que d'autres threads aient fini leur tâche comme ceux du bloc 1, ce qui est l'avantage de cette décomposition. Cependant certains threads du bloc 2 sont à nouveau inactifs, à cause du faible nombre de voisins. C'est pourquoi, ce type de décomposition n'est efficace que pour les systèmes avec un nombre moyen de voisins très supérieur au nombre de threads par bloc.

La somme des forces ne peut être calculée indépendamment comme pour une décomposition par force, néanmoins, cette interdépendance n'est qu'entre les threads d'un même bloc. Ainsi, à l'aide de processus de synchronisation tels que des barrières et des réductions de blocs, l'addition des forces partielles calculées par chaque thread du bloc est possible, sans utilisation d'opérations atomiques ou de mémoires additionnelles

importantes. Pour ces raisons, ce schéma de décomposition par bloc est très adapté aux systèmes de SRD-MD avec force de couplage, ainsi que tous les systèmes ayant un nombre de voisins très important. Cette approche n'est par contre pas adaptée pour les systèmes ayant un nombre de voisins faible, notamment ceux ayant un nombre moyen de voisins inférieur au nombre de threads par bloc, car dans ce cas, une partie des threads de chaque bloc serait très souvent inactive (voir figure III.4).

III.3 Résultats

III.3.1 Les résultats des simulation : comparaison avec l'étude de Référence [1]

Notre simulation de SRD-MD avec force de couplage utilise une combinaison d'une grille régulière avec des listes de Verlet pour structure accélératrice, comme décrit dans la partie II.1 du chapitre précédent. Chaque type d'interaction nécessite cette structure accélératrice, car elles ont chacune un rayon de coupure différent. Cependant, nous avons choisi de n'utiliser qu'une seule grille régulière avec liste de Verlet pour le calcul des forces d'interaction colloïde-fluide et fluide-colloïde, en utilisant la troisième loi de Newton pour calculer les uns à partir des autres. Bien que cette approche nécessite d'utiliser des opérations atomiques, cela reste plus efficace que l'utilisation de deux structures accélératrices pour le calcul de ces interactions. En effet, en associant un bloc ou un thread par particule de fluide, dans le cadre d'une décomposition par bloc ou par particule, une très large partie des threads serait inactive, car une grande partie des particules de fluide n'interagit avec aucun colloïde. De plus, l'ajout d'une grille et de positions triées selon cette grille est aussi plus coûteuse en mémoire. Les simulations de SRD-MD ont été exécutées avec différentes fractions volumiques de colloïdes allant de $\Phi_c = 0.05$ à $\Phi_c = 0.2$ afin d'étudier la cinétique de l'agrégation et comparer les courbes obtenues avec les résultats de la Référence [1]. À l'état initial, les colloïdes et les particules de fluide sont distribués de manière aléatoire, en évitant les interpénétrations. Quelques itérations de SRD-MD sont ensuite exécutées en ne prenant en compte que les forces d'interaction répulsives colloïde-colloïde et colloïde-fluide, afin de stabiliser la température du système. Après cette étape, les itérations de SRD-MD sont calculées en prenant en compte les interactions décrites par les équations (III.4) et (III.5). La figure III.5 montre des captures d'écran de ces simulations aux états initial et final. À l'état final, il est clair que les colloïdes agrégés ne sont pas répartis de manière homogène dans la boîte de simulation. De plus, pour la fraction volumique la plus basse (5%), une large partie de l'espace n'est remplie que de particules de fluide ce qui implique qu'une grande partie des cellules des différentes grilles régulières liées aux colloïdes sont vides, contrairement au cas avec la fraction volumique

la plus haute (20%). La figure III.6 représente le nombre d'agrégats n_A composés d'au moins deux colloïdes en fonction du temps, obtenu dans nos simulations et dans celles de la version CPU de la Référence [1]. Il est ainsi montré que notre version GPU permet d'obtenir des comportements similaires à la version CPU de référence, ce qui valide donc notre implémentation GPU. De plus, la version GPU permet, grâce à sa plus grande puissance de calcul, de pouvoir simuler des systèmes plus grands, pour un temps de simulation total raisonnable, et ainsi, d'affiner les résultats obtenus avec de meilleures statistiques. Comme le montre la figure III.6, le processus d'agrégation se décompose en deux étapes. Dans un premier temps, le nombre d'agrégats augmente car les colloïdes isolés s'attirent pour former des dimères. Dans un second temps, le nombre d'agrégats diminue à cause de la coalescence des agrégats qui se poursuit jusqu'à ce qu'il ne reste plus qu'un seul cluster. Au cours de la simulation, à cause de la restructuration des particules, le temps de calcul des noyaux GPU varie d'une itération à l'autre. Dans la suite, le temps de calcul est moyenné sur un nombre d'itérations significatif afin de prendre en compte ce phénomène, pour déterminer l'efficacité des différentes méthodes.

III.3.2 La performance de notre schéma de décomposition par bloc

Notre schéma de décomposition par bloc a été testé avec différentes cartes graphiques NVIDIA. En plus des cartes GTX 690 et K20m déjà présentés dans le paragraphe II.1.3, des cartes plus récentes ont pu être utilisées, dont une Tesla K40m et une Titan-X possédant 12 Go de mémoire. Dans un premier temps, la taille des blocs doit être fixée à une valeur devant être multiple de la taille d'un warp (soit $warpSize = 32$ dans le cas de cartes graphique NVIDIA). Ce choix a une influence sur les performances de notre schéma de décomposition : plus la taille des blocs est grande, plus la répartition des charges de travail entre thread est équilibrée mais en revanche, plus la synchronisation entre les threads de différents blocs devient coûteuse. En effet, l'algorithme requiert de synchroniser les threads d'un même bloc avant de pouvoir faire la somme des forces partielles que chaque thread vient de calculer ou pour remplir la liste de Verlet lors de sa réactualisation. Le cas pour lequel la taille des blocs est celui d'un warp est particulier, car les threads d'un warp sont physiquement synchronisés, ce qui signifie qu'aucune méthode de synchronisation telle que des barrières n'est requise. La Tableau III.2 compare l'efficacité des noyaux *computeAcc* et *computeAccAndVerlet* selon les tailles de blocs. Les résultats montrent que la taille de bloc optimal pour notre système est $blocSize = 64$, soit un bloc composé de deux warps. Par la suite, toutes les simulations présentées ont cette taille de bloc en paramètre. Dans le but de comparer l'efficacité de notre algorithme pour des simulations de SRD-MD avec force de couplage, les différents schémas de décomposition présentés dans

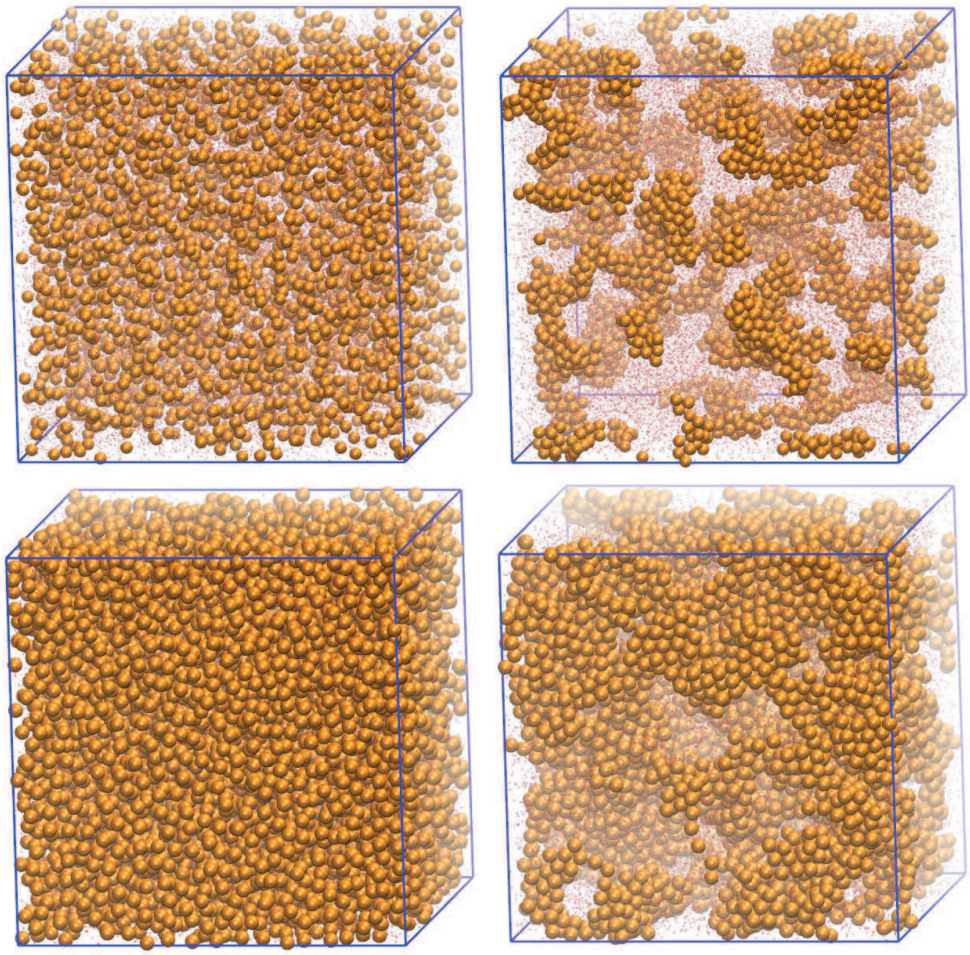


FIGURE III.5 – Captures d'écran de simulations réalisées avec une fraction volumique de 5% (en haut) et 20% (en bas) dans leurs états initial (à gauche) et final (à droite). Les colloïdes sont représentés en orange et les particules de fluide en rouge. Pour une question de visibilité, seulement une particule de fluide sur 300 est représentée [48].

taille de bloc	computeAcc	computeAccAndVerlet
32	3.82 ms	12.22 ms
64	2.92 ms	9.45 ms
128	3.14 ms	9.92 ms
256	3.27 ms	10.28 ms

Tableau III.2 – Comparaison du temps moyen de calcul des noyaux *computeAcc* et *computeAccAndVerlet* en utilisant un schéma de décomposition par bloc, avec différentes valeurs de taille de bloc. Les temps ont été moyennés sur les 1000 premières itérations, sur une simulation de 500 colloïdes et 788981 particules de fluide avec $\rho_c = 0.10$, s'exécutant sur une GPU K20m. On retrouve des écarts de performance similaires pour un nombre de colloïdes plus élevé et des fractions volumiques différentes

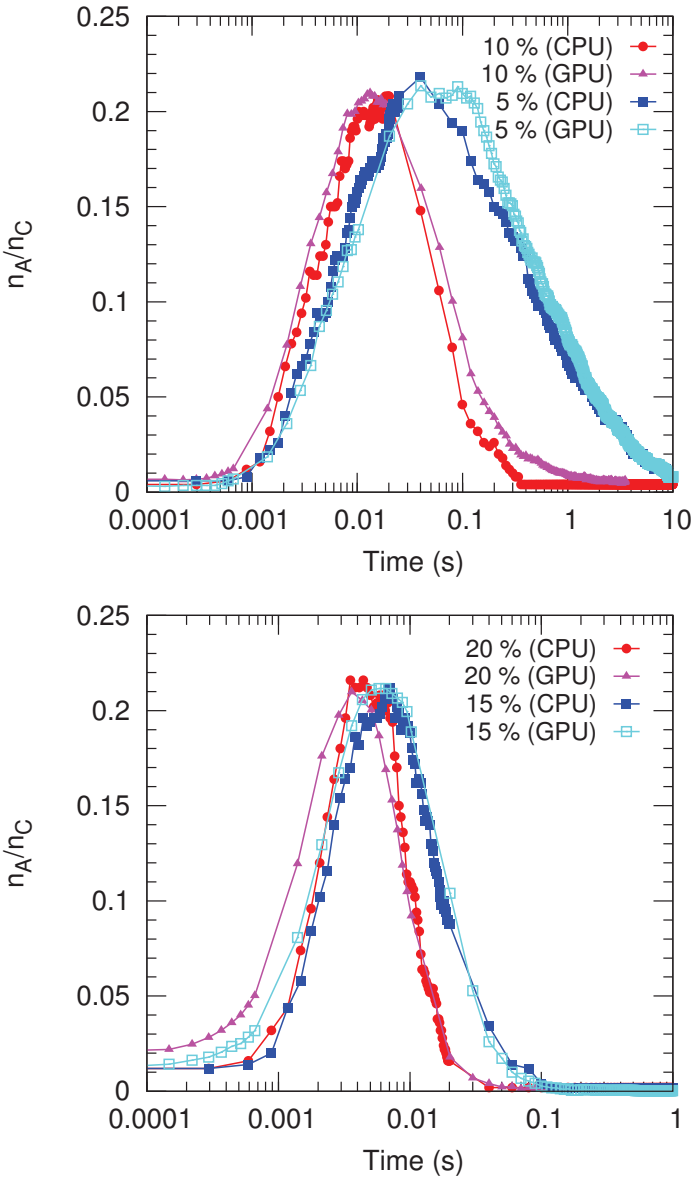


FIGURE III.6 – Évolution du nombre d'agrégats n_A divisé par le nombre de colloïdes n_C en fonction du temps (un agrégat est composé d'au moins deux colloïdes). Pour la version GPU, les résultats ont été moyennés sur 3 simulations réalisées avec 3000 colloïdes pour les fractions volumiques de 5% et 4000 colloïdes pour la fraction volumique de 10%, 15% et 20%. Ces résultats sont comparés avec la simulation CPU de référence réalisée avec 500 colloïdes pour les mêmes fractions volumiques [1].

Colloïdes	fluide	décomp cell	décomp particle	décomp bloc
500	788981	1040 ms	206 ms	69 ms
2000	3120942	2091 ms	402 ms	275 ms
4000	6311854	4201 ms	812 ms	559 ms

Tableau III.3 – Temps d'une itération d'une simulation de SRD-MD avec $\Phi_C = 0.1$, s'exécutant sur un GPU K20m (moyenné sur les 1000 premières itérations). Une itération est composée de 8 étapes de MD et d'une étape de SRD. Les résultats montrent une progression linéaire du temps de calcul en fonction de la taille du système, excepté pour le système le plus petit avec la décomposition par particule et par cellule, ce qui est dû à la non exploitation de toute la capacité de calcul du GPU avec cette décomposition sur les petits systèmes.

la partie III.2.2.1 ont été implémentés, excepté le schéma de décomposition par force car il est trop coûteux en mémoire et qui est moins efficace pour des potentiels d'interaction simples, comme l'a montré la littérature [49].
Les tests ont été exécutés pour différentes tailles de systèmes et différentes fractions volumiques de colloïdes. Par la suite, nous allons nous concentrer sur les résultats obtenus avec la fraction volumique $\Phi_C = 10\%$, une valeur où le ratio entre particules de fluide et colloïdes est très grand et où les effets hydrodynamiques sont assez importants pour justifier l'utilisation du modèle SRD-MD plutôt qu'un modèle plus simple. Les performances obtenues avec les différents schémas de décomposition ont été rapportées dans le Tableau III.3. Comme il était attendu, la décomposition par cellule est la moins efficace dans le cas d'une simulation SRD-MD, à cause de la fraction volumique colloïdale faible, ce qui implique qu'une partie non négligeable des cellules est sans colloïde. Cette méthode devient également moins efficace à mesure que la simulation avance, car le phénomène d'agrégation tend à rendre la distribution des colloïdes dans l'espace de moins en moins homogène comme le montre la figure III.5, déséquilibrant de plus en plus la répartition de la charge des calculs. À partir de ce constat, il est aussi attendu que cette approche soit plus efficace dans les cas de fractions volumiques plus importantes, néanmoins, même avec $\Phi_C = 0.25$, cette méthode reste la plus inefficace.

Le schéma de décomposition par particule, standard dans le cas de simulations de MD, obtient des performances moyennes, ce qui s'explique par le très grand nombre de particules de fluide voisines. Par exemple, dans le cas de la simulation avec 500 colloïdes et 788981 particules de fluide ($\Phi_C = 10\%$), chaque colloïde a en moyenne 18000 voisins, et ce nombre peut varier de l'ordre de 1000 à 30000. Ainsi, le temps d'inactivité des threads dans certains cas peut être très important. De plus, dans le cas du plus petit système à 500 colloïdes, le nombre de threads lancés dans ce cas est très inférieur au nombre de processus simultanés qu'un GPU peut exécuter. C'est pourquoi, le temps de calcul pour une itération est si élevé dans ce cas précis. Ainsi, ces résultats montrent

	Temps moyen(ms)	Nombre d'appel	Ratio
updateNeighbors	24.35	1/8	4.7%
computeAcc	3.87	8	48.29%
updatePosition	1.11	8	13.7%
checkIfUpdate	0.37	8	4.6%
updateVeloc	0.72	8	8.9%
computeAvgVeloc	4.17	1	6.5%
rotateVeloc	1.36	1	2.1%

Tableau III.4 – Temps moyen de calcul des principaux noyaux GPU d'une simulation de SRD-MD avec le ratio global que représente chaque noyau sur le temps total de la simulation, avec un schéma de décomposition par bloc (500 colloïdes, $\Phi_C = 0.1$). La troisième colonne représente le nombre moyen d'appels du noyau dans une itération. "1/8" vient du fait que les listes de Verlet fluide-colloïde sont réactualisées toutes les 8 itérations en moyenne et "8" correspond aux huit étapes de MD dans une itération SRD.

que notre schéma de décomposition par bloc est le plus adapté pour une simulation de SRD-MD avec force de couplage. Il permet d'exploiter totalement le GPU même pour les petits systèmes avec des temps linéaires dès le système à 500 colloïdes contrairement aux deux autres décompositions. Malgré les synchronisations supplémentaires qu'il requiert, l'absence de temps important d'inactivité des threads, permet d'avoir des performances 45% plus rapides que celles obtenues avec la décomposition par particule, lorsque tous deux exploitent totalement les capacités de calculs du GPU (voir Tableau III.3). Le Tableau III.4 décrit le temps moyen de calculs pour les principales étapes qui composent une itération (8 étapes de MD + 1 étape de SRD) dans le cas d'une décomposition par blocs. Il montre que le goulot d'étranglement de l'algorithme est le noyau *computeAcc*, qui représente approximativement la moitié du temps total de la simulation.

Les simulations ont été exécutées avec différentes tailles de systèmes pour différentes fractions volumiques colloïdales, en utilisant notre schéma de décomposition par bloc. Les performances en temps pour les systèmes où $\Phi_C = 0.1$, ont été reportées dans le Tableau III.5. Les résultats montrent que la complexité de notre méthode est linéaire en fonction de la taille du système, et les temps totaux des simulations obtenus sont faibles pour des simulations de MD. La limite des simulations est donc la consommation mémoire du GPU, empêchant de pouvoir lancer de plus longues simulations avec des systèmes plus importants, ce qui est discuté dans le prochain paragraphe. En effet, avec le GPU K20m le système le plus grand qui peut être exécuté pour $\Phi_C = 0.1$ est aux alentours de 7500 colloïdes. Cela est dû au nombre très important de particules de fluide, qui pour ce cas atteint 11834715 particules de fluide dont il faut stocker les attributs dans le GPU. Néanmoins, la parallélisation GPU des simulations de SRD-MD permet de diminuer de

Colloïdes	Fluide	Temps/itération (ms)				Total time (h)			
		GTX 690	K20m	K40m	GTX TitanX	GTX 690	K20m	K40m	GTX TitanX
500	788 981	81	64	61.4	53	1.2	1.0	0.9	0.8
2000	3 120 942	315	261.9	240	134	4.8	4	3.7	2.0
3500	5 487 872	571	526.8	431	244	8.7	6	6.6	3.7
4000	6 311 854	XXX	545	480	274	XXX	8.3	7.3	4.2
7500	11 834 715	XXX	1052	936	532	XXX	16	14.3	8.1
8000	12 623 708	XXX	XXX	975	547	XXX	XXX	14.9	8.4
14000	22 091 468	XXX	XXX	XXX	1 142	XXX	XXX	XXX	17.6

Tableau III.5 – Temps moyen d'une itération et temps total d'une simulation de SRD-MD pour $\Phi_C = 0.1$ durant 55000 itérations sur un GPU GTX690, K20m, K40 et GTX Titan X. Une itération est composée de 8 étapes de MD et 1 étape de SRD. "XXX" signifie que la consommation mémoire de la simulation est trop importante pour pouvoir être exécutée sur le GPU.

façon drastique le temps de simulation par rapport à la version CPU, et ainsi d'augmenter considérablement la taille des systèmes simulés. A titre de comparaison, pour une même configuration, dans le cas des plus petits systèmes, une simulation sur CPU met un mois pour être exécutée pour deux heures pour la version GPU.

III.3.3 La consommation mémoire

La consommation mémoire est la limite de notre simulation de SRD-MD nous empêchant de pouvoir simuler des systèmes à plus grande échelle, à cause du nombre très important de particules de fluide à modéliser comparé au nombre de colloïdes. Pour une simulation avec une fraction volumique de $\Phi_C = 0.1$, le nombre de particules de fluide est 1578 fois plus grand que le nombre de colloïdes. Toutes les données de la simulation doivent être stockées dans la mémoire globale du GPU, afin d'éviter les transferts mémoires entre CPU et GPU très coûteux. Or la capacité mémoire des GPUs est bien importante que celle d'un CPU : la Tesla K20m possède 5 Go de mémoire et la carte GTX 690 2Go uniquement. Cependant, les deux GPU de dernières générations dont nous avons pu avoir accès, avec une capacité mémoire bien plus importante de 12 Go, montre que cette limite par rapport au CPU tend à diminuer. De plus, une partie de cette mémoire est déjà utilisée par le système de base, pour l'affichage, donc cette limite ne peut pas être atteinte.
Chaque particule requiert de stocker sa position, sa position triée, sa vélocité, son accélération, son accélération précédente, son déplacement depuis la dernière réactualisation de la liste de Verlet et des clés de hachage pour l'associer aux cellules des différentes grilles. Chaque particule de fluide requiert également une clé de hachage associée à sa cellule de SRD et un buffer mémoire temporaire utilisé pour des

calculs d'analyse tel que le calcul de la température du système. Chaque colloïde requiert également deux listes de Verlet de respectivement $nbMaxNeighborCC$ et $nbMaxNeighborCF$ éléments, pour les interactions colloïde-colloïde et colloïde-fluide. Chaque cellule nécessite deux index des premières et dernières particules qu'elle contient. Chaque cellule de SRD nécessite un compteur des particules présentes dans la cellule et la vitesse du centre de masse ainsi qu'un axe aléatoire de rotation. Dans le but de diminuer la consommation, nous avons utilisé le même espace mémoire pour stocker tous les buffers temporaires tels que les accélérations ou les positions non triées. Enfin, chaque générateur aléatoire a besoin de mémoire pour stocker son état.
La double précision est préférée pour stocker les données flottantes pour une question de validité des simulations. Dans un premier temps, les données avec trois coordonnées (position, vélocité, accélération, etc.) ont d'abord été stockées dans une structure *vector4*, dans une organisation mémoire nommée *Array of Structure* (AoS), comme dans les chapitres précédents. Cette organisation mémoire a pour avantage de diminuer le nombre de transactions lors d'accès à la mémoire du GPU. Les résultats présentés dans le tableau III.5 sont issu d'une implémentation utilisant cette organisation de cette mémoire. Afin de réduire au maximum la consommation mémoire, nous avons réimplémenté notre simulation, en séparant dans 3 buffers différents les coordonnées x, y et z, ce qui est appelé une organisation en *Array of Structure* (SoA). Il s'est révélé qu'en plus de réduire la consommation mémoire, cette organisation mémoire permet d'améliorer les performances comme le montre le Tableau III.6, cette amélioration des performances étant importante pour les architectures les plus anciennes (30% plus rapide pour la GTX690 et 15% plus rapide pour la K20m), tout en permettant de pouvoir simuler de plus grand système. Cette organisation mémoire peut être utilisé pour tous les autres simulations présentées jusqu'à présent et permettre d'obtenir une amélioration des performances du même ordre. La différence de performance entre une organisation SoA et AoS a été étudié dans [50], qui indique que l'organisation en SoA favorise l'accès coalescent aux données, tandis que celle en AoS permet de réduire le nombre de transactions, les threads chargeant les données par paquet de 128 bit. Dans le cas de calcul sur des données organisés de façon coalescente, comme c'est le cas dans les simulations de fluide, une organisation SoA est optimale.

Le Tableau III.7 résume la consommation mémoire d'une notre simulation , avec une organisation mémoire en SoA. Il liste uniquement la mémoire utilisée par les principaux buffers en fonction du nombre de colloïdes, de particules de fluide, de cellules de grille régulière et de SRD, sans prendre en compte la mémoire utilisée pour les tests. Par exemple, pour une simulation avec la fraction volumique colloïdale de $\Phi_C = 0.10$, $\#colloides = 10\,500$, $\#fluids = 16\,658\,282$, $\#cell_{grid} = 15625$, $\#cell_{SRD} = 2299968$, $nbMaxNeighborCC = 50$ and $nbMaxNeighborCF = 24050$, la mémoire totale utilisée outre celle concernant l'analyse est : $132\,o * 16\,658\,282 + 148\,o * 10\,500 + 8\,o * 15625 + 28\,o *$

Colloïdes	Fluide	Temps/itération (ms)		Total time (h)	
		GTX690	K20m	GTX690	K20m
500	788 981	59	57	0.9	0.9
2 000	3 120 942	242	235	3.7	3.6
4 000	6 311 854	493	474	7.5	7.2
6 000	9 487 161	XXX	792	XXX	12.1
8 000	12 623 708	XXX	971	XXX	14.8
10 500	16 658 282	XXX	1 179	XXX	18.0
11 000	17 317 669	XXX	XXX	XXX	XXX

Tableau III.6 – Temps moyen d'une itération et temps total d'une simulation de SRD-MD pour $\Phi_C = 0.1$ durant 55000 itérations, avec une organisation mémoire en SoA, sur un GPU GTX690, K20m et GTX970. Une itération est composée de 8 étapes de MD et 1 étape de SRD. "XXX" signifie que la consommation mémoire de la simulation est trop importante pour pouvoir être exécutée sur le GPU.

$2299968 + 4\,o * 10\,500 * 24100 = 3.28\,Go.$

III.4 Conclusion

Dans ce chapitre, nous avons présenté un nouvel algorithme pour des simulations de SRD-MD avec force de couplage sur GPU. Cet algorithme combine des précédents travaux sur la SRD et sur la MD sur GPU et propose un nouveau schéma de décomposition afin de s'adapter à la spécificité de la partie MD du modèle qui est liée à notre force de couplage. Ce nouveau schéma associe à chaque bloc de threads un colloïde et toutes ses interactions à calculer. Par rapport à la décomposition standard associant un thread par colloïde, cette stratégie permet d'éviter les divergences des warps liées aux problèmes de branches divergentes. En effet, dans la méthode standard, les threads doivent attendre que tous les threads de leur warp ont terminé de calculer leur interaction pour passer aux colloïdes suivants. Or plus la variation du nombre de voisins est importante entre colloïdes, plus le temps d'attente est important. Pour les systèmes ayant un grand nombre de particules voisines, notre schéma de décomposition par bloc devient ainsi plus avantageux. Cependant un bloc étant composé d'au minimum 32 threads, notre stratégie n'est pas adaptée pour les systèmes ayant en moyenne moins de 32 voisins et en pratique, moins de 64, car nos résultats montrent que cette méthode est optimale avec des blocs de 2 warps. Ainsi, dans le cas de nos simulations de SRD-MD avec force de couplage, la décomposition par bloc permet une accélération de 45% par rapport aux méthodes standards de la littérature.

La limite de nos simulations est la consommation mémoire très importante due au nombre très élevé de particules de fluide à représenter dans la SRD. Afin de réduire

élément	consommation mémoire
fluide	sorted position = 24 o * $\#fluids$ velocity = 24 o * $\#fluids$ previous acceleration = 24 o * $\#fluids$ displacement = 24 o * $\#fluids$ temporary buffer (for unsorted position, acceleration, center of mass velocity) = 24 o * $\#fluids$ hash grid = 4 o * $\#fluids$ hash SRD = 4 o * $\#fluids$ **total = 132 o *** $\#fluids$ **+ 4 o *** $\#fluids$ ***** $nbMaxNeighbor$
colloïdes	position = 24 o * $\#colloids$ sorted position = 24 o * $\#colloids$ velocity = 24 o * $\#colloids$ acceleration = 24 o * $\#colloids$ previous acceleration = 24 o * $\#colloids$ displacement = 24 o * $\#colloids$ hash grid = 4 o * $\#colloids$ verletCC = 4 o * $\#colloids$ * $nbMaxNeighborCC$ verletCF = 4 o * $\#colloids$ * $nbMaxNeighborCF$ **total = 148 o *** $\#colloids$ **+** **4 o *** $\#colloids$ ***** $(nbMaxNeighborCC + nbMaxNeighborCF)$
grille régulière	index start = 4 o * $\#cell_{grid}$ index end = 4 o * $\#cell_{grid}$ **total = 8 o *** $\#cell_{grid}$
grille SRD	counter = 4 o * $\#cell_{SRD}$ rotation axis = 24 o * $\#cell_{SRD}$ **total = 28 o *** $\#cell_{SRD}$

Tableau III.7 – Liste de tous les buffers stockés sur le GPU.

la consommation mémoire, nous avons changer l'organisation mémoire en AoS avec des *vector4* utilisé précédemment, par une organisation en SoA avec 3 buffers distincts pour les coordonnées x, y et z, pour un gain de 25% de mémoire. De plus, nous avons constaté que cette organisation mémoire favorise la coalescence et donc les performances, surtout pour les architectures plus anciennes. Ainsi, les précédentes simulations peuvent aussi appliquer une organisation mémoire en SoA afin de réduire la consommation mémoire et améliorer légèrement les performances.

La consommation mémoire reste une limite, qui nous empêche de simuler des systèmes très larges avec des GPUs standards dont la capacité mémoire est limité. Cependant, certains GPUs de dernière génération possèdent jusqu'à 32Go de mémoire permettant de dépasser cette limite en utilisant notre algorithme. Il existe également la solution d'utiliser un cluster de GPU, pour lequel il faudrait adapter notre méthode pour être parallélisée sur un cluster. Toutefois, même avec nos ressources actuelles, la parallélisation sur GPU de ce type de simulation de SRD-MD permet d'ouvrir des perspectives nouvelles dans l'étude de ce modèle, accélérant considérablement les temps de calcul et permettant une étude à une plus grande échelle.

Chapitre IV :

Application de la SRD à la synthèse d'images de fluides réalistes

Dans les chapitres précédents, nous avons présenté différentes simulations de MD dans le cadre des suspensions colloïdales. Dans la partie II.4, nous avons montré que les mêmes concepts permettant de simuler des suspensions colloïdales peuvent être appliqués à des simulations de nanoalliages à l'échelle atomique. Les simulations à base de systèmes de particules sont présentes dans bien d'autres domaines, notamment les simulations de fluide en informatique graphique. Par convention de langage, dans ce chapitre, nous allons désigner par simulations de fluide, les simulations de fluide réalisées en informatique graphique pour les opposer aux simulations de type MD. Le problème à résoudre dans ces simulations est le même que pour les précédentes simulations de MD, c'est-à-dire calculer les trajectoires des particules de fluide modélisées dans le but de reproduire un phénomène physique réel. Cependant, si les simulations de MD devaient simuler un phénomène de la manière la plus physiquement réaliste possible dans un temps raisonnable, les simulations de fluide recherchent plutôt une physique *vraisemblable*, mais obtenue en temps réel. Dans la première partie de ce chapitre, nous allons introduire plus précisément ce que sont les simulations de fluide. La seconde est consacrée aux descriptions de différentes approches permettant de résoudre les équations de ces simulations. Enfin la dernière présente nos travaux sur l'utilisation des concepts de la SRD pour simuler des phénomènes étudiés en informatique graphique. Par convention de langage, durant ce chapitre, le terme particule de fluide ne désigne plus comme précédemment les particules de solvant, mais tous types de particules de gaz ou de liquide.

IV.1 Les simulations de fluides

La simulation de fluide est un domaine de l'informatique graphique dont le but est de pouvoir produire des animations physiquement vraisemblables de phénomènes liés aux fluides, qui comprennent les liquides, les gaz et les plasmas. Leurs applications se trouvent principalement dans les milieux du jeu vidéo, de l'infographie et du cinéma, et plus généralement dans les milieux utilisant l'animation et la synthèse d'images. A la problématique principale qui est la résolution des équations du mouvement des particules de fluide, vient alors s'ajouter celle de pouvoir générer un rendu réaliste à partir du nuage de particules obtenu. De plus, les systèmes modélisés sont de nature très différente de ceux présentés jusqu'ici. Les systèmes des chapitres précédents étaient tous à l'échelle mésoscopique voire atomique. En revanche, en informatique graphique, les phénomènes que l'on cherche à simuler, tels que le déferlement des vagues, le mélange de deux fluides dans un verre ou le vent, sont à l'échelle macroscopique. Ainsi les modèles employés sont différents ou alors sont adaptés pour fonctionner à cette nouvelle échelle. Le changement de nature des systèmes à simuler entraîne également d'autres modifications par rapport aux simulations précédentes. Premièrement, il n'y a plus toujours une boîte de simulation

avec des conditions périodiques au bord. La simulation doit effectuer une étape de gestion des bords et de manière plus générale, la collision des particules de fluide avec d'autres objets tels que des murs, un sol ou des débris selon les simulations. Deuxièmement, le fluide n'occupe plus tout l'espace de simulation. Ainsi, il existe une interface entre les particules de fluide modélisées et l'air, qui implique le phénomène de tension de surface. Un autre phénomène à prendre en compte est l'incompressibilité de certains fluides qui doit être assurée au cours de toute la simulation. D'autres types de problématiques sont à résoudre selon le phénomène étudié, comme pour les simulations multiphases qui cherchent à représenter le mélange de différents fluides. Tous ces points sont décrits plus précisément dans les paragraphes suivants, afin de mieux comprendre les problématiques posées par les simulations de fluide.

IV.1.1 Les équations du mouvement

Comme pour les simulations de type MD, la problématique principale d'une simulation de fluide en informatique graphique est de résoudre les équations régissant le mouvement du fluide. Dans les simulations de fluides, ces équations sont celles de Navier Stokes. Elles ont deux formulations possibles, une eulérienne et une lagrangienne selon la manière dont est représenté le fluide. Dans sa forme la plus générale, l'équation de Navier-Stockes exprime la dérivée particulaire $\frac{D}{Dt}$ en fonction du temps :

$$\frac{Dv_i(t)}{Dt} = -\frac{1}{\rho_i}\nabla p_i(t) + \nu\nabla^2 v_i(t) + \frac{F_i^{autre}(t)}{m_i} \tag{IV.1}$$

où ρ_i est la densité de la particule de fluide, p_i sa pression, ν sa viscosité cinématique et m_i sa masse. Le terme $\frac{1}{\rho_i}\nabla p_i$ représente l'accélération des particules due à la différence de pression dans le fluide. La force lui correspondant domine généralement toutes les autres forces. Elle conserve le volume du fluide et sachant que la masse des particules est constante, des petits écarts de densité sont préférables pour avoir une simulation de qualité conservant le volume du fluide si le but est de simuler des fluides incompressibles comme de l'eau. Le terme $\nu\nabla^2 v_i$ dénote l'accélération due aux forces de friction entre les particules possédant des vitesses différentes. Bien que ν est une donnée physique connue pour les fluides, elle est souvent déterminée de façon empirique par l'utilisateur afin d'obtenir un comportement correct. Enfin le terme $\frac{F_i^{autre}}{m_i}$ correspond aux autres forces appliquées à la particule comme la gravité ou le vent. Les différents modèles que nous verrons ont pour but de résoudre l'équation (IV.1) de Navier Stokes. C'est un problème complexe dont les modèles ne peuvent que donner une résolution approximative. La résolution de ce problème nécessite de résoudre un problème de voisinage. Notre solution proposée pour nos simulations de MD ne convient pas pour la majorité des simulations de fluide de la littérature, car il a été montré que les méthodes de conservation rendaient la simulation

plus lentes et étaient trop coûteuses en mémoire [51]. Cela s'explique par le fait que les déplacements des particules sont bien plus rapides, et qu'il n'y a pas de phénomènes d'agrégation, ce qui implique un changement bien plus rapide du voisinage.

IV.1.2 La gestion des collisions

L'étape de gestion des collisions permet de prendre en compte les interactions entre le fluide et une surface rigide, afin d'empêcher l'interpénétration du fluide avec d'autres objets ou pour empêcher que le fluide ne sorte du domaine dans le cas où il est enfermé. La façon de résoudre ce problème dépend du modèle employé et notamment de la façon dont est modélisé le fluide. La principale difficulté est d'éviter les problèmes de pénétration sans ajouter de l'énergie dans le système.

IV.1.3 La tension de surface

Le gestion de la tension de surface permet de simuler l'interface entre le fluide et l'air (ou tout autre gaz), plus communément nommée surface. A la surface, le système tend vers un équilibre correspondant à une configuration minimisant l'énergie, qui a pour conséquence de diminuer l'aire de l'interface. Cela est la conséquence de la force de tension de surface. Un moyen de modéliser cette force est de modéliser l'air. Il faut alors modéliser les interactions air-fluide, ce qui revient à résoudre un autre problème, celui d'une simulation de fluide multiphase. De plus, dans le cadre d'une synthèse d'images, où la majorité du temps, seules l'animation et la visualisation du fluide nous intéressent, la modélisation de l'air peut représenter un coût excessif réduisant les performances. Ce coût peut être réduit en ne modélisant qu'une couche à la surface du fluide. Il faut alors ajouter une étape pour détecter la surface du fluide. Une autre approche est de modéliser une force qui n'est appliquée qu'à la surface du fluide. Le problème est alors de pouvoir assurer que cette force n'agit qu'à la surface sans affecter tout le système, tout en modélisant bien tous les effets liés à cette force, c'est-à-dire une minimisation de la courbure, pas de groupement de particules de fluide et aucune dissipation de la quantité de mouvement.

IV.1.4 L'incompressibilité

La compressibilité d'un fluide est mesurée par la variation de son volume lorsqu'elle subit une force de pression externe. Certains fluides tels que l'eau ou le sang, sont dits incompressibles, ce qui signifie que leur volume reste constant même sous une force de pression. Garantir l'incompressibilité pour ces fluides est un enjeu majeur pour obtenir des simulations réalistes. C'est ce qui permet, par exemple, d'éviter qu'au cours de la simulation, le volume de l'eau dans un verre ne fluctue. Dans une simulation

numérique, une certaine marge d'erreur peut être tolérée, le but étant d'avoir une simulation vraisemblable. On peut considérer que si le fluide n'excède pas un seuil limite de compressibilité, la simulation est correcte. Les différents modèles doivent donc pouvoir offrir un moyen de garantir que ce seuil limite ne peut être dépassé.

IV.1.5 La gestion des fluides multiphases

Certains types de phénomènes nécessitent de prendre en compte plusieurs types de fluides dans le système, même lorsque l'on ne veut étudier la dynamique que d'un seul des fluides du système. D'autres travaux de la littérature se focalisent sur des simulations de fluides multiphases comme dans le cas de mélanges. Un moyen d'intégrer et de généraliser les interactions entre plusieurs fluides dans les modèles est donc primordial dans ce type de simulations.

IV.1.6 Le rendu

Créer un rendu réaliste du fluide nécessite de traiter le problème de reconstruction de la surface. Les données de départ et donc les méthodes varient en fonction des modèles utilisés. Dans le cas de la SRD et autres modèles se basant sur des systèmes de particules, le problème revient à extraire une surface à partir d'un nuage de points. Notre étude visant à introduire un nouveau modèle afin de modéliser le fluide, et non une nouvelle méthode pour obtenir un rendu réaliste, nous avons utilisés une simple visualisation par particules. Pour avoir un rendu réaliste, nous pouvons ensuite utiliser une librairie tiers, *openvdb* [52] pour traiter ce problème. Il permet de reconstruire une surface à partir d'une nuage de points, puis de traiter le résultat brut obtenu avec des fonctions de filtrage pour améliorer le rendu.

Les deux parties suivantes, sont consacrées à l'état de l'art de deux modèles de la littérature répondant aux problématiques du domaine de la simulation de fluide. Le premier est le modèle SPH, un des plus étudiés dans la littérature, parmi les modèles lagrangiens. Le second, PIC/FLIP, est une description d'un modèle hybride, eulérien-lagrangien utilisant à la fois une grille et un système de particule pour modéliser le fluide, comme la SRD.

IV.2 Le modèle SPH

L'approche *Smoothed Particle Hydrodynamics* (SPH), créée par Lucy en 1977 [53] et par Gingold et Monaghan [54] de façon indépendante, est une méthode pour calculer

les dérivées spatiales n'utilisant pas de grille comme les approches eulériennes, mais en effectuant des interpolations sur des systèmes de particules. Tout d'abord utilisée pour simuler des phénomènes astrophysiques, cette méthode a été adaptée pour des simulations de fluide réalistes en temps réel par Müller [55] en 2003. L'avantage de cette méthode est de simplifier la formulation des équations qui régissent les mouvements des fluides, transformant notamment le calcul du gradient de la pression par des forces entre paires de particules. Depuis, la recherche sur les animations de fluide réalistes en temps réel employant la méthode SPH s'est démocratisée et est aujourd'hui l'un des concepts majeurs des simulations de fluide en informatique graphique [51].
Le fluide est représenté par un ensemble de particules, comme pour toutes les méthodes dites lagrangiennes. Chaque particule i possède des quantités telles qu'une masse m_i, un volume V_i, une densité ρ_i et exerce une pression p_i. Ainsi, les quantités d'une particule représentent les quantités locales du fluide à la position r_i. Les particules, ainsi que les quantités qu'elles portent, sont déplacées avec le flux du fluide à partir de la vitesse locale v_i selon :

$$\frac{dr_i(t)}{dt} = v_i(t) \tag{IV.2}$$

Le terme v_i se calcule avec l'équation de Navier-Stokes, mais cette fois-ci dans sa formulation lagrangienne. La dérivation $\frac{Dv_i(t)}{Dt}$ (voir équation (IV.1)) correspond à la dérivée de la vitesse selon le temps du point échantillon, qui est dans le cas du SPH et des méthodes lagrangiennes, la particule elle-même, déplacée selon sa vitesse. Ainsi, on a simplement $\frac{Dv_i(t)}{Dt} = \frac{dv_i(t)}{dt}$ et $\frac{dr_i(t)}{dt} = v_i$. D'où la formulation lagrangienne de l'équation de Navier-Stockes :

$$\frac{dv_i(t)}{dt} = -\frac{1}{\rho_i}\nabla p_i(t) + \nu\nabla^2 v_i(t) + \frac{F_i^{autre}(t)}{m_i} \tag{IV.3}$$

Avant de pouvoir résoudre cette équation, il faut expliquer les concepts d'interpolation et de dérivée spatiale du modèle SPH qui permettent une approximation du résultat de l'équation. Ces notions sont décrites dans le paragraphe suivant.

IV.2.1 Le concept du SPH

L'interpolation L'idée centrale dans une approche SPH est la méthode d'interpolation. De manière générale, en dehors du contexte de simulation de fluide, toute fonction peut se calculer à partir d'un système de particules. Toute quantité A_i en toute position r_i peut être calculée en intégrant les particules du voisinage :

$$A_i = \int A_j W_{ij} dr_j \tag{IV.4}$$

ce qui est simplifié dans le cadre d'une simulation numérique par la somme des particules dans le voisinage de i :

$$A_i \approx \sum_j \frac{m_j}{\rho_j} A_j W_{ij} \tag{IV.5}$$

où W_{ij} est une fonction noyau. Cette fonction définit la forme du voisinage. Elle varie en fonction des simulations, mais est toujours de la forme :

$$W_{ij} = W\left(\frac{\|r_i - r_j\|}{h}\right) = W(q) \tag{IV.6}$$

où h est la longueur de lissage.
Le choix de cette fonction noyau va influer sur la précision de la somme effectuée dans l'équation (IV.5). Plus le nombre de particules comprises dans ce voisinage est élevé, plus le calcul est précis, mais coûteux. Il faut donc le choisir en fonction d'un compromis entre performance et précision de la simulation. Il n'y a pas de consensus pour un choix de fonction noyau idéal. Cependant certaines sont utilisées couramment dans la littérature, comme la fonction spline cubique [56, 57].

Les dérivées spatiales Afin de pouvoir résoudre l'équation (IV.3), il est nécessaire de comprendre ce que sont les opérateurs de dérivées spatiales ∇, $\nabla\cdot$ et ∇^2. ∇ est l'opérateur gradient défini par :

$$\nabla q = \left(\frac{\partial q}{\partial x}, \frac{\partial q}{\partial y}, \frac{\partial q}{\partial z}\right)$$

Il représente la façon dont un scalaire varie dans l'espace.
$\nabla\cdot$ est l'opérateur de divergence défini par :

$$\nabla \cdot \vec{u} = \frac{\partial \vec{u}}{\partial x} + \frac{\partial \vec{u}}{\partial y} + \frac{\partial \vec{u}}{\partial z}$$

Il ressemble à l'opérateur de gradient, mais représente la variation globale d'un vecteur.
∇^2 est l'opérateur laplacien défini par :

$$\nabla^2 q = \frac{\partial^2 q}{\partial^2 x} + \frac{\partial^2 q}{\partial^2 y} + \frac{\partial^2 q}{\partial^2 z}$$

C'est l'application de l'opérateur gradient suivi de celui de la divergence. Il représente l'échange du champ de scalaire avec son voisinage. Par exemple si la valeur de $\nabla^2 q$ est faible, cela signifie que la valeur du scalaire q est faible par rapport à la moyenne de son voisinage. Ces dérivées spatiales peuvent être calculées de différentes manières pour être appliqués à A_i. Dans le modèle SPH, les formulations originales pour $\nabla A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla W_{ij}$ et $\nabla^2 A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W_{ij}$ ne sont pas employées à cause des forces asymétriques qui entraînent des instabilités. D'autres approximations ont été étudiées

pour les simulations SPH et les suivantes sont généralement préférées [58, 59] :

$$\nabla A_i = \rho_i \sum_j m_j \left(\frac{A_j}{\rho_i^2} + \frac{A_j}{\rho_j^2} \right) \nabla W_{ij}, \tag{IV.7}$$

$$\nabla^2 A_i = 2 \sum_j \frac{m_j}{\rho_j} A_{ij} \frac{r_{ij} \cdot W_{ij}}{r_{ij} \cdot r_{ij} + 0.01h^2}, \tag{IV.8}$$

avec $A_{ij} = A_i - A_j$ et $r_{ij} = r_i - r_j$, la distance entre i et j

Ces formulations du gradient (équation (IV.7)) et du laplacien (équation (IV.8)) vont respectivement permettre de calculer les termes d'accélération des particules due à la différence de pression $\frac{1}{\rho_i(t)} \nabla p_i(t)$ et aux forces de friction $\nu \nabla^2 v_i(t)$ de l'équation (IV.3).

IV.2.2 La base de l'algorithme SPH

L'algorithme SPH consiste à résoudre l'équation (IV.3) de Navier-Stockes sous sa forme lagrangienne. Pour cela, il faut calculer à chaque itération et pour chaque particule, ces trois termes :

- les forces de pression $\frac{1}{\rho_i} \nabla p_i(t)$,
- les forces de friction ou viscosité $\nu \nabla^2 v_i(t)$,
- les autres forces externes $\frac{F_i^{autre}(t)}{m_i}$ telles que la gravité, le vent, etc, selon les simulations.

En utilisant les concepts SPH vus précédemment, ces termes peuvent être calculés. La densité ρ_i se calcule avec la méthode d'interpolation donnée par l'équation (IV.5), qui dans ce cas est simplifiée :

$$\rho_i = \sum_j \frac{m_j}{\rho_j} \rho_j W_{ij} = \sum_j m_j W_{ij} \tag{IV.9}$$

La pression p_i pourrait également être calculée par interpolation. Cependant, de nombreuses alternatives sont proposées dans la littérature, dont une des plus simples [60–62] est l'utilisation des équations d'état, dont la forme dans ce contexte exprime p_i en fonction de ρ_i par :

$$p_i = k \left(\left(\frac{\rho_i}{\rho_0} \right)^7 - 1 \right) \tag{IV.10}$$

où k est la constante de raideur et ρ_0 est la pression au repos du fluide. L'intérêt d'exprimer p_i ainsi est de pouvoir se servir des paramètres k et ρ_0 pour contrôler la contribution des forces de pression sur la dynamique du fluide. k permet de pouvoir ajuster la pression : plus k est élevé, plus les forces de pression tendent à amener les particules de fluide dans une configuration de repos, c'est à dire où la densité $\rho_i = \rho_0$.

Algorithm 7 Etapes d'une simulation de fluide SPH

1: **for all** particule i **do** // en parallèle
2: $Voisin$ = calculer les voisins j
3: ρ_i = calculer densité i à partir de $Voisin$ (équation (IV.9))
4: p_i = calculer pression i à partir de ρ_i (équation (IV.10))
5: **for all** particule i **do** // en parallèle
6: $F_i^{pression} = -\frac{m_i}{\rho_i}\nabla p_i$ (avec l'équation (IV.7))
7: $F_i^{visco} = m_i \nu \nabla^2 v_i(t)$ (avec l'équation (IV.8))
8: $F_i^{autres} = m_i g$ (pour le cas de la gravité)
9: $F_i^{totale} = F_i^{pression} + F_i^{visco} + F_i^{autres}$
10: **for all** particule i **do** // en parallèle
11: $v_i(t + \Delta t) = v_i(t) + \Delta t \frac{F_i^{totale}}{m_i}$ (2^e loi de Newton)
12: $r_i(t + \Delta t) = r_i(t) + \Delta t v_i(t + \Delta t)$ (équation (IV.2))

L'algorithme (7) résume les différentes étapes de base d'une itération de simulation SPH utilisant pour calculer la pression l'équation d'état (IV.10), qui est alors appelée simulation SESPH. Le but est de pouvoir calculer toutes les positions $r_i(t)$ des particules du système. Tout d'abord, il faut déterminer l'ensemble des voisins j de la particule i à partir de la fonction noyau W_{ij} de l'équation (IV.6). Ce voisinage permet de calculer ρ_i (équation (IV.9), puis à partir de ρ_i de calculer p_i (équation (IV.10)). Ensuite, pour chaque particule, les trois types de forces de l'équation de Navier-Stockes (pression, viscosité et autres) peuvent être calculées à partir des termes précédents et des dérivées partielles (équations (IV.7) et (IV.8)). Enfin, avec la troisième loi de Newton, cette somme de forces permet de calculer l'accélération des particules, qui permet d'obtenir la vitesse une fois intégrée selon le temps. La méthode d'intégration présentée dans l'algorithme est celle d'Euler-Cromer, une approche couramment utilisée [63, 64]. Le pas de temps Δt doit respecter les conditions de Courant-Friedrich-Levy (CFL), soit $\Delta t \leqslant \lambda \frac{a}{||v^{max}||}$ avec $\Delta t \approx 0.4$, λ étant un scalaire, a étant le diamètre des particules et v^{max} la vitesse maximale de toutes les particules [58]. Cela permet de limiter le déplacement des particules de fluide lors d'une itération en fonction du diamètre des particules. Pour $\lambda = 1$, cela signifie par exemple qu'une particule ne peut pas se déplacer à une distance supérieure à son diamètre. L'algorithme (7) ne présente que les étapes de base : il reste maintenant à comprendre comment les problématiques de gestion des collisions, de tension de surface, d'incompressibilité et de fluides multiphases peuvent être résolues avec le modèle SPH.

IV.2.3 La gestion des collisions

Afin que les particules de fluides ne pénètrent pas des objets ou des murs, il est nécessaire d'ajouter à l'algorithme 7 des étapes de gestion des collisions. Une des méthodes

les plus simples et les plus utilisées est de placer des particules dans les bords des objets rigides. Ces particules n'ont pas la même dynamique que les particules de fluide. Elles se déplacent en fonction de l'objet auquel elles appartiennent et exercent une force sur le fluide, qui est souvent une force de répulsion basée sur la distance entre les particules [60,65]. Ainsi, la gestion des collisions s'effectue dans le calcul de la pression. Les particules qui pénètrent un objet sont repoussées en dehors de l'objet à la prochaine itération, via ces forces de répulsion. Cependant cette méthode bien que très simple à mettre en place, nécessite de réduire Δt pour que les pénétrations des particules de fluides dans les objets soit les plus superficielles possibles. En effet, plus les particules pénètrent dans l'objet, plus les forces de répulsion pour les faire ressortir sont fortes, ce qui peut causer des forces de pression trop élevées dans le système. Le faible Δt que requiert cette méthode nuit donc aux performances en temps de la simulation. Il faut alors calculer plus d'itérations pour un même temps de simulation, ce qui revient à diminuer la taille du système pour rester dans la contrainte du temps réel.

Afin d'éviter de diminuer Δt, il est possible d'utiliser une méthode de résolution par contraintes [55]. Cette méthode vérifie pour chaque particule si elle pénètre un objet. Si c'est le cas, alors la particule est déplacée en dehors de l'objet, en modifiant directement sa position. Un phénomène qui apparaît avec ces deux méthodes, est un regroupement de particules de fluide à la surface des objets, ce qui s'explique par la carence de particules de fluide près de l'interface fluide-objet. Cela pose problème dans les calculs des quantités du fluide calculées par interpolation tels que ρ_i, où le faible nombre de voisins va créer une mauvaise approximation de la quantité. Ainsi la densité est sous évaluée et les forces de pression qui en résultent ne sont pas assez importantes pour éviter des interpénétrations entre particules de fluide.

Les solutions proposées consistent à prendre en compte les objets et les murs dans le calcul des densités des particules de fluide qui les entourent. La plupart des solutions utilisent les particules des bords pour contribuer aux calculs de densité comme si elles étaient des particules de fluide. Il faut alors déterminer pour elles aussi une densité. Certaines méthodes calculent la densité des particules du bord comme pour des particules de fluide, à partir de leur voisinage [66, 67]. D'autres recopient les densités des particules de fluide voisines [64, 68, 69]. La manière dont sont réparties les particules de bord dans l'objet, c'est-à-dire son échantillonnage, joue aussi un rôle important sur la densité et la pression. Dans les références [64, 66, 69] les auteurs utilisent un échantillonnage précalculés des objets tandis que d'autres calculent un échantillonnage à la volée [68]. Le choix de la méthode dépend de la forme des objets, selon la complexité de la surface avec des régions convexes et concaves.

IV.2.4 La tension de surface

Deux catégories de méthodes permettent de reproduire le phénomène de tension de surface : les modèles microscopiques et macroscopiques. Les modèles macroscopiques se basent sur le *color field* c, qui est défini ainsi : $c = 1$ pour les particules de fluide et $c = 0$ ailleurs, ∇c représente alors la normale n de la surface pointant vers le fluide et la courbure κ peut être calculée par $\kappa = -\nabla^2 c$. La tension de surface résultant de l'interaction entre le fluide et l'air s'applique en suivant cette normale n et tend à minimiser la courbure κ. Afin que la tension de surface n'agisse qu'à l'interface de deux fluides, le *color field* est normalisé [70]. Cependant, l'approximation faite sur l'opérateur laplacien avec l'équation (IV.8) dans le modèle SPH est sujette à erreur pour le calcul de la surface [71]. Les modèles microscopiques considèrent des forces de cohésion entre les particules pour imiter les forces attractives entre les molécules [56, 72], en ajoutant une force aux particules de la surface. Cela permet d'éviter d'utiliser l'opérateur laplacien. Par exemple, dans la référence [72], l'équation d'état de van der Waals est utilisée pour calculer des forces de cohésion. Cependant, que ce soit pour les modèles macroscopiques ou microscopiques, tous les effets de la tension de surface ne peuvent être modélisés correctement avec un seul modèle. Ainsi, dans la référence [73], la tension de surface est modélisée par une combinaison des différentes approches décrites ci-dessus.

IV.2.5 L'incompressibilité

L'incompressibilité est un enjeu majeur dans les simulations où le fluide est dit incompressible tel que l'eau. En réalité, l'objectif est d'obtenir une compressibilité faible entre 0.1 et 1%, ce qui devient l'étape la plus coûteuse dans une simulation SPH [74]. De nombreuses méthodes ont été proposées dans la littérature. Le but n'étant pas d'être exhaustif, mais de comprendre la manière dont l'incompressibilité peut être garantie avec le modèle SPH, deux méthodes vont être présentées ici, une non itérative et une itérative. La première méthode, le Weakly Compressible SPH (WCSPH) présentée dans [56], fait partie des méthodes non itératives. Elle consiste dans le cadre d'une simulation SESPH, utilisant l'équation (IV.10), à se servir de la valeur de la constante de raideur k afin de garantir l'incompressibilité du fluide. Plus k est élevée, plus les forces de pression sont fortes et amènent le fluide à revenir à sa densité de repos ρ_0 rapidement. Cependant, le problème en terme de performance de cette méthode, est qu'il est alors nécessaire de réduire considérablement le pas de temps Δt afin de respecter les conditions CFL. De plus, il est nécessaire de définir la valeur de k pour garantir le taux d'incompressibilité η souhaité, tout en ayant le plus grand Δt possible. C'est ce que la simulation WCSPH entreprend, en définissant k par :

$$k = \frac{\rho_0 c_s^2}{7}$$

où c_s est la vitesse du son dans le fluide qui est fixé dans [56] d'après la relation :

$$\frac{\Delta\rho}{\rho_0} \approx \frac{\mid v_{f_{max}} \mid^2}{c_s^2}$$

où $\Delta\rho = \rho - \rho_0$ et $v_{f_{max}}$ est la vitesse la plus élevée de toutes les particules de fluide durant toute la simulation. Pour garantir un taux de compressibilité maximal η, il faut alors définir c_s telle que $\frac{|v_{fmax}|^2}{c_s^2} < \sqrt{\eta}$. Bien que cette méthode permet de calculer une itération très rapidement comparée aux méthodes itératives, le Δt très faible en fait une méthode bien moins performante.
La deuxième méthode, nommée Predicted Corrected Incompressible SPH (PCISPH), est une méthode itérative proposant un système de prédiction et de correction des vitesses et des positions des particules [75]. Le but est de pouvoir éviter la diminution du pas de temps de l'approche WCSPH. Cette approche utilise un procédé itératif. Tout d'abord, elle calcule les vitesses v_i* et les positions r_i* intermédiaires, sans prendre en compte les forces de pression. Puis le processus itératif commence, dans lequel, à partir de r_i*, les densités ρ_i* et ensuite les pressions p_i* sont calculées et enfin la force de pression est ajoutée aux v_i* et r_i*. La variation de ρ_i* par rapport à la densité cible est alors estimée, et tant que cette variation est supérieure aux taux η, le processus itératif se renouvelle. Ainsi le calcul d'une itération de PCISPH, correspond en réalité à plusieurs calculs pour la même itération avec des forces de pression à chaque fois renouvelées. Le calcul d'une seule itération est donc plus long, mais la simulation reste bien plus performante qu'avec la première méthode.

IV.2.6 La gestion des fluides multiphases

Le modèle SPH permet d'intégrer facilement plusieurs types de liquides. Deux particules de fluide différentes se distinguent simplement par une masse m_i et une densité au repos ρ_0 différentes. Les autres quantités se calculent comme auparavant, par interpolation. Cependant, cela ne fonctionne correctement que pour des fluides dont le ratio de densité est inférieur à 10. En effet, dans le cas contraire, la méthode d'interpolation SPH devient imprécise à cause de la trop forte discontinuité au niveau des interfaces [58]. La méthode peut être adaptée en remplaçant pour le calcul de $\rho_i = \sum_j m_j W_{ij}$ par $\rho_i = m_i \sum_j W_{ij}$: ainsi, la valeur de ρ_i n'est plus affectée par la masse de ses voisins, mais toujours par leur noyau W_{ij}.

IV.3 Le modèle PIC/FLIP

Les modèles Particules In Cell (PIC) et Fluid Implicit Particle (FLIP) sont des modèles hybrides, utilisant à la fois un système de particules comme les approches lagrangiennes telles que SPH, et à la fois une grille comme les approches eulériennes, ce que fait également la SRD. Les modèles hybrides de ce type ont été développés dans le but de combiner les qualités des deux modèles et de combler leurs lacunes. Nous allons présenter dans un premier temps ce qui différencie les méthodes eulériennes et lagrangiennes afin de montrer quelles sont leurs lacunes.

IV.3.1 Les approches lagrangienne et eulérienne

Les approches lagrangiennes et eulériennes sont les deux manières les plus communes de modéliser un fluide. Comme nous l'avons vu avec le SPH, avec le point de vue lagrangien, on représente le fluide comme un ensemble de particules qui forment le volume du fluide (voir équation (IV.3)). C'est de ce point de vue que découlent toutes les simulations que nous avons vues jusqu'à présent, qui sont des simulations basées sur des systèmes de particules. Cette approche a l'intérêt d'être intuitive et de représenter correctement la partie advection, c'est à dire que les quantités du fluide sont bien transportées par son flux. Cependant, les lacunes de la méthode se situent dans la méthode d'interpolation comme nous l'avons montré dans les paragraphes IV.2.3 et IV.2.6. L'interpolation devient très approximative et pose des problèmes de réalisme lorsque la discontinuité dans le champ d'une quantité est trop forte, ou lorsque le nombre de particules dans une région est trop faible. Ceci entraîne une approximation dans le calcul des forces de pression ce qui explique notamment les problèmes du modèle avec la contrainte de l'incompressibilité.
Le point de vue eulérien représente le fluide comme un milieu continu divisé dans les cellules d'une grille. Il se base toujours sur l'équation de Navier-Stokes générale (IV.1), mais sous la forme eulérienne, l'équation ne décrit pas le mouvement d'éléments ponctuels de fluides, mais d'une région. Ainsi, dans l'équation (IV.1), pour la dérivée de la vitesse, on a $\frac{Dv_i}{Dt} = \frac{\partial v_i}{\partial t} + v_i \cdot \nabla v_i$, d'où la forme eulérienne de l'équation de Navier-Stokes :

$$\frac{\partial v_i}{\partial t} = \frac{1}{\rho_i}\nabla p_i(t) + \nu\nabla^2 v_i(t) + \frac{F_i^{autre}(t)}{m_i} - v_i \cdot \nabla v_i \tag{IV.11}$$

Cela revient, en pratique à se baser sur des particules fixes alignées sur la grille, qui contrairement à la méthode lagrangienne, ne sont pas advectées en fonction de leur vélocité. Les problèmes de régions peu denses, avec trop peu de particules pour que la méthode d'interpolation soit précise, ne se posent donc pas avec une approche eulérienne, et par conséquent la précision du calcul de la densité, de la pression et de l'incompressibilité

non plus. De plus, le voisinage étant constant, il peut être calculé simplement et n'est pas une étape coûteuse. Cependant, en contrepartie, la partie advection est plus approximative, utilisant une approche semi-lagrangienne, qui se base sur le même point de vue lagrangien du transport des quantités dans le flux, mais permet de le faire à partir de particules fixées sur une grille. L'idée des méthodes hybrides est donc de combiner ces deux approches afin de pouvoir réaliser la partie advection avec des particules lagrangienne et la partie calcul des pressions par une grille eulérienne.

IV.3.2 La méthode FLIP

Les méthodes hybrides PIC et FLIP utilisent une même approche se basant sur deux types de particules pour représenter le fluide. D'un côté, des particules lagrangiennes dont le mouvement est libre, qui permettent de réaliser la partie advection en mettant à jour les positions des particules. De l'autre, des particules fixées aux cellules d'une grille eulérienne, qui permettent de calculer les forces de pression et de résoudre l'équation (IV.11) afin de calculer la nouvelle vélocité des particules libres. Il faut alors avoir une méthode pour transférer les vélocités entre les particules lagrangiennes et leur cellule eulérienne afin de lier les deux parties de l'algorithme. La différence entre les méthodes PIC et FLIP concerne ces points, et c'est ce qui explique les approximations de ces types de méthodes. Le modèle PIC, introduit en 1957 par Harlow [76], utilise une double interpolation pour le transfert des vélocités : une pour calculer les vélocités des particules de la cellule à partir des particules lagrangiennes, une autre pour mettre à jour la vélocité des particules lagrangiennes en les remplaçant par les particules de leur cellule. Cette méthode présente un problème de dissipation à cause de cette double interpolation, dont l'approximation tend à lisser la vélocité des particules peu à peu. Ainsi ce modèle rend les fluides plus visqueux que ce qui est attendu. Pour éviter ce problème, en 1986, Brackbill *et al.* proposent le modèle FLIP, une variante du modèle PIC évitant la deuxième interpolation pour mettre à jour la vélocité de particules lagrangiennes [77]. A la place, Brackbill propose de n'interpoler que la variation de la vélocité pour l'ajouter à celle des particules lagrangiennes. Ainsi, le lissage des vélocités n'est pas accumulé, ce qui permet d'obtenir un fluide moins visqueux, mais, en contrepartie, cette méthode provoque des bruits visibles à la surface. Les méthodes PIC et FLIP ont été adaptées en 2005 par Zhu *et al.* pour traiter le cas de fluides incompressibles [78], en combinant les modèles afin d'atténuer les problèmes de dissipation de PIC et le bruit à la surface de FLIP.

Au début de la simulation, la phase d'initialisation consiste à placer aléatoirement dans chaque cellule, huit particules lagrangiennes. Les particules fixées aux cellules sont placées au centre des arêtes de la cellule, ce qui correspond à une grille *staggered Marker and Cell* (MAC). L'algorithme 8 récapitule les étapes d'une itération d'une simulation

Algorithm 8 Etapes d'une itération de simulation PIC/FLIP

1: **for all** cellule i **do** // en parallèle
2: transférer les vélocités v_{i_L} des particules lagrangiennes voisines à i aux vélocités des particules fixes v_i par interpolation (équation IV.5)
3: sauvegarder la vélocité des particules de i dans $v_{i_{old}}$ (FLIP)
4: calculer des forces externes aux particules de i et mettre à jour leur v_i (IV.11)
5: transférer v_i aux vélocités de leur particules lagrangiennes v_{i_L} par interpolation (PIC)
6: ajouter l'interpolation de ses différences des vélocités $v_i - v_{i_{old}}$ à la vélocité des particules lagrangiennes v_{i_L} (FLIP)
7: **for all** particule lagrangienne i **do** // en parallèle
8: calculer la nouvelle position r_i à partir de v_{i_L}

PIC/FLIC. Tout comme pour une simulation SPH, l'algorithme se décompose en deux parties : le calcul de la vélocité des particules de fluide en résolvant l'équation de Navier-Stokes et l'advection des particules de fluide. Les paragraphes suivants détaillent les étapes spécifiques à une simulation PIC/FLIP comparée à une simulation SPH. Le calcul des forces externes se déroule de la même manière que pour une simulation SPH.

Transfert des vélocités à la grille Afin de pouvoir résoudre l'équation de Navier-Stokes avec la grille, il faut transférer les vélocités des particules lagrangiennes aux particules fixes de leur cellule. La vélocité des particules fixes se calcule par une interpolation trilinéaire des vélocités des particules lagrangiennes voisines. Pour cela, il est donc nécessaire d'effectuer une recherche de voisinage, dans le rayon défini par une distance inférieure à deux fois la taille d'une cellule.

Calcul de la pression Le calcul de la pression avec la forme eulérienne de l'équation de Navier Stokes (équation IV.11) est plus complexe. Il consiste à résoudre une équation de Poisson $\nabla^2 p = \nabla \cdot v$. La partie $\nabla \cdot v$ est complexe dans un système eulérien, car elle ne s'applique pas à une particule, mais à une cellule, soit à un ensemble de particules fixes. Cela revient donc à résoudre un grand système d'équations linéaires, ce qui fait que le calcul de la pression est le goulot d'étranglement des simulations eulériennes et également eulériennes/lagrangiennes, représentant plus du tiers du temps de la simulation [79].

Transfert de la vélocité de la grille vers les particules Une fois les nouvelles vélocités calculées dans la grille, il est nécessaire de les transférer aux particules lagrangiennes afin de pouvoir réaliser la partie d'advection. C'est sur ce point que les méthodes PIC et FLIP divergent. Avec la méthode PIC, cela s'effectue à partir d'une interpolation trilinéaire avec les particules de la grille. La méthode FLIP calcule la variation entre l'ancienne et la nouvelle vélocité, et ajoute aux anciennes vélocités des

particules lagrangiennes, l'interpolation de ces variations. Dans une méthode PIC/FLIP, ces deux apports sont additionnés ensemble, en les pondérant par un coefficient selon le fluide à simuler. Pour un fluide visqueux, une forte pondération de la méthode PIC est avantageux, et au contraire, une forte pondération de la méthode FLIP s'adapte mieux au cas non visqueux [78].

Les diverses méthodes appliqués aux simulations SPH, répondant à ces problématiques, sont également applicables aux simulations PIC/FLIP, exceptées celles qui concernent le calcul des forces de pression. C'est la cas par exemple de la gestion des collisions, car cette partie s'effectue avec une grille avec PIC/FLIP. De plus, les problématiques liées au problèmes d'incompressibilité ne se posent plus.

IV.3.3 La gestion des collisions

Afin de pouvoir gérer des collisions dans le cas de murs fixes, il faut ajouter une étape à l'algorithme 8 afin de déterminer quelles cellules correspondent à du fluide, de l'air ou un solide et modifier la phase d'initialisation. Durant l'initialisation, les cellules qui correspondent à des objets solides ou des murs sont marquées. La génération de huit particules par cellule ne s'effectue que sur les cellules correspondant à du fluide. Au cours de la simulation, à chaque itération, une étape détermine pour chaque cellule si elle correspond à du fluide, de l'air ou un solide. Celles qui ont été marquées comme un solide à l'initialisation correspondent à du solide, celles contenant des particules de fluide correspondent à du fluide et les autres correspondent à de l'air. Après, le calcul de la vélocité à partir des forces par la grille, il faut vérifier si aucune des vélocités des particules fixes de la cellule ne pointent vers une cellule correspondant à un solide. Si tel est le cas, il faut alors projeter la vélocité de telle sorte qu'elle se dirige le long de la cellule solide. Avec une grille *staggered MAC*, comme la vélocité de la grille est répartie dans plusieurs particules fixes, il suffit de mettre à zéro la vélocité des particules se trouvant dans cette cellule solide. Le problème de cette méthode est que les murs doivent être alignés à la grille, pour ne pas subir des artefacts dus à la voxelisation du mur, particulièrement visibles lorsque la résolution de la grille est faible. Diverses méthodes pour mieux prendre en compte la gestion des collisions ont été proposées dans la littérature, cependant aucune d'entre elles ne corrigent ce problème pour tous les cas [80].
Afin de pouvoir gérer des collisions avec des objets non fixes, il faut tout comme pour les simulations SPH, modifier le calcul de la pression, afin de prendre en compte un couplage fluide-solide, en ajoutant des particules fixes sur les objets solides [80, 81].

IV.3.4 La gestion des fluides multiphases

Afin de pouvoir simuler le mélange de deux fluides avec des densités différentes, les méthodes PIC/FLIP doivent déterminer l'interface entre les deux fluides, ce qui est plus précisément réalisé avec la représentation par particules plutôt que par une grille. Hong *et al.* utilise le *levelset*, qui permet de déterminer une isosurface dans un champ de scalaires, afin de calculer les interfaces entres les fluides [82]. Cette surface peut être soit recalculée à chaque itération, ou bien être déplacée pendant la phase d'advection, en ajoutant des particules sur cette interface. L. Boyd *et al.* ajoute également des particules à l'interface, pour conserver une interface lisse en empêchant les particules des deux fluides de se mélanger sans contrôle [79]. Au niveau de l'interface, les calculs doivent se faire indépendamment pour les deux types de fluides, que ce soit pour la partie advection, ou le calcul des forces de pression.

IV.4 Les simulations de fluide SRD

Le modèle de SRD présenté dans le chapitre III présente des lacunes pour pouvoir être utilisé dans le contexte de l'animation et la synthèse de fluides réalistes. La SRD est une modélisation simple du fluide à l'échelle mésoscopique qui diffère du comportement d'un fluide vu à l'échelle macroscopique. En effet, comme nous l'avons vu dans les simulations des chapitres précédents, un fluide n'est jamais au repos à l'échelle mésoscopique, tandis qu'il tend vers un état stable à l'échelle macroscopique. De plus, les simulations à l'échelle mésoscopique représentent des systèmes où les particules de fluide remplissent la quasi totalité de l'espace de simulation. Il n'y a donc pas de simulations SRD dans la littérature où une partie de l'espace est vide (correspondant à un fluide tel que l'air que l'on souhaite éviter de modéliser pour des questions de performances), ni de surface à gérer et donc de tension de surface. Enfin, le point essentiel que la SRD ne prend pas en compte est la gestion de la compressibilité. Le fluide étant représenté par des particules ponctuelles, il n'y a pas de gestion des interpénétrations entre particules de fluide. Malgré ces lacunes, le modèle SRD présente l'avantage d'être très rapide, car il ne nécessite pas de recherche de voisinage. Dans un premier temps, nous avons étudié des cas spécifiques de simulations de fluide, qui ont fait l'objet d'étude dans la littérature d'informatique graphique, pour lesquelles ces lacunes ne présentent pas de problème. Puis nous avons étudié un cas de simulation de fluide plus classique dans le domaine de l'informatique graphique, pour lequel le modèle doit être modifié afin de pouvoir résoudre ces problématiques.

IV.4.1 Les simulations de décomposition spinodale

La première simulation que nous avons étudiée est la décomposition spinodale. C'est un processus de séparation de deux fluides mélangés, qui a déja fait l'objet d'études avec des simulations SPH [83], mais également par des simulations de SRD [84]. A l'état initial, deux types de particules de fluides sont placés aléatoirement dans une boîte de simulation, sur laquelle on applique des conditions périodiques. L'état final est atteint lorsque les deux types de particules forment des régions distinctes. Il est donc nécessaire de pouvoir inclure deux types de particules et de contrôler leur mouvement afin de les séparer. Le modèle de SRD permet d'intégrer plusieurs types de particules, simplement en les différenciant par leur masse. Cependant, cela ne permet pas de pouvoir influer sur leur déplacement afin d'obtenir une séparation en deux phases, ou traiter le cas de deux fluides avec la même masse devant avoir des comportements différents. Il faut donc modifier le modèle de SRD standard.

IV.4.1.1 Le modèle de SRD avec deux types de particules

Pour influer sur le mouvement des particules, l'étape à modifier est celle de la rotation des vélocités des particules. Celle-ci ne doit plus se faire aléatoirement, mais en fonction du type de la particule et de ses particules voisines. Le but est de diriger les deux types de particules dans des directions opposées et vers les cellules contenant majoritairement des particules de sa population. Pour cela nous avons repris le modèle de Rothman *et al* proposé pour la séparation de phase dans le cas de simulations de gaz [85]. Ce modèle ajoute aux particules une couleur notée C_n, avec $C_n = 1$ ou $C_n = -1$ selon leur type. Ce nouvel attribut permet de calculer deux vecteurs pour chaque cellule, le *color field* f et le *color flux* q. le vecteur f représente le gradient lié à la différence de couleur entre une cellule et ses voisines et est défini par :

$$f = \sum_{i}^{N_{cellVoisine}} \frac{c_i}{\mid c_i \mid^2} \Delta_i \tag{IV.12}$$

avec c_i la distance entre le centre de la cellule courante et sa cellule voisine i et Δ_i la somme des couleurs des particules présentes dans la cellule i.
Le vecteur q représente la quantité de mouvement au sein de la cellule courante, pondérée par la couleur, qui est définie par :

$$q = \sum_{n=1}^{N_{particuleInCell}} C_n(v - v_{cm}) \tag{IV.13}$$

avec v la vélocité de la particule n et v_{cm} la vélocité du centre de masse des particules de la cellule courante.
Ces deux vecteurs permettent de déterminer la matrice de rotation R, permettant de pivoter le vecteur q dans la direction de f tel que :

$$\frac{f}{|\ f\ |} = R \cdot \frac{q}{|\ q\ |} \tag{IV.14}$$

Ainsi R est la matrice de rotation avec pour angle celui entre f et q et l'axe de rotation est la normale à la surface contenant q et f.

IV.4.1.2 L'implémentation GPU

La modification du modèle de SRD nécessite de remplacer le noyau calculant un axe de rotation aléatoire pour chaque cellule, par un nouveau noyau. Dans le cas d'une cellule contenant les deux types de particules, ce nouveau noyau détermine un angle et un axe de rotation en fonction de q et f. Dans les autres cas, l'axe se calcule aléatoirement comme précédemment, et l'angle est alors celui fixé en paramètre. Avant de pouvoir exécuter cette étape, il faut auparavant calculer f et q pour chaque cellule, ainsi que toutes les données intermédiaires Δ_i et c_i nécessaires au calcul de ces deux vecteurs. c_i étant constante au cours du temps, elle peut être précalculée. Pour les Δ_i, il suffit de réinitialiser leur valeur au début de l'étape de collision, puis de parcourir toutes les particules afin d'ajouter leur C_n au Δ_i associé à leur cellule par opérations atomiques. Ce calcul peut être ajouté au noyau calculant v_{cm}. Le calcul de q pour chaque cellule, nécessite un nouveau noyau, s'exécutant après avoir calculé v_{cm}. Ce noyau parcourt chaque particule et ajoute par opération atomique $C_n(v - v_{cm})$ au q de sa cellule. Enfin, f est calculé dans un noyau parcourant toutes les cellules i et ses voisines. Tout cela est résumé dans l'algorithme 9.

IV.4.1.3 Résultats

La figure IV.1 montre les résultats de cette simulation de décomposition spinodale avec 800k particules avec un pas de temps $\Delta t = 4$, une vitesse initiale des particules tirée aléatoirement à partir d'une variance $\sigma = 1$, une taille de cellule de SRD $a_0 = 20$ et un nombre moyen de particules par cellule $\gamma = 10$. Les paramètres Δt et σ influent tous deux sur la distance parcourue par les particules durant une itération. Comme il n'y a que des phases de SRD ces deux paramètres sont interchangeables, et c'est donc la valeur $\Delta t \times \sigma$ qui importe : plus elle est élevée, plus les particules se déplacent vite et plus la décomposition se fait rapidement, car moins l'interface entre les deux populations est stable. a_0 est à mettre en rapport avec Δt et σ, pour déterminer à quelle vitesse les particules passent d'une cellule de SRD à l'autre. Ainsi, son influence est l'inverse de celle des deux précédents

paramètres : plus a_0 est petit, plus la décomposition se fait rapidement. Enfin γ influe sur la précision du contrôle des rotations. Différentes configurations finales sont possibles, selon les paramètres, mais également selon les positions initiales des particules. Une interface séparant les deux groupes bien plane est la configuration finale la plus courante (voir la figure IV.1). Cependant, le système peut également trouver un état stable, avec plusieurs groupes distincts d'une même population. En effet, si aucune particule d'un même type

Algorithm 9 étape de collision de SRD modifiée pour une séparation de phase

//étape de translation globale
calcule un vecteur de translation aléatoire v_{trans}
for all particule p_i **do** en parallèle
    $p_{Shift_i} = translation(p_i, vtrans)$
//étape du calcul de la vélocité du centre de masse
remise à zéro de Δ_i
//étape de la somme des vélocités
for all particule p_i **do** en parallèle
    //translation des particules
    //ajoute la vélocité et incrémente le compteur avec des opérations atomiques
    détermine $cell_i$, la cellule de SRD contenant p_{Shift_i}
    incrémente avec une opération atomique $counter_i$ associé à $cell_i$
    calcule la somme $sumVeloc_{cell_i} += v_{Shift}$ par opérations atomiques
    calcule Δ_i $+= C_n$ par opération atomique
//application de la division
for all cellule de SRD $cell_i$ **do** en parallèle
    calcule v_{cm_i} de $cell_i$, à partir de $sumVeloc_{cell_i}$ et $counter_i$
//étape du calcul du color flux
remise à zéro des q
for all particule p_i **do** in parallel
    calcule le color flux $q_i += C_n(v - v_{cm})$ par opérations atomiques associé $cell_i$,
//étape de rotation des vélocités
remise à zéro des f
for all cellule $cell_i$ **do** en parallèle
    if $nb_{typeFluidInCell} = 2$. **then**
        for all cellule $cell_j$ voisine à i **do** en parallèle
            calcule $f_i += \frac{c_i}{|c_i|^2}\Delta_j$ associé à $cell_i$
        calcule $angleRotation_i = angleEntre(f, q)$ associé à $cell_i$
        calcule $axeRotation_i = normal(f, q)$ associé à $cell_i$
    else
        calcule $axeRotation_i$ associé à $cell_i$
        affecte $angleRotation_i = defaultAngle$ associé à $cell_i$
for all particule p_i **do** en parallèle
    applique la rotation sur la vélocité $veloc_i$ de p_i, à partir de $rotationAxis_i$ et la valeur v_{cm_i} associée à la cellule contenant p_i

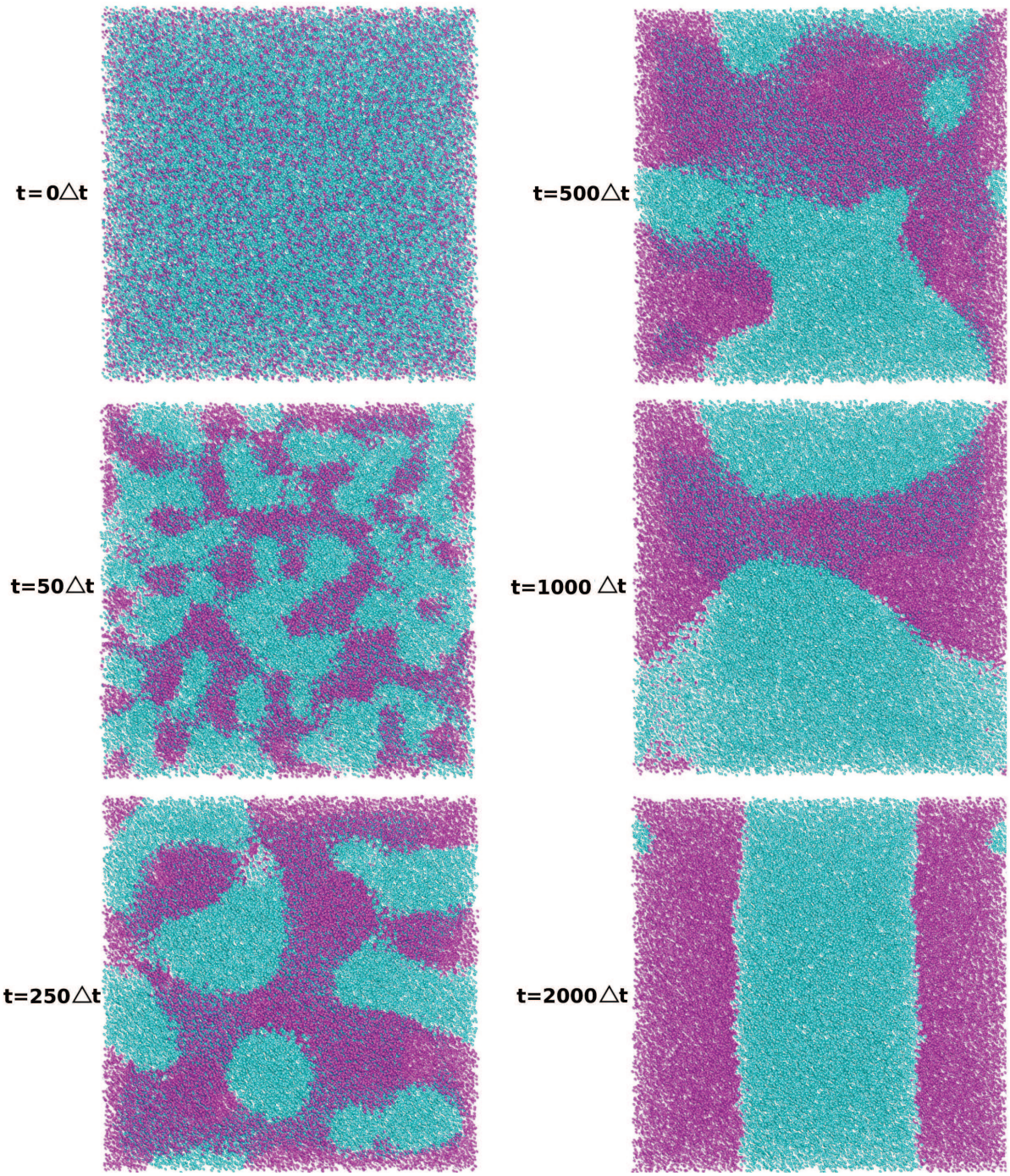


FIGURE IV.1 – Captures d'écran d'une simulation de décomposition spinodale avec 800k particules, $\Delta t = 4$, $\sigma = 1$, $a_0 = 20$ et $\gamma = 10$. A l'état initial, les deux populations de particules sont réparties aléatoirement. Au cours de la simulation, des groupes d'une même population se forment et se rejoignent, jusqu'à parvenir à un état où les groupes ne peuvent plus converger et restent stables.

ne se trouve dans les cellules voisines d'un groupe, il n'y a aucune force qui tend à faire migrer ce groupe vers un autre du même type. De plus, les configurations où il ne reste plus que deux groupes peuvent prendre différentes formes, avec des interfaces formant des

surfaces courbes plus où moins complexes. Les paramètres réduisant le déplacement des particules en dehors de leur cellule de SRD favorisent ces dernières configurations.

IV.4.2 Les simulations d'une déformation de bulle d'air

La deuxième simulation que nous avons étudiée, est la déformation d'une bulle d'air remontant vers la surface. Le comportement des bulles dépend de leur taille. Les petites ont une forme quasi-sphérique et leur trajectoire est droite. Les plus grosses ont une forme ellipsoïdale, arrondie sur leur partie haute et plus aplatie voire creuse sur leur partie basse, à cause de la pression plus faible dans leur zone de sillage. De plus, elles ont une trajectoire en spirale ou en zigzag [86]. Ce sont les grosses bulles que nous cherchons à simuler, dont le mouvement asymétrique a déjà fait l'objet de simulations basées sur des méthodes avec des grilles eulériennes ou SPH [87]. Cette simulation est assez similaire à celle de la décomposition spinodale. La bulle d'air et l'eau se distinguent par une couleur C_n, initialisée au début de la simulation à $C_n = -1$ pour les particules présentes dans la bulle et $C_n = 1$ pour les autres. Pour que la bulle d'air ne se disloque pas dans l'eau, nous avons utilisé la même méthode de séparation de phase que pour la simulation précédente. Deux vecteurs q et f sont calculés afin de déterminer un axe et un angle de rotation pour chaque cellule de SRD. De plus, la boîte de simulation est deux fois plus longue dans l'axe y qu'en x et z. Les conditions périodiques ne s'appliquent que sur l'axe y. En x et en z, un mur est placé. La gestion des collisions entre un mur et une particule s'effectue en replaçant les particules ayant pénétrées le mur sur sa surface et en multipliant par -1 la coordonnée du vecteur vitesse correspondant à l'axe normal au mur. Une force extérieure qui ne s'applique qu'aux particules d'air permet de faire remonter la bulle.

La figure IV.2 montre les résultats de cette simulation de bulle d'air avec 200k particules, un pas de temps $\Delta t = 1$, une vitesse initiale des particules tirée aléatoirement à partir d'une variance $\sigma = 0.6$, une taille de cellule de SRD $a_0 = 20$, un nombre moyen de particules par cellule de $\gamma = 20$, le diamètre de la bulle d'air $a_{bulle} = 135$ et la force verticale $f_{air} = 0.08$. Les choix des paramètres sont primordiaux pour obtenir le comportement attendu. f_{air}, σ et Δt affectent la distance parcourue par les particules du système durant une itération : f_{air} influe sur le mouvement vertical des particules d'air, σ sur les déplacements liés aux rotations de toutes les particules et Δt influe sur ces deux paramètres. Plus la bulle remonte vite par rapport au déplacement lié aux rotations, plus un mouvement de spirale est observé. En effet, cette remontée rapproche les particules d'air et d'eau qui se repoussent alors, faisant osciller la bulle. Cependant, quand la bulle remonte trop vite, une traînée de particules d'air au dessous est observée. Ce sont des particules qui finissent par se désolidariser du reste de la bulle. Si la bulle remonte très lentement, la bulle reste quasi sphérique, ne créant plus de différence de pression entre

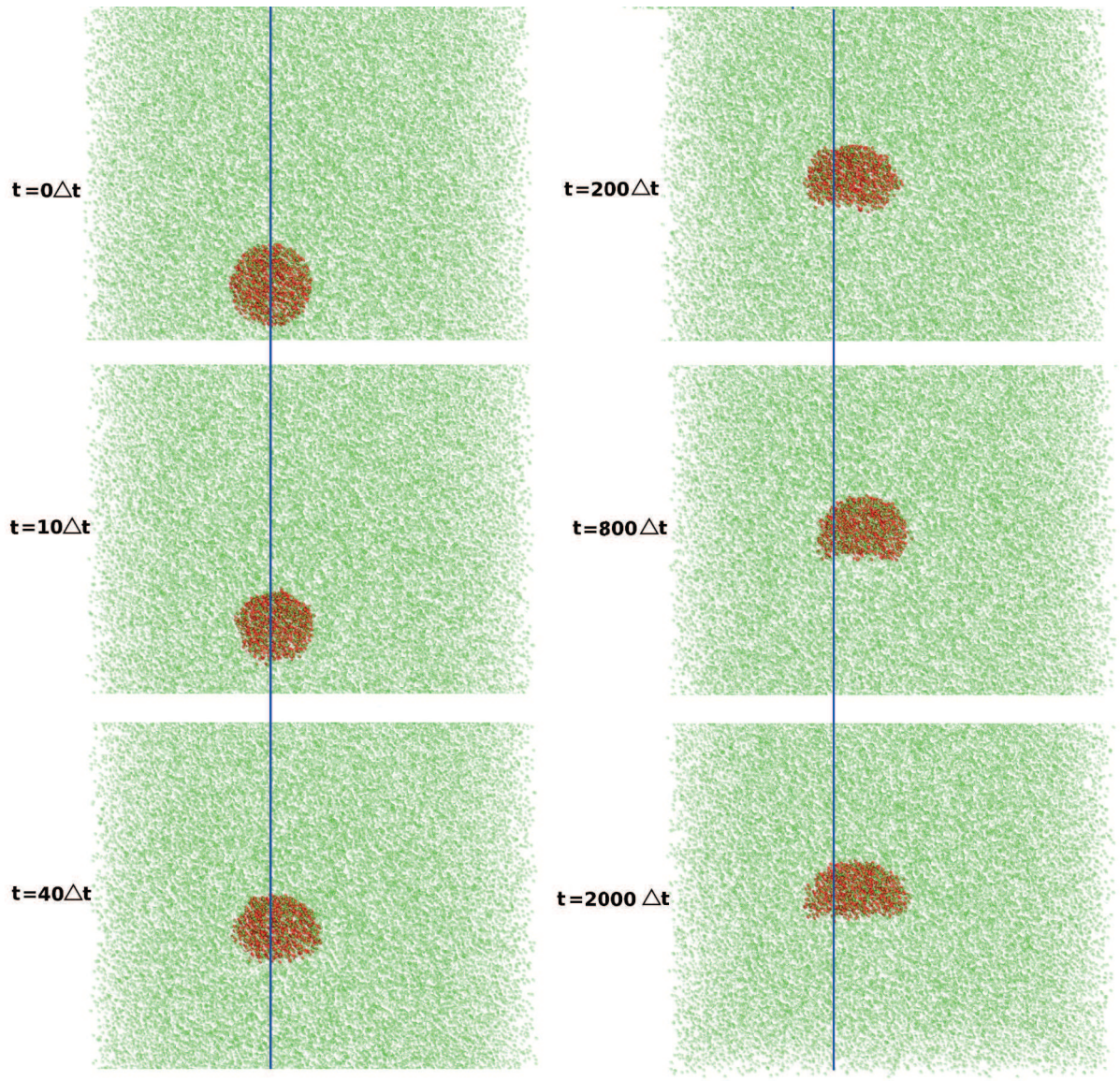


FIGURE IV.2 – Captures d'écran d'une simulation de bulle d'air avec 200k particules, $\Delta t = 1$, $\sigma = 0.6$, $a_0 = 20$, $\gamma = 20$, $a_{bulle} = 135$ et $f_{air} = 0.08$. A l'état initial, une bulle d'air est placée en bas d'une boîte de simulation. Au début de la simulation, la bulle est sphérique, à cause du placement aléatoire des particules dans une sphère. Durant les premières itérations, la bulle se contracte rapidement, à cause de la méthode de séparation de phase. Puis, on peut voir que la bulle d'air se déforme peu à peu, avec un côté arrondi au dessus et creux en bas. De plus, la bulle a un léger mouvement d'oscillation lors de sa remontée et a tendance à dériver plus sur un côté, ce qu'on peut constater en regardant la variation du centre de la bulle par rapport à la ligne bleue (la verticale passant par son centre, à l'état initial de la simulation).

ses parties haute et basse.

Plus γ est grand, moins les particules d'air se désolidarisent du reste de la bulle, les rotations séparant l'air de l'eau devenant plus précises. La taille de la bulle influe sur

la forme de celle-ci. Pour des bulles de petites tailles ($a_{bulle} = 85$), la forme sphérique est à peu près conservée. Cela s'explique par le faible mouvement qu'elles engendrent au niveau de l'eau, ne créant pas de courant la faisant remonter rapidement (voir La figure IV.3b). Les bulles de tailles intermédiaires ($a_{bulle} = 135$) atteignent rapidement la forme sphérique au-dessus et creuse en dessous, une fois leur vitesse maximale atteinte, suite au mouvement sur l'eau qu'elles engendre en remontant. Les bulles de grandes tailles ($a_{bulle} = 205$) quant à elle, n'atteignent cette forme qu'après une phase où elles se retrouvent très étirées verticalement (voir La figure IV.3a).

L'objectif ici était de pouvoir reproduire le comportement de la bulle d'air. Cependant, celui de l'eau ne semble pas correspondre à un fluide à l'échelle macroscopique. En effet, les particules d'eau se déplacent de manière aléatoire dès le départ au lieu d'être à peu près stables, avant même que la bulle d'air n'engendre un mouvement de l'eau.

Ces simulations montrent les possibilités du modèle de SRD, qui permet de calculer des informations sur le voisinage d'une particule, sans effectuer de recherche de voisinage. Dans la simulation de décomposition spinodale, cela a permis de déterminer où diriger les particules vers celle de leur type, grâce à des données associées à leur cellule de SRD et ses voisines, calculées grâce à des opérations atomiques. Ce type de modèle ressemble donc à un modèle PIC/FLIC où les particules peuvent se déplacer d'une cellule à l'autre sans contrainte. Cependant, ces simulations ne correspondent pas aux cas les plus communs étudiés en animation et synthèse d'images de fluide. Le mouvement aléatoire des particules d'eau ne correspond pas au comportement d'un fluide à l'échelle macroscopique. Nous allons dans le prochain paragraphe essayer d'adapter la SRD pour le cas d'une simulation standard.

IV.4.3 Les simulations d'une rupture de barrage

Afin de pouvoir utiliser ce type de simulation dans le cadre d'animation et synthèse d'images de fluides réalistes, il est nécessaire de pouvoir sortir du cas où les fluides emplissent tout l'espace de simulation. Cela reviendrait dans la simulation précédente à ajouter une surface à la place de conditions périodiques. La bulle d'air en remontant vers la surface devrait alors la déformer. Cela nécessite de gérer une interface eau-air pour simuler une tension de surface, ainsi qu'une force de gravité pour que l'eau se retrouve en dessous, et l'air au dessus. La gravité et l'ajout d'une surface impliquent qu'il est également nécessaire de gérer la compressibilité du fluide. Avec le modèle actuel, la gravité impliquerait que toutes les particules d'eau se retrouveraient au fond, et donc que le volume d'eau tendrait à devenir nul. Le souci est que rien ne permet de conserver une distance de repos entre deux particules de fluide. Nous allons décrire dans la partie suivante

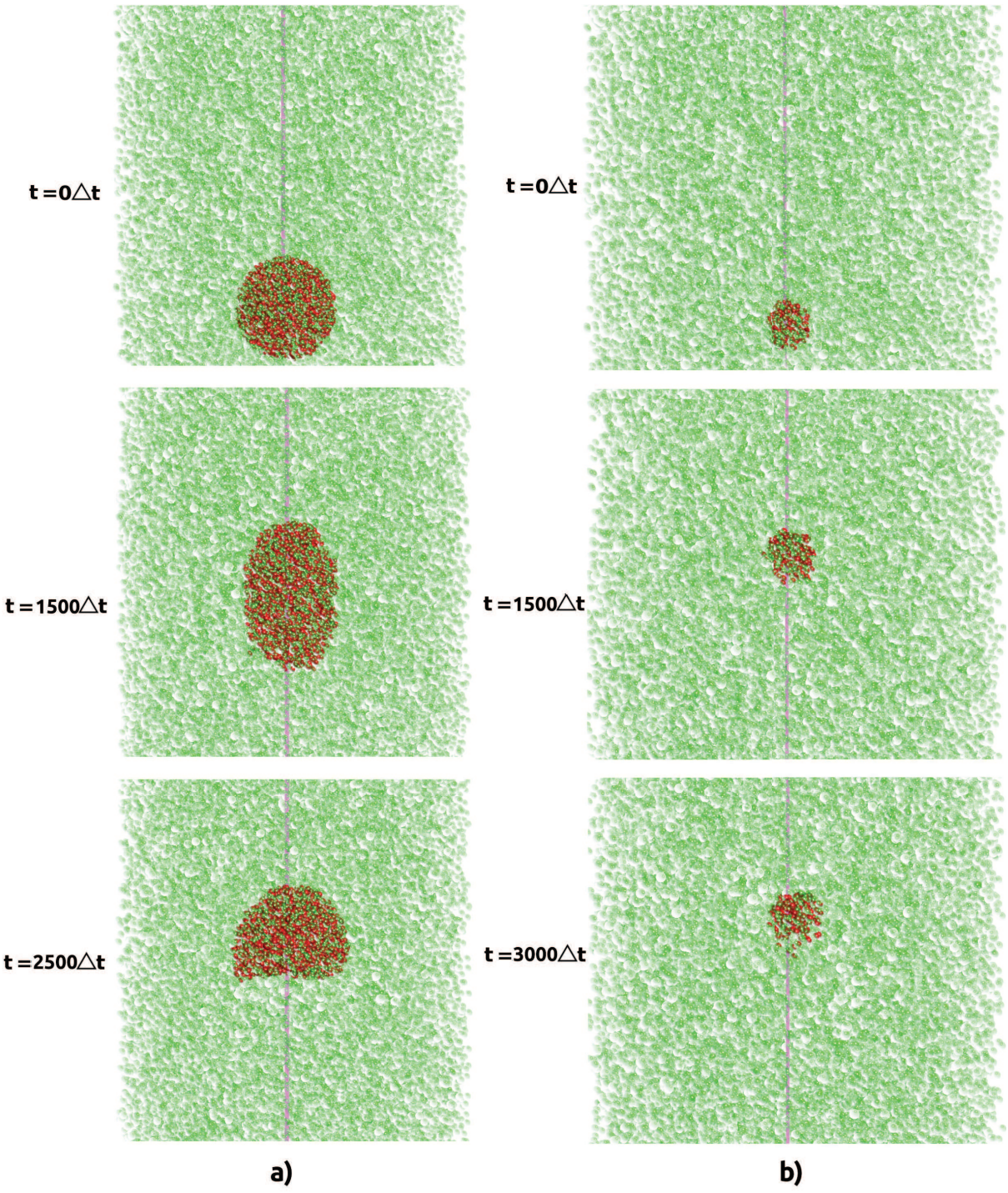


FIGURE IV.3 – Captures d'écran de deux simulations de bulle d'air avec 200k particules, $\Delta t = 1$, $\sigma = 0.6$, $a_0 = 20$, $\gamma = 20$ et $f_{air} = 0.08$. La taille de la bulle est $a_{bulle} = 205$ pour la simulation avec une grande bulle (a) et $a_{bulle} = 85$ pour celle avec une petite bulle (b). Pour la simulation avec une grande bulle (gauche), on constate que la bulle passe par une phase avec une forme allongée verticalement avant de se comporter comme la bulle de taille intermédiaire, avec une forme arrondie au dessus et aplatie en dessous et aura ensuite alors un léger mouvement en spirale à partir de là. Dans la simulation avec la petite bulle (droite), celle ci conserve un forme à peu près sphérique, en remontant avec une trajectoire droite.

les modifications à apporter au modèle de SRD afin de pouvoir simuler ces phénomènes, dans le cadre d'une simulation d'une rupture de barrage. Dans cette simulation, les particules d'eau se trouvent à l'état initial à gauche de la boîte de simulation. Elles chutent alors sous l'effet de la gravité et forment une vague déferlante pour peu à peu trouver un état stable. Cette simulation a été réalisée par Müller *et al.* pour introduire le modèle SPH, dans les démonstrations du modèle SPH [55]

IV.4.3.1 La gestion de la compressibilité

Afin de pouvoir contrôler la distance entre les particules de fluide, nous avons ajouté des forces de pression au modèle de SRD. Le but est d'éviter que deux particules de fluide ne puissent s'interpénétrer, et que les particules de fluide tendent à atteindre un état de repos stable. Diverses manières de calculer la pression ont été testées. Celle donnant le comportement le plus proche d'une simulation SPH, est une adaptation de celle proposée par Clavet *et al.* [88]. Cette approche contrôle la compressibilité en repoussant ou attirant les particules voisines d'une particule i selon sa densité locale ρ_i et la densité au repos visée ρ_0 à partir d'une pression P_i. Cependant, ce processus ne suffit pas à éviter des regroupements de particules. En effet, il est possible qu'une particule atteigne ρ_0 en attirant fortement une petite partie des particules voisines et en repoussant d'autres fortement. Ainsi, on obtiendrait un ensemble de petits groupes de particules, sans cohérence. Pour éviter ce problème Clavet et *al.* propose d'ajouter un second terme à la pression, nommé *pression proche*, que nous avons utilisé pour éviter ce même problème [88]. Cette nouvelle force dépend de la distance entre les deux particules, mais également d'un nouveau terme de densité, nommé *densité proche*, défini par :

$$\rho_i^{proche} = \sum_{j \in N(i)} (1 - r_{ij}/h)^3 \tag{IV.15}$$

où $N(i)$ est l'ensemble des voisins de la particule i défini par un noyau d'interpolation en pic de rayon h. La force de pression qui en résulte est définie par :

$$P_i^{proche} = k^{proche} \rho_i^{proche} \tag{IV.16}$$

où k^{proche} est la constante de raideur. La pression proche et la pression permettent alors de calculer le vecteur de déplacement D_{ij} à appliquer à chaque paire de particules selon l'expression :

$$D_{ij} = \Delta t^2 (P_i(1 - r_{ij}/h) + P_i^{proche}(1 - r_{ij}/h)^2) r_{ij} \tag{IV.17}$$

Ce vecteur de déplacement s'applique à la particule voisine j et le déplacement opposé est appliqué à la particule i. Seule la partie basée sur la pression P_i^{proche}, qui permet

Algorithm 10 Calcul et application de la pression proche

1: *//calcul de* P_{near}
2: **for all** particule i **do**
3: $\rho_i^{near} = 0$
4: **for all** particule j voisine de i **do**
5: q $= r_{ij}/cellSize$
6: **if** $q < 1.$ **then**
7: $\rho_i^{proche} += (1. - q)^3$ (équation (IV.15))
$P_i^{near} = \rho_i^{proche} k_{near}$ (équation (IV.16))
8: *// application de la pression*
9: $dx = 0$
10: **for all** particule j voisine de i **do**
11: q $= r_{ij}/cellSize$
12: **if** $q < 1.$ **then**
13: $\overrightarrow{D} = \Delta t^2 P_i^{proche}(1 - q)^2 \overrightarrow{r_{ij}}$ (équation (IV.18))
14: $pos_j = pos_j + \overrightarrow{D}/2$ (opération atomique)
15: $dx = dx - \overrightarrow{D}/2$
16: $pos_i = pos_i + dx$

d'éviter que des particules soit trop proches, nous intéresse et a été adaptée dans le modèle SRD. Il est alors nécessaire de modifier le calcul de déplacement pour n'utiliser que P_i^{proche} ce qui donne :

$$D_{ij} = \Delta t^2 P_i^{proche}(1 - r_{ij}/h)^2 r_{ij} \tag{IV.18}$$

L'algorithme 10 est l'adaptation de la méthode de pression proche pour notre simulation SRD. Cependant cet algorithme n'est pas optimisé pour une implémentation GPU. En effet, l'application des vecteurs de déplacements D_{ji} à toutes les particules voisines de i nécessite de nombreuses opérations atomiques. Elles peuvent être évitées en séparant l'algorithme en deux kernels. Le premier calcule P_i^{proche} et la stocke en mémoire globale. Dans le second, chaque thread va pouvoir alors calculer la somme de tous les vecteurs de déplacement d'une particule i à partir de toutes les pressions proches P_j^{proche} et P_i^{proche}. Ce deuxième kernel est résumé par l'algorithme 11.

Le calcul de la pression et le déplacement des particules de fluide qui en résulte s'ajoutent aux phases de rotation des vélocités comme pour le cas de la SRD-MD. Ainsi, une itération est composée de n étapes de calcul de pression suivies d'une étape de rotation. Cela devrait permettre de diminuer l'effet de diffusion du fluide présent dans les précédentes simulations. En effet, la pression permet de faire tendre le système vers un état stable en diminuant progressivement la vélocité des particules. La rotation ne faisant que tourner ces vélocités, elle n'empêche donc pas la stabilité des particules une fois celles-ci au repos.

Algorithm 11 Algorithme modifié de l'application de la pression proche

1: **for all** particule i **do**
2: $dx = 0$
3: **for all** particule j voisine de i **do**
4: q $= r_{ij}/cellSize$
5: **if** $q < 1.$ **then**
6: $\overrightarrow{D_{ij}} = \Delta t^2 P_i^{proche}(1-q)^2\overrightarrow{r_{ij}}$
7: $\overrightarrow{D_{ji}} = \Delta t^2 P_j^{proche}(1-q)^2\overrightarrow{r_{ji}}$
8: $pos_i = pos_i - \overrightarrow{D_{ij}}/2 + \overrightarrow{D_{ji}}/2$

IV.4.3.2 La gestion de la tension de surface

Afin de pouvoir simuler la tension de surface, nous avons réutilisé les méthodes de séparation de phase que nous avons utilisé pour la décomposition spinodale. La tension de surface provient de l'interaction entre les particules d'air et d'eau. Cependant, nous ne voulons pas modéliser explicitement les particules d'air, pour une question de performance. Ainsi, l'idée est de reprendre le modèle de deux types de particules, sans avoir à modéliser explicitement le deuxième type de particules qu'est l'air. Pour cela, nous allons considérer que les cellules de SRD ayant un nombre de particules d'eau N_e inférieur à la moyenne γ, contient le nombre de particules d'air N_a tel que $N_e + N_a = \gamma$. Il est alors possible de calculer l'angle et l'axe de rotation de la même manière que pour la simulation de décomposition spinodale.

IV.4.3.3 Résultats

La figure IV.4 montre les résultats d'une rupture de barrage avec 200k particules, $\Delta t = 0.3$, $\sigma = 1$, $a_0 = 20$, le nombre moyen de particules d'eau dans les cellules non vides $\gamma = 10$, la force de gravité $g = 4.$, le rayon d'interpolation $h = 32$ et $k^{proche} = 2.9$. g et k^{proche} sont les paramètres qui influent sur la formation de la vague engendrée par la chute. g donne de la vitesse aux particules d'eau qui suite à leur chute, vont se retrouver compressées. k^{proche} détermine alors la valeur de la force de pression qui va les faire remonter de l'autre côté la boîte de simulation. Le rapport de ces deux paramètres détermine également la compressibilité du fluide. Plus g est élevé par rapport à k^{proche}, plus les particules de fluide sont proches les unes des autres lorsqu'elles atteignent un état stable, tandis que plus k^{proche} est grand face à g, plus les particules sont éloignées au repos. Ainsi, le paramètre γ indique uniquement le nombre de particule moyen par cellule à l'initialisation.
Le principal problème est le retour du fluide à un état stable. En effet, dans notre simulation, les particules d'eau restent toujours en mouvement à cause de la force de pression P^{proche}, qui ne permet pas aux particules de fluide de trouver un état au repos.

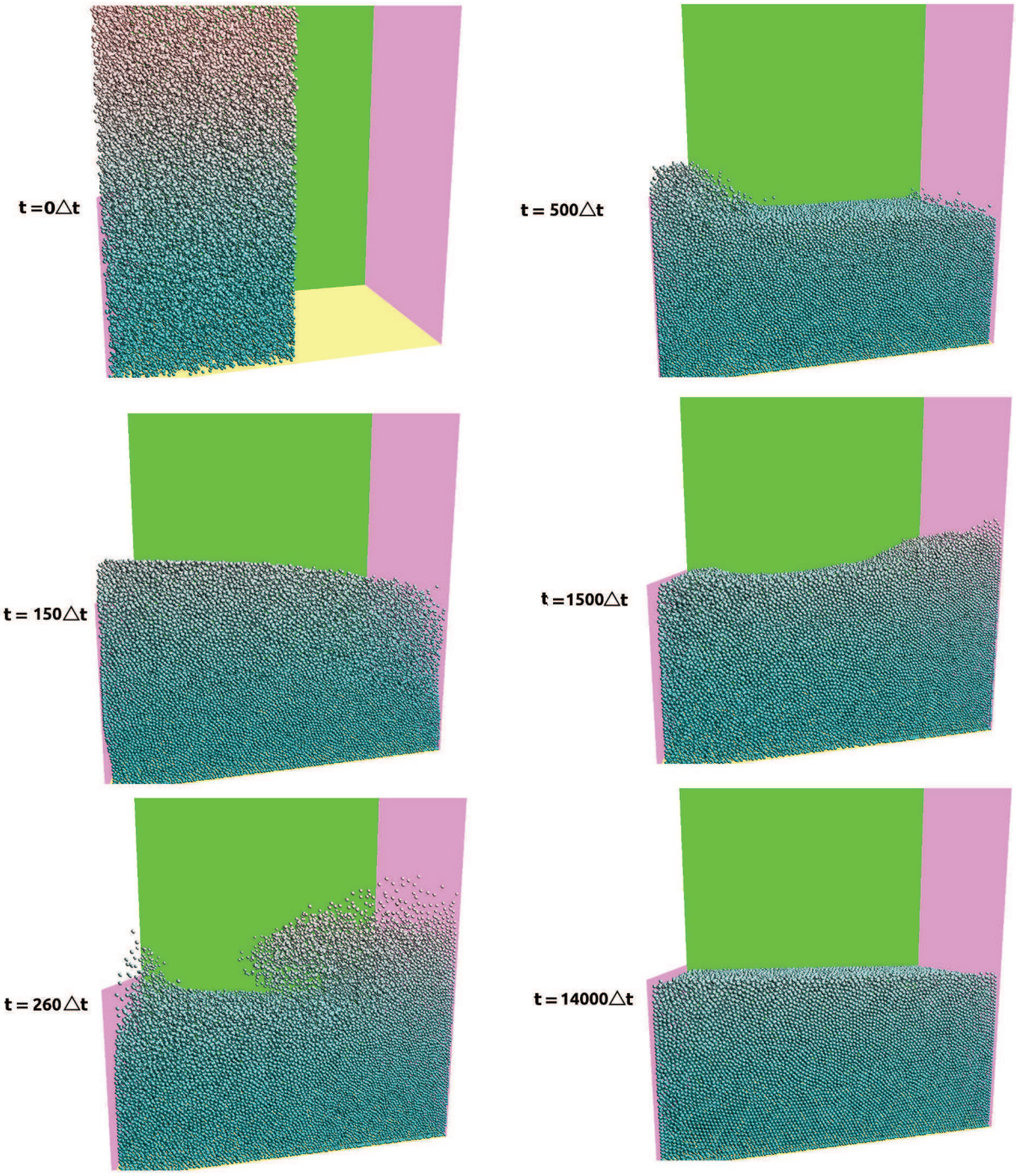


FIGURE IV.4 – Captures d'écran de simulations d'une simulation d'une rupture de barrage avec 200k particules, $\Delta t = 0.3$, $\sigma = 1$, $a_0 = 20$, $g = 4.$, $\gamma = 10$, $h = 32$ et $k^{proche} = 2.9$. A l'état initial, les particules d'eau sont placées sur la moitié gauche de la boîte de simulation, puis tombent en chute libre. Au cours de leur chute, les particules d'eau atteignent le bord de droite de la boîte de simulation, et sous l'effet de la pression, remonte pour former une vague qui retourne vers le bord de gauche.

Le second est la vitesse de la simulation, qui n'arrive à produire un résultat cohérent que pour un pas de temps faible ce qui ne permet pas de réaliser cette simulation pour 200k particules en temps réel. Cette simulation est encore en cours d'étude afin d'améliorer les résultats sur ces deux points. L'idée est de modifier le système de pression afin que les particules de fluide arrivent à retrouver un état de repos où les particules s'immobilisent. De plus, ce calcul de pression nécessite une recherche de voisinage, qui fait perdre l'intérêt à l'utilisation du modèle de SRD, rendant l'algorithme de notre simulation aussi complexe qu'une simulation SPH. Ainsi, l'idée serait ensuite d'adapter le calcul de pression afin qu'elle ne s'applique plus à une particule, mais à une cellule de SRD par des calculs de données se faisant par les cellules et ses voisines, comme pour les simulations de décomposition spinodale et de la bulle d'air.

Chapitre V :

Conclusion et perspectives

Les travaux de recherche de cette thèse ont poursuivi deux objectifs. Le principal a été l'étude de modèles de dynamique moléculaire et leur optimisation sur GPU. La seconde a été leur adaptation dans le domaine de l'animation et la synthèse de fluides réalistes.

Le premier projet a été la parallélisation d'une simulation de dynamique brownienne (BD) déjà existante en version séquentielle pour une suspension colloïdale simple. La BD est un modèle simple ne représentant pas les molécules de fluide, mais simplement leur effet par des forces de friction et des forces aléatoires. Ces travaux ont abouti à une amélioration de la méthode de recherche de voisinage qui se base sur une combinaison entre une grille régulière et des listes de Verlet. La grille régulière permet de réactualiser le voisinage élargi, en utilisant la méthode du tri par base associée à un z-index, pour le stocker dans des listes de Verlet. L'amélioration vient de la manière de détecter lorsqu'il est nécessaire de réactualiser les voisinages, qui introduit un test simple sur les déplacements des particules depuis la dernière réactualisation. Cette méthode permet un gain de 15% sur les performances, comparées à celles de la littérature. Au cours de cette thèse, cette méthode a été réutilisée dans des simulations de suspensions colloïdales plus complexes.
Ce premier travail a été exploité pour l'étude d'une hétéroagrégation à l'interface de deux suspensions, ce qui a fait l'objet d'une publication dans le journal PCCP [32]. Elle ajoute un autre type de particules, possédant une charge opposée, et des murs sur l'axe z à la place de conditions périodiques. Ces travaux ont eu pour but d'analyser les paramètres favorisant la cristallisation des particules et montrent ce qu'apporte la parallélisation GPU pour ce type d'étude. L'utilisation du GPU a permis de réaliser des simulations d'une durée inatteignable avec les versions séquentielles.
La recherche de voisinage a ensuite été adaptée pour l'étude de l'hétéroagrégation entre deux populations de particules de silice et d'alumine, qui ont une taille très différente. Pour avoir des performances convenables, elle nécessite d'utiliser trois grilles et trois listes de Verlet pour les différentes interactions, avec la troisième liste permettant de calculer à la fois les interactions silice-alumine et alumine-silice. De plus, le grand rapport en taille entre les particules requiert de modifier le modèle de BD classique, qui tend pour ce système à immobiliser les grandes particules lorsqu'elles sont entourées de petites. Quand de nombreuses petites particules sont adsorbées par une plus grande, formant un complexe, le coefficient de diffusion de ce complexe devient sous évalué à cause de l'accumulation des forces de friction. La solution proposée pour palier ce problème a été de décomposer le mouvement des particules qui forment un complexe silice-alumine en deux étapes. La première déplace l'ensemble du complexe comme s'il ne formait qu'une seule particule. La seconde réorganise la position des silices. Cette amélioration a été présentée dans la conférence VRIPHYS 2015 [18]. Enfin, dans une collaboration avec l'Université de Gênes, notre méthode de recherche de voisinage a pu être employée dans un contexte différent, celui des simulations atomiques de nanoalliages. Cela montre qu'elle peut s'adapter à

toutes les simulations de type MD, où les voisinages peuvent être conservés durant de nombreuses itérations. Les simulations ont été portées sur GPU, permettant d'étudier dans le cas des nanoalliages, des systèmes avec un nombre d'atomes plus important en un temps plus court.

Dans un deuxième temps, le modèle de SRD-MD a été étudié. C'est un modèle plus complexe, qui représente explicitement les molécules de fluide par des particules qui ont une dynamique très simple, basée sur des rotations aléatoires identiques pour les particules d'une même cellule de SRD. Le nombre de particules de fluide étant très important, la parallélisation GPU d'une simulation de SRD-MD est primordiale. Il a déjà fait l'objet de publications dans la littérature pour des types de SRD-MD simples. Notre étude s'est donc tournée vers le modèle de SRD-MD avec force de couplage. C'est une variante du modèle plus complexe, qui ajoute des interactions entre les particules de fluide et les colloïdes afin d'empêcher leur interpénétration avec les colloïdes. La recherche de voisinage s'adapte de la même manière que pour la simulation avec des particules à deux tailles très différentes, et la partie SRD du modèle a été implémentée avec l'approche utilisée par Chen *et al.* [2]. Une nouvelle optimisation majeure a été développée dans la manière de calculer les forces entre les particules de fluide et de colloïdes. Cela provient du ratio bien plus important entre les deux types de particules que dans les simulations précédentes. Cette configuration favorise l'emploi d'un nouveau schéma de décomposition par bloc, associant tous les calculs des forces d'interaction d'un colloïde à un bloc de threads. Par rapport à la décomposition standard associant un thread par colloïde, cette stratégie permet d'éviter les problèmes de branches divergentes, ce qui permet un gain en performance de 45% sur cette étape de notre simulation. Ces travaux ont donné lieu à une publication soumise dans le journal CPC [48].

Enfin, le troisième projet a consisté en l'adaptation du modèle SRD dans le cadre de l'animation et la synthèse d'images réalistes. Premièrement, deux simulations simples ont été réalisées en ajoutant au modèle de SRD, la gestion de deux types de particules. Les deux types de particules doivent tendre à se repousser l'un l'autre pour rejoindre les autres particules de leur type. Pour cela, l'algorithme utilise les cellules de SRD afin de calculer des informations en relation avec les types de particules contenus par la cellule de SRD et ses voisines, sans avoir à faire de calcul de voisinage. Ces informations vont permettre d'appliquer une rotation aux particules de fluide, non plus aléatoire, mais dirigeant les deux types de fluides vers des directions opposées et vers les autres particules de leur type. Ceci permet de réaliser une simulation de décomposition spinodale, où à partir d'une configuration aléatoire de deux types de particules dans la boîte de simulation, le système tend vers un état où les particules identiques se regroupent et forment des phases distinctes stables. Elle permet aussi de simuler la remontée d'une bulle d'air dans de l'eau, en ajoutant une force verticale ne s'appliquant qu'aux particules d'air. La bulle d'air se

déforme alors avec une partie sphérique au dessus, et creuse en dessous, en se déplaçant verticalement en spirale, ce qui correspond au comportement des bulles de grandes tailles. Afin de réaliser des simulations plus standard du domaine, telles qu'une rupture de barrage, il faut ajouter au modèle de SRD, en plus de la gravité, des notions telles que la gestion de la compressibilité et la tension de surface. La tension de surface est obtenue en modélisant implicitement l'air, en considérant que toutes les cellules de SRD contiennent au moins le nombre moyen de particules γ. Les cellules ayant un nombre de particules d'eau inférieur contiennent alors implicitement des particules d'air. Pour la compressibilité, il a fallu ajouter des forces de pression et donc une recherche de voisinage. Plusieurs méthodes de calculs de pression ont été testées, et la méthode de Clavet *et al.* de *pression proche* [88] a donné les meilleurs résultats. Ces ajouts permettent d'obtenir dans la simulation, une vague qui se forme après la chute des particules. Cependant, la simulation s'avère lente à cause d'un pas de temps faible nécessaire pour avoir un comportement correct et à cause du coût de la recherche de voisinage pour le calcul de pression. De plus, le système ne parvient pas à trouver un état stable, à cause des forces de pression.

Ces travaux sur l'adaptation du modèle SRD pour des simulations de fluide en informatique graphique sont toujours en cours. L'objectif est de modifier le calcul de la pression afin de pouvoir augmenter le pas de temps pour obtenir une simulation temps réel avec de grands systèmes. Par la suite, le but serait d'effectuer ce calcul de pression sans passer par les particules, mais uniquement par le biais des cellules de SRD, afin d'éviter une recherche de voisinage. Le but est de se servir uniquement de l'étape de rotation pour appliquer la pression. En effet, cette étape perd de son intérêt avec notre méthode actuelle, car les changements de direction des particules d'eau sont bien plus liés aux étapes de calcul de pression, qu'aux phases de rotation. L'idée avec le calcul de pression locale à une cellule, est d'utiliser les étapes de rotation afin de diriger les particules d'eau vers les cellules de basse pression, avec un coefficient multipliant leur vitesse selon la force de pression de la cellule. Ainsi, une cellule de SRD pourrait être vue comme l'équivalent d'une cellule d'une grille eulérienne, et les particules de fluide permettraient le calcul des données des cellules ainsi que la phase d'advection, comme dans les modèles hybrides de type PIC/FLIP.

Du côté du domaine des suspensions colloïdales, les prochaines études concernent des simulations avec des colloïdes non sphériques et anisotropes (de type plaquette par exemple), qui peuvent être modélisés par un ensemble de particules liées entre elles. La parallélisation est alors primordiale, afin d'avoir des systèmes de grande taille et d'appliquer des calculs plus complexes tels que la rotation de ces colloïdes. De plus, la collaboration avec l'Université de Gênes est toujours en cours, afin de paralléliser d'autres outils nécessaires aux simulations de nanoalliages tels que le problème de la minimisation globale.

Listes des acronymes et termes techniques

adsorption	fixation d'une entité sur une surface
BD	dynamique brownienne
bloc	groupe de *warps* logiquement liés
buffer	mémoire tampon
cache	mémoire temporaire très rapide d'accès, mais de très faible capacité
CPU	Central Processing Unit (processeur principal)
CSA	Complexe Silice-Alumine
CUDA	interface de programmation permettant d'exploiter les GPUs NVIDIA
diffusion	migration des particules dans un milieu liquide ou gazeux
dimère	molécule ne comportant que deux sous-unités
double	nombre flottant à double précision (64 bit)
float	nombre flottant (32 bit)
GPU	Graphical Processing Unit (carte graphique)
grille régulière	structure accélératrice partitionnant l'espace
int	nombre entier
HIs	interactions hydrodynamiques
liste de Verlet	liste conservant le voisinage pendant plusieurs itérations
MD	dynamique moléculaire
mémoire coalescente	organisation de la mémoire de telle sorte que les cœurs adjacents accèdent à des cases mémoires contiguës
noyau	partie de code défini par le programmeur, s'exécutant sur le GPU
OpenCL	interface de programmation permettant d'exploiter de façon hétérogène tous types de périphériques de calcul parallèles (CPUs et GPUs)
opération atomique	opération qui s'assure le monopole de l'accès aux données afin d'éviter les problèmes de concurrence
PRNG	générateur de nombres pseudo aléatoires

problème de concurrence	problème lié aux accès simultanés à une même donnée par plusieurs processus
réduction	algorithme parallèle récursif divisant la taille de la donnée en entrée à chaque itération
SIMD	paradigme de programmation résolvant un problème en exécutant la même série d'instructions sur des entrées différentes
SPH	Smoothed Particle Hydrodynamics
SRD	Stochastic Rotation Dynamics
suspension colloïdale	fluide diphasique comportant de petites particules de 10 nm à 1μ m (colloïdes), suspendues dans un liquide (solvant)
thread	fil d'exécution
warp	groupe de 32 cœur physiquement liés

Listes des symboles utilisés

Δt	pas de temps
η	viscosité
γ	nombre moyen de particules de fluide par cellule de SRD
Γ_i	force stochastique
κ	inverse de la longueur de Debye
ϕ	fraction volumique
ψ	potentiel de surface
Ξ_i	force de friction
ζ	coefficient de friction
a_i	rayon de la particule i
A_{ij}	constante d'Hamaker
d_i	diamètre de la particule i
F_{ij}	force d'interaction entre les particules i et j
k_B	constante de Boltzmann
L_{box}	longueur de la boîte de simulation
m_i	masse de la particule i
N	nombre total de particules
N_i	ensemble des particules dans le voisinage de la particule i
n_i	nombre d'itérations consécutives sans réactualisation du voisinage
N_v	Nombre maximal de voisins
R	pourcentage massique de silice introduite dans les systèmes mixtes alumine-silice
$R(\alpha)$	matrice de rotation d'angle constant α
R_A	rayon d'adsorption
R_C	rayon de coupure
R_L	rayon de conservation
r_{ij}	distance entre les centres des particules i et j
R_{shell}	différence entre les rayons de conservation et de coupure
T	température
t	temps
v_i	vecteur vélocité de la particule i
v_{cm}	vélocité au centre de masse d'une cellule de SRD
V_{ij}	potentiel d'interaction entre les particules i et j

Bibliographie

[1] A. Tomilov, A. Videcoq, M. Cerbelaud, M. A. Piechowiak, T. Chartier, T. Ala-Nissila, D. Bochicchio, and R. Ferrando. Aggregation in colloidal suspensions : Evaluation of the role of hydrodynamic interactions by means of numerical simulations. *The Journal of Physical Chemistry B*, 117(46) :14509–14517, 2013.

[2] Z. Chen, J. Kingsley, X. Huang, and E. Tüzel. Accelerating a novel particle-based fluid simulation on the GPU. In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pages 1–6, Sept 2013.

[3] E. Westphal, S. P. Singh, C.-C. Huang, G. Gompper, and R. G. Winkler. Multiparticle collision dynamics : GPU accelerated particle-based mesoscale hydrodynamic simulations. *Computer Physics Communications*, 185 :495–503, February 2014.

[4] M. Cerbelaud, A. Videcoq, P. Abelard, C. Pagnoux, F. Rossignol, and R. Ferrando. Self-assembly of oppositely charged particles in dilute ceramic suspensions : predictive role of simulations. *Soft Matter*, 6 :370–382, 2010.

[5] M.P. Allen and D.J. Tildesley. *Computer simulation of Liquids*. Oxford University Press : Oxford, 1987.

[6] Donald L. Ermak and J. A. McCammon. Brownian dynamics with hydrodynamic interactions. *The Journal of Chemical Physics*, 69(4) :1352–1360, 1978.

[7] M. Cerbelaud, A. Videcoq, P. Abélard, C. Pagnoux, F. Rossignol, and R. Ferrando. Heteroaggregation between Al_2O_3 submicrometer particles and SiO_2 nanoparticles : Experiment and simulation. *Langmuir*, 24(7) :3001–3008, 2008.

[8] M. Cerbelaud, A. Videcoq, P. Abélard, and R. Ferrando. Simulation of the heteroagglomeration between highly size-asymmetric ceramic particles. *Journal of Colloid and Interface Science*, 332(2) :360 – 365, 2009.

[9] Anthony Y Kim, Kip D Hauch, John C Berg, James E Martin, and Robert A Anderson. Linear chains and chain-like fractals from electrostatic heteroaggregation. *Journal of Colloid and Interface Science*, 260(1) :149 – 159, 2003.

[10] M. A. Piechowiak, A. Videcoq, F. Rossignol, C. Pagnoux, C. Carrion, M. Cerbelaud, and R. Ferrando. Oppositely charged model ceramic colloids : Numerical predictions and experimental observations by confocal laser scanning microscopy. *Langmuir*, 26(15) :12540–12547, 2010.

[11] William T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 1983.

[12] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995.

[13] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH fluids in computer graphics. *Eurographics State-of-The-Art-Reports*, pages 21–42, 2014.

[14] Di-Bao Wang, Fei-Bin Hsiao, Cheng-Hsin Chuang, and Yung-Chun Lee. Algorithm optimization in molecular dynamics simulation. *Computer Physics Communications*, 177(7) :551 – 559, 2007.

[15] Markus Ihmsen, Nadir Akinci, Markus Becker, and Matthias Teschner. A parallel SPH implementation on multi-core CPUs. *Computer Graphics Forum*, 30(1) :99–112, 2011.

[16] Pedro Gonnet. Pairwise verlet lists : Combining cell lists and verlet lists to improve memory locality and parallelism. *Journal of Computational Chemistry*, 33(1) :76–81, 2012.

[17] Andrew J. Proctor, Cody A. Stevens, and Samuel S. Cho. GPU-optimized hybrid neighbor/cell list algorithm for coarse-grained MD simulations of protein and RNA folding and assembly. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics*, BCB'13, pages 633–640, 2013.

[18] Công Tâm Tran, Benoît Crespin, Manuella Cerbelaud, and Arnaud Videcoq. Brownian Dynamics Simulation on the GPU : Virtual Colloidal Suspensions. In Fabrice Jaillet, Florence Zara, and Gabriel Zachmann, editors, *Workshop on Virtual Reality Interaction and Physical Simulation*. The Eurographics Association, 2015.

[19] Loup Verlet. Computer "experiments" on classical fluids. I. thermodynamical properties of Lennard-Jones molecules. *Phys. Rev.*, 159 :98–103, Jul 1967.

[20] C.M. Eastman and S.F. Weiss. Tree structures for high dimensionality nearest neighbor searching. *Information Systems*, 7(2) :115 – 122, 1982.

[21] George Marsaglia, Arif Zaman, and Wai Wan Tsang. Toward a universal random number generator. *Statistics & Probability Letters*, 9(1) :35 – 39, 1990.

[22] William H. Press, Saul a. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in Fortran 77 : the Art of Scientific Computing. Second Edition*, volume 1. 1996.

[23] W. Humphrey, A. Dalke, and K. Schulten. VMD : visual molecular dynamics. *J. Molecular Graphics*, 14 :33–38, 1996.

[24] Alexander Stukowski. Visualization and analysis of atomistic simulation data with ovito–the open visualization tool. *Modelling and Simulation in Materials Science and Engineering*, 18(1) :015012, 2010.

[25] Jmol : an open-source Java viewer for chemical structures in 3D with features for chemicals, crystals, materials and biomolecules, 2009.

[26] Shane Cook. *CUDA Programming : A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[27] Aaftab Munshi, Benedict Gaster, Timothy G. Mattson, James Fung, and Dan Ginsburg. *OpenCL Programming Guide*. Addison-Wesley Professional, 1st edition, 2011.

[28] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '03, pages 41–50, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[29] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. Interactive SPH simulation and rendering on the GPU. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 55–64, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[30] Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical Report Ottawa, Ontario, Canada, 1966.

[31] David Van Der Spoel, Erik Lindahl, Berk Hess, Gerrit Groenhof, Alan E. Mark, and Herman J. C. Berendsen. Gromacs : fast, flexible, and free. *Journal of Computational Chemistry*, 26(16) :1701–1718, 2005. QC 20120301.

[32] Manuella Cerbelaud, Công Tâm Trân, Riccardo Ferrando, Benoît Crespin, and Arnaud Videcoq. Interdiffusion and crystallization of oppositely charged colloids. *Phys. Chem. Chem. Phys.*, pages 31094–31102, 2017.

[33] J. Lyklema, editor. *Fundamentals of interface and colloid science : Volume 1*. Academic Press, london edition, 1991.

[34] R. Hogg, T.W. Healy, and D.W. Fuerstenau. *Trans. Fraday Soc.*, 62 :1638–1651, DOI : 10.1039/TF9666201638, 1966.

[35] Aleena Laganapan, Manuella Cerbelaud, Riccardo Ferrando, Công Tâm Tran, Benoît Crespin, and Arnaud Videcoq. Computer simulations of heteroaggregation with large size asymmetric colloids. *Journal of Colloid and Interface Science*, 2017.

[36] Riccardo Ferrando, Julius Jellinek, and Roy L. Johnston. Nanoalloys : from theory to applications of alloy clusters and nanoparticles. *Chemical Reviews*, 108(3) :845–910, 2008. PMID : 18335972.

[37] A. Malevanets and R. Kapral. Mesoscopic model for solvent dynamics. *J. Chem. Phys.*, 110 :8605, 1999.

[38] A. Malevanets and R. Kapral. Solute molecular dynamics in a mesoscale solvent. *J. Chem. Phys.*, 112 :7260, 2000.

[39] M. Cerbelaud, M.A. Laganapan, T. Ala-Nissila, R. Ferrando, and A. Videcoq. Shear viscosity in hard-sphere and adhesive colloidal suspensions with reverse non-equilibrium molecular dynamics. *Soft Matter*, pages –, 2017.

[40] J.T. Padding and A.A. Louis. Hydrodynamic interactions and brownian forces in colloidal suspensions : Coarse-graining over time and length scales. *Phys. Rev. E*, 74 :031402, 2006.

[41] G. Gompper, T. Ihle, D. M. Kroll, and R. G. Winkler. Multi-Particle Collision Dynamics : A Particle-Based Mesoscale Simulation Approach to the Hydrodynamics of Complex Fluids. *Advances in polymer science*, 221 :1 – 87, 2009.

[42] T. Ihle and D. M. Kroll. Stochastic rotation dynamics : A galilean-invariant mesoscopic model for fluid flow. *Phys. Rev. E*, 63 :020201, January 2001.

[43] Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig. Accelerating molecular dynamics simulations using Graphics Processing Units with CUDA. *Computer Physics Communications*, 179(9) :634–641, November 2008.

[44] Jacobus Antoon Van Meel, Axel Arnold, Daan Frenkel, SF Portegies Zwart, and Robert G Belleman. Harvesting graphics power for md simulations. *Molecular Simulation*, 34(3) :259–266, 2008.

[45] Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *Journal of Computational Physics*, 227(10) :5342–5359, May 2008.

[46] A Zhmurov, RI Dima, Y Kholodov, and V Barsegov. Sop-GPU : Accelerating biomolecular simulations in the centisecond timescale using graphics processors. *Proteins : Structure, Function, and Bioinformatics*, 78(14) :2984–2999, 2010.

[47] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, 117(1) :1–19, March 1995.

[48] Công Tâm Tran, Benoît Crespin, Manuella Cerbelaud, and Arnaud Videcoq. Colloidal suspension by SRD-MD simulation on GPU (en cours de soumission). *Computer Physics Communications*, 2018.

[49] Lorenzo Rovigatti, Petr Sulc, I.Z. Reguly, and Flavio Romano. A comparison between parallelization approaches in molecular dynamics simulations on GPUs. 36, 01 2015.

[50] Gang Mei and Hong Tian. Impact of data layouts on the efficiency of GPU-accelerated IDW interpolation. *SpringerPlus*, 5(1) :104, Feb 2016.

[51] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH Fluids in Computer Graphics. In Sylvain Lefebvre and Michela Spagnuolo, editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association, 2014.

[52] Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. Openvdb : An open-source data structure and toolkit for high-resolution volumes. In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, pages 19 :1–19 :1, New York, NY, USA, 2013. ACM.

[53] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82 :1013–1024, December 1977.

[54] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics-theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181 :375–389, 1977.

[55] Matthias Muller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 154–159, 2003.

[56] Markus Becker and Matthias Teschner. Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, pages 209–217, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[57] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible SPH. *ACM Trans. Graph.*, 31(4) :62 :1–62 :8, 2012.

[58] J. J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30 :543–574, 1992.

[59] Joseph P. Morris, Patrick J. Fox, and Yi Zhu. Modeling low reynolds number incompressible flows using SPH. *J. Comput. Phys.*, 136(1) :214–226, September 1997.

[60] J.J. Monaghan. Simulating free surface flows with SPH. *J. Comput. Phys.*, 110(2) :399–406, February 1994.

[61] Karthik Raveendran, Chris Wojtan, and Greg Turk. Hybrid smoothed particle hydrodynamics. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 33–42, New York, NY, USA, 2011. ACM.

[62] Toon Lenaerts and Philip Dutré. Mixing fluids and granular materials. In *Computer Graphics Forum (Proceedings of Eurographics 2009)*, volume 28, pages 213–218, March 2009.

[63] Markus Ihmsen, Nadir Akinci, Gizem Akinci, and Matthias Teschner. Unified spray, foam and air bubbles for particle-based fluids. *The Visual Computer*, 28(6-8) :669–677, 2012.

[64] Hagit Schechter and Robert Bridson. Ghost SPH for animating water. *ACM Trans. Graph.*, 31(4) :61 :1–61 :8, July 2012.

[65] Joseph John Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8) :1703 – 1759, 2005.

[66] Markus Ihmsen, Nadir Akinci, Marc Gissler, and Matthias Teschner. Boundary handling and adaptive time-stepping for PCISPH. In Kenny Erleben, Jan Bender, and Matthias Teschner, editors, *VRIPHYS*, pages 79–88. Eurographics Association, 2010.

[67] Barbara Solenthaler, Jürg Schläfli, and Renato Pajarola. A unified particle model for fluid–solid interactions. *Computer Animation and Virtual Worlds*, 18(1) :69–82, 2007.

[68] X. Y. Hu and N. A. Adams. A multi-phase SPH method for macroscopic and mesoscopic flows. *J. Comput. Phys.*, 213(2) :844–861, April 2006.

[69] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible SPH. *ACM Trans. Graph.*, 31(4) :62 :1–62 :8, July 2012.

[70] B. Solenthaler and R. Pajarola. Density contrast SPH interfaces. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 211–218, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[71] Jihun Yu, Chris Wojtan, Greg Turk, and Chee Yap. Explicit Mesh Surfaces for Particle Based Fluids. *Computer Graphics Forum*, 2012.

[72] S. Nugent and H. Posch. Liquid drops and surface tension with smoothed particle applied mechanics. *Physical Review E*, 62(4) :4968–4975, October 2000.

[73] Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile surface tension and adhesion for SPH fluids. *ACM Trans. Graph.*, 32(6) :182 :1–182 :8, November 2013.

[74] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics*, 20(3) :426–435, March 2014.

[75] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. *ACM Trans. Graph.*, 28(3) :40 :1–40 :6, July 2009.

[76] M W. Evans and F H. Harlow. The particle-in-cell method for hydrodynamic calculations. 01 1957.

[77] J U Brackbill and H M Ruppel. Flip : A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *J. Comput. Phys.*, 65(2) :314–343, August 1986.

[78] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3) :965–972, July 2005.

[79] Landon Boyd and Robert Bridson. Multiflip for energetic two-phase fluid simulation. *ACM Trans. Graph.*, 31(2) :16 :1–16 :12, April 2012.

[80] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[81] Olivier Génevaux, Arash Habibi, and Jean michel Dischler. Simulating fluid-solid interaction. In *in Graphics Interface*, pages 31–38, 2003.

[82] Jeong-Mo Hong and Chang-Hun Kim. Discontinuous fluids. *ACM Trans. Graph.*, 24(3) :915–920, July 2005.

[83] Leonardo Di G. Sigalotti, Jorge Troconis, Eloy Sira, Franklin Peña Polo, and Jaime Klapp. Diffuse-interface modeling of liquid-vapor coexistence in equilibrium drops using smoothed particle hydrodynamics. *Phys. Rev. E*, 90 :013021, Jul 2014.

[84] Yasuhiro Hashimoto, Yu Chen, and Hirotada Ohashi. Immiscible real-coded lattice gas. *Computer Physics Communications*, 129(1) :56 – 62, 2000.

[85] Daniel H. Rothman and Stéphane Zaleski. Lattice-gas models of phase separation : interfaces, phase transitions, and multiphase flow. *Rev. Mod. Phys.*, 66 :1417–1479, Oct 1994.

[86] Mingming Wu and Morteza Gharib. Experimental studies on the shape and path of small air bubbles rising in clean water. 14, 05 2002.

[87] Bo Ren, Yuntao Jiang, Chenfeng Li, and Ming C. Lin. A simple approach for bubble modelling from multiphase fluid simulation. *Computational Visual Media*, 1(2) :171–181, Jun 2015.

[88] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, pages 219–228, New York, NY, USA, 2005. ACM.

Simulations de fluides complexes à l'échelle mésoscopique sur GPU

Résumé :

Les suspensions colloïdales ont été étudiées par simulations numériques à partir de deux modèles : la dynamique Brownienne (BD) et la SRD-MD (Stochastic Rotation Dynamics - Molecular Dynamics). Ces études ont consisté à reprendre des travaux existants pour les porter sur GPU, tout en cherchant différentes optimisations possibles adaptées à ces simulations. Une amélioration de la recherche de voisinage de la littérature a pu être utilisée pour toutes ces simulations de type BD. Une simulation de SRD-MD avec couplage de force qui n'avait pas encore été parallélisée sur GPU dans la littérature, a été implémentée en utilisant un nouveau schéma de décomposition adapté à cette simulation, améliorant considérablement les performances. Ces simulations ont pu donner lieu par la suite à des études sur des suspensions colloïdales plus complexes : une hétéroagrégation entre deux suspensions avec des particules de même taille, une hétéroagrégation entre deux populations de colloïdes de tailles très différentes, et en dehors des suspensions colloïdales, une simulation de nanoalliages. Enfin, le modèle de SRD a été adapté afin d'être utilisé dans le cadre d'animation physique de fluide réaliste dans le contexte de l'informatique graphique. Des adaptations du modèle pour y incorporer des notions comme la gestion de la compressibilité, de la tension de surface ont dues être apportées. Des premiers résultats ont pu permettre de réaliser quelques simulations, dont une chute d'eau dans une verre.

Mots clés : Simulations de fluide ; Dynamique Brownienne ; Dynamiques de Rotation Stochastique - Dynamique moléculaire ; Suspensions colloïdales ; Recherche de voisinage ; GPU.

Complex Fluid simulations at mesoscopic scale on GPU

Abstract :

Colloïdal suspensions have been studied by means of numerical simulation, using two physical models : Brownian dynamics and Stochastic Rotation Dynamics - Molecular Dynamics. These studies consist in parallizing colloïdal simulations from previous studies on GPU, and find some new optimisations for these specific simulations. An improvement of the neigborhood search has been implemented in all our BD type simulations. A SRD-MD with force coupling have been implemented for the first time in the literature, using a new decomposition scheme, which improves significantly its performances. Then, theses simulations have been adapted to study more complex colloidal suspensions : an interfacial heteroaggregation of colloidal suspensions, a heteroaggregation between two types of particles with a large size ratio, and outside this context, a nanoalloy simulation. Finally, the SRD model has been adapted to realistic fluid animtion from computer science context. Theses adaptations require to add to SRD model, the notion of compressibility and surface tension. First results have been released, like a pouring water into a glass simulation.

• Fluids simulation ; Brownian Dynamics ; Stochastic Rotation Dynamics - Molecular Dynamics ; Colloidal suspensions ; Neighborhood search ; GPU.

LABO - UMR CNRS n° xxxx
xxx, rue xxx - 87000 LIMOGES