Équilibrage dynamique de charge sur supercalculateur exaflopique appliqué à la dynamique moléculaire

Raphaël Prat

▶ **To cite this version:**

Raphaël Prat. Équilibrage dynamique de charge sur supercalculateur exaflopique appliqué à la dynamique moléculaire. Analyse numérique [cs.NA]. Université de Bordeaux, 2019. Français. NNT : 2019BORD0174 . tel-02413331

HAL Id: tel-02413331
https://theses.hal.science/tel-02413331

Submitted on 16 Dec 2019

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

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

université de BORDEAUX

FROM RESEARCH TO INDUSTRY
cea

Thèse présentée pour obtenir le grade de

Docteur de l'université de Bordeaux

École Doctorale de Mathématiques et Informatique de Bordeaux

Spécialité Informatique

Raphaël Prat

Équilibrage dynamique de charge sur supercalculateur exaflopique appliqué à la dynamique moléculaire

Sous la direction de Raymond Namyst & Laurent Colombet

Thèse prévue le 09 octobre 2019
à la Maison de la Simulation

Jury			
M. Baaden, Marc	Directeur de Recherche	Institut de Biologie Physico-Chimique	Examinateur
M. Calvin, Christophe	Expert International	Cea Saclay	Rapporteur
M. Colombet, Laurent	Chargé de recherche	Cea Dam-Île de France	CoDirecteur de thèse
M. Fortin, Pierre	Maître de Conférence	Sorbonne Université	Examinateur
M. Mehaut, Jean-Francois	Professeur	Université Grenoble Alpes	Rapporteur
M. Namyst, Raymond	Professeur	Université de Bordeaux	Directeur de thèse
M. Roman, Jean	Professeur	Inria	Examinateur

Commissariat à l'énergie atomique et aux énergies alternatives (Cea)
Centre Dam-Île de France – Bruyères-le-Châtel, 91297 Arpajon Cedex
☎ +33 (0)1 69 26 40 00

Équilibrage dynamique de charge sur supercalculateur exaflopique appliqué à la dynamique moléculaire

Raphaël PRAT

2016 – 2019

1 Remerciements

Maintenant que ma thèse au CEA s'est achèvée, je vais donc remercier toutes les personnes que j'ai pu côtoyer, pour ce qu'elles ont pu m'apporter aussi bien sur le plan technique que personnel.

Tout d'abord, je remercie tous les membres du jury pour avoir pris du temps pour évaluer mon travail. Je remercie Jean Roman pour avoir accepté de présider le jury. Je remercie mes deux rapporteurs Christophe Calvin et Jean François Méhaut pour leurs rapports et leurs commentaires. Je remercie mes deux examinateurs Marc Baaden et Pierre Fortin pour leurs questions intéressantes.

Je tiens tout particulièrement à remercier Laurent Colombet et Thierry Carrard pour votre encadrement quotidien et Raymond Namyst pour ton encadrement malgré la distance. Laurent, j'ai vraiment apprécié ta bonne humeur, tes références à Kaamelot et toutes les discussions que l'on a eu durant ces trois années. Thierry, si on enlève la fois où tu es rentré dans mon bureau en m'annonçant que mon travail était faux, je te remercie aussi d'avoir pris le temps de m'écouter avec mes idées farfelues. Raymond, je te remercie d'avoir pris le temps de faire toutes ces visioconférences malgré ton emploi du temps surchargé, de ton soutien et de tes commentaires pertinents.

Je remercie le CEA de m'avoir accueilli pendant ces trois ans ainsi que les personnes qui y travaillent. Notament Laurent Soulard pour m'avoir encadré en stage et pour toute l'aide que tu m'as apporté sur de nombreux points sur la physique, Olivier Durand pour avoir bêta testé ma version de l'AMR. Je remercie Nicolas Pineau, Claire Lemarchand et Ronan Madec pour les débats à la pause matinale et Sandra Boullier pour avoir répondu à toutes mes questions idiotes ainsi que pour m'avoir aidé à effectuer mon premier fax. Je remercie les stagiaires pour la bonne ambiance qu'ils amènent à chaque fois : Sébastien, Sami, Quentin, Loïc, Tristan, Luis, Théo, Robinson, Alexandre, Régis et Lucas. Je remercie pour cette même raison les thésitifs et post-docs : Richard, Thibaud, Luc, Jean Baptiste 1 et 2, Nils, David, Lucas, Gérôme, Xavier, Ahmed, Jean-Charles, Emmanuel, Ioannis, Aloïs, Augustin et Guillaume.

Je remercie particulièrement Jean Vicomte (ou Giovanni Viciconte), le plus français des Italiens, pour avoir enduré/partagé ces trois ans de thèse avec moi. C'était très plaisant de rencontrer quelqu'un d'aussi POSITIF. Je ne saurais comment te remercier pour les vacances de fin de thèse dans le village le plus beau du monde.

Paul Lafourcade, que dire, je te remercie pour avoir été une source d'inspiration au quotidien et surtout pour ta patience à toute épreuve. Sans oublier les .gifs, le temps passé à me plaindre et ta patience encore une fois.

Estelle dirand, tu as eu le plaisir de me côtoyer pendant 2 ans, je te laisse donc quelques lignes blanches que tu pourras remplir pour me remercier davantage :

Bon ok, merci pour ces deux ans et pour toutes les discutions que l'on a pu avoir.

Finalement Nicolas Bruzy (désolé mais je n'ai pas trouvé pas de photo), c'est vraiment dommage que nous n'ayons pas pu être co-bureau plus longtemps. Je tiens à te remercier pour avoir été aussi drôle durant ces 6-7 mois, n'oublie pas de prendre soin de ce bureau.

Je remercie ma famille pour de nombreuses raisons, mes parents biologiques Agnès Prat-Dutel et Rémi Prat pour m'avoir fait, éduqué, encouragé et m'avoir écouté durant de longues heures pendant ma rédaction. Ma grand mère Roberte Dutel pour avoir finalement pu venir à ma soutenance de thèse grâce à l'aide de Sylvie et Dominique Maignan. Je remercie Patrice pour avoir fait le boulot de ma mère (relecture du manuscrit). Je remercie aussi le reste de ma famille.

Sonia, cette fois je ne t'oublierai pas et pas seulement parce que tout le monde m'a répété de ne pas t'oublier à l'écrit. Au passage je maintiens que je te gardais vraiment pour la fin, mais sur le coup j'ai eu un petit black out (encore désolé). Cela fait 8 ans que tu fais partie de ma vie, que nous avons partagé déjà beaucoup d'étapes de nos vies (prépa, école d'ingé et thèse). Tu m'as toujours aidé

quand j'en avais besoin (révisions statistiques), j'espère pouvoir te rendre ne serait-ce que 1% de tout ce que tu as fait pour moi (mais je ne rédigerai pas ton manuscrit de thèse). Je te remercie donc pour tout ce que tu as fait pour moi, pour la personne que tu es (belle, petite, intelligente, joviale, marrante, curieuse, battante, etc...) et surtout je te remercie d'avance pour tout ce qui m'arrivera dans le futur avec toi. Merci Sonia 182.

Table des Matières

2 Introduction

Contexte

Pour approfondir notre compréhension des phénomènes physiques, les chercheurs élaborent des modèles afin d'effectuer des simulations numériques validées par des expériences. Par exemple dans le domaine de la physique de la matière condensée, les chercheurs du CEA cherchent à étudier le comportement d'un liquide ou d'un solide de l'échelle macroscopique à l'échelle microscopique. Dans cette thèse, nous nous intéressons aux simulations réalisées avec la méthode de la Dynamique Moléculaire classique (Dm) [4, 114]. Elle consiste à reproduire numériquement le déplacement d'un ensemble de particules à une échelle atomique sur quelques nanosecondes. L'objectif est de comprendre la structure interne d'un matériau sous certaines conditions (température, choc, etc) comme par exemple une plaque métallique (étain, cuivre, etc) ou un matériau organique (azote, carbone, etc). En Dm, le déplacement d'un atome est déduit des interactions entre celui-ci et l'ensemble des autres atomes. La nature des interactions dépend de nombreux facteurs comme le type des atomes.

Pour réaliser des simulations de Dm, les physiciens s'appuient sur la puissance de calcul des supercalculateurs. Celle-ci n'a cessé d'augmenter au cours des dernières décennies [111] au point que l'ordinateur le plus puissant du monde en 1996 est moins puissant que nos téléphones portables actuels. Cette émergence de nouveaux supercalculateurs pouvant atteindre jusqu'à des millions de cœurs de calcul (10 649 600 cœurs pour le calculateur Sunway TaihuLight [57]) a permis d'étudier en dynamique moléculaire le comportement de systèmes de particules toujours plus complexes. En effet, suite à l'évolution des supercalculateurs de ces dernières décennies, les systèmes d'atomes simulés sont passés de quelques milliers d'atomes à plusieurs dizaines de milliards d'atomes. Les physiciens se sont alors intéressés à l'étude de phénomènes plus complexes comme l'évolution d'un choc dans une plaque métallique [47] ou l'impact de nano-gouttes d'étain sur une surface solide [148, 79]. Dans cette thèse nous nous intéresserons à l'optimisation de simulations ayant les caractéristiques suivantes :

1. la répartition extrêmement inégale de la densité d'atomes;
2. le déplacement rapide des atomes;
3. une grande partie du domaine de la simulation ne contient pas d'atome;
4. uniquement des interactions à courte portée.

Pour de telles simulations, puisque la distribution de la densité des atomes est extrêmement inégale, la répartition de la charge entre les cœurs de calcul d'un supercalculateur est extrêmement

déséquilibrée. Les simulations ont alors une durée plus importante. D'autres simulations de ce type ne sont même pas concevables à cause du temps d'exécution trop important ou parce que les ressources mémoires sont insuffisantes.

Jusqu'à présent des solutions de répartition de charge telles que les méthodes de partitionnement [29] permettaient d'améliorer l'équilibrage de la charge entre les cœurs de calcul. Toutefois, ces solutions ne sont plus suffisantes pour réaliser les simulations envisagées par le Cea sur des dizaines de milliers de cœurs.

Pour calculer efficacement les interactions à courte portée, les codes de Dm utilisent un maillage en cellules [5]. Pour les simulations dont la densité d'atomes est extrêmement inégale, une zone très importante du domaine ne contient généralement pas ou peu d'atomes. Les mailles créées dans cette zone sont alors inutiles car il n'y a pas ou peu de calculs à faire. De plus, ces mailles engendrent des surcoûts de stockage mémoire mais aussi de calcul car les algorithmes de Dm parcourent l'ensemble des mailles. C'est pourquoi la répartition des calculs entre les ressources d'un supercalculateur est particulièrement difficile à mettre en place compte tenu de ces mailles.

Objectifs de la thèse

Dans cette thèse, nous proposons une solution novatrice en Dm pour résoudre ces problèmes d'efficacité. Pour cela, nous étudions l'utilisation de la méthode de raffinement de maillage adaptatif (Amr) [17, 16] pour les simulations de dynamique moléculaire afin de réduire le surcoût des mailles ne contenant pas d'atome qui dégradent la répartition de la charge sur les supercalculateurs. Pour cela nous développons cette méthode dans le code Exaflopique de Simulation Temporelle Atomistique et Moléculaire Parallèle (ExaSTAMP) [30, 29]. Ce code est développé depuis 6 ans au Cea afin d'être exécuté sur les dernières générations de supercalculateurs tel que Tera-1000-2 qui est composé 8 256 Knls (Nœuds), soit environ 561 408 cœurs de calcul.

Les objectifs de cette thèse sont :

- d'incorporer notre solution basée sur l'Amr dans un code de Dm;
- d'optimiser la structure Amr afin qu'elle tienne compte des trois niveaux de parallélisation : vectorisation, intra-Nœud et inter-Nœuds;
- d'élaborer différentes stratégies de parallélisation afin de répartir équitablement la charge entre les ressources de calcul;
- d'étudier l'impact des méthodes de partitionnement sur des simulations comme l'impact d'une nano-goutte d'étain sur une surface solide;
- de réaliser le plus efficacement possible des cas de production sur les supercalculateurs du Cea.

Organisation

La thèse est organisée en 3 parties : état de l'art, contributions et validation. Le chapitre 3 explique brièvement les notions importantes pour comprendre le fonctionnement des simulations de dynamique moléculaire. Une description de l'évolution matérielle et logicielle des supercalculateurs est réalisée dans le chapitre 4. Elle comprend notamment un aperçu des méthodes de parallélisation actuellement utilisées par la plupart des codes scientifiques. Finalement, dans le chapitre 5, nous décrivons les différents types de maillage et notamment ceux obtenus par les méthodes de raffinement de maillage adaptatif et de l'octree. Les méthodes de partitionnement géométrique ou

de graphe sont ensuite introduites, puis nous concluons par une brève description des principaux codes de DM.

Les chapitres 6, 7 et 8 décrivent mes contributions et les différentes étapes qui ont permis d'élaborer une structure AMR adaptée aux 3 niveaux de parallélisme dans un code de DM. En effet, dans le chapitre 6, nous mettons en place la structure AMR dans le code EXASTAMP et nous cherchons à optimiser les calculs en améliorant la réutilisation de caches mémoire et l'utilisation de la vectorisation. L'élaboration de la parallélisation intra-NŒUD (OPENMP [35]) basée sur des tâches est développée et étudiée dans le chapitre 7. Les performances obtenues sur les dernières générations de processeurs avec un très grand nombre de threads sont alors supérieures à des codes comme LAMMPS [109]. Ensuite, nous abordons dans le chapitre 8 la conception de la parallélisation inter-NŒUDS (MPI [31]) combinée avec les méthodes de partitionnement du domaine. Nous proposons également un système astucieux permettant de répliquer efficacement les atomes dans les zones fantômes avec une structure AMR.

La dernière partie traite de la validation des contributions effectuées, cf. chapitre 9. Les tests sont alors réalisés sur des simulations exécutées avec un très grand nombre de cœurs de calcul, entre 16 384 et 65 536 cœurs.

3 La Dynamique Moléculaire classique

Généralités

La physique de la matière condensée est un domaine de recherche consistant à étudier les propriétés d'un liquide ou d'un solide à un niveau macroscopique ou à un niveau microscopique. Les matériaux considérés peuvent être organiques (carbone, azote, etc) ou métalliques comme du cuivre ou de l'étain. Le Cea cherche à étudier la réponse de ces matériaux à des sollicitations dynamiques comme une onde de compression ou de décompression. Les phénomènes physiques étudiés peuvent par exemple être la compréhension de l'évolution d'un choc dans un matériau métallique ou le changement de phase (liquide/solide) d'un matériau.

Pour simuler ces phénomènes, plusieurs approches sont possibles dont la plus traditionnelle est la mécanique des milieux continus. Elle repose sur la résolution des équations de Navier-Stockes. Dans le cadre de cette thèse, nous nous intéressons à des simulations comme l'évolution d'un micro-jet d'étain résultant de la propagation d'un choc avec une surface libre ou l'impact d'une nano-goutte d'étain sur une surface solide. Ces simulations ont pour vocation de comprendre le comportement d'un phénomène comme un choc sur un matériau à un niveau atomique, ce qui n'est pas possible avec la mécanique des milieux continus. Pour cela, la méthode numérique de la Dynamique Moléculaire *ab initio* (AIMD) restitue la physique d'un petit système d'atomes en tenant compte des électrons. La méthode AIMD permet de simuler quelques milliers d'atomes d'hydrogène à quelques centaines d'atomes avec beaucoup d'électrons. Pour les simulations envisagées, ce nombre d'atomes n'est pas suffisant.

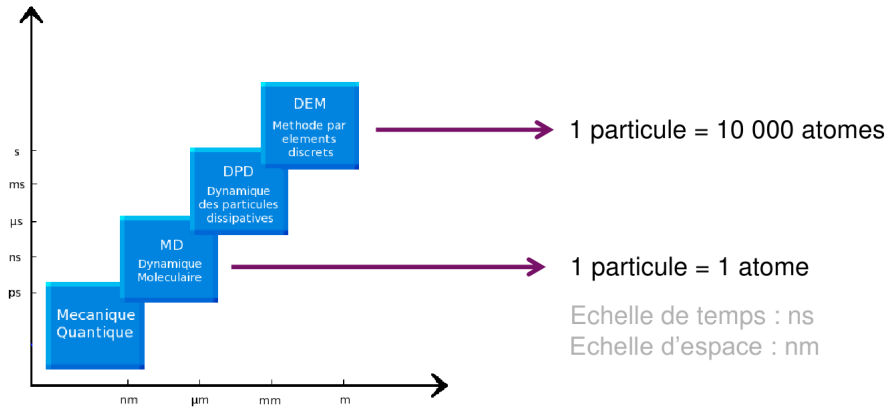


Figure 3.1 | Méthodes utilisées selon le temps d'acquisition et la taille à observer.

Dans le cadre de cette thèse, la méthode numérique que nous utilisons pour comprendre les phénomènes de l'évolution d'un micro-jet ou de l'impact d'une nano-goutte d'étain sur une surface solide est la Dynamique Moléculaire classique (DM). Elle permet de restituer la physique sur quelques nanosecondes d'un système d'atome décrit au niveau atomique. D'un point de vue statistique, si l'on considère un très grand nombre d'atomes, la DM se comportent comme de la physique des milieux continus. À noter que plusieurs méthodes comme la Dynamique des Particules Dissipatives (DPD) et la Méthode par Éléments Discrets (DEM) tentent de faire la jonction entre l'échelle macroscopique et microscopique (voir figure 3.1).

La DM est basée sur la résolution numérique de l'équation du mouvement de Newton $\ddot{r} = \frac{\vec{f}}{m}$ où la position cartésienne de la particule est définie par $r(r_x, r_y, r_z)$, sa vitesse par $\dot{r}$, son accélération par $\ddot{r}$ et sa masse par m. La somme des forces $\vec{f}$ exercées sur la particule considérée dépend de la nature des interactions entre les particules. Ces interactions sont modélisées par des potentiels empiriques ou champs de force. En DM, les potentiels correspondent à la partie restituant toute la physique du phénomène étudié. Une description algorithmique et numérique des principaux potentiels utilisés dans les code de DM est effectuée dans la section 3.4.

Chaque simulation est réalisée selon un ensemble thermodynamique en rendant invariant au cours du temps des grandeurs physiques comme par exemple le volume, la température ou la pression. Les principaux ensemble thermodynamique utilisés dans les codes de DM sont décrits dans le paragraphe 3.2. En se plaçant sous tel ou tel ensembles thermodynamique, les systèmes particulaires peuvent être étudiés selon divers simulations comme par exemple lorsqu'un système est isolé de tout autre système (ensemble Microcanonique) ou lorsqu'un système est au contact d'un plus grand lui imposant alors sa température (ensemble Canonique). Pour effectuer l'intégration en temps du système, plusieurs schémas numériques sont spécifiés dans le paragraphe 3.3.

Dans ce chapitre nous décrivons les notions nécessaires afin de comprendre les simulations comme l'éjection de matière ou l'impact de nano-gouttes sur une surface solide. Ces simulations sont détaillées dans la section 3.5.

Les ensembles thermodynamiques

Un ensemble thermodynamique est un cas particulier des ensembles statistiques [58] pour lequel certaines conditions telles que l'énergie totale du système (E), la pression (P) ou la température (T) sont constantes en moyenne. Dès lors, chaque ensemble thermodynamique nous permet d'observer des systèmes de particules en leur appliquant différentes conditions comme par exemple une température ou une pression élevée.

Par exemple l'un des ensembles les plus utilisés en DM est l'ensemble Microcanonique (NVE) [59], il permet de modéliser un système isolé thermodynamiquement, c'est-à-dire qu'au cours du temps il n'y a pas de transfert de particules (N) ni d'énergie (E) avec l'extérieur et que le domaine du système ne varie pas (V).

L'ensemble Canonique (NVT) [59] permet de considérer un système de volume et de température constante en moyenne et dont le nombre de particule ne varie pas.

En pratique, un second système bien plus grand impose sa température (T) au système considéré. Il est alors considéré comme une contrainte extérieure. Afin de modéliser ce transfert de température, on ajoute un thermostat au système. Il existe diverses méthodes permettant d'inclure ce thermostat dans l'équation du mouvement comme par exemple le thermostat d'Andersen [8], de Berendsen [13] ou de Nosé-Hoover [67, 102, 103, 66, 50].

Une autre possibilité est d'introduire la dynamique de Langevin [2, 122], c'est-à-dire que lors du calcul des accélérations des particules par l'équation du mouvement de Newton, un terme de dissipation et un terme de fluctuation sont introduits. Ces termes dépendent de la viscosité notée γ. La viscosité correspond au phénomène de résistance à l'écoulement se produisant dans la matière. Pour une particule r, on obtient alors l'accélération d'une particule, notée $\ddot{r}$ et de masse m, par l'équation 3.1 et par le théorème de fluctuation-dissipation [26]. T étant la température, K_b la constante de Boltzmann et R(t) un processus gaussien stationnaire, c'est-à-dire indépendant du temps et de moyenne nulle. Généralement cet ensemble est utilisé afin de thermaliser un système avant de repasser dans un ensemble NVE.

$$m\ddot{r} = -f - \gamma\dot{r} + \sqrt{2\gamma K_b T R(t)} \tag{3.1}$$

Un dernier ensemble présenté est l'ensemble « T-P » ou NPT, pour lequel le système est en contact avec un réservoir d'énergie mais dont le volume du système varie au cours du temps. Cela permet de contraindre la pression (barostat) du système en plus de sa température (thermostat).

Concernant les phénomènes physiques que l'on cherche à reproduire numériquement, cf. section 3.5, deux ensembles thermodynamique sont utilisés dans notre cas : NVE et NVT. L'ensemble NVT est principalement utilisé pour thermaliser le système à une température donnée (initialisation de la configuration des atomes) alors que l'ensemble NVE est utilisé pour effectuer la simulation après l'initialisation lorsque des conditions impliquant un déséquilibre local sont imposées au système.

Schémas Numériques

Dans ce paragraphe nous allons énumérer les principaux schémas d'intégration permettant d'approximer le déplacement des particules au cours du temps durant une simulation de DM. Pour cela on résout numériquement les équations différentielles du type $\ddot{r} = \frac{\partial^2 r}{\partial t_n^2} = F(r)$. Selon les schémas numériques utilisés par les codes de DM, le déplacement des atomes sera plus ou moins en adéquation avec la réalité.

Intégrateur de Verlet

Sachant que le temps est discrétisé selon un pas de temps donné ($\Delta t > 0$), le but est de déterminer les positions cartésiennes $r^n(r_x^n, r_y^n, r_z^n)$ des particules au temps $t_n = n\Delta t$ par une approximation de la trajectoire des particules. Pour une simulation, le temps t_n sera par la suite appelé pas de temps n. Les schémas d'intégration utilisés en DM se basent sur l'intégrateur de Verlet [139]. Celui-ci repose sur l'utilisation d'une dérivée centrée afin d'approximer l'accélération au pas de temps n notée $\ddot{r}^n$, avec $\ddot{r}^n = \frac{\partial^2 r^n}{\partial t^2} = \frac{r^{n+1} - 2r^n + r^{n-1}}{\Delta t^2}$. On obtient alors la relation :

$$r^{n+1} = 2r^n - r^{n-1} + \ddot{r}^n \Delta t^2 + O(4) \tag{3.2}$$

Les développements de Taylor ci-dessous de r^{n+1} et r^{n-1} permettent d'établir que l'intégrateur de Verlet est d'ordre 4. On note $\dot{r}^n$ la dérivée temporelle première (vitesse) et $\dddot{r}^n$ la dérivée temporelle troisième du vecteur position r^n.

$$\begin{cases} r^{n+1} &= r^n + \Delta t\dot{r}^n + \frac{\Delta t^2}{2}\ddot{r}^n + \frac{\Delta t^3}{6}\dddot{r}^n + O(4) & (1) \\ r^{n-1} &= r^n - \Delta t\dot{r}^n + \frac{\Delta t^2}{2}\ddot{r}^n - \frac{\Delta t^3}{6}\dddot{r}^n + O(4) & (2) \\ r^{n+1} &= 2r^n - r^{n-1} + \ddot{r}^n\Delta t^2 + O(4) & (1) + (2) \end{cases}$$

Le problème de l'intégrateur de Verlet est qu'il n'incorpore pas le calcul des vitesses, essentiel pour le calcul de certaines grandeurs physiques comme l'énergie cinétique du système ou la température. De plus, pour calculer la position r^{n+1} on a besoin des instances des vecteurs positions r^n et r^{n-1}, ce qui augmente le stockage.

Verlet Vitesse

Une solution est d'utiliser le schéma d'intégration Verlet Vitesse [131], aussi connu sous le nom de méthode de Störmer-Verlet. Basé sur l'intégrateur de Verlet, il se différencie de celui-ci en incluant le calcul des vitesses. Les vitesses (dérivées des positions) sont obtenues par une dérivée centrée, i.e. équation 3.3.

$$\dot{r}^n = \frac{r^{n+1} - r^{n-1}}{2\Delta t}. \tag{3.3}$$

À partir de l'intégrateur de Verlet (équation 3.2) et de l'équation 3.3 on obtient l'algorithme de Verlet Vitesse:

$$\begin{cases} r^{n+1} = r^n + \Delta t \dot{r}^n + \frac{\Delta t^2}{2}\ddot{r}^n + O(3) \\ \dot{r}^{n+1} = \dot{r}^n + \frac{\ddot{r}^n + \ddot{r}^{n+1}}{2}\Delta t \end{cases} \tag{3.4}$$

Pour calculer la vitesse au pas de temps n+1 ($\dot{r}^{n+1}$), on ajoute une étape intermédiaire en temps $t_{n+1/2} = t_n + \Delta t/2$ décrite par l'équation 3.5 :

$$\dot{r}^{n+1/2} = \frac{r^{n+1} - r^n}{\Delta t} = \dot{r}^n + \frac{\Delta t}{2}\ddot{r}^n \tag{3.5}$$

En insérant l'étape 3.5 dans l'algorithme de Verlet Vitesse (3.4) on obtient le système d'équations 3.6 suivant :

$$\begin{cases} r^{n+1} = r^n + \Delta t \dot{r}^n + \frac{\Delta t^2}{2}\ddot{r}^n \\ \ddot{r}^{n+1} = \frac{\vec{f}}{m} \\ \dot{r}^{n+1/2} = \dot{r}^n + \frac{\Delta t}{2}\ddot{r}^n \\ \dot{r}^{n+1} = \dot{r}^{n+1/2} + \frac{\Delta t}{2}\ddot{r}^{n+1} \end{cases} \tag{3.6}$$

À noter que l'accélération $\ddot{r}^{n+1}$ est obtenue à partir de l'équation du mouvement de Newton. Finalement, ce schéma est d'ordre 2 et réversible en temps [136], c'est-à-dire qu'une fois le pas de temps n atteint, le processus peut être inversé dans le but de revenir à l'état initial du système.

Les interactions

Les schémas numériques utilisés en DM nécessitent à chaque pas de temps d'avoir connaissance de l'accélération de chaque particule. L'accélération d'une particule est déduite des interactions entre cette particule et les autres particules du système. En Dynamique Moléculaire classique, les interactions sont classées selon 3 familles potentiels. À noter que ceux sont les potentiels qui restitue la physique de la méthode.

Classification

La première famille est composée des interactions à courte portée, c'est-à-dire les interactions avec les particules incluses dans le voisinage «proche» d'une particule. Ces interactions sont modélisées par différents types de potentiels comme les potentiels de pair, Embedded Atom Model (EAM) ou Modified Embedded Atom Model (MEAM). Lorsque la distance entre deux particules est supérieure à un Rayon de coupure (RCUT), la valeur de ces potentiels est considérée comme négligeable, c'est-à-dire égale à 0. En pratique, pour une simulation dont les atomes ont uniquement des interactions à courte portée, le calcul des forces est l'étape qui peut coûter jusqu'à 95% du temps de calcul d'une simulation de DM pour des potentiels coûteux en calcul comme les MEAM. Les potentiels sont décrits dans les paragraphes allant de 3.4.3 à 3.4.7.

La deuxième famille est formée par les interactions à longue portée. Elle prend par exemple en compte les forces électrostatiques entre une particule et toutes les particules du système. Afin de calculer de telles interactions, il existe plusieurs méthodes comme la méthode d'Ewald [51] et ses dérivées comme la méthode Particle–Particle-Particle–Mesh Ewald (P3M) [65].

Dans le cas d'un système incluant des molécules, il faut également tenir compte des interactions intramoléculaires, modélisant par exemple l'élongation, la flexion ou la torsion de la molécule.

Les simulations envisagées dans la partie 3.5 reproduisent des phénomènes sur des métaux. Or, pour des simulations de métaux, les interactions à longue portée sont négligeables et il n'y a pas d'interactions intramoléculaires. C'est pourquoi nous concentrons nos explications uniquement sur les interactions à courte portée.

Listes de voisins

Avant de détailler les principaux potentiels utilisés pour modéliser les interactions à courte portée, nous allons nous intéresser aux méthodes permettant de sélectionner les interactions non nulles entre les atomes. En effet, les interactions et, par extension, les forces sont considérées comme négligeables lorsque la distance entre deux atomes est supérieure au maximum des RCUT.

Ainsi, l'un des enjeux pour les simulations comportant des interactions à courte portée est de déterminer pour chaque atome du système la liste des atomes les plus «proches», nommée liste de voisins. Le calcul des interactions est alors en $O(N)$. Afin de déterminer la liste des atomes voisins, l'approche triviale est de tester à chaque pas de temps si les distances relatives entre un atome et l'ensemble des atomes du système sont inférieures au rayon de coupure. Cette stratégie a une complexité algorithmique en $O(N^2)$, N étant le nombre d'atomes composant le système.

C'est pourquoi, si le nombre d'opération pour construire des listes de voisins croît en fonction du carré du nombre d'atomes alors que le calcul des interactions ne dépend que du voisinage «proche» des atomes (complexité linéaire), le nombre d'atomes simulé dépend alors uniquement de la construction des listes de voisins. Afin de réduire la complexité algorithmique et le taux de rafraîchissement des listes de voisins, deux méthodes sont utilisées : la méthode des listes de Verlet et la méthode des cellules liées.

Méthode des listes de Verlet

La méthode des listes de Verlet [139] a été introduite dans le but de réduire la fréquence de rafraîchissement des listes de voisins, notées N^n, n étant le pas de temps. Le principe, illustré par la figure 3.2, est d'ajouter au RCUT un Rayon de Verlet (RVERLET) pour créer une liste de Verlet pour chaque atome, notée V^n. La liste de voisins est alors un sous-ensemble de la liste de Verlet, i.e. $N^n \in V^n$. Cette méthode assure que, tant qu'aucun atome du système ne s'est déplacé de plus de

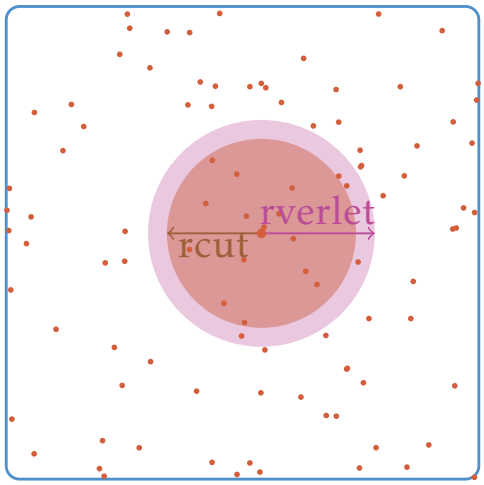


Figure 3.2 | Méthode de construction des listes de voisins par la méthode des cellules liées.

$\frac{1}{2}$ rVerlet, la liste de voisins au pas de temps $n' > n$ est incluse dans la liste de Verlet construite au pas de temps n, c'est-à-dire $N_{n'} \in V^n$. À noter que la construction des listes de Verlet a une complexité algorithmique en $O(N^2)$. En pratique lorsque le système est peu dynamique car les atomes se déplacent lentement, les listes de Verlet ne sont pas fréquemment rafraîchies. Lorsque le système est extrêmement dynamique, comme lorsque le système d'atomes est soumis à des conditions extrêmes (haute pression, haute température, etc), les listes de Verlet risquent d'être mises à jour à chaque pas de temps de la simulation. Une solution est d'augmenter le Rayon de Verlet au détriment d'un stockage plus important d'atomes et donc une augmentation du nombre de candidats à exclure pour trouver les listes de voisins pour chaque pas de temps.

Méthode des cellules liées

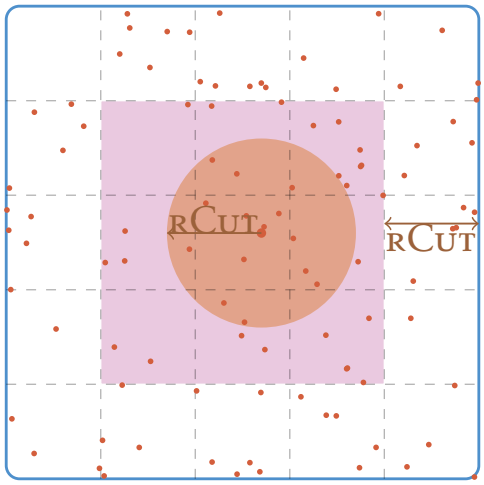


Figure 3.3 | Méthode de construction des listes de voisins par la méthode des cellules liées

La méthode des cellules liées [5], illustrée par la figure 3.3, consiste à mailler le domaine de la simulation en cellules. Les cellules sont des parallélépipèdes rectangles de volumes égaux dont la largeur est supérieure au maximum des rayons de coupure. Les atomes sont alors projetés en fonction de leurs coordonnées cartésiennes sur le maillage en cellules. Ainsi par construction, pour

chaque atome, l'ensemble des atomes qui lui sont voisins sont inclus dans la même cellule ou dans l'une des 8 cellules adjacentes en 2D ou 26 en 3D. La complexité algorithmique de la recherche des voisins passe alors en $O(N)$. La méthode des cellules liées se combine avec la méthode des listes de Verlet en incluant le rayon de Verlet lors de la détermination de la taille des cellules. En pratique les logiciels de dynamique moléculaire sont généralement basés sur cette combinaison de méthodes.

Les potentiels interatomiques

Pour les simulations de matériaux denses, à partir d'une distance supérieur à rCut les interactions entre les atomes sont écrantées par des atomes plus «proches». On fait alors l'hypothèse que les interactions entre deux atomes trop éloignés sont négligeables et ne sont donc pas calculées. Soit N le nombre d'atomes de la configuration atomique, la position et la masse de chaque atome i sont respectivement notées r_i et m_i. La distance relative (vecteur) et la distance euclidienne (scalaire) entre l'atome i et j sont respectivement notées $\bar{r_{ij}}$ et r_{ij}, elles sont respectivement définies par $\bar{r_{ij}} = r_j - r_i$ et $r_{ij} = \|\bar{r_{ij}}\|_2$ (norme de l'espace euclidien à 3 dimension). La force exercée par l'atome j sur l'atome i est notée $\overrightarrow{f_{ij}}$. La somme des forces appliquées à l'atome i est déduite de l'énergie potentielle E_p par l'équation 3.8, $\overrightarrow{\nabla}$ étant l'opérateur gradient de l'espace euclidien à 3 dimensions.

$$\sum_j \overrightarrow{f_{ij}} = -\overrightarrow{\nabla}\left(\sum_j E_p(r_{ij})\right). \tag{3.7}$$

Ainsi l'équation du mouvement de newton peut être écrite :

$$m_i \ddot{r}_i = -\overrightarrow{\nabla}\left(\sum_j E_p(r_{ij})\right). \tag{3.8}$$

Pour les simulations qui nous intéressent (voir paragraphe 3.5) on distinguera principalement trois types de potentiel: les potentiels de paire, paragraphe 3.4.4, les potentiels Eam, paragraphe 3.4.5 et le potentiel Meam, paragraphe 3.4.6.

Potentiels à deux corps

La famille des potentiels de paire est généralement utilisée pour décrire les interactions entre deux atomes d'un liquide ou d'un gaz. Dans ce cas, chaque interaction entre deux atomes dépend uniquement de la distance entre ces atomes. L'énergie potentielle d'un atome est alors approchée par la somme des interactions de paire (équation 3.9). Par la suite, pour simplifier les écritures, lorsque l'on considère deux atomes i et j, ceux-ci sont inclus dans leurs propres voisinages respectifs et N_i est la liste des atomes voisins de l'atome i.

$$E_p(r_i) = \sum_{j \in N_i} \left[E_p(r_{ij})\right], \quad \texttt{sachant que} \quad ||r_{ij}||_2 = ||r_i - r_j||_2 < \text{rCut}. \tag{3.9}$$

L'une des particularités de ces potentiels est leur propriété symétrique, c'est-à-dire que $\overrightarrow{f_{ij}} = -\overrightarrow{f_{ji}}$. Cette propriété évite de calculer deux fois la même interaction et ainsi cela préserve de précieux cycles de calcul. Il existe de nombreux potentiels de paire comme le potentiel Lennard Jones (Lj) [69], les potentiels de Morse [99] prenant en compte la spectroscopie et l'énergie de vibration, les potentiels de Buckingham [23](exponential-6) étant entre un potentiel Lj et un potentiel de Morse. Les potentiels comme Molière [97] ou Ziegler–Biersack–Littmark (ZBL) [150] sont utilisés pour

des particules énergétiques (potentiels coulombiens). Nous allons détailler davantage le potentiel Lennard Jones car nous l'utiliserons tout au long de ce manuscrit.

$$E_p(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]. \tag{3.10}$$

Le potentiel Lennard Jones, dont la formule analytique est écrite dans l'équation 3.10, est composé d'un premier terme répulsif ($\left(\frac{\sigma}{r}\right)^{12}$) obtenu empiriquement et d'un deuxième terme attractif ($\left(\frac{\sigma}{r}\right)^{6}$) correspondant à la force de Van Der Walls [86]. Les paramètres ϵ et σ désignent respectivement la profondeur du puits du potentiel et la distance pour laquelle la valeur du potentiel est nulle. La distance d'équilibre (aussi répulsif qu'attractive) est alors égale à $2^{\frac{1}{6}}\sigma$. Des exemples de valeur de paramètres sont fournis dans le livre [5] et reportés dans la Table 3.1.

Element	σ(nm)	ϵ(meV)	Element	σ(nm)	ϵ(meV)
H	0.281	0.741	*He*	0.228	0.879
C	0.335	4.412	*N*	0.331	3.214
O	0.295	5.308	*F*	0.283	4.550
Ne	0.272	4.050	*S*	0.352	15.770
Cl	0.335	14.951	*Ar*	0.341	10.324

Table 3.1 | Paramètres du potentiel Lennard-Jones pour différents éléments [5]. Les paramètres σ et ϵ sont déterminés pour un rCut donné. La valeur de rCut est égale à 2.5σ.

Au vu de sa simplicité et de son coût arithmétique peu élevé, le potentiel LJ permet d'évaluer les performances des différents logiciels en effectuant des comparaisons sur des cas simples. À noter qu'il ne masque pas les autres étapes de calculs durant les simulations de DM. Il permet donc d'évaluer un grand nombre de portion des codes.

Modèle de l'atome immergé: EAM

Ce modèle a été introduit par Daw et Baskes [36, 37, 54, 38] afin de prendre en compte la densité électronique du système en plus d'un terme de paire. Ainsi l'énergie potentielle, voir équation 3.11, est obtenue à partir d'un terme de pairn noté Φn et de la densité atomique de l'atome, notée $F(\rho_i)$. À noter que Φ dépend uniquement de la distance entre deux atomes, donc comme pour le potentiel LJ, une partie des calculs sont effectués deux fois (propriété de symétrie) et peuvent donc être évités.

$$\begin{cases} E_p(r_i) &= \sum_{j \in N_i} \left[\frac{1}{2} \Phi(r_{ij}) \right] &+ F(\rho_i) \\ \rho_i &= \sum_{j \in N_i} f(r_{ij}) \end{cases} \tag{3.11}$$

EAM est donc une famille de potentiels qui se distingue par le choix des fonctions Φ, F et ρ. Il existe un très grand nombre de variantes pour ces fonctions. Elles peuvent notamment être tabulées [145] afin d'accélérer les calculs au détriment de la précision. Nous allons décrire la version Sutton-Chen [130] utilisée pour des métaux comme le cuivre.

$$\begin{cases} \Phi(r) &= \epsilon \left(\frac{a_0}{r}\right)^l \\ F(\rho) &= -c\epsilon\sqrt{\rho} \\ \rho_i &= \sum_j \left(\frac{a_0}{r_{ij}}\right)^m \end{cases} \tag{3.12}$$

Les paramètres ϵ, c et a_0 sont déterminés à partir des paramètres d'équilibre et d'énergie d'une lattice cubique faces centrées correspondant au matériau. Les paramètres m et l sont des paramètres sans dimension avec $l > m > 0$. Ils permettent de modéliser différents métaux en adaptant autant que possible les constantes élastiques, voir table 3.2.

Element	m	n	$a_0(A)$	$\epsilon(eV)$	c
Ni	6	9	3.52	157.07	39.432
Cu	6	9	3.61	123.82	39.432
Rh	6	12	3.80	4937.1	144.41
Pd	7	12	3.89	479.0	108.27
Ag	6	12	4.09	2541.5	144.41
Ir	6	14	3.84	2448.9	334.94
Pt	8	10	3.92	198.33	34.408
Au	8	10	4.08	127.93	34.408
Pb	7	10	4.95	5576.5	45.778
Al	6	7	4.05	331.47	16.399

Table 3.2 | Paramètres pour un potentiel Sutton-Chen selon la nature des atomes [130]

.

Cependant, à cause de son caractère centrosymétrique, les résultats obtenus pour des métaux comme l'étain est faussée à cause des liaisons chimiques qui ne sont pas prises en compte par un terme angulaire.

Potentiel MEAM

Nous avons vu que le potentiel EAM prenait en compte dans son calcul tous les atomes voisins à chaque atome car ceux-ci influencent l'énergie potentielle. Le potentiel MEAM introduit par Daw et Baskes [37] se différencie du potentiel EAM en prenant également en compte les dépendances angulaires avec les atomes voisins et l'obstruction d'une interaction entre deux atomes par un troisième (écrantage). Bien que complexe, nous détaillons ce potentiel car c'est le potentiel retranscrivant le plus fidèlement les interactions entre les atomes des matériaux utilisés des simulations présentées dans la section 3.5.

Comme pour le potentiel EAM, le potentiel MEAM, voir l'équation 3.13, est composé d'une énergie de paire (Φ) et d'un terme lié à la densité électronique ($F(\rho_i)$). On note E_i l'énergie potentielle de l'atome i et N_i la liste des atomes voisins de l'atome i :

$$E_i = \sum_{j \in N_i} \left[\frac{1}{2} \Phi(r_{ij}) \right] + F(\rho_i). \tag{3.13}$$

Φ est obtenue comme étant la somme de l'énergie calculée, notée E_i^μ, et de la fonction F, avec comme paramètre la densité atomique de l'atome i notée ρ_i.

$$\Phi(r_{ij}) = \frac{2}{Z}(E_i^\mu(r_{ij}) - F(\rho_i/Z)). \tag{3.14}$$

Soit $a(r) = \alpha(\frac{r}{r_0} - 1)$, r_0 une constante et δ une autre constante, E^μ s'écrit alors sous la forme :

$$E^\mu(r) = -E_0.e^{-a(r)}.\left(1 + a(r) + \delta\frac{r}{r_0}[a(r)]^3\right). \tag{3.15}$$

Le terme $F(\rho_i/Z)$ est défini par ($A.E_0$ une constante) :

$$F(x) = A.E_0.x.ln(x).$$

Soient $R_{ij} = r_{ij}/r_0 - 1$ et $\rho_i = \sum_j \rho_i(r_i j)$, on obtient alors l'écriture suivante :

$$\rho_i(r_{ij}) = \sqrt{s_0.t_0.e^{-\beta_0.R_{ij}} + s_1.t_1.e^{-\beta_1.R_{ij}} + s_2.t_2.e^{-\beta_2.R_{ij}} + s_3.t_3.e^{-\beta_3.R_{ij}}}.$$

$(s_i)_{i\in\{0,1,2,3\}}$, $(\beta_i)_{i\in\{0,1,2,3\}}$ et $(t_i)_{i\in\{0,1,2,3\}}$ étant des constantes définies comme paramètres du potentiel.

Terme d'écrantage :

Le potentiel MEAM prend en compte le fait qu'un troisième atome puisse faire écran durant le calcul d'une interaction entre deux atomes. Ce phénomène est illustré par la Figure 3.4.

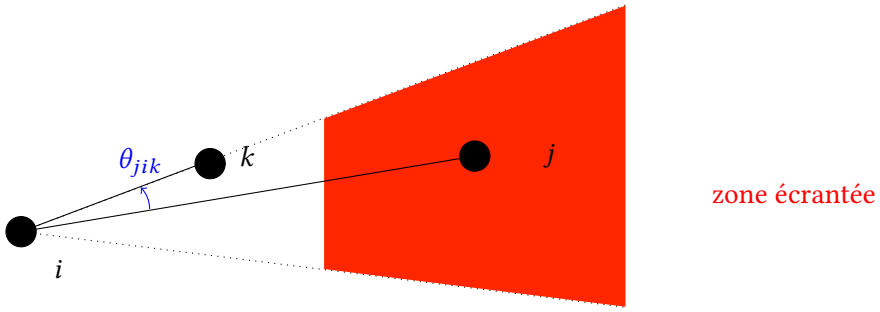


Figure 3.4 | Représentation de l'interaction entre l'atome *i* et *j* perturbée par l'atome k. Cette perturbation atténue cette interaction.

Dans ce cas, si l'atome k (voisin des atomes i et j) est positionné entre l'atome *i* et j, l'interaction entre l'atome *i* et *j* va diminuer, voire devenir nulle si l'angle $\theta_{jik} = \widehat{jik}$ est proche de 0. Pour prendre en compte ce phénomène, un terme d'écrantage noté S_{ij} est introduit dans l'équation 3.13. Il sera multiplié par Φ et ρ_i. Le terme d'écrantage est obtenu à partir du terme C :

$$C_{ikj} = \frac{2(X_{ik} + X_{kj}) - (X_{ik} - X_{kj})^2 - 1}{1 - (X_{ik} - X_{kj})^2}.$$

Avec $X_{ik} = (\frac{r_{ik}}{r_{ij}})^2$ et $X_{kj} = (\frac{r_{kj}}{r_{ij}})^2$, le terme d'écrantage s'écrit sous la forme de l'équation 3.16.

$$S_{ij} = \Pi_{k\neq i,j} S_{ikj}. \tag{3.16}$$

Avec :

$$S_{ikj} = \begin{cases} 0 & si \quad C_{ikj} \leq Cmin \\ exp\left[-\frac{Cmax - C_{ikj}}{C_{ikj} - Cmin}\right] & si \quad Cmin < C_{ikj} < Cmax \\ 1 & si \quad C_{ikj} \geq Cmax \end{cases} \tag{3.17}$$

Cmin et Cmax sont des constantes définissant la zone d'écrantage. Lorsque le terme d'écrantage est intégré au potentiel, l'énergie de paire Φ est redéfinie comme étant égale à $\Phi(r_{ij}) * S_{ij}$.

Termes angulaires :

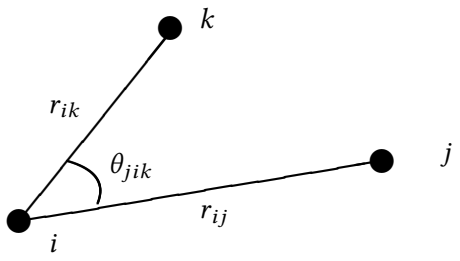


Figure 3.5 | Représentation de l'interaction entre les atomes i et j perturbée par l'atome k.

Comme dit précédemment, le potentiel MEAM prend en compte les dépendances angulaires lors du calcul des ρ_i (voir Figure 3.5).

$$\rho_i = \frac{2.\rho_i^{(0)}}{1 + e^{-\Gamma}}. \tag{3.18}$$

Avec $\rho_i^{(0)} = \sum_{j \in N_i} e^{-\beta^{(0)}(\frac{r_{ij}}{r^0} - 1)}$ représente la densité électronique de symétrie sphérique. Lors du calcul de Γ, les dépendances angulaires proviennent des termes $\rho_i^{(1,2,3)}$ représentant les contributions non sphériques à la densité. On pose :

$$\Gamma = \sum_{h=1}^{3} t^{(h)} \left(\frac{\rho_i^{(h)}}{\rho_i^{(0)}} \right)^2. \tag{3.19}$$

Les termes $\rho_i^{(h)}$ sont détaillés par les équations allant de 3.20 à 3.23. Ils sont obtenus à partir des termes des harmoniques sphériques utilisés par Biswas et Hamann. À noter que les paramètres $(t^{(h)})_{h=1,2,3}$ sont des constantes.

$$\rho_i^{(0)} = \sum_{j \in N_i} S_{ij} \rho_i^{a(0)}(r_{ij}). \tag{3.20}$$

$$(\rho_i^{(1)})^2 = \sum_{\alpha} \left(\sum_{j \in N_i} S_{ij} \rho_i^{a(1)}(r_{ij}) \frac{\bar{r_{ij}}^{\alpha}}{r_{ij}} \right)^2 \Bigg|_{\alpha = x,y,z}. \tag{3.21}$$

$$(\rho_i^{(2)})^2 = \sum_{\alpha,\beta} \left(\sum_{j \in N_i} S_{ij} \rho_i^{a(2)}(r_{ij}) \frac{\bar{r_{ij}}^{\alpha} \bar{r_{ij}}^{\beta}}{r_{ij}^2} \right)^2 \Bigg|_{\alpha,\beta = x,y,z} - \frac{1}{3} \left(\sum_{j,k=1(\neq i)}^{N} \rho_i^{a(2)}(r_{ij}) \right)^2. \tag{3.22}$$

$$(\rho_i^{(3)})^2 = \sum_{\alpha,\beta,\gamma = x,y,z} \left(\sum_{j \in N_i} S_{ij} \rho_i^{a(3)}(r_{ij}) \frac{\bar{r_{ij}}^{\alpha} \bar{r_{ij}}^{\beta} \bar{r_{ij}}^{\gamma}}{r_{ij}^3} \right) - \frac{2}{5} \sum_{\alpha} \left(\sum_{j \in N_i} \rho_i^{a(1)}(r_{ij}) \frac{\bar{r_{ij}}^{\alpha}}{r_{ij}} \right)^2 \Bigg|_{\alpha = x,y,z}. \tag{3.23}$$

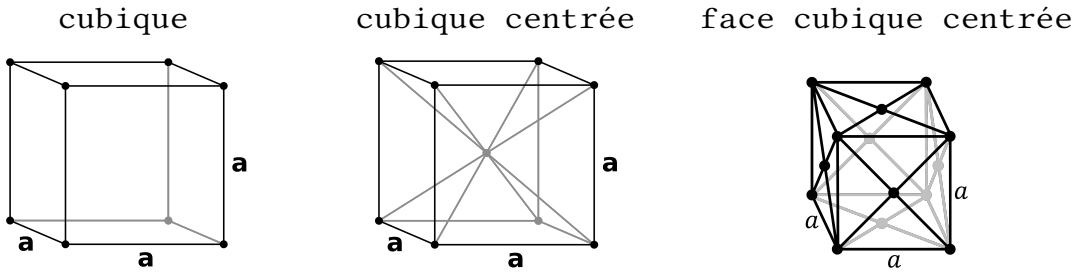
Autres Potentiels

Les potentiels de paire, Eam et Meam sont les plus courants lors des simulations des métaux en physique de la matière condensée. Il en existe d'autres types comme les potentiels à 3-corps [126, 127, 143] avec par exemple les potentiels de Vashishta [138]. Comme les potentiels Eam et Meam, il existe d'autres potentiels à N-corps comme les potentiels "Bond order potential" ou les potentiels de Tersoff [133] et Brenner [20]. Afin d'accélérer le calcul de potentiels très coûteux, une solution envisageable est de tabuler certaines fonctions [145], c'est-à-dire d'interpoler les résultats à partir d'un tableau de résultats, au détriment de la précision. Cette solution peut s'avérer utile afin de produire des résultats préliminaires à moindre coût. Des potentiels comme le potentiel Spectral Neighbor Analysis Potential (SNAP) [135] sont basés sur des techniques de machine learning afin de reproduire les énergies, les forces et les tenseurs de contrainte d'un grand nombre de petites configurations d'atomes pour mieux appréhender les phénomènes physiques complexes.

Phénomènes physiques et simulation numérique

Les phénomènes physiques sont presque inobservables à l'échelle atomique via des expérimentations. De plus, ces expérimentations ne permettent pas d'observer la totalité du phénomène et donc d'en comprendre le fonctionnement. L'éjecta de matière, paragraphe 3.5.2, et l'impact d'une nano-goutte sur une surface libre, paragraphe 3.5.3 sont par exemples des phénomènes physiques encore méconnus à l'échelle atomique et nécessitent des simulations particulièrement complexes et coûteuses. C'est pourquoi, les physiciens s'appuient sur des simulations numériques. Pour réaliser les simulations numériques, celles-ci ont besoin d'une disposition initiale du système d'atomes . Pour cela, les configurations initiales sont généralement obtenues à partir de la génération de cristaux parfaits, voir paragraphe 3.5.1.

Structure cristalline



Une structure cristalline est définie par l'arrangement des atomes dans un cristal. Un solide cristallin est constitué par la répétition dans l'espace d'un motif atomique ou moléculaire, appelé maille. L'ensemble de ces mailles forme alors un réseau cristallin. En dynamique moléculaire, les gaz ou métaux sont généralement générés à partir d'une maille cubique centrée ou cubique à faces centrées et thermalisés pendant un certain nombre de pas de temps afin de reproduire la physique du matériau pour une température donnée. Dans ce cas on applique des conditions périodiques au bord du domaine afin de reproduire le comportement du matériau à l'échelle macroscopique. Les conditions périodiques consistent à répliquer les atomes aux bords du domaine de la simulation de l'autre côté du domaine par un décalage de la position cartésienne concernée.

La densité d'atomes d'un cristal parfait est alors homogène dans tout le volume du cristal. De plus, la taille de la maille est généralement choisie afin que les atomes soient à l'équilibre, c'est-à-dire que la somme des forces exercées sur l'atome est nulle. Les atomes vont alors vibrer autour de

leurs positions initiales s'il n'y a pas de contraintes externes, comme un choc ou une déformation, appliquées au cristal. Dans ce cas, la disposition des atomes évolue peu au cours du temps. Ces simulation sont alors dites «statiques».

Éjecta de matière

Dans ce manuscrit, nous nous intéressons aux simulations numériques visant à reproduire des phénomènes complexes dont la densité d'atomes est extrêmement variable spatialement et temporellement. Ces simulations sont dites «hétérogènes» et «dynamiques». Notre étude portant sur l'optimisation d'un code de DM sur les supercalculateurs a pour objectif de réaliser de telles simulations, c'est pourquoi la validation se fera principalement sur deux simulations.

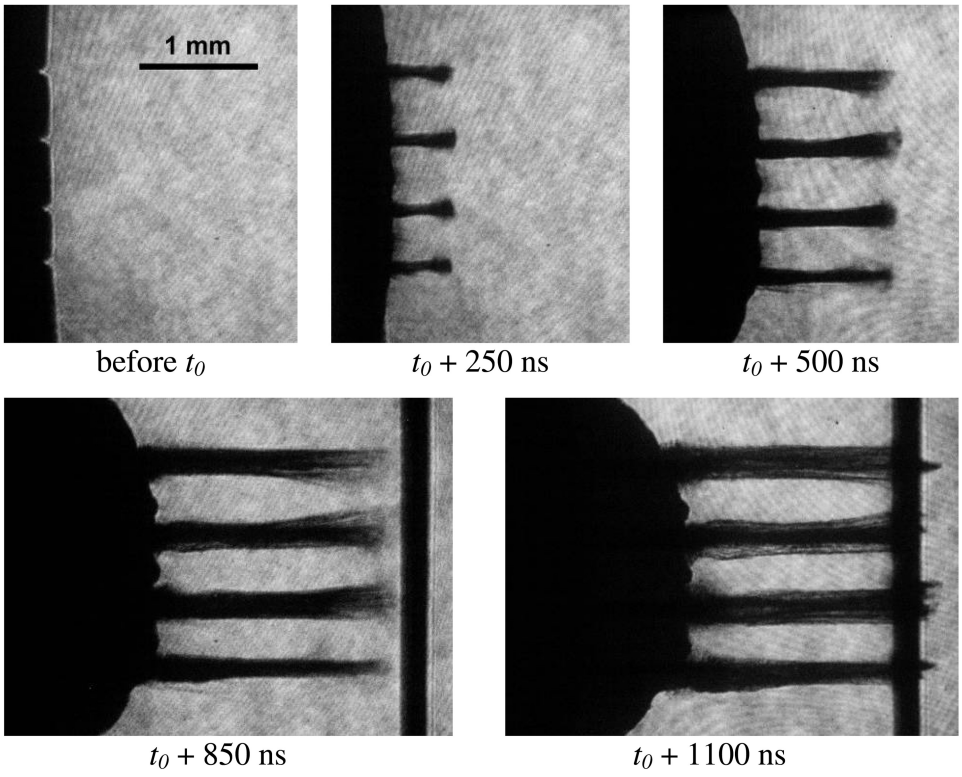


Figure 3.6 | 5 ombroscopies sont présentées dans l'article [40] derrière une plaque d'étain d'une épaisseur de 300 µm, avec des rainures triangulaires dont le demi-angles est de 30° dans la surface libre. Un choc laser de 73 GPa est appliqué au temps t_0 sur un point de 3.6 mm de diamètre sur la face avant (à gauche). Sa réflexion par les rainures génère quatre micro-jet sous vide, vers la droite. Leurs vitesses de pointe (3.84 km/s) et celle de la surface libre plane (1.39 km/s) peuvent être déduites de cette séquence. La dernière image montre l'impact des jets contre une feuille de cuivre placée à 3 mm de l'échantillon.

La première simulation reproduit le phénomène d'éjection de la matière, aussi nommé micro-jet. Lorsqu'un métal comprenant une surface libre et rugueuse est mis sous choc, c'est-à-dire qu'une onde de choc parcourt le métal, l'interaction entre la surface libre et le front de choc crée une perturbation de la surface selon un processus de Richtmyer-Meshkov [117, 91]. Lorsque le métal comprend une surface libre avec des rainures espacées régulièrement, l'instabilité prend la forme d'une nappe 2D s'effritant au fur et à mesure qu'elle s'étend. Cette expérience est illustrée par la figure 3.6, elle montre dans ce cas la génération d'un micro-jet.

Concernant l'initialisation, l'ensemble thermodynamique choisi est NVT afin de thermaliser le système d'atomes pour une température donnée, comme décrit par la figure 3.7 de l'article [47]. Par réplication d'une maille, une plaque de dimensions L_x, L_y et L_z est générée, c'est dans cette

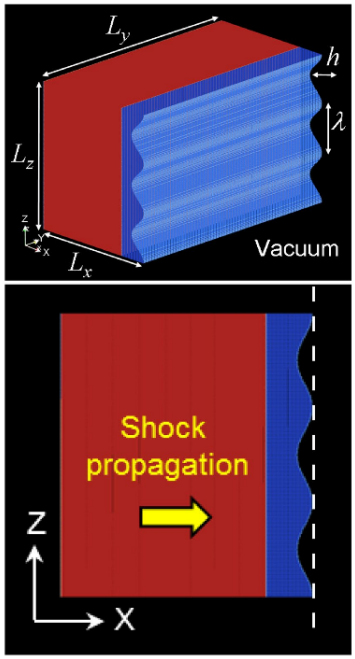


Figure 3.7 | Description du système [47] simulé avec une rugosité sinusoïdale définie par h (profondeur des rainures) et λ (largueur des rainures), le bloc en rouge est obtenu en répliquant une maille adaptée à l'élément selon L_x, L_y et L_z. L'emplacement de la surface libre au moment de la simulation initiale est représenté par une ligne pointillée blanche.

plaque que l'onde de choc va se propager. Une surface comprenant un motif sinusoïdal, dont les oscillations sont d'une profondeur h et d'une largeur λ est placée à droite de la plaque. Des conditions périodiques selon les axes y et z sont imposées. Selon la direction x, on applique une condition libre à droite et un mur à gauche, c'est-à-dire qu'on impose un potentiel de paire entre l'atome et sa projection sur la limite du domaine. Ce potentiel est uniquement composé d'un terme répulsif.

Après thermalisation du système, la simulation est effectuée dans l'ensemble NVE. Les conditions aux limites restent inchangées. Le matériau est alors projeté contre un piston (vers la gauche du domaine), c'est-à-dire qu'un front de choc va se former et se propager de gauche à droite. La figure 3.8 présente l'avancement d'une telle simulation au cours du temps et elle est réalisée par le logiciel ExaSTAMP [30, 29] au CEA.

Impact d'une nano-goutte sur une surface solide

Pour valider mon étude, une deuxième simulation est aussi introduite. Celle-ci permettra notamment de mettre en relief les résultats obtenus pour la simulation d'un micro-jet. De plus, une deuxième simulation permet d'éviter des erreurs de raisonnement en dissociant les optimisations spécifiques à un scénario des optimisations génériques. Le deuxième scénario reproduit le phénomène d'un impact d'une nano-goutte sur une surface solide, aussi nommé éclaboussement [148, 79].

Ce phénomène est répandu dans de nombreuses applications techniques. Les simulations permettent par exemple de comprendre l'évolution des jets d'encre durant l'impression d'un document, l'utilisation de sprays, le traitement thermique des aciers, le refroidissement d'une surface chaude comme l'aube d'une turbine, les cylindres dans les laminoirs pour la production d'acier, les impacts de laser ou les puces semi-conductrices. Lors de la fabrication de micro-matériel lors de la soudure de cartes de circuits imprimés ou de circuits micro-électroniques. C'est en 1908 que Worthington et Mason [146] décident d'investiguer les impacts selon différentes configurations (projectile rentrant

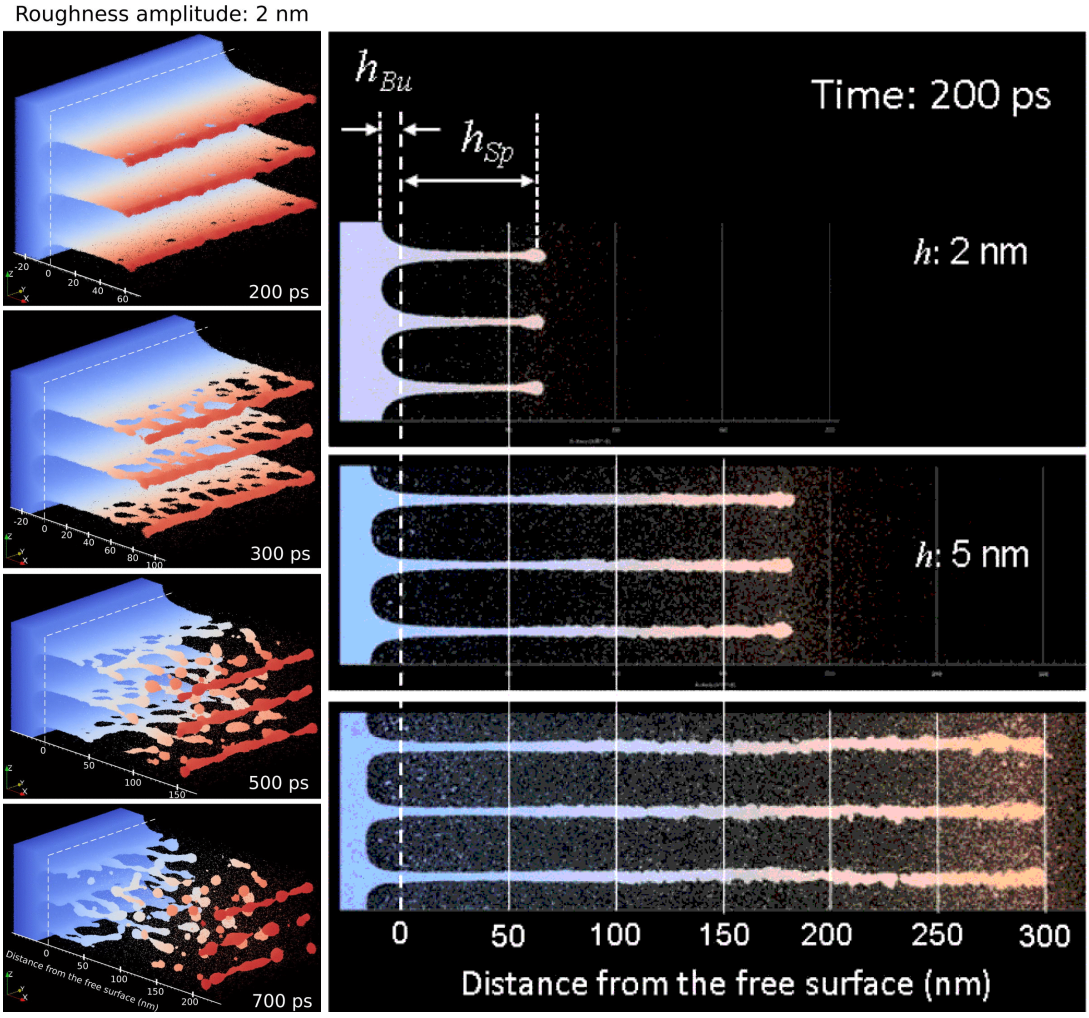


Figure 3.8 | Simulation d'un micro-jet [47] selon les paramètres suivants : la maille de base est une maille BCC d'étain de paramètre 3.7 Å($L_x \times L_y \times L_z$ = 111 nm × 148 nm × 92.5 nm) formant un ensemble de 60 millions d'atomes. La mise en vitesse est d'environ 1570m/s et la pression du système est de 65GPa.

dans une plaque de blindage, impact de goutte d'eau sur de l'eau, ...). Une centaine d'années plus tard ce phénomène n'est toujours pas totalement maîtrisé.

Dans notre cas, nous nous intéresserons à la simulation d'impact d'une nano-goutte d'étain [10] sur une surface d'étain ou sur une condition aux limites de type "mur", voir figure 3.9. Le dénouement de l'impact de la gouttelette dépend de la vitesse de la goutte, de son angle d'impact avec le mur ou la surface, de la taille de la goutte, des propriétés du matériau (densité, viscosité, visco-élasticité et autres effets non newtoniens), de la tension de surface, de la rugosité et de la mouillabilité de la surface. Il est à noter que ce type de simulation numérique en DM n'est actuellement réalisable par les logiciels qu'à de petites échelles.

Finalement, la simulation d'un éjecta de matière et de l'impact d'une nano-goutte d'étain sur une surface solide sont tout deux représentatifs des simulations actuellement en cours sur les supercalculateurs du CEA. Ces simulations vont ainsi nous permettre d'évaluer les optimisations réalisées dans ce manuscrit sur la répartition de la charge.

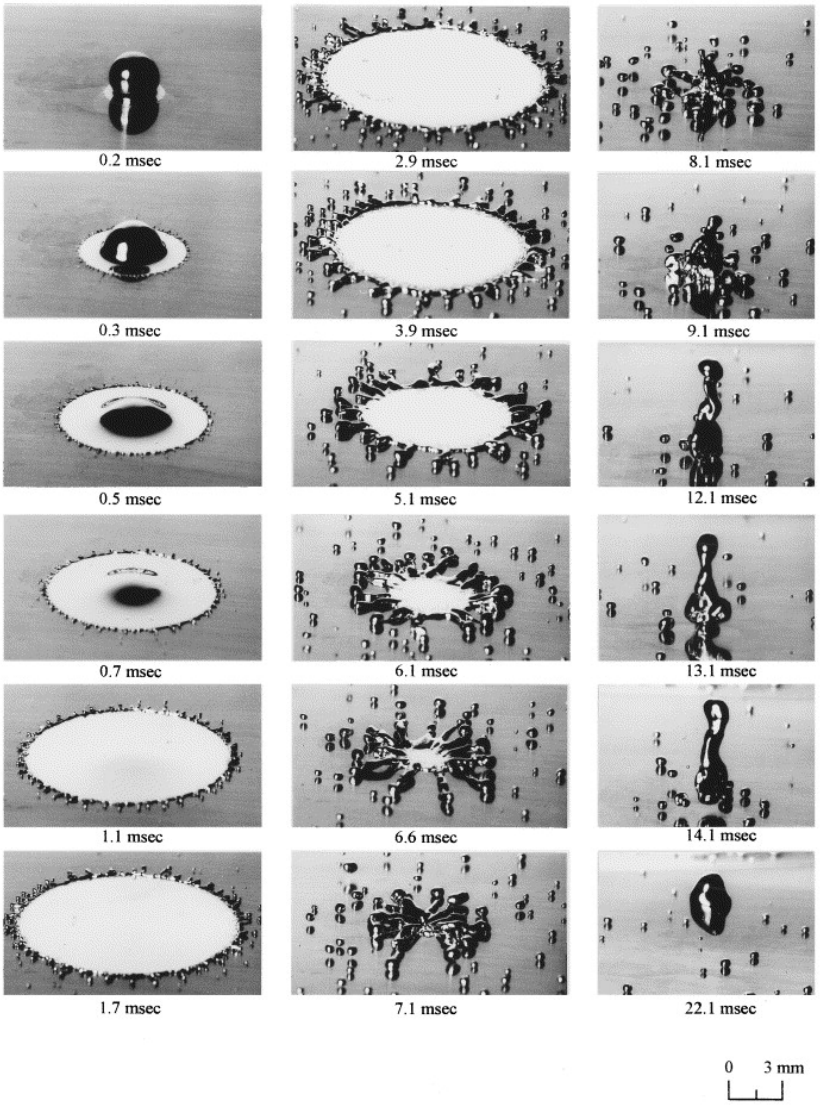


Figure 3.9 | Aziz, Shiraz et Chandra proposent durant leur étude en 2000 une série de photographies de l'éclaboussure d'une gouttelette d'étain fondu durant l'impact avec une vitesse initiale de 4 m/s sur une surface en acier inoxydable à une température de 240°C.

4 Le Calcul Haute Performance

Au cours des 50 dernières années, la puissance de calcul des supercalculateurs n'a cessé de progresser [111]. Ainsi, de nombreux domaines de la physique ont pu fortement évoluer grâce aux nouvelles capacités des simulations numériques. Cette évolution engendrée par cette puissance de calcul a notamment rendu possible la modélisation de phénomènes plus complexes tout en améliorant leur précision sur des domaines spatiaux toujours plus étendus. Ainsi, les simulations numériques permettent aujourd'hui de reproduire fidèlement certaines expériences physiques. Si bien que ces dernières sont parfois remplacées purement et simplement par des simulations moins onéreuses. Par exemple, il est possible de concevoir par simulation le refroidissement d'un réacteur en hydrodynamique ou d'étudier l'écoulement de l'air autour d'une aile d'avion en aérodynamique.

Afin de comprendre comment la puissance de calcul des supercalculateurs a augmenté, nous présentons leur évolution matérielle. Le but est de comprendre l'impact de cette évolution sur la conception des codes de simulations et l'évolution des modèles de parallélisation. Actuellement, un supercalculateur est un ensemble de Nœuds de calcul communiquant entre eux via un réseau d'interconnexion. Un Nœud est composé d'une mémoire vive, de périphériques d'entrée-sortie et d'un processeur. Chaque processeur contient un ensemble de cœurs chargés de traiter des flux d'opérations indépendamment les uns des autres. Après une présentation dans le paragraphe 4.1.1 de l'architecture des processeurs mono et multicœurs constituant un Nœud, nous aborderons les notions de hiérarchie mémoire dans le paragraphe 4.1.2. Une description du traitement des calculs par les cœurs est effectuée dans le paragraphe 4.1.3. Une attention particulière est apportée au traitement vectoriel des calculs, cf. paragraphe 4.1.4. Nous abordons également très brièvement les capacités des cartes graphiques car bien qu'elles commencent à être présentes dans de nombreux supercalculateurs, cf. paragraphe 4.1.5, nous ne les utiliserons pas.

Pour comprendre comment les ressources matérielles d'un supercalculateur sont exploitées, nous exposons les aspects logiciels permettant d'écrire un code de simulation en un flux d'instructions exécutées sur un supercalculateur. L'utilisation d'algorithmes parallèles dans les codes de simulation permet de répartir les calculs entre les Nœuds d'un supercalculateur. Pour cela, les développeurs s'appuient sur des interfaces de programmation pour paralléliser les calculs. Les calculs sont répartis entre les Nœuds via une parallélisation en mémoire distribuée, cf. paragraphe 4.2.1. Au sein de chaque Nœud, une parallélisation en mémoire partagée est généralement privilégiée pour répartir les calculs entre les cœurs, cf. paragraphe 4.2.2. Une dernière forme de parallélisme située au niveau de chaque cœur est la vectorisation, aussi nommée parallélisation de données, cf. paragraphe 4.2.3.

Évolutions matérielles

L'architecture d'un Nœud de calcul

Dans ce paragraphe nous allons décrire brièvement l'architecture des processeurs composant les supercalculateurs et correspondant à un Nœud de calcul. Les premiers supercalculateurs étaient composés d'un seul processeur monocœur d'une puissance de calcul faible, nommé Unité Centrale de Traitement (Uct). Ils effectuaient peu d'opérations par unité de temps. En 1963 le Cea disposait d'un calculateur IBM 7030 « STRETCH » [125]. Le centre de calcul figurait alors parmi les plus puissants existant à l'extérieur des États-Unis avec une puissance de l'ordre de 0.37 Megaflops.

Le processeur a pour rôle d'effectuer les calculs. Il communique avec la mémoire vive ainsi qu'avec les périphériques d'entrées-sorties. La mémoire vive (ram) est dite volatile, les données sont uniquement stockées durant le temps de la simulation.

Un processeur (Uct) est composé en partie d'Unités Arithmétiques et Logiques (Alu) effectuant des opérations élémentaires comme les additions ou les multiplications. Il contient également des caches mémoire (petits espaces de stockages rapides d'accès) et des registres permettant de stocker les données récupérées depuis la ram ainsi que les instructions à exécuter. L'évolution des processeurs est directement liée à la capacité à miniaturiser des transistors (loi de Moore [98] : *le nombre de transistors dans un circuit intégré double approximativement tous les deux ans*).

La première étape vers nos supercalculateurs actuels a été la conception et le développement des architectures dites Multiprocesseur Symétrique à mémoire Partagée (Smp). Elle consiste à juxtaposer deux ou plusieurs processeurs. Les processeurs partagent la totalité de la mémoire vive, c'est pourquoi on parle de systèmes à mémoire partagée. Il existe deux types de machines multiprocesseurs se différenciant l'une de l'autre par leurs manières d'accéder et de répartir la mémoire selon les processeurs : Uniform Memory Access (Uma) et Non Uniform Memory Access (Numa).

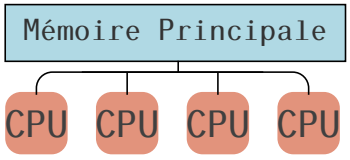


Schéma 4.1 | Schéma d'une machine composée de 4 processeurs (Uct) pour une architecture mémoire Uniform Memory Access.

Une architecture mémoire Uma, illustrée par le schéma 4.1, consiste à relier les processeurs à une unique mémoire principale en utilisant soit un bus mémoire, soit des commutateurs transversaux ou un réseau à plusieurs étages. Le principal défaut des systèmes mémoire Uma résulte de l'engorgement lorsque de nombreux processeurs effectuent des accès concurrentiels aux mêmes données.

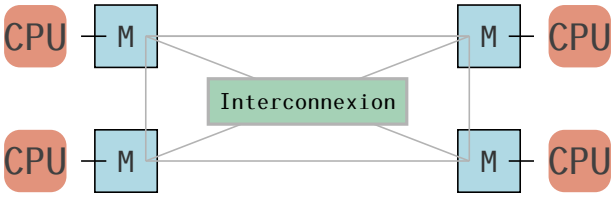


Schéma 4.2 | Schéma d'une machine composée de 4 processeurs avec une architecture mémoire Non Uniform Memory Access. M correspond à une partie de la mémoire vive distribuée.

Le système Numa, illustré par le schéma 4.2, a été adapté pour les systèmes composés d'un grand nombre de processeurs en palliant les limites des architectures Uma. Dans le contexte d'un système Numa, la mémoire vive est distribuée entre les processeurs formant ainsi un ensemble de nœuds Numa. Les nœuds Numa communiquent entre eux en échangeant des données via par exemple un réseau d'interconnexion comme l'Infiniband® [107] ou le Bull eXascale Interconnect (Bxi) [41]. Le temps d'accès d'un processeur à une donnée d'un autre nœud Numa est variable et dépend du nœud Numa à considérer. C'est pourquoi les accès sont dits non uniformes, on parle généralement d'effet Numa. À noter que les systèmes Numa composent actuellement la totalité des supercalculateurs.

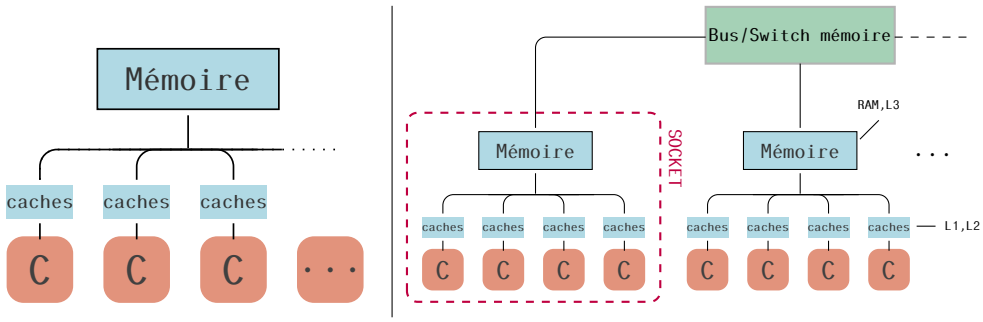


Schéma 4.3 | Architecture mémoire Uma (gauche) versus Numa (à droite) d'un processeur multicœurs.

Depuis les années 2010, la tendance est d'utiliser des processeurs multicœurs. Les cœurs sont ce qu'on appelait précédemment processeurs. Chaque cœur est composé d'Alu, de registres et de caches mémoire permettant d'exécuter indépendamment des autres cœurs un fil d'instruction. Les cœurs partagent généralement le cache de plus haut niveau alors que le cache L1 est généralement local au cœur (plus rapide d'accès mais plus petit). Ainsi, les processeurs à architectures multicœurs s'apparentent à une machine Uma ou Numa, voir schéma 4.3. Dans la suite de ce travail, un nœud Numa sera nommé une Socket lorsque l'on considère un processeur multicœurs et un Nœud (noté N) pour une machine multiprocesseurs. Un Nœud est donc un ensemble de Sockets. Les Sockets d'un même Nœud sont reliées entre elles par des bus et communiquent via des protocoles spécialisés.

Chaque choix de composant va influencer les performances des supercalculateurs. Par exemple on peut choisir d'utiliser peu de cœurs avec une grande fréquence de calcul ou plutôt opter pour davantage de cœurs moins rapides mais nécessitant davantage de synchronisation. Est-il plus intéressant d'augmenter le nombre de cœurs par processeur ou le nombre de processeurs ? D'autres problèmes annexes mais cruciaux voient le jour comme la consommation électrique des processeurs, leur refroidissement ou l'espace de stockage (Big DATA).

En juin 2019, le supercalculateur le plus puissant était le calculateur SUMMIT [111] du laboratoire national d'Oak Ridge dans le Tennessee, avec ses 2 397 824 cœurs (3.07GHz) ainsi que ses 27 648 processeurs graphiques Nvidia Volta. Il a ainsi pu atteindre une puissance de calcul environ égale à 143,500 **PFlops** [1] soit de l'ordre de 10^8 fois plus rapide que Cray-1.

Hiérarchie mémoire des processeurs multicœurs

Au fur et à mesure des avancées technologiques des processeurs, la mémoire de ceux-ci a évolué en devenant de plus en plus hiérarchique (plusieurs niveaux). Par exemple, dans les systèmes Uma et Numa, un niveau mémoire nommé cache mémoire est présent. Ce sont des stockages mémoire de

[1] un million de milliard d'opérations par seconde

l'ordre de quelques Mega Octets (MO) dont les temps d'accès aux données sont plus rapides qu'à la mémoire vive du Socket, cf. figure 4.4. En raison de sa rapidité d'accès, lorsqu'une donnée est chargée depuis la mémoire vive du Socket, celle-ci est «automatiquement» copiée dans les caches mémoire, car cette donnée est susceptible d'être réutilisée. En pratique, c'est un fragment contigu de la mémoire qui est copié dans l'une des lignes du cache mémoire du niveau 1 (nommé L1). Plus le niveau de cache est élevé, plus la capacité mémoire est importante et plus la latence est élevée. C'est pourquoi le cache L1 est le premier que l'on commence à remplir. Si le cache est totalement rempli par de nouvelles lignes de cache, les plus anciennes sont déplacées dans le cache de niveau supérieur. Afin de garantir la cohérence entre les lignes de cache des différents niveaux, des protocoles de transmission entre les caches existent tels que le "*cache coherence*" Numa (ccNuma).

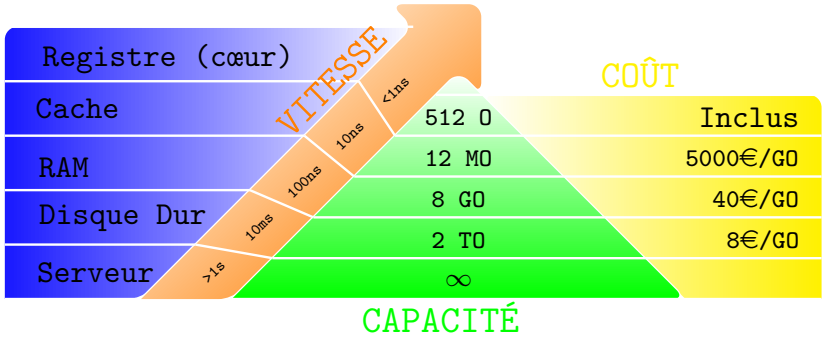


Figure 4.4 | Hiérarchie des accès mémoire selon la latence (temps), la capacité mémoire (octets) et le coût (euros). Les valeurs sont données à titre indicatif.

La latence des accès à la mémoire est corrélée à la proximité entre celle-ci et les Unités Arithmétiques et Logiques, cf. la figure 4.4. Cependant, plus les accès mémoire sont rapides et plus l'espace de stockage en question est faible. À noter que dans le cas d'une architecture Numa, les caches L1 et L2 sont généralement propres à chaque cœur et le cache L3 est commun aux cœurs présents dans une Socket. Ainsi, pour minimiser les effets Numa, tant qu'une donnée doit être réutilisée, celle-ci doit être conservée au sein des caches mémoire pour un accès rapide.

Flux d'exécution

Lors de la compilation d'un code de calcul, le compilateur traduit le code (haut niveau) en langage machine/assembleur qui est une suite d'instructions (dépendant de la machine) que l'on nomme flux d'instructions. Actuellement, les compilateurs effectuent des pré-traitements complexes afin d'optimiser un code sur une architecture cible. En pratique, la compilation est trop complexe pour être décrite précisément car elle est dépendante des compilateurs et des architectures qui évoluent chaque année. En théorie, dans le contexte d'un Processeur à jeu d'instructions réduit (Risc), une instruction est effectuée en 5 étapes :

1. IF : Chargement de l'instruction à exécuter;
2. ID : Décode l'instruction et adresse les registres;
3. EX : L'instruction est exécutée par les Unités Arithmétiques et Logiques;
4. MEM : Rapatriement / écriture d'une donnée entre un registre et la mémoire vive;
5. WB : Le résultat est transféré dans un registre;

Instruction	Cycle d'horloge									
	1	2	3	4	5	6	7	8	9	10
Inst. 1	*IF*	*ID*	*EX*	*MEM*	*WB*					
Inst. 2		*IF*	*ID*	*EX*	*MEM*	*WB*				
Inst. 3			*IF*	*ID*	*EX*	*MEM*	*WB*			
Inst. 4				*IF*	*ID*	*EX*	*MEM*	*WB*		
Inst. 5					*IF*	*ID*	*EX*	*MEM*	*WB*	
Inst. 6						*IF*	*ID*	*EX*	*MEM*	*WB*
	Initialisation				1 résultat par cycle					

Lors de l'exécution d'un programme, les instructions suivent un flux d'exécution séquentielle, c'est-à-dire que les instructions sont traitées par les cœurs les unes après les autres. Lorsque les instructions sont pipelinées, 5 étapes différentes de 5 instructions peuvent être effectuées simultanément. Dans le cas idéal où chaque étape dure un cycle d'horloge, voir le tableau ci-dessus, les instructions sont exécutées à un cycle d'intervalle. En effet, lorsque l'étape EX de l'instruction 1 est effectuée, l'étape ID de l'instruction 2 est également effectuée ainsi que l'étape IF de l'instruction 3. En 10 cycles d'horloge, 6 instructions ont été réalisées au lieu de 2. Pour un jeu d'instructions du type Risc, on parle de pipeline classique de Risc. Néanmoins, si la séquence comporte des instructions conditionnelles (ex: if(cond) ou else en C++), le pipeline des instructions est brisé. À noter que les processeurs actuels prévoient le cas le plus probable et créent le pipeline correspondant à cette possibilité.

Instruction Simd

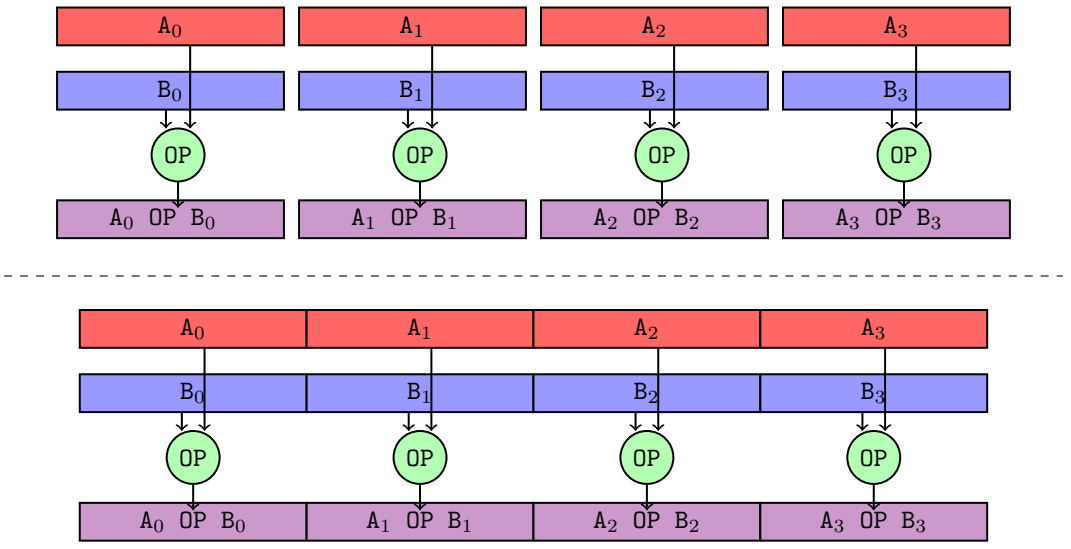


Figure 4.5 | Traitement SISD (haut) et Simd (bas) d'une opération OP prenant les blocs des tableaux A et B.

Dans les années 1980, les calculateurs ont évolué en effectuant des calculs vectoriels [49]. On peut par exemple citer le calculateur Cray-1 [118]. Selon la taxonomie de Flynn [53] adaptée aux calculateurs par Duncan et al. [46], les architectures des supercalculateurs sont classées en 4 classes

en fonction de leur capacité à effectuer un ou plusieurs flux d'instructions sur un ou plusieurs flux de données. Lorsque le calculateur exécute un flux d'instructions sur un flux de données, le calculateur utilise des instructions dites scalaires ou Simple Instruction Simple Donnée (Sisd) (architecture de Von Neumann [140]). Lorsque les instructions sont appliquées sur plusieurs flux de données comme c'est le cas pour les processeurs vectoriels, elles sont dites Simple Instruction Multiple Données (Simd). Les données sont chargées dans des registres dits vectoriels dont la taille dépend du type d'instructions et des données (SSE, AVX, AVX512 / int, float, double). À noter que ces registres ne sont généralement pas spécialisés et peuvent contenir des entiers ou des floats. Les données sont ensuite traitées simultanément par des Alus spécialisées, même si en pratique des Alus classiques peuvent être utilisées, on parle alors de parallélisme de données, cf. figure 4.5.

Des instructions de masquage permettent notamment d'appliquer une opération sur une partie des données. De plus, comme dans le cas d'instructions SISD, les instructions Simd peuvent être pipelinées. Dans certains cas le pipelinage peut être amélioré par la technique de "chaining" des instructions, c'est-à-dire que pour effectuer : $A + B * C$, les instructions correspondant à la multiplication et à l'addition sont "fusionnées" afin d'éviter le stockage de la valeur intermédiaire $B * C$. Néanmoins ce choix incombe au compilateur et il est difficile de prévoir comment les instructions sont chaînées.

Processeur graphique

Bien que la majorité des supercalculateurs soit uniquement composé de processeurs Uct, la tendance à l'utilisation de Uct + Gpu ne cesse de s'accroître. Les Gpus ont été introduits dans le domaine du Calcul Haute Performance afin de répondre à des problématiques demandant un calcul très intensif et répétitif pour très peu de communications (entre les cœurs ou entre les cœurs et la mémoire) comme le calcul matriciel ou l'infographie. Les processeurs graphiques sont composés d'une mémoire partagée et d'un très grand ensemble de cœurs, de l'ordre de plus de 1 000 pour les plus puissants. Ils possèdent des registres vectoriels et des Alus Simd synchronisés. Les cœurs sont rassemblés en blocs de cœurs, chaque bloc de cœurs possède un cache L1 et un cache d'instructions. Les blocs de cœurs effectuent alors un flux d'instructions sur de multiples données. Les codes de calcul nécessitent d'être adaptés pour ce type de processeur. Or, nous n'exploitons pas l'utilisation de Gpus au cours de cette thèse, ceux-ci ne seront donc pas développés davantage.

Évolutions logicielles

En parallèle de la complexification de l'architecture matérielle des supercalculateurs (cf. figure 4.6), de nouveaux modèles de programmation ont été mis en place pour utiliser ces machines le plus efficacement possible.

Les supercalculateurs sont constitués d'un ensemble de Nœuds, chacun est composé d'un ou plusieurs Sockets qui sont elles-mêmes composées d'une mémoire vive, de caches et de cœurs. Les cœurs sont généralement capables d'effectuer des instructions vectorielles (Simd). Afin de prendre en compte les spécificités des supercalculateurs, 3 niveaux de parallélisme sont utilisés. Le parallélisme en mémoire distribuée permet de distribuer la mémoire et les calculs sur les différents Nœuds. La parallélisation en mémoire partagée permet une parallélisation intra-Nœud. Le dernier niveau est la vectorisation permettant d'appliquer une même opération à plusieurs données. Pour illustrer ces parallélisations, des exemples en C++ sont présentés en adaptant l'algorithme pour obtenir le résultat R de la somme des éléments d'un tableau issu du produit de deux vecteurs A et B : $R = \sum_i A_i * B_i$ (cf listing 3.1).

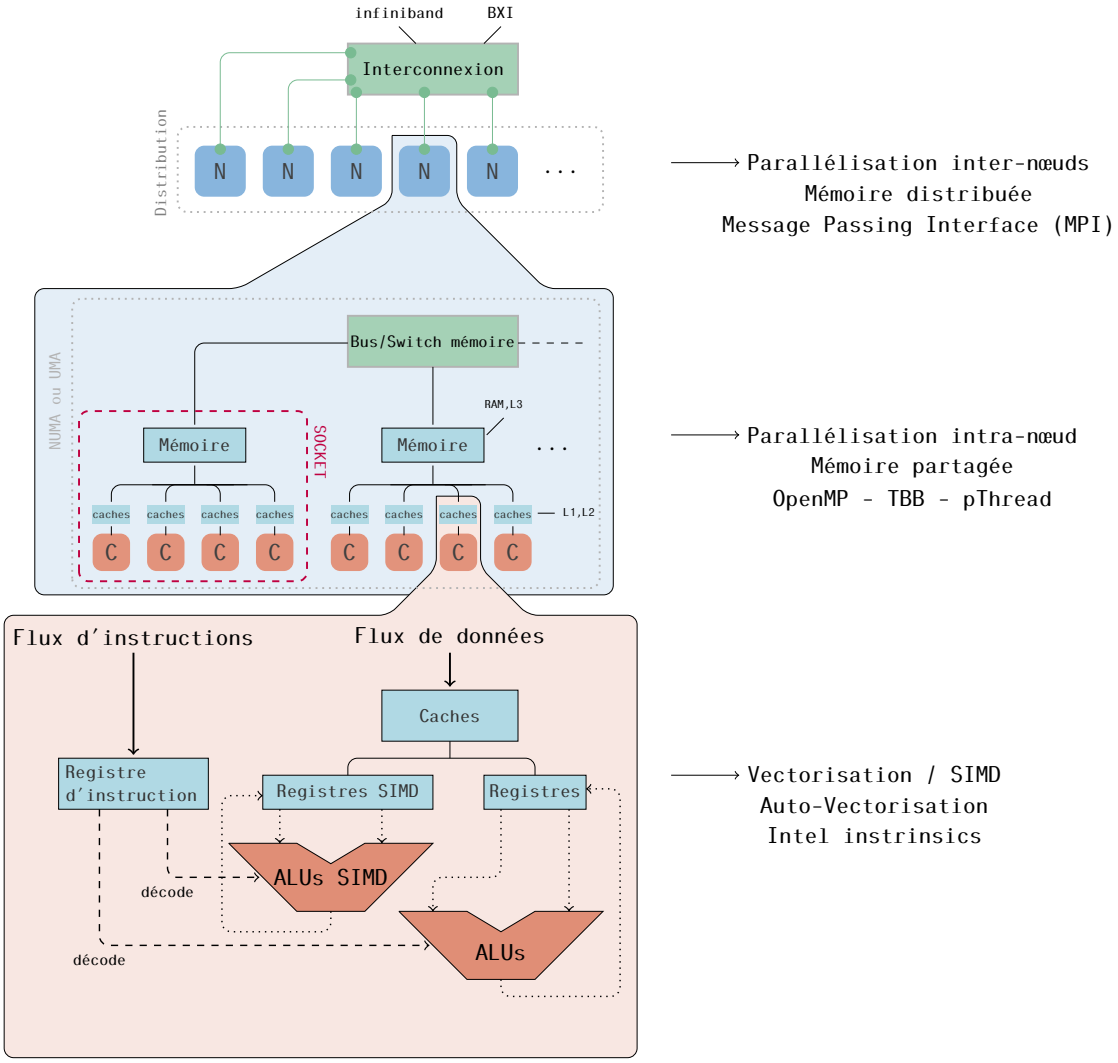


Figure 4.6 | Représentation hiérarchique de l'architecture matérielle d'un supercalculateur associée aux trois niveaux de parallélisme. Les ressources (Mémoire et Calculs) d'une simulation sont distribuées sur les Nœuds (N) de la machine. Chaque nœud Uct est un ensemble de Sockets composés d'une mémoire, de caches et de cœurs (C) pouvant accéder à la mémoire des autres Sockets. Les calculs sont répartis entre les cœurs. Chaque cœur traite son flux d'instructions sur de multiples flux de données (Simd) ou sur un seul (SISD).

```
1  std::vector<double> A,B;
2  double R(0.);
3
4  // Remplissage des tableaux
5
6  for(size_t it = 0 ; it < A.size() ; it++)
7    R += A[it] + B[it];
```

Listing 4.1 | Programme séquentiel effectuant l'addition des termes d'un produit de deux vecteurs.

Parallélisation inter-Nœuds

Dans le cas d'une parallélisation inter-Nœuds, les ressources de la simulation sont distribuées entre plusieurs processus qui ont accès aux ressources de calcul soit d'un cœur, soit d'un ensemble de

cœurs ou même d'un Nœud du calculateur. À noter que les processus n'ont pas accès aux données des autres processus. Chaque processus effectue la partie du calcul qui lui est attribuée. Si un processus doit récupérer des données ou des résultats de calculs effectués par un autre processus, ils se communiquent les informations via un système d'échange de messages

La norme la plus répandue depuis 1993 pour gérer l'envoi et la réception d'informations est le Message Passing Interface (Mpi) [31]. Dans le contexte d'un supercalculateur, les ressources de la simulation sont distribuées sur les processus Mpi. Chaque processus Mpi se différencie des autres par son rang Mpi. Les processus Mpi peuvent dialoguer avec les autres via un communicateur MPI_COMM_WORLD ou un sous-ensemble de processus Mpi via d'autres communicateurs instanciés par l'utilisateur. La norme Mpi propose notamment des communications entre deux processus Mpi (point à point) via des messages Mpi bloquants ou non bloquants : MPI_Send ou MPI_ISend et MPI_Recv ou MPI_IRecv. Ces messages sont définis par les rangs des processeurs émetteurs-récepteurs, un tag et un communicateur. Si durant la simulation un ensemble de processus Mpi a besoin de diffuser ou de récupérer une donnée ou un ensemble d'informations disponibles sur l'ensemble des processus, la norme Mpi propose également des communications collectives telles que MPI_Reduce ou MPI_Broadcast. Cependant lorsque le nombre de processus Mpi est très élevé, c'est-à-dire supérieur à 10 000 processus Mpi, les communications collectives deviennent onéreuses. C'est pourquoi, dans le cas d'un supercalculateur, on privilégie l'association d'un processus Mpi à un Nœud pour diminuer le nombre de communications. En affectant un processus Mpi par Socket, on limite également les effets Numa. En plus de cette parallélisation inter-Nœuds, chaque processus Mpi peut effectuer une parallélisation intra-Nœud (nommée **MPI+X** [68]) pour paralléliser les calculs entre les cœurs.

En reprenant l'algorithme précédent (cf listing 3.2), chaque processus Mpi se voit attribuer une partie des tableaux A et B en fonction de leurs rangs Mpi et effectue en local la somme : $\sum_i A_i * B_i$, i étant une collection d'indices attribués selon le rang Mpi. Chaque processus Mpi calcule alors une partie de la solution, une opération de réduction (collective Mpi) sur l'ensemble des processus Mpi est effectuée pour obtenir la solution globale sur le processus de rang 0.

```
1  MPI_Init(&argc, &argv);
2  int rang;
3  MPI_Comm_rank(MPI_COMM_WORLD, &rang);
4  int bloc = nbElementTotal/nbProcessusMPI; // si nbElementTotal>nbProcessurMPI
5  int nbElementLocal;
6  if(rang != nbProcessusMPI-1) nbElementLocal=bloc;
7  else nbElementLocal = nbElementTotal-bloc*(nbProcessusMPI-2);
8  std::vector<double> A_local,B_local;
9  double R_local(0.), R(0.);
10
11 // Chaque processus MPI remplit  sa partie des tableaux en fonction de son rang
12
13 for(size_t it = 0 ; it < nbElementLocal ; it++)
14   R_local += A_local[it] + B_local[it];
15
16 MPI_Reduce(
17   &R_local,       // variable à envoyer
18   &R,             // variable de réception
19   1,              // nombre d'éléments
20   MPI_DOUBLE,     // type
21   MPI_SUM,        // opération
22   0,              // rang du processus MPI receveur
23   MPI_COMM_WORLD) // communicateur
24 MPI_Finalise();
```

Listing 4.2 | Programme effectuant l'addition des termes d'un produit de deux vecteurs parallélisé avec MPI.

Parallélisation intra-Nœud

Pour chaque processus Mpi, une couche supplémentaire de parallélisme est utilisée afin de paralléliser les calculs entre les cœurs. Pour cela, un ou plusieurs processus légers nommés threads sont associés à chaque cœur. Chaque thread a accès aux données de la mémoire vive et exécute un flux d'instructions. On parle alors de parallélisme par thread en mémoire partagée [90]. Chaque thread a une mémoire qui lui est dédiée (nommée TLS) et une pile d'exécution. Dans le cas de l'hyperthreading, ou Simultaneous Multithreading (Smt) [137], plusieurs threads sont associés à un même cœur avec leurs propres registres mais partagent les mêmes Alus et les mêmes caches.

Plusieurs outils logiciels sont utilisés pour gérer les threads. Par exemple, Posix Thread (pThread) [25] est une interface permettant explicitement de créer des threads, de les détruire, contrôler leurs flux d'instructions, de les mettre en pause ou de choisir le cœur associé au thread. Des librairies avec un plus haut niveau d'abstraction basé sur pThread comme Open Multi-Processing (OpenMP) [35] et Intel Thread Building Block (Tbb) [115] sont disponibles dans différents langages de programmation comme le Fortran, le C ou le C++. Ils permettent d'utiliser implicitement des threads, par exemple, OpenMP utilise des directives \#pragma omp principalement devant une boucle de calcul à paralléliser.

Durant une simulation, un thread est désigné comme maître, il est le seul à exécuter les instructions en dehors des sections parallèles, les autres threads sont alors mis en attente. Lors d'une section parallèle, le travail est découpé en blocs d'instructions indépendantes les unes des autres (un bloc = une tâche). Ces blocs sont alors répartis entre le thread maître et les autres threads selon une politique d'attribution des tâches nommée ordonnanceur. L'ordonnanceur est propre à chaque standard : OpenMP, Tbb, ou autres.

En ce qui concerne l'utilisation de Gpus, chaque cœur possède un ou plusieurs threads associés. Tous les threads d'un bloc de cœurs effectuent le même flux d'instructions. Pour faciliter l'écriture des codes sur Gpus, des interfaces spécifiques telles que CUDA [119] pour les Gpus Nvidia, ou OpenACC [144] et OpenCL [129] ont été développées. Néanmoins le passage d'un code écrit pour être exécuté sur Uct en Gpu requière de nombreux ajustements des opérateurs ainsi que des structures de données.

En reprenant l'algorithme de la somme des termes d'un produit de deux vecteurs (cf listing 3.3), toutes les opérations de multiplications sont indépendantes les unes des autres. Il est donc possible de découper la boucle de calcul en blocs et d'incrémenter une variable R locale pour chaque thread. Ensuite le programme effectuera une somme séquentielle entre les R locaux. L'utilisation d'une variable locale R est nécessaire afin d'éviter les accès concurrents à une même donnée R pouvant induire une erreur d'écriture ou de lecture de celle-ci. L'utilisation d'une variable locale est implicite grâce à la directive reduction(+:R). Une autre solution aurait pu être d'utiliser des opérations atomiques d'OpenMP pour éviter que deux threads mettent à jour la même donnée en même temps.

```
1  std::vector<double> A,B;
2  double R(0.);
3
4  #pragma omp parallel
5  {
6    // Remplissage des tableaux en parallele si les données sont indépendantes
7  }
```

```
8
9   #pragma omp parallel for reduction(+:R) schedule(auto)
10  for(size_t it = 0 ; it < A.size() ; it++)
11    R += A[it] * B[it];
```

Listing 4.3 | Programme effectuant l'addition terme à terme de deux vecteurs parallélisé avec OpenMP.

Le modèle de parallélisation par tâche [18] permet la gestion de l'exécution simultanée d'un ensemble de fonctions pouvant opérer sur des données, différentes ou non. Chaque tâche est définie par un ensemble de calculs à exécuter sur un ensemble de données. Les tâches ont une granularité correspondant au compromis entre le nombre de calculs, la taille des données et le nombre de cœurs nécessaires. La granularité des tâches est directement corrélée à leur nombre. Un trop petit nombre de tâches entraîne une pénurie pour certains threads alors qu'un trop grand nombre a pour effet un surcoût d'ordonnancement des tâches.

Si deux tâches ne peuvent être exécutées simultanément, c'est-à-dire qu'une tâche dépend du résultat d'une autre, une dépendance entre celles-ci doit être définie. Dans ce cas, l'exécution des tâches s'effectue selon un Graphe orienté acyclique (Dag) dont les sommets et les arêtes seront respectivement les tâches et les dépendances. Le nombre de tâches effectuées en parallèle est égale au minimum entre le nombre de cœurs et le nombre de tâches disponibles et indépendantes entre elles. L'intérêt d'une parallélisation basée sur les tâches est de laisser à l'ordonnanceur le soin de gérer la répartition des tâches. Le choix d'exécuter une tâche prend en compte les ressources disponibles, les dépendances entre les tâches (en amont et en aval), priorité donnée à la tâche, le Nœud de calcul, etc.

De nouvelles stratégies d'ordonnancement ont vu le jour tel que le "vol de tâches" [19] consistant à distribuer dans un premier temps un ensemble de tâches sur les threads, puis au cours de la simulation, lorsqu'un thread est à court de tâches, celui-ci va voler une tâche à un autre thread. Bien que le fait de voler une tâche ait un coût, celui-ci est amorti par l'équilibrage de charge qu'il entraîne. Cette technique est notamment utilisée par Intel Cilk [18] et Tbb.

OpenMP intègre le modèle de parallélisation par tâche depuis sa version 3.0 avec les directives \#pragma omp task ou taskloop grainsize(Size). Ces directives sont régulièrement mises à jour avec notamment l'apparition de la clause *depend* permettant de créer les dépendances entre les tâches (version 4.0). Des supports d'exécution avec des stratégies d'ordonnancement différentes sont apparus, notamment pour effectuer des simulations sur des machines hétérogènes (multi-Uct/Gpu). C'est le cas de StarPU [9] : chaque tâche est définie par l'utilisateur par une structure nommée codelet. Le codelet est une structure contenant les opérateurs de calcul et un ensemble de données. Les opérateurs peuvent être écrits sous différentes formes selon l'architecture cible. Les tâches sont assignées aux différentes unités de calcul en fonction des dépendances, des priorités et de la localité des données.

Ainsi, en reprenant l'algorithme précédent avec la librairie OpenMP (cf listing 3.4), le lancement des tâches est défini via la directive \#pragma omp taskloop ne modifiant pas le code. Une autre stratégie aurait été de découper la boucle en blocs, de définir un opérateur séquentiel pour un bloc, sans oublier une directive \#pragma omp atomic lors de l'incrémentation de la variable R. Les tâches auraient été créées via la directive \#pragma omp task par le thread maître.

```
1   std::vector<double> A,B;
2   double R(0.);
3   size_t SIZE;
4
5   // Remplissage des tableaux et listDeBlocs
```

```
6
7   //implémentation 1
8   #pragma omp parallel taskloop reduction(+:R) grainsize(SIZE)
9   for(size_t it = 0 ; it < A.size() ; it++)
10    R += A[it] * B[it];
```

Listing 4.4 | Programme effectuant l'addition terme à terme de deux vecteurs parallélisés avec OpenMP avec des tâches explicites.

Vectorisation

L'utilisation d'instructions Simd est un enjeu majeur pour les architectures récentes de supercalculateurs qui voient la taille de leurs registres vectoriels augmenter. Pour cela de nombreux codes de calcul ont été revisités afin que les structures de données utilisées dans les opérateurs de calculs soient contiguës en mémoire via l'utilisation de fonctions du type __assume_aligned(ptr, 64). Les opérateurs de calcul ont également été adaptés afin d'éviter les dépendances entre les éléments d'un même tableau. Il peut s'avérer plus efficace d'effectuer davantage de calculs intermédiaires mais en utilisant la vectorisation. Par exemple, quand une instruction conditionnelle du type $if(cond)\{do\_something();\}else\{0;\}$ est présente dans la boucle de calcul, il est préférable d'effectuer vectoriellement $do\_something()$ sur l'ensemble des éléments du tableau en appliquant un masque sur les données ne respectant pas la condition $cond$.

Par défaut, la plupart des compilateurs actuels peuvent aisément vectoriser les boucles du type : tableau 1 = tableau 2 * opération * tableau 3, mais ne s'autorisent généralement pas à vectoriser les boucles plus complexes. De plus, des compilateurs comme celui d'Intel (icpc) effectuent une estimation de la boucle de calcul pour déterminer si la vectorisation aura un impact favorable. Cette estimation est déduite du coût scalaire d'une itération de la boucle de calcul, du coût vectoriel d'une itération de la boucle divisé par la taille du vecteur, du facteur de déroulement de la boucle et du coût des initialisations et finalisations des vecteurs avant et après la boucle de calcul.

En C/C++, l'utilisation de directives \#pragma omp simd permettent de forcer l'utilisation des instructions Simd lors de la compilation (cf listing 3.5). Une méthode plus agressive est d'instrumenter directement le code avec des instructions Simd comme les instructions Intel Intrinsics. Pour cela l'utilisateur manipule des classes C++ de vecteurs Simd intrinsics dont la taille dépend des registres vectoriels, par exemple 256-bits = 4 doubles ou 8 floats pour de la vectorisation en AVX2. Néanmoins, l'utilisation de ces instructions doit être adaptée pour chaque jeu d'instructions vectorielles (intel) ce qui limite la portabilité du code (par exemple sur les processeurs ARM) et augmente sa maintenance.

```
1   std::vector<double> A,B,C;
2
3   // Remplissage des tableaux
4
5   #pragma omp simd reduction(+:R)
6   for(size_t it = 0 ; it < A.size() ; it++)
7     R += A[it] + B[it];
```

Listing 4.5 | Programme effectuant l'addition terme à terme de deux vecteurs et utilisant l'auto-vectorisation.

Conclusion

Suite aux récentes évolutions des architectures des supercalculateurs, les algorithmes et architectures logicielles ont dû être adaptés afin de prendre en compte la hiérarchie architecturale de ceux-ci. Ainsi une parallélisation en mémoire distribuée utilisant par exemple la bibliothèque standard MPI permet de répartir les calculs et les données entre les NŒUDS ou les SOCKETS du supercalculateur. Au sein de chacun de ces NŒUDS, une parallélisation en mémoire partagée s'effectue entre les cœurs. De plus, pour prendre en compte les spécificités des processeurs vectoriels, un travail sur la disposition de données et l'écriture des opérateurs est aussi nécessaire afin d'utiliser les instructions dites SIMD.

Dans le chapitre suivant, nous réalisons une description des méthodes usuelles de répartition de charge entre les processus MPI ainsi qu'une description des méthodes d'équilibrage de charge. Une présentation est également proposée sur des maillages utilisés dans les codes de simulation ainsi que les codes de Dynamique Moléculaire.

5 État de l'art

La première motivation de cette thèse est d'améliorer la répartition de la charge de calcul sur des supercalculateurs récents pour des simulations de Dynamique Moléculaire classique dont la densité d'atomes est fortement non uniforme. Pour cela les deux simulations principalement visées sont le développement d'un micro-jet et l'impact de nano-goutte(s) d'étain(s) sur une plaque. Pour ce type de calculs, la matière est concentrée sur une faible portion du domaine de simulation et les déplacements des particules sont très dynamiques. Les principaux codes de DM, comme nous le détaillerons en section 5.4, utilisent un maillage pour construire les listes de voisins pour chaque particule. Le maillage est généré par la méthode des cellules liées et ne tient pas compte des spécificités de la simulation. C'est pourquoi lorsque des parties du domaine à mailler ne contiennent pas ou peu d'atomes, le maillage engendre des calculs et l'utilisation de la mémoire inutilement.

Dans le but d'introduire un maillage s'adaptant dynamiquement à la physionomie de la simulation et ainsi éviter le maillage des volumes ne contenant pas d'atome, nous étudions dans le paragraphe 5.1 l'utilisation d'autres maillages utilisés dans plusieurs domaines de simulations numériques. Une attention particulière est portée à l'intégration des maillages pour des simulations numériques dites à N-corps comme c'est le cas pour la DM.

Afin de tenir compte de la structure des supercalculateurs décrite dans le chapitre 4, une première forme de parallélisme est utilisée pour distribuer les mailles entre les NŒUDS ou les sockets. Cependant, pour les simulations cibles, il ne suffit pas d'envoyer simplement sur chaque NŒUD le même nombre de mailles car la répartition de la charge de calcul obtenue entre les NŒUDS est généralement très déséquilibrée. C'est pourquoi, pour mieux répartir la charge de calcul, des méthodes de répartition dynamique de charge sont étudiées en commençant par la méthode de partitionnement spatial de domaine classique, voir paragraphe 5.2, suivies par les méthodes de partitionnement géométriques et de graphe, voir paragraphe 5.3.

Maillage adaptatif

Définitions

En DM, le maillage a pour but de déterminer rapidement quels sont les atomes inclus dans le voisinage de chaque atome. Cependant, nous verrons que la définition et l'objectif d'un maillage diffère selon le type de simulations envisagées. Je vais donc introduire différentes notions et définitions.

Tout d'abord, la notion de maillage de domaine est particulièrement présente en mécanique des fluides ou plus généralement, en mécanique des milieux continus. La mécanique des fluides permet de simuler à un niveau macroscopique des gaz, des liquides ou des plasmas. Elle consiste

notamment à résoudre les équations de Navier-Stokes [128] à chaque pas de temps. Dans ce cas, le maillage consiste à discrétiser spatialement le domaine de simulation en mailles (différences finies, éléments finis,volumes finis, ...). La solution aux équations du problème est calculée en chacune de ces mailles et l'erreur locale de la solution est directement liée à l'espacement entre les mailles. Dans ce cas, le maillage est directement utilisé par les méthodes numériques et il est donc très important pour la précision. Par conséquent, il est crucial au bon déroulement de la simulation.

Dans le cas des simulations à N-corps, l'objectif principal est de localiser spatialement les éléments les uns par rapport aux autres pour la détection de collision ou la recherche des plus proches voisins. Dans ce cas ,le maillage participe indirectement aux calculs présents dans les méthodes numériques. Pour cela, le maillage d'un domaine consiste en une décomposition géométrique du domaine en cellules élémentaires. Chaque cellule élémentaire couvre un espace distinct des autres cellules élémentaires et leur union couvre la totalité du domaine. Les cellules élémentaires peuvent être identiques ou être de formes variées. Le maillage est dit conforme si l'interface entre deux cellules élémentaires comporte le même nombre de nœuds, c'est-à-dire les extrémités formant l'espace couvert par la cellule.

La conformité du maillage est importante pour des simulations comme par exemple dans le domaine de la mécanique des fluides car dans le cas contraire elle conduit à des instabilités numériques dues aux discontinuités. Néanmoins, cette conformité n'est pas nécessaire pour les simulations à N-corps car le maillage n'induit pas d'erreurs numériques. Par la suite, pour les simulation à N-corps, le terme de grille est préféré au maillage. Les cellules élémentaires sont nommées cellules.

Dans le cadre de cette thèse, les grilles évoquées sont uniquement celles employables pour la Dynamique Moléculaire classique mais aussi pour n'importe quelle simulation à N-corps. Pour cela, différents types de grilles existent comme les grilles structurées, adaptées structurées, multi-blocs et non structurées. Une attention particulière est portée aux grilles à blocs structurés utilisant la méthode de Raffinement de Maillage Adaptatif (AMR) (décrite dans le paragraphe 5.1.5).

Grilles structurées

Pour des simulations dont la densité des éléments est équitablement répartie comme c'est le cas en DM pour une simulation d'un cristal parfait dont la densité d'atomes est uniforme, les grilles généralement utilisées sont les grilles dites structurées [55]. Elles sont générées à partir de la réplication d'une cellule comme un rectangle ou un triangle dans toutes les directions jusqu'à couvrir l'ensemble du domaine. La grille obtenue est alors conforme. La connectivité entre les cellules obéit à un motif précis. À noter que la grille structurée la plus courante est la grille cartésienne. Celle-ci est obtenue à partir de cellules en forme de carré ou de cube, voir figure 5.1. Un autre exemple est la grille oblique lorsque la cellule élémentaire a la forme d'un parallélogramme en 2D ou d'un parallélépipède en 3D. L'avantage des grilles structurées est leur faible coût de génération et de stockage car elles ne nécessitent pas de stocker l'emplacement et la connectivité des cellules.

Concernant les simulations de Dynamique Moléculaire classique vues dans le paragraphe 3.5, la méthode des cellules liées se sert d'une grille cartésienne dont la cellule est un parallélépipède rectangulaire. Les cellules ont une largeur, une hauteur et une profondeur supérieures à RCUT, celui-ci est imposé par la physique. Le principal défaut des grilles structurées est leur manque de maniabilité. En effet si l'on souhaite raffiner la grille dans certaines zones de la simulation en réduisant la taille des cellules, la totalité de la grille doit être adaptée. Ainsi la précision obtenue dans

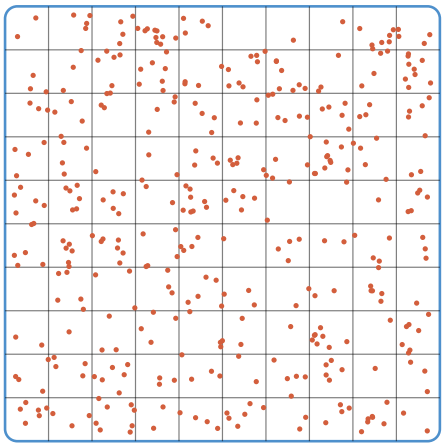


Figure 5.1 | Grille structurée pour une simulation à N-corps dont la cellule à la forme d'un carré.

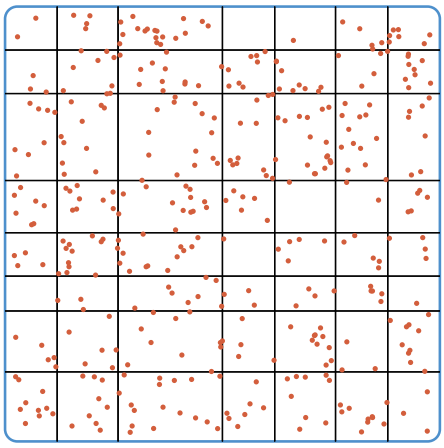


Figure 5.2 | Grille adaptée structurée pour une simulation à N-corps. Les cellules alignées de droite à gauche ont la même hauteur et les cellules alignées de haut en bas en la même largeur.

certaines zones ne compense pas les surcoûts de calculs et de stockage que les cellules entraînent dans les autres zones.

Une possibilité pour pallier aux surcoûts de calculs dûs aux cellules ne contenant presque ou pas d'atomes est d'utiliser une grille adaptée structurée. Cette grille est formée de cellules de même forme dont la taille varie. Elles possèdent toutes le même paterne de connectivité et la grille obtenue est alors conforme. La grille adaptée structurée s'apparente donc à une grille structurée déformée comme la grille rectilinéaire (composée de rectangles), voir figure 5.2, ou une grille curviligne. Elles ont l'avantage d'être plus malléables que les grilles structurées car elles permettent de mailler finement les espaces à fortes densités et donc d'atténuer les défauts de la grille structurée.

Les grilles curvilignes sont particulièrement intéressantes pour simuler des domaines avec des parois rondes. Cependant, pour conserver leurs structures, la taille des cellules est adaptée pour toutes les cellules dans l'alignement de la déformation. Ainsi, si le système comporte plusieurs espaces à mailler finement ou si l'espace à mailler finement n'est pas dans le sens d'un des axes, la grille formée peut devenir équivalente à une grille structurée comportant donc ses défauts.

Pour plus de détails et d'exemples sur les grilles structurées, le livre [55] donne un aperçu des principales grilles et des méthodes utilisées pour les générer.

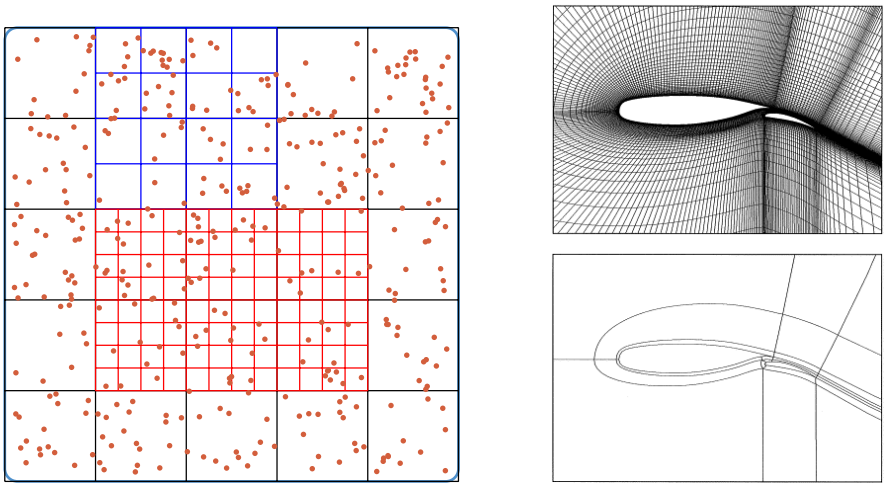


Figure 5.3 | (figure de gauche) Grille multi-blocs pour une simulation à N-corps. Le domaine est décomposé en trois blocs de couleur noire, rouge et bleue dont la taille des cellules est variable. (figure de droite) Grille multi-blocs et topologie des blocs pour le profil aérodynamique NLR 7301 d'une aile avec un volet [62]

Grilles multi-blocs

Au lieu d'adapter la grille en faisant varier la taille des cellules selon un axe, une autre stratégie est d'adapter localement la taille des cellules de la grille. Pour cela, des grilles multi-blocs sont utilisées. Elles sont habituellement composées d'un ensemble de blocs rectangulaires dont chaque bloc est une grille structurée. Les cellules entre deux blocs ont généralement la même forme mais pas la même taille, voir figure 5.3. L'intérêt de cette grille est de combiner l'avantage d'utiliser des grilles structurées avec des cellules raffinées finement dans les espaces à forte densité sans répercussion pour les autres blocs de la grille. La grille peut être reconstruite dynamiquement au fur et à mesure de la simulation afin de tenir compte de l'évolution de la simulation. Par exemple ce type de grille est utilisée pour calculer le profil aérodynamique d'une aile avec un volet, voir la figure 5.3 [62].

Pour ce type de grille, puisque les blocs sont des grilles structurées, chaque bloc est défini par : les informations d'une cellule (taille et forme), le nombre de cellules et l'ordonnée du bloc dans grille. Pour les cellules à l'interface entre deux blocs, comme elles ne sont pas de la même taille, une table de connectivités doit être stockée. Le stockage des tables de connectivités peut être évité si les cellules des blocs sont de mêmes formes. La méthode Raffinement de Maillage Adaptatif (Amr) permet de générer ce type de grille. Cette méthode est détaillée dans le paragraphe 5.1.5.

Grilles non structurées

Jusqu'à présent nous avons uniquement introduit des grilles construites autour d'une certaine structure engendrant des défauts dans certains cas. Afin d'éviter d'être contraint par une structure pré-déterminée à l'avance, une dernière variété de grille est la grille non structurée.

Elle consiste en une décomposition spatiale du domaine en cellules de tailles et formes différentes et dont les cellules sont connectées à un nombre variable d'autres cellules, voir figure 5.4. L'avantage des grilles non structurées est qu'elles s'adaptent à de nombreuses géométries grâce à la possibilité de raffiner localement la grille par des méthodes Amr. La grille peut être remaillée localement au cours de la simulation. L'inconvénient de ces grilles est qu'il est nécessaire de stocker la table des cellules ainsi que leurs connectivités. Des méthodes permettent de générer des grilles non structurées comme par exemple la méthode de triangularisation de delaunay [78, 81, 28] , la méthode de Voronoï [142, 63] ou la méthode frontale [85]. Les grilles non structurées ont peu d'intérêt en

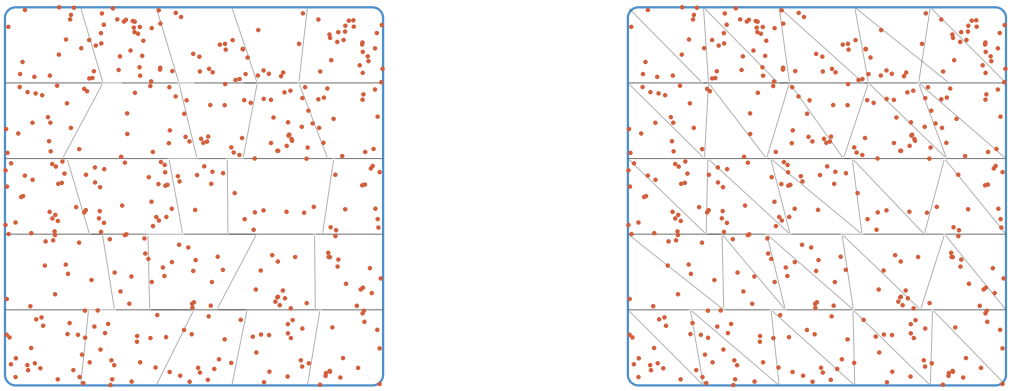


Figure 5.4 | Grille non structurée dont les cellules sont des parallélogrammes (gauche) et une une non-structurée formée par des cellules triangulaires (droite).

DM car le stockage du maillage est coûteux, il y a perte du pattern pour trouver les cellules voisines efficacement et des conditions sur la forme et la taille des cellules doivent être conservées. Elles sont utilisées pour générer des maillages d'éléments finis non structurés comme par exemple en hydrodynamique.

Méthode de Raffinement de Maillage Adaptatif

Les deux simulations mises en évidences dans le chapitre 3 sont extrêmement dynamiques et hétérogènes. Nous avons vu dans les paragraphes précédents que les grilles à blocs structurés semblent être les plus à même de tenir compte de la densité des atomes. Pour cela, la méthode Raffinement de Maillage Adaptatif (AMR) permet de générer ce type de grille et de l'adapter dynamiquement tout au long de la simulation. La grille générée par cette méthode est alors nommée grille AMR.

Premières applications

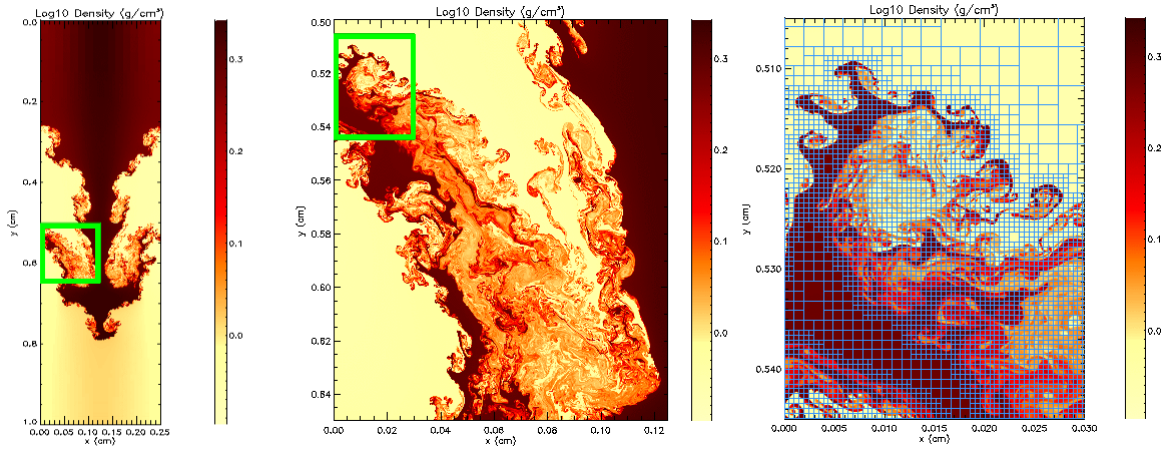


Figure 5.5 | Simulation de l'instabilité de Rayleigh–Taylor par le code FLASH [56] en utilisant une grille AMR à 10 niveaux de raffinement.

Avant de décrire la méthode AMR adaptée pour les simulations à N-corps. Nous rappelons comment cette méthode a été insufflée dans divers domaines physiques. Une des premières grilles

Amr a été développée par Berger [17, 16] pour un code de simulation d'hydrodynamique en 1984. Son objectif est de réduire les temps de calculs tout en conservant une bonne précision de la solution. En hydrodynamique, le domaine de simulation est discrétisé en un ensemble de cellules élémentaire formant traditionnellement une grille cartésienne. La solution de l'équation est calculée en chacun des nœuds se trouvant aux extrémités des cellules. L'espacement entre les nœuds détermine une erreur locale et donc influence la précision de la solution calculée. l'Amr consiste à agglomérer les cellules de la grille dont les Nœuds de calcul n'ont pas besoin d'être proches pour conserver leurs précisions. L'erreur est donc négligeable en ces points. L'agglomération est déterminée par des critères tels que la masse volumique ou par le critère d'extrapolation de Richardson [116]. L'étape d'agglomération est répétée tant que les critères sont validés. Chaque passe constitue alors un niveau de raffinement. À noter que différents niveaux de raffinement peuvent être obtenus pour différents espaces de la simulation.

Cette méthode est intéressante pour des simulations à densités hétérogènes car elle réduit les coûts de calculs pour certaines parties de la simulation. Pour des questions de coût de stockage, la stratégie pour générer la grille n'est pas d'agglomérer les cellules de la grille mais de générer une grille cartésienne grossière puis de raffiner chaque cellule. Tant que les nœuds aux extrémités des cellules élémentaires ne valident pas les critères de raffinement et de stabilité des schémas d'intégrations, les cellules sont raffinées.

Actuellement de nombreux logiciels permettent de modéliser des phénomènes issus de la mécanique des milieux continus. En hydrodynamique des logiciels comme CASTRO [7, 149], FLASH [56] et MAESTRO [101] intègrent l'Amr pour la résolution des équations de radiation en astrophysique (flux stellaires). On retrouve aussi ce système dans le logiciel LMC [39] pour des simulations de combustion. Des fonctions sont disponibles dans des bibliothèques facilitant la mise en œuvre des techniques Amr : BoxLib [83], FLASH [45] (extension), PARAMESH [87] ou Cactus [60] (sur lequel est basé Einstein Toolkit [84]).

Amr et simulations à N-Corps

Les méthodes Amr ont été déclinées pour les simulations à N-Corps, comme par exemple pour simuler la formation des galaxies [147]. Pour cela, le domaine de la simulation est décomposé en un ensemble de cellules formant une grille structurée et grossière, qui sont équivalentes à des nœuds pour les simulations d'hydrodynamique. L'Amr consiste donc à raffiner récursivement les cellules contenant un grand nombre de particules.

De nombreux logiciels utilisent l'Amr pour divers problèmes physiques tels que l'astrophysique avec Enzo [22](N-Corps et hydrodynamique), RAMSES [134](N-Corps et hydrodynamique), Nxy [6] ou le code Charm [94, 95] avec la librairie Chombo [32].

La méthode Amr peut utiliser des cellules de formes différentes : triangle, carré, rectangle ou hexagone. Comme nous l'avons vu précédemment, la méthode des cellules liées génère une grille cartésienne dont les cellules ont la forme d'un parallélépipède rectangulaire.

De plus, la combinaison de la méthode Amr avec une structure en octrees (paragraphe suivant) rend possible de mailler les espaces les plus denses tout en concevant des propriétés intéressantes. La grille formée est une grille à blocs structurés avec Raffinement de Maillage Adaptatif.

La méthode des Octrees : k-d-tree et Pbe

La méthode des octrees est un cas particulier de la méthode Partition Binaire de l'Espace (Pbe) [15]. Cette méthode consiste à subdiviser récursivement un espace en plusieurs ensembles convexes délimités par des hyperplans. Cette subdivision donne lieu à une représentation du système au

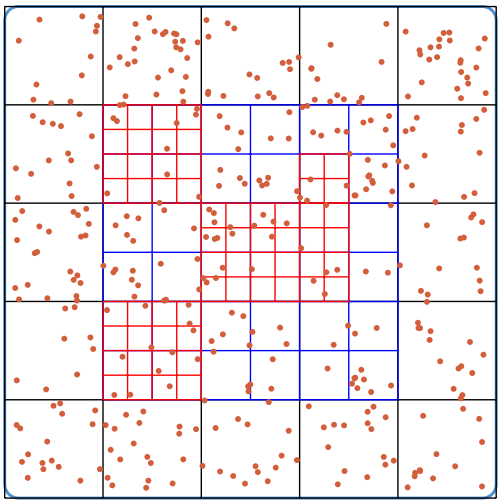


Figure 5.6 | Grille non structurée générée par la méthode des octrees.

moyen d'une arborescence de données appelée arbre Pbe. Les k-d-trees [12] sont un cas particulier des arbres Pbe car la subdivision se fait toujours selon un hyperplan dont sa normale est orthogonale à un axe du système.

En $3D$, un octree (quadtree en $2D$) est arbre dont chaque nœud est récursivement divisé en 8 espaces convexes de même volume et séparés par des plans parallèles aux 3 axes du système, soit des parallélépipèdes rectangulaires. Ainsi en Dm, la méthode des octrees combinée avec l'Amr, illustrée par la figure 5.6, permet de subdiviser récursivement le domaine jusqu'à atteindre les cellules de la taille du Rayon de coupure dans les espaces à forte densité d'atome. Chaque subdivision correspond à un niveau de raffinement, et les cellules qui ne sont pas raffinées sont appelées des cellules feuilles.

L'arborescence d'un octree permet de parcourir rapidement les cellules de la grille sans avoir besoin de stocker les positions cartésiennes de celles-ci. Puisqu'un octree est une représentation spatiale de la décomposition d'un domaine, l'arborescence de l'arbre fournit les informations nécessaires pour trouver la connectivité de chaque feuille de ce dernier. Enfin, une grille basée sur un octree est peu coûteuse à raffiner au cours du temps car l'algorithme de raffinement peut repartir incrémentalement de l'arborescence de l'octree.

Ainsi cette méthode permet d'adapter dynamiquement la grille à l'évolution de la simulation en tenant compte de ses spécificités, comme par exemple de la densité des atomes. Pour cela, un critère de raffinement est utilisé pour définir si un nœud de l'octree doit être raffiné ou inversement si 8 nœuds doivent être agglomérés.

Finalement cette méthode permet d'adapter localement la grille en fonction d'un ou de plusieurs critères de raffinement. À noter que l'adaptation d'une partie de la grille n'influe pas sur le reste de la grille. De plus la méthode Amr tient compte de l'évolution de la simulation même si celle-ci ne concerne qu'une partie du domaine.

Index de Morton

Les octrees ont d'autres propriétés intéressantes à exploiter. Par exemple lorsqu'un octree est complet, c'est-à-dire que toutes ses feuilles sont au niveau du raffinement le plus élevé, chaque niveau de l'arbre correspond à une grille cartésienne. L'index de [100] est un outil permettant de déterminer la position d'une cellule dans l'arborescence d'un octree à partir de sa position cartésienne sur une grille d'un niveau de raffinement donné. Celui-ci est obtenu par une manipulation des

coordonnées cartésiennes d'une cellule retranscrit en binaire et du niveau de raffinement. Tout d'abord, lorsqu'une cellule est raffinée, les cellules raffinées créées ont chacune une coordonnée locale (x,y,z). Elles sont numérotées de 1 à 8 comme illustré sur la figure 5.7. La variable x est égale à 0 ou 1 si la cellule est sur le bord gauche ou droit. La variable y est égale à 0 ou 1 pour la face avant ou arrière et la variable z est égale à 0 ou 1 pour le bord en bas ou bord en haut. L'index local de Morton, noté M', est un entier de 3 bits défini tel que :

$$M' = x * 4 + y * 2 + z = (xyz)_2$$

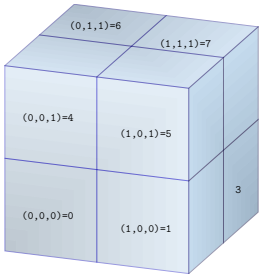
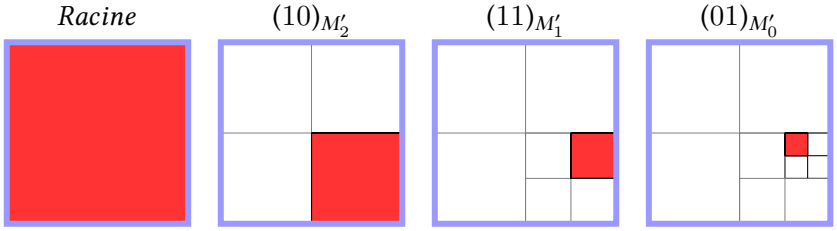


Figure 5.7 | Numérotation des cellules filles à l'intérieur d'une cellule avec une indexation de Morton aussi appelée parcours en Z.

L'index de Morton d'une cellule consiste à écrire les coordonnées cartésiennes d'une cellule en base binaire, c'est-à-dire $(x, y, z)_{10}$=$(x_n...x_2x_1x_0, y_n...y_2y_1y_0, z_n...z_2z_1z_0)_2$. L'index de Morton s'écrit alors M=$(x_ny_nz_n...x_2y_2z_2x_1y_1z_1x_0y_0z_0)_2$. Chaque triplet de bits correspond à un index local de Morton. L'index de Morton s'écrit alors M=$(M'_n...M'_2M'_1M'_0)$. Obtenir l'index de Morton permet donc de récupérer le chemin dans l'arborescence de l'octree. Par exemple en 2D, pour une cellule C de position cartésienne (x=6,y=3) et dont le niveau de raffinement est de 3, son index de Morton est 45. Pour cela, on retranscrit la position x et y en binaire puis on obtient l'index de Morton en binaire que l'on retranscrit en base 10 :

$$\begin{array}{ccccc} x & = & (6)_{10} & = & (110)_2 \\ y & = & (3)_{10} & = & (011)_2 \\ morton & = & (101101)_2 & = & (45)_{10} \end{array}$$

Table 5.1 | Chemin de la cellule C donnée par l'index Morton M=$(45)_{Base10}$ =$(00)_{M'_3}(10)_{M'_2}(11)_{M'_1}(01)_{M'_0}$, () est donné en binaire et M' correspond aux indexes locaux.



Nous utilisons l'index de Morton pour parcourir efficacement les éléments d'un octree, c'est-à-dire sans passer deux fois par le même élément et deux éléments successifs sont spatialement proches. Il permet aussi de déduire le chemin d'accès d'une cellule feuille dans ce dernier.

Méthode de partitionnement spatial de domaine

La parallélisation en mémoire distribuée a été abordée dans le chapitre 4. Elle consiste à distribuer une partie de la simulation (calculs) sur chaque Nœud du supercalculateur. Dans le domaine de la Dm, la méthode la plus simple pour distribuer les calculs entre les Nœuds est d'employer la méthode de partitionnement spatial de domaine [109]. Le principe de cette méthode est de scinder un domaine contenant des particules en plusieurs sous-domaines tout en tenant compte des conditions limites entre les sous-systèmes. Chaque sous-domaine est alors exécuté sur un Nœud du supercalculateur associé a un processus Mpi ou à une autre bibliothèque de communication. Chaque processus Mpi exécute un ou plusieurs threads (parallélisation intraNœud).

Comme illustré par la figure 5.8, le domaine de simulation est découpé en sous-domaines. Chaque sous-domaine a la forme d'un parallélépipède rectangulaire. En $3D$, chaque sous-domaine a alors 26 sous-domaines voisins. Afin que chaque particule, et plus particulièrement celles au bord des sous-domaines, puisse avoir l'ensemble des atomes inclus dans son voisinage, une zone fantôme autour du domaine est mise en place afin de rapatrier les informations manquantes en les dupliquant. Cette méthode est particulièrement utilisée dans les codes de dynamique moléculaire pour paralléliser les calculs en distribuant les sous-domaines sur les Nœuds d'un supercalculateur.

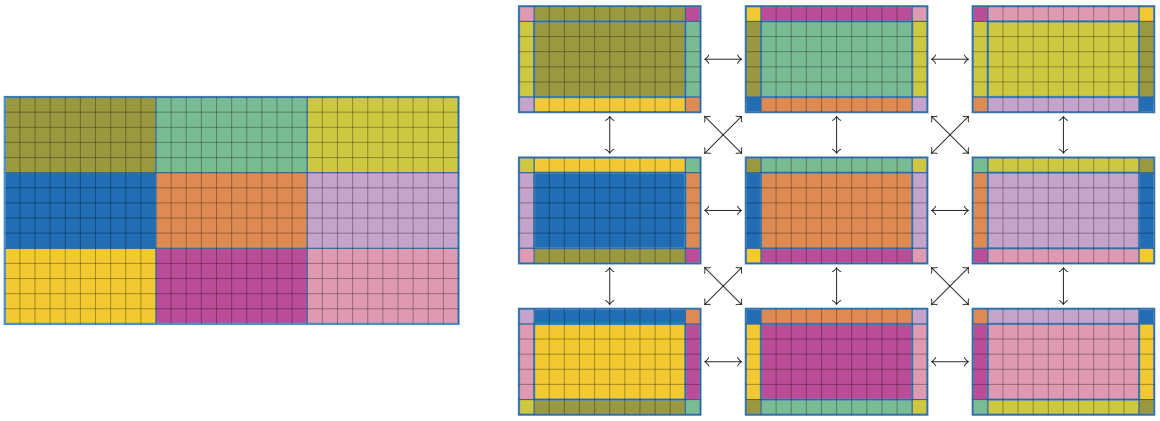


Figure 5.8 | Application de la méthode de décomposition spatiale de domaine en $2D$, le domaine de la simulation (gauche) est alors décomposé en 9 sous-domaines (couleurs). Les zones fantômes sont adaptées afin de récupérer les informations pour chaque sous-domaine. En $2D$, chaque sous-domaine a besoin de récupérer les informations des 8 sous-domaines adjacents.

Répartition de charge

Bien que la méthode de partitionnement spatial de domaine distribue les calculs entre les ressources d'un supercalculateur, celle-ci ne tient pas compte de la charge attribuée à chaque sous-domaine. Or, lorsque les simulations sont hétérogènes, la charge de calcul entre les sous-domaines est hétérogène. C'est pourquoi certains Nœuds reçoivent très peu de calculs et attendent donc que les autres aient fini leurs calculs. Afin d'éviter ce cas de figure, des méthodes d'équilibrage de charge sont utilisées.

Pour cela, un poids est attribué à chaque élément d'une simulation à N-corps, aussi bien arithmétique (nombre d'opérations) que de stockage. Lorsque la distribution des éléments n'est pas uniforme pour une boite de simulation donnée, la méthode classique de partitionnement spatial du domaine ne permet pas de répartir équitablement les calculs et les données entre les différents processus Mpi. C'est pourquoi, des techniques de répartition de charge tenant compte de divers critères permettent d'obtenir des sous-domaines dont la somme des poids est environ équivalente.

Pour cela, deux familles de partitionnement existent : les partitionnements géométriques 5.3.1 et les partitionnements de graphes 5.3.2.

Partitionnement géométrique

La première famille que nous allons décrire est composée des méthodes de partitionnement dites géométriques. Elles sont basées sur des découpages géométriques d'un domaine comme par exemple en scindant un domaine en deux selon un plan. Elles sont généralement de l'ordre de 10 fois moins coûteuses que les méthodes de partitionnement de graphe et 100 fois moins pour une méthode de partitionnement d'hypergraphe [113]. Pour les simulations à N-Corps ne comportant pas de grille, un poids est attribué à chaque élément. Dans le cas d'une simulation avec grille, un poids est attribué à chaque cellule, il correspond souvent à la somme des poids attribués aux éléments inclus dans la cellule. Les poids peuvent dépendre de nombreux facteurs comme le type d'élément pondéré par le nombre d'atomes proches et donc de la nature des interactions qu'il est susceptible d'avoir. Il existe de nombreux partitionnements géométriques, les plus courants sont le partitionnement Recursive Coordinate Bisection (RCB), le partitionnement Recursive Inertial Bisection (RIB), le partitionnement Space Filling Curve (SFC) et plus particulièrement le partitionnement Hilbert SFC.

Recursive Coordinate Bisection

Le partitionnement Recursive Coordinate Bisection (RCB) est obtenu à partir de la méthode Partition Binaire de l'Espace [15]. Le domaine de simulation est divisé en deux sous-domaines séparés par un hyperplan (une surface en $3D$) orthogonal, à l'un des axes des coordonnées de sorte que la charge de travail soit équitablement distribuée entre les deux sous-domaines. Le choix de l'hyperplan s'effectue dans la direction orthogonale à l'axe des coordonnées selon lequel le sous-domaine est le plus allongé. Les sous-domaines sont alors re-divisés récursivement jusqu'à ce que le nombre de sous-domaines soit égal au nombre de processus Mpi. Ainsi en $3D$ la forme des sous-domaines obtenue est un parallélépipède rectangulaire et la table de connectivité entre les domaines est simple à obtenir car elle est déduite de l'arborescence de l'arbre de découpage. Un autre avantage de la méthode RCB est la possibilité d'adapter dynamiquement le partitionnement, c'est-à-dire que la taille des sous-domaines peut être adapté selon la dernière découpe.

Recursive Inertial Bisection

Le partitionnement Recursive Inertial Bisection (RIB) [123, 132] est une variante du partitionnement RCB. La différence s'effectue par le choix des hyperplans utilisés pour découper récursivement les sous-domaines en deux. Tout d'abord l'algorithme trace l'axe principal d'inertie selon la distribution des éléments (poids) pour chaque sous-domaine. Les axes principaux d'inertie correspondent aux axes formant le repère d'inertie. Le repère d'inertie correspond au repère orthonormé tel que le tenseur d'inertie associé au domaine soit une matrice diagonale. L'axe principal d'inertie est l'axe selon lequel la distribution des objets du sous-domaine est le plus «allongé».

Les sous-domaines sont séparés par un hyperplan orthogonal à l'axe principal d'inertie, répartissant la charge équitablement. les sous-domaines sont alors récursivement subdivisés jusqu'à ce que le nombre de sous-domaines soit égal au nombre de processus Mpi. Les sous-domaines obtenus sont des volumes convexes. Comme pour le partitionnement RCB, le partitionnement est dynamique.

Hilbert Space Filling Curve

Les deux méthodes de partitionnement précédentes sont basées sur la méthode Pbe et ne tiennent pas compte des possibilités des grilles Amr. À contrario, le partitionnement Space Filling Curve (SFC) [108, 141] s'applique pour n'importe quelles grilles Amr ou grilles non-structurées. Le principe est d'affecter un poids à chaque cellule et de définir une courbe ne passant jamais deux fois par la même cellule. Un sous-domaine est obtenu lorsque la somme des poids cumulée dépasse la charge totale divisée par le nombre de sous-domaines voulu.

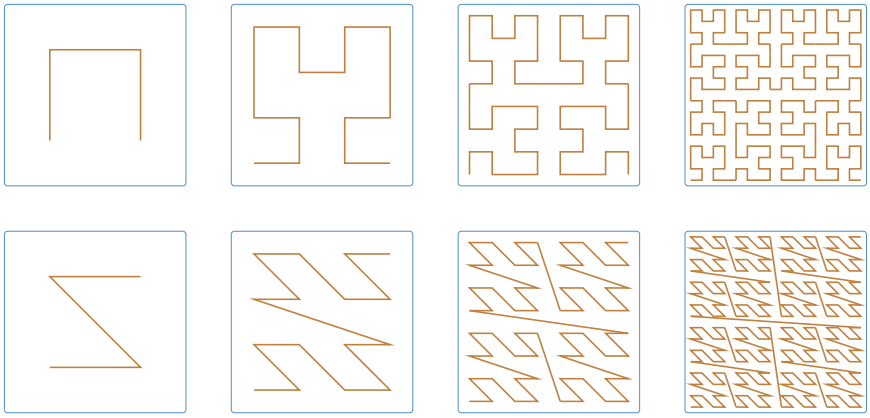


Figure 5.9 | Exemple d'une courbe de Peano en dimension 2 : Hilbert (haut) et Lebesgue (bas)

L'un des choix de courbes les plus utilisés est la courbe de Hilbert (figure 5.9). C'est une Courbe de Peano [104], c'est-à-dire une courbe remplissant l'espace d'un hypercube de dimension n, et faisant partie de la famille des fractales, c'est-à-dire dont le motif est invariant selon l'échelle. Cette courbe permet donc de parcourir n'importe quelle grille cartésienne avec ou sans Amr en dimension n sans passer deux fois par le même élément et dont deux éléments successifs sont généralement "voisins". Chaque élément du domaine est numéroté par son index de Hilbert. Chacun de ces éléments reçoit un poids selon la charge de calcul associée. Les éléments sont alors distribués sur les sous-domaines. Pour cela, ils sont attribués au même sous-domaine tant que la somme des poids des éléments parcourus par la courbe de Hilbert ne dépasse la charge totale divisée par le nombre de sous-domaine voulu.

De nombreuses déclinaisons avec d'autres courbes de Peano existent tel la courbe de Lebesgue (courbe en Z), cf figure 5.9. Celle-ci correspond au parcours des éléments d'un octree selon un indexage de Morton. Cependant la courbe d'Hilbert reste la plus utilisée car elle n'intègre pas de discontinuité (les éléments successifs sont adjacents). Un dernier intérêt à utiliser des courbes de Peano est lorsque le domaine est re-partitionné (dynamiquement) en sous-domaines. Si la distribution des poids associés aux cellules n'a pas variée à outrance, les sous-domaines obtenus conservent une grande partie des cellules qui leur ont déjà été attribués.

Partitionnement de graphe et hypergraphe

Comme expliqué dans le paragraphe précédent, les méthodes géométriques tiennent uniquement compte des poids associés à chaque élément et partitionnent donc l'espace selon un certain motif géométrique sans tenir compte des relations entre les éléments. Au contraire, des méthodes de partitionnement de graphe qui tiennent compte de ce lien. Bien qu'elles soient généralement plus coûteuses en calcul et en mémoire que les partitionnements géométriques (d'environ d'un ordre de grandeur) elles apportent généralement une meilleure répartition de la charge de calcul. Un graphe

est un ensemble de sommets reliés par des arrêtes (orientées ou non) alors qu'un hypergraphe est une généralisation des graphes non orientés dont les arrêtes peuvent relier plus de deux sommets.

En Dynamique Moléculaire classique, les cellules formant la grille sont les sommets d'un graphe et la table de connectivité constitue les arêtes. Les méthodes de répartition de graphe consisteront à regrouper les sommets en sous-domaines dont la forme géométrique ne rentre pas en ligne de compte. La répartition de la charge tient compte du poids pour chaque sommet du graphe tout en cherchant à minimiser le nombre d'arêtes, pondérés par leurs valeurs, l'interface des sous-domaines. Bien que coûteuses, ces méthodes ont l'avantage de prendre en compte la connectivité entre les éléments du graphe et donc de chercher à minimiser les communications entre les sous-domaines.

Dans ce paragraphe, nous présentons les méthodes utilisées dans ce manuscrit, c'est-à-dire celles fournies par l'Api Metis (paragraphe 5.3.2) et SCOTCH (paragraphe 5.3.2).

Metis

Metis est une librairie fonctionnant sous la forme d'une Api. Elle est développée par le Karypis Lab. Metis permet de partitionner des graphes structurés et non structurés, des maillages en élément finis et d'ordonner ses matrices creuses en matrices diagonales par blocs. Metis est fondé sur les algorithmes de partitionnement de graphe dits à multi-niveaux, illustrés par la figure 5.10, comme l'algorithme *multilevel recursive bisection* [72], *multilevel k-way graph partitioning* [73] et *multi-constraint* [70]. Ils sont composés de trois phases : une phase d'agglomération, une phase de partitionnement de graphe et une phase de raffinement.

Durant la phase d'agglomération, un graphe grossier issu du graphe fourni par l'utilisateur est généré en agglomérant un ensemble de sommets proches en un sommet. Le poids des sommets formé est égal à la somme des poids de leurs sommets fusionnés. De même que les poids des arêtes entre les sommets sont obtenus en effectuant la somme des poids des arêtes entre les sommets de l'ancien graphe. Pour cela, plusieurs algorithmes existent comme le *random matching* [24] ou *Heavy edge matching* [72].

La phase de partitionnement consiste à appliquer un algorithme de partitionnement sur le graphe le plus grossier. Pour cela plusieurs algorithmes sont disponibles comme l'algorithme de bissection spectrale [110, 11], bissection géométrique[93] ou basés sur des heuristiques [77].

La dernière phase consiste à effectuer la transformation inverse à la phase d'agglomération jusqu'à obtenir le graphe initial partitionné. À chaque raffinement de graphe, les sommets sont divisés. Les sommets divisés se retrouvent dans les sous-domaines de leur sommet grossier. De plus, des algorithmes de raffinement de partitionnement sont utilisés, sur les interfaces entre les sous-domaines afin d'affiner le partitionnement, comme par exemple l'algorithme de Kernighan–Lin [77, 52].

Extension parallèle de Metis : ParMetis

ParMetis [76] est la librairie de Metis dont les algorithmes de partitionnement multi-niveaux sont parallélisés [74, 80] en mémoire distribuée (Mpi) et en mémoire partagée. ParMetis peut ainsi effectuer la répartition de la charge d'une simulation contenant jusqu'à plusieurs milliards de sommets. ParMetis inclus notamment des routines pour des simulations avec des grilles Amr [120]. Pour cela l'utilisateur doit fournir une grille grossière et l'arborescence de la structure en octree de chaque élément grossier de la grille.

Une dernière librairie liée à Metis est hMetis. Elle permet notamment de partitionner des hypergraphes. Elle est basée sur l'algorithme à multi-niveaux d'hypergraph [71, 75].

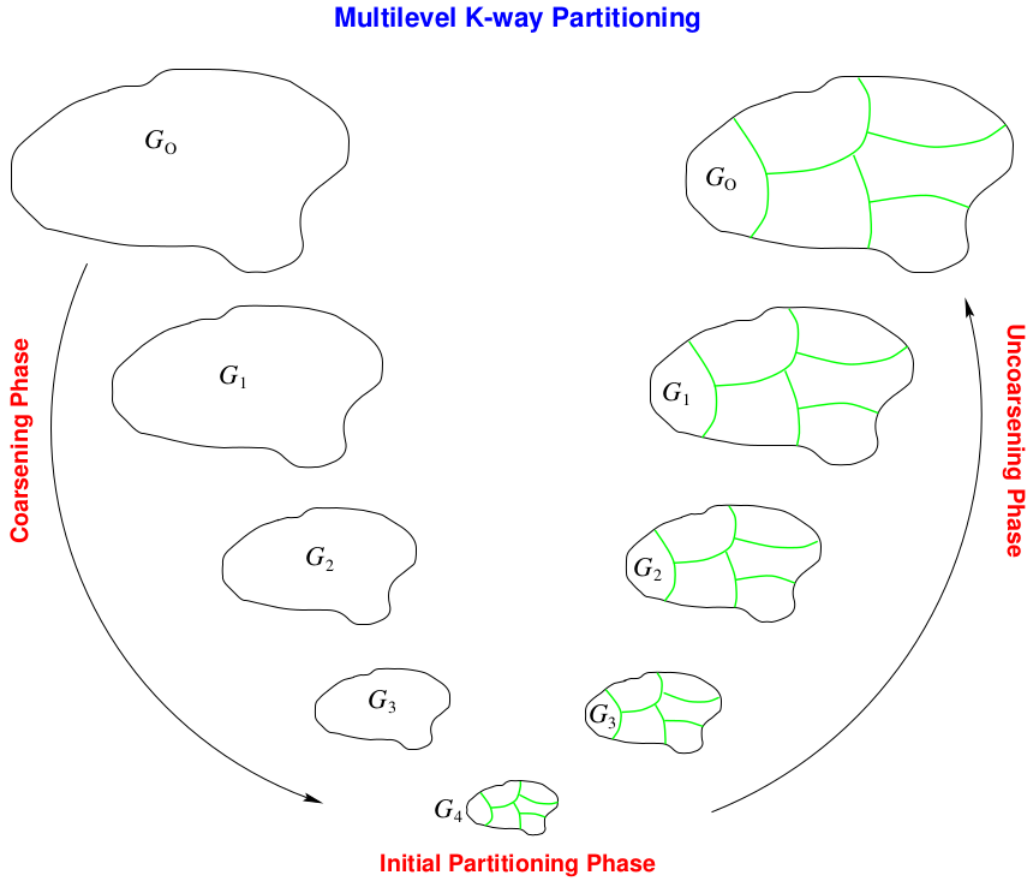


Figure 5.10 | *Multilevel k-way graph partitioning* [74] : L'algorithme s'effectue en trois phases, une phase d'agglomération pour simplifier le graphe, une phase d'initialisation du partitionnement sur le graphe simplifié puis une phase de raffinement afin de récupérer le graphe initial.

SCOTCH

SCOTCH [106] est une librairie proposant des interfaces en C et Fortran pour partitionner des graphes structurés et non-structurés, des maillages en élément finis et des matrices creuses. SCOTCH est développée par le laboratoire LaBri de l'université de Bordeaux. SCOTCH est basée sur l'algorithme Dual Recursive Bipartitioning (Drb). Drb est fondé sur l'idée des algorithmes "diviser et conquérir" et inclut de nombreuses variantes. PT-SCOTCH [27] est une extension de SCOTCH incluant une parallélisation utilisant l'interface Mpi et pouvant s'appuyer sur un parallélisme par Posix Thread. PT-SCOTCH est généralement mis en compétition avec ParMetis car les deux bibliothèques s'appliquent dans les mêmes contextes et ont généralement des résultats similaires. Dans la suite de la thèse seul ParMetis est utilisée car il est suffisant pour démontrer les performances des développements proposés.

Raffinement de graphe

Une extension des partitionnements de graphe sont les méthodes de raffinement de graphe. Elles consistent à partitionner un graphe déjà existant dont les poids des sommets et des arêtes ont évolué. Ces méthodes sont généralement moins coûteuses que de partitionner un nouveau graphe. Pour cela il existe deux stratégies principales pour raffiner un graphe : le scratch-remap [33] et

les méthodes de diffusion [77, 121]. La première consiste dans un premier temps à calculer une nouvelle partition pour un graphe (FROM SCRATCH), ensuite l'idée est d'adapter la différence entre le partitionnement actuel et le nouveau afin de minimiser les coûts de migration des sommets/cellules en DM. Les méthodes de diffusion consistent à migrer les sommets au bord de chaque partition du graphe jusqu'à atteindre le seuil de déséquilibre souhaité.

En DM, entre deux partitionnements, bien que les atomes aient évolués spatialement, ceux-ci n'ont pas drastiquement changé de position. Pour cela, lors de la répartition de la charge, les méthode de raffinement de graphe sont privilégiées au détriment des méthodes de partitionnement de graphe car le raffinement de graphe permet d'éviter au maximum que les atomes migrent d'un processus MPI à un autre.

Les codes de Dynamique Moléculaire classique

Dans le contexte de la Dynamique Moléculaire classique, de nombreux codes de simulation ont été développés en tenant compte des récentes évolutions des supercalculateurs. Afin d'optimiser la répartition de charge des codes de DM, les développements sont réalisés dans le code de DM : EXASTAMP. Celui-ci est présenté dans le paragraphe 5.4.1. Afin de valider les développements, ceux-ci sont comparés avec les optimisations déjà réalisées dans le code de référence LAMMPS. LAMMPS est présenté dans le paragraphe 5.4.2. Dans le paragraphe 5.4.3, une liste non-exhaustive d'autres logiciels de DM est réalisée.

ExaSTAMP

Le code Exaflopique de Simulation Temporelle Atomistique et Moléculaire Parallèle (EXASTAMP) [30] est un code de simulation de Dynamique Moléculaire classique développé au CEA depuis 2013 afin de renforcer le code STAMP pour les futurs supercalculateurs exaflopiques. EXASTAMP est écrit en C++ et reprend les principaux fondements des codes de DM afin de paralléliser les calculs. Il utilise les trois niveaux de parallélismes MPI + OPENMP/TBB+vectorisation.

EXASTAMP incorpore plusieurs ensembles thermodynamiques (voir chapitre 3) comme l'ensemble NVE, NVT et NPT. Les potentiels analytiques d'interactions inter-atomiques à courte portée LJ, exp-6, EAM shutton-chen et vniitf, MEAM Baskes, sont intégrés et permettent de répondre à la grande majorité des simulations DM négligeant les interactions à longue portée. Des potentiels purement numériques comme le potentiel SNAP sont aussi intégrés. EXASTAMP incorpore un système *in situ* [43] permettant de traiter un ensemble d'analyses pendant la simulation pour éviter les surcoûts de stockage. Une extension d'EXASTAMP incorporant les simulations moléculaires comme les polymères est en train d'être réalisée.

Rappelons qu'en Dynamique Moléculaire classique, une simulation consiste à faire évoluer un ensemble d'atomes ou de molécules dans un volume donné (nommé domaine) au cours du temps. Le code EXASTAMP est principalement conçu pour traiter les interactions à courte portée comme c'est le cas pour les simulations présentées dans le paragraphe 3.5. C'est pourquoi, afin de déterminer rapidement le voisinage de chaque atome, son architecture est basée sur la méthode des cellules liées pouvant optionnellement être combinée avec la méthode de la liste de Verlet. L'architecture d'EXASTAMP a été définie selon trois niveaux hiérarchiques et illustrée par la figure 5.11 afin de tirer partie de la structure des supercalculateurs : les cellules, la grille et les sous-domaines. Ces trois niveaux sont détaillés dans les paragraphes suivants.

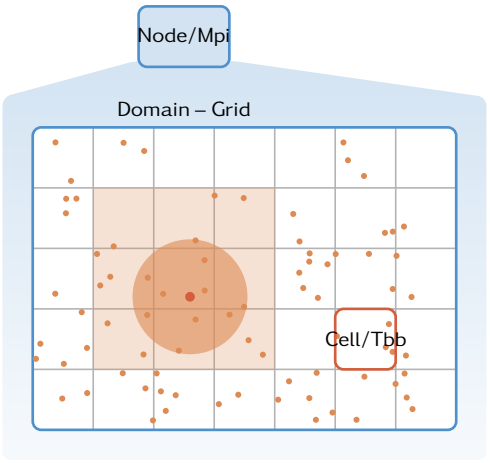


Figure 5.11 | Architecture du logiciel ExaSTAMP comprenant 3 formes de parallélisme, Mpi entre les domaines, Tbb/OpenMP entre les cellules et vectorisation des opérateurs appelés par les cellules.

Cellules et vectorisation

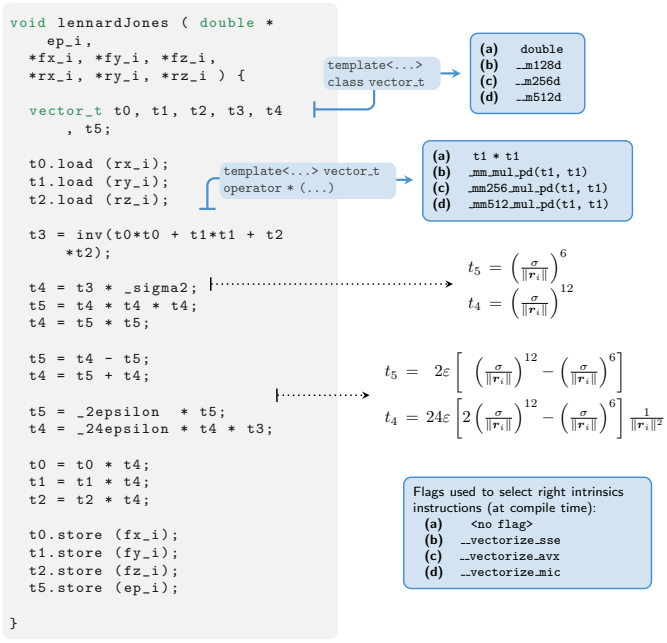


Figure 5.12 | Écriture de l'opérateur du potentiel Lennard Jones dans ExaSTAMP écrit avec le jeu d'instructions Intel intrinsics. ExaSTAMP utilise un niveau d'abstraction en utilisant une classe vector_t pouvant intégrer différents types d'instructions (sse, avx-256bits et avx-512bits).

Comme introduit dans le paragraphe 5.1.2, la cellule est la maille élémentaire utilisée par la méthode des cellules liées pour décomposer le domaine de simulation en une grille structurée. Chacune de ces cellules est définie par sa position sur la grille cartésienne et une structure de stockage. Le stockage est une structure de tableaux contenant chacun un champ d'information d'une particule comme sa position ou sa vitesse. Le nombre de champs varie selon le type de la particule (atome, molécule). Les listes de Verlet, décrites dans le paragraphe 3.4.2, sont aussi stockées au niveau de la cellule. Le but est de donner à la classe cellule toutes les informations mémoire nécessaires pour lui appliquer des opérateurs de calcul comme le calcul du potentiel ou la recherche de voisins.

Les opérateurs de calcul sont écrits en utilisant des instructions Intel Intrinsics (Simd) [29], voir figure 5.12, permettant ainsi d'effectuer le parallélisme de données, décrit dans le paragraphe 4.2.3. De cette façon, ExaSTAMP exécute les opérateurs plus rapidement sur les dernières générations de processeurs car la tendance est à l'augmentation du nombre de données traitées par instruction Simd.

la grille et parallélisation intra-Nœud

La grille correspond à un ensemble de cellules utilisées pour construire les listes de Verlet. Le parallélisme en mémoire partagée avec les Api OpenMP ou Tbb s'applique en parcourant les cellules de la grille. Une cellule est donc traitée séquentiellement. Lorsque le nombre d'atomes par cellule est équivalent, cette stratégie permet de diviser le temps de calcul avec un thread par le nombre de threads utilisés comme l'atteste la figure 5.13 sur différentes machines multicœurs.

ExaSTAMP permet de gérer des ensembles de cellules formant une grille cartésienne formant un parallélépipède rectangulaire nommée grille rectilinéaire, mais aussi une grille structurée dont la forme est quelconque. Elle est nommée grille any. La grille any peut notamment gérer un ensemble de cellules formant plusieurs volumes non-connexes. Néanmoins cette souplesse se fait au détriment d'un surcoût d'appel de fonctions testant si la cellule appartient à la grille.

Sur chaque grille, les cellules sont classées dans des structures nommées *traversals* pour permettre des parcours de cellules sur des cas particuliers comme les cellules sur le bord de la grille ou celles dans les zones fantômes ou celles excluant les cellules fantômes. La grille aura donc pour rôle d'associer les opérateurs et les cellules.

De plus, pour éviter que plusieurs threads modifient la même donnée en même temps dans les portions multithreads, ExaSTAMP utilise des mutex (verrous mémoire, cf paragraphe 4.2.2). Ces mutex assurent que lorsqu'un thread modifie une donnée, aucun autre thread ne peut accéder à cette donnée (chargement ou stockage). Néanmoins l'utilisation de mutex a un impact négatif sur la parallélisation en mémoire partagée à cause des coûts d'appels des mutex et des synchronisations

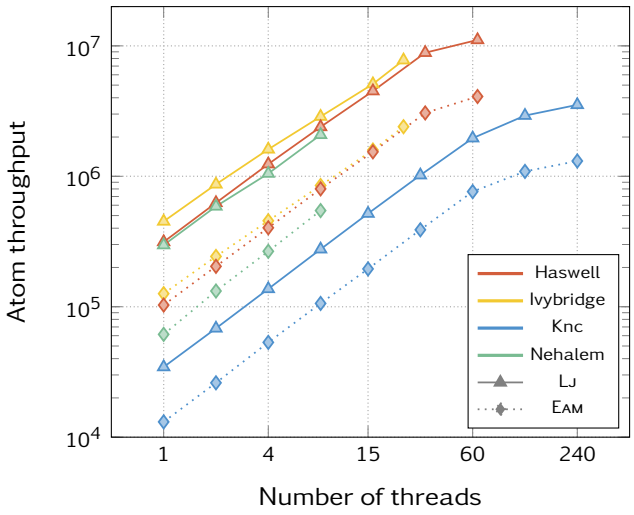


Figure 5.13 | Comparaison du nombre d'atomes traités par seconde avec un potentiel Lj et Eam sur divers processeurs multicœurs.

Les sous-domaines et parallélisation inter-Nœud

Le domaine de la simulation est divisé en sous-domaines. Les sous-domaines sont répartis entre les différents processus Mpi nommés node par la méthode de partitionnement spatial de domaine (paragraphe 5.2) et une grille rectilinéaire ou any est associée à chaque sous-domaine. En pratique un node est associé à un Socket ou un Nœud du supercalculateur. Dans le cas d'une simulation dont la distribution de la charge de calcul ne suit pas une loi uniforme, comme par exemple un éjecta de matière ou l'impact de goutte, ExaSTAMP permet de répartir la charge entre les sous-domaines via l'Api Zoltan. Zoltan permet d'effectuer les méthodes de partitionnement géométrique (RCB, RIB et SFC) ainsi que les méthodes de partitionnement de graphe (ParMetis et PT-SCOTCH). La forme des sous-domaines obtenue par chaque méthode, exceptée la méthode RCB, nécessite une certaine souplesse, c'est pourquoi la grille any a été implémentée.

Limitation actuelle d'ExaSTAMP

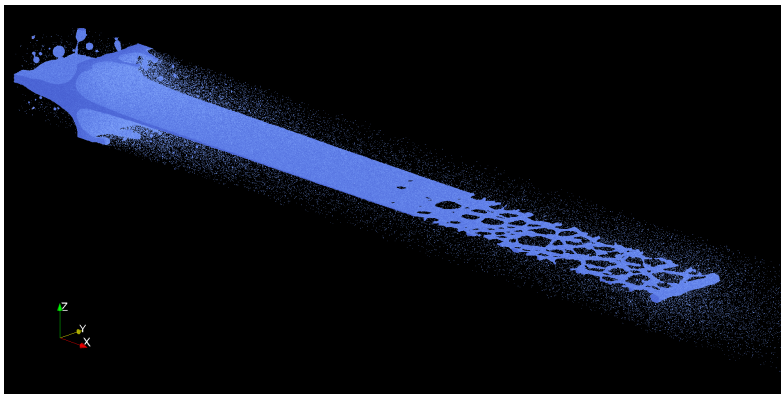


Figure 5.14 | Simulation d'un micro-jet après une nanoseconde. La simulation a été effectuée avec le logiciel ExaSTAMP sur un système de 200 millions d'atomes d'étain et dont les interactions sont modélisées par un potentiel Eam (rCut=5.6Å). On peut notamment observer que la majorité du domaine ne contient pas d'atome causant alors des surcoûts de la méthode des cellules liées.

Lors d'une simulation d'un micro-jet, la nappe induite par la propagation d'un choc dans une rainure de la paroi se déplace selon une direction. Le domaine de la simulation est alors ajusté au fur et à mesure de la progression de la nappe. Le maillage en cellule généré possède un nombre de cellules excédant le nombre d'atomes. Cette expérience a été réalisée par E. Cieren durant sa thèse [29] sur le développement d'un micro-jet d'étain d'environ 200 millions d'atomes, voir figure 5.14. Par une étude sur l'impact de la taille des cellules sur l'empreinte mémoire et sur la vitesse de traitement des atomes avec un potentiel Eam, il a pu mettre en évidence qu'en doublant le volume des cellules, l'empreinte mémoire est divisée par quatre et le temps de calcul par deux.

Taille des cellules (Å)	6.00	6.60	7.00	7.50	8.00	8.50	9.00	10.00
Nombre de cellules (10^6)	804.9	604.8	506.9	412.1	339.6	283.1	238.5	173.9
Mémoire (MB/core)	1279.2	979.0	837.4	693.4	576.2	488.3	417.5	306.3
Temps par particule par cœur (10^{-5} s)	4.23	4.01	3.43	2.84	2.36	2.00	1.71	1.92

Table 5.2 | Evolution de l'utilisation de la mémoire par cœur et du temps à traiter les interactions (Eam) d'une particule par cœur en fonction de la taille des cellules.

L'empreinte mémoire totale de la simulation est donc principalement définie par l'empreinte mémoire engendrée par le stockage des atomes, des listes de voisins et du coût mémoire des structures des cellules. Le nombre d'atomes et d'interactions ne variant pas, ceux-ci ont un coût de stockage presque fixe. De plus lorsque la taille des cellules augmente, le nombre d'atomes à migrer à chaque pas de temps diminue. Finalement, si l'empreinte mémoire de la simulation varie autant c'est à cause de l'augmentation du stockage des cellules. Concernant la réduction du temps de calcul, plusieurs facteurs peuvent rentrer en compte :

- La répartition de charge obtenue avec moins de cellules est de meilleure qualité car les poids des cellules vides (nombreuses) ne viennent pas dégrader la répartition.
- Le coût d'appel des opérateurs de calcul n'est pas négligeable lorsque le nombre de cellules vides devient excessif. Ainsi appeler l'opérateur du potentiel sur des millions de cellules impacte les performances de la simulation. À noter que la recherche de voisins n'est pas impactée par des cellules plus grandes grâce à une pré-détermination des candidats (atomes voisins) potentiels. Lorsque l'on cherche les atomes voisins entre des atomes de deux cellules adjacentes, un masque est utilisé afin d'éliminer tout les atomes ne pouvant interagir avec n'importe quel atome de l'autre cellule.
- Une dernière possibilité est que la taille des cellules crée naturellement un effet de *cache blocking* permettant une meilleure réutilisation des données dans les caches L1.

Ainsi la raison principale pour laquelle ExaSTAMP perd en performance pour cette simulation est dûe au maillage en cellules qui n'est pas adapté aux parties du domaine ne contenant pas ou peu d'atomes.

LAMMPS : Un logiciel de référence

Large-Scale Atomic/Molecular Massively Parallel Simulator (Lammps) [109] est l'un des simulateurs de Dynamique Moléculaire classique le plus utilisé sur des supercalculateurs. Il est développé principalement par le laboratoire national Sandia du département de l'énergie des USA. Lammps intègre un très grand nombre de fonctionnalités pour la simulation de systèmes d'atomes, de polymères, biologiques, métalliques ou à l'échelle mésoscopique, utilisant une variété de champs de forces et de conditions aux limites. Comme ExaSTAMP, l'architecture du logiciel Lammps, voir figure 5.15, utilise la méthode des cellules liées et des listes de Verlet. Lammps permet d'effectuer des simulations allant jusqu'à plusieurs milliards d'atomes sur des dizaines de milliers de cœurs.

Bien que la version standard de LAMMPS soit uniquement parallélisée avec la librairie Mpi, de nombreux packages ont été développés pour intégrer la parallélisation en mémoire partagée : comme USER-OMP ou USER-INTEL [21]. Le package KOKKOS [48] a été intégré pour optimiser Lammps sur les accélérateurs (Intel Xeon Phi et GPU) en se basant sur le support d'exécution du même nom. Malgré des optimisations spécifiques pour l'architecture Knl, la version OpenMP de Lammps est 10 à 15% plus lente que la version standard sur un seul processeur.

On notera que Lammps diffère d'ExaSTAMP par sa stratégie de stockage des atomes. Les atomes sont stockés dans une seule et unique structure de tableaux par sous-domaine contrairement au stockage par cellule. Lammps effectue notamment sa parallélisation en mémoire partagée en parcourant les atomes au lieu des cellules.

Concernant l'équilibrage de charge, Lammps propose de partitionner l'espace en affectant des poids à chaque particule (atome/molécule). Lammps permet uniquement des sous-domaines en

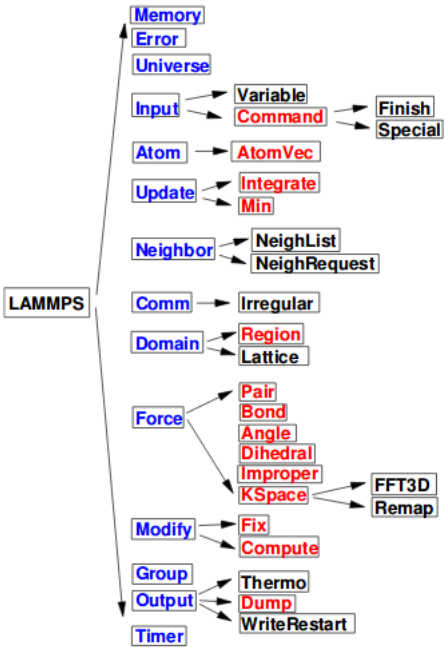


Figure 5.15 | Hiérarchie des classes à l'intérieur du code source de la partie standard de LAMMPS.

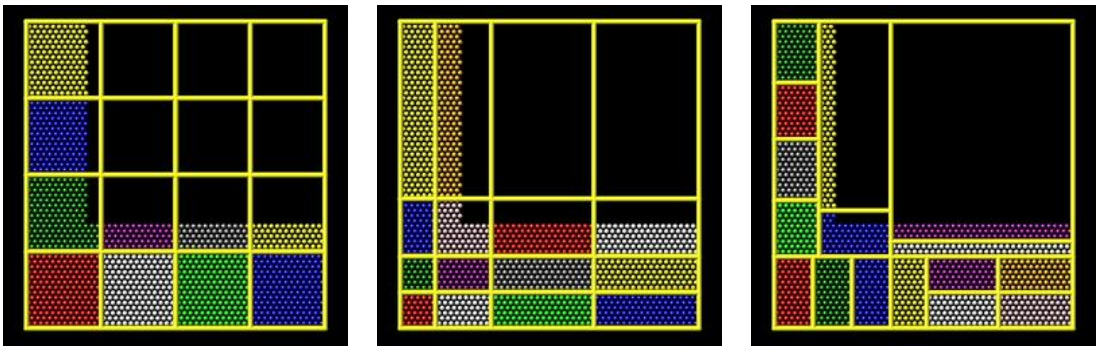


Figure 5.16 | Équilibrage de charge du logiciel LAMMPS. Les images représentent une simulation déséquilibrée sans répartition (gauche), avec la méthode shift (milieu) et la méthode RCB (droite).

forme de pavé, c'est pourquoi uniquement la méthode shift ou la méthode RCB, voir figure 5.16, sont utilisées pour un équilibrage statique et dynamique.

Finalement, LAMMPS est utilisé tout au long de notre campagne d'expérimentation afin de comparer les développement réalisés dans EXASTAMP concernant l'optimisation de la répartition de la charge de calcul pour les 3 niveaux de parallélisme.

Autres Logiciels

DL_POLY

DL_POLY [124] est un logiciel de DM développé par le laboratoire de Daresbury. Il apporte un ensemble de fonctionnalités et fichiers de données, est écrit en fortran 90, est conçu pour faciliter les simulations DM et est utilisé pour la production. Ce logiciel est conçu pour fonctionner sur UCT avec OPENMP et MPI, ainsi que sur GPU. L'une des lacunes de DL_POLY est qu'il n'intègre

pas d'algorithme d'équilibrage de charge pour compenser une mauvaise distribution de densité. Compte tenu de ce dernier point, il est difficilement envisageable d'utiliser DL_POLY pour des simulations comme le développement d'un micro-jet ou l'impact d'une nano-goutte d'étain sur une surface solide.

Gromacs

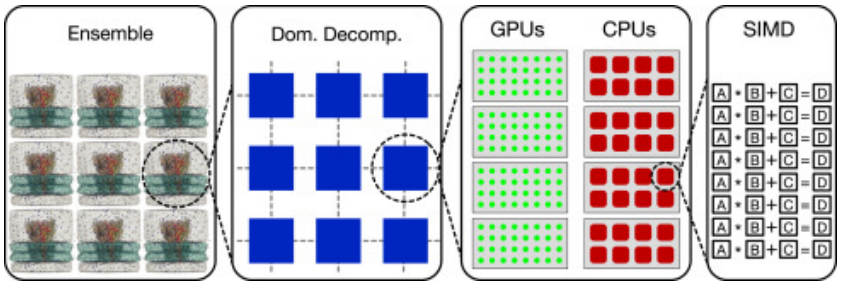


Figure 5.17 | Parallélisme [1] à plusieurs niveaux dans Gromacs. Les registres Simd sont utilisés pour paralléliser les interactions entre les particules pour chaque cœur. OpenMP est utilisé pour le parallélisme à l'intérieur des sous-domaines tandis que les interactions non liées sont gérées par les Gpu ou autres accélérateurs. Le Mpi avec équilibrage de charge est utilisé pour décomposer une simulation en domaines sur plusieurs Nœuds d'un calculateur.

GROningen MAchine for Chemical Simulations (Gromacs) [14] est un code développé depuis 1995. Gromacs est implémenté en C avec des fonctions Simd et CUDA pour la portabilité sur carte graphique. Comme les autres logiciels de Dm, Gromacs contient des algorithmes de recherche de voisins basés sur la méthodes des cellules liées et les listes Verlet. Alors qu'il était initialement conçu pour les simulations de molécules biochimiques comme les protéines, les lipides et les acides nucléiques qui ont beaucoup d'interactions liées, il est possible d'effectuer des simulations "tout-atome" incluant uniquement des interactions à courte et à longue portée avec des potentiels tel que le Lennard Jones ou le potentiel de Buckingham. Afin d'équilibrer la charge, Gromacs utilise la méthode Eighth Shell [64, 82].

Mini-applications

Plusieurs versions simplifiées de codes complexes Dm ont récemment été développées pour servir de banc d'essai pour diverses optimisations sur des architectures multicœurs. Le logiciel CoMD a été conçu pour étudier les propriétés dynamiques de divers liquides et solides dans le cadre du projet ExMatEx [96]. L'implémentation de référence de CoMD utilise OpenMP, Mpi, OpenCL, et peut utiliser des listes de Voisins.

MiniMd est une version simplifiée de Lammps écrite en C++. MiniMd est un logiciel de simulation de Dm parallèle écrit en C++ et destiné à être utilisé sur des supercalculateurs parallèles et de nouvelles architectures à des fins de tests. Grâce à sa simplicité (près de 5000 de lignes), il est possible d'explorer des optimisations potentielles beaucoup plus rapidement. Bien que seuls les potentiels des modèles Lennard Jones et Eam soient supportés, les performances et le comportement de mise à l'échelle de miniMD sont représentatifs de codes beaucoup plus grands et complexes avec parallélisation hybride Mpi + OpenMP.

Conclusion

Dans ce chapitre, les méthodes de maillage pour les simulations à N-corps ont été abordées dans une première partie. Celles-ci permettent principalement d'accélérer la recherche de voisins ou la détection de collisions. Nous avons décrit les grilles à blocs structurés avec raffinement de maillage adaptatif. Ces maillages sont présents dans une grande majorité des codes d'hydrodynamique ou en cosmologie afin d'adapter le maillage aux différentes densités du domaine de simulation. Cependant, ceux-ci n'ont pas été introduit en Dynamique Moléculaire classique alors que certaines simulations comme l'impact d'une nano-goutte d'étain sur une surface solide génère un très grand nombre de cellules vides.

De plus, dans une deuxième partie, les principales méthodes de répartition de la charge ont été décrites. Celles-ci sont utilisées pour pallier les défauts de la méthode de partitionnement spatial de domaine et les domaines obtenus par la majorité des méthodes ne sont plus des parallélépipèdes rectangulaire. De plus, ces méthodes ont déjà été introduites dans de nombreux codes de simulation en Dm. Dans notre étude, les méthodes RCB, RIB, SFC et ParMetis sont utilisées.

Dans une dernière partie, l'architecture du logiciel ExaSTAMP a été particulièrement détaillée car celui-ci servira de base pour les développements de cette thèse. ExaSTAMP a l'avantage d'être un code proposant les fonctionnalités principales de la plupart des codes Dm ainsi qu'un système de répartition de charge. Néanmoins celui-ci perd en efficacité lorsque les simulations sont trop extrêmes comme le développement d'un micro-jet ou l'impact de nano-goutte car la grille structurée est trop coûteuse quand elle maille les espaces vides. L'idée principale de cette thèse repose sur l'implémentation d'une grille à blocs structurés avec la méthode de raffinement de maillages adaptatifs. Afin de pouvoir comparer les contributions de cette thèse, le logiciel de référence Lammps a été brièvement introduit ainsi que les logiciels les plus répandus.

6 Développement d'une grille à blocs structurés AMR adaptée à la dynamique moléculaire pour les nouvelles architectures de processeur

Dans le contexte des simulations de Dynamique Moléculaire classique, répartir équitablement la charge de calcul entre les ressources d'un supercalculateur n'est pas une opération triviale. Surtout si la simulation considérée a une densité d'atomes non uniforme et qu'elle évolue rapidement comme c'est le cas lorsqu'une nano-goutte d'étain est projetée contre une surface solide.

Comme nous l'avons vu dans le chapitre 5, une large majorité des simulations de DM utilisent les méthodes des cellules liées et des listes de Verlet pour construire les listes des atomes voisins nécessaires aux calculs des interactions à courte portée. Celles-ci nécessitent l'utilisation d'une grille structurée en cellules permettant d'accélérer la construction des listes de voisins nécessaires pour le calcul des interactions à courte portée. La majorité des codes de simulation en DM s'appuient sur cette grille structurée pour exploiter les trois niveaux de parallélisation vus dans la partie 4.2 : vectorisation, en mémoire partagée et en mémoire distribuée. Pour les simulations dont la densité d'atomes est uniforme, les temps de simulation observés dans le paragraphe 5.4.1 par le code ExaSTAMP ont une scalabilité quasi linéaire, c'est-à-dire que le temps à n cœurs est environ égal au temps à un cœur divisé par n. Néanmoins pour les simulations hétérogènes comme l'évolution d'un micro-jet, la grille structurée engendre des surcoûts de calcul à cause des cellules contenant peu ou pas d'atomes. Comme observé dans le paragraphe 5.4.1, elles impactent la répartition des calculs sur les threads et dégradent la répartition de charge réalisée par les méthodes de partitionnement.

Dans le paragraphe 6.1, une étude de la méthode des cellules liées est réalisée sur des scénarios basiques avec le logiciel ExaSTAMP afin de mettre en évidence les limites de l'utilisation de la grille structurée. Par la suite nous discutons des avantages et inconvénients de l'intégration de la méthode de Raffinement de Maillage Adaptatif dans les codes de DM.

Afin de démontrer les avantages de l'utilisation de la méthode AMR pour des simulations de DM, comme par exemple pour les deux simulations cibles présentées dans le paragraphe 3.5, une implémentation d'une grille AMR est effectuée au sein du code ExaSTAMP. Le processus d'élaboration de la grille AMR et son développement sont exposés dans le paragraphe 6.2. Nous démontrons que l'AMR résout en grande partie les problèmes engendrés pour les simulations cibles.

Dans le but de tenir compte de l'architecture des supercalculateurs, les méthodes d'optimisations utilisées pour améliorer la réutilisation des données stockées dans les caches mémoires sont abordées dans le paragraphe 6.3. Ensuite dans le paragraphe 6.4, nous nous intéressons aux optimisations des opérateurs lors du calcul du potentiel pour une structure AMR. Pour cela nous étudions l'impact du parallélisme de données bas niveau en instrumentant les opérateurs avec des instructions SIMD. En plus de cette optimisation, nous introduisons une variante novatrice de la méthode des listes de Verlet : les blocs de Verlet, qui, à notre connaissance, n'a pas été étudiée dans la littérature. Elle

permet d'optimiser l'opérateur de construction des listes de voisins et du potentiel tout en réduisant l'empreinte mémoire de la simulation.

Choix d'une méthode palliant les surcoûts engendrés par une grille structurée

Dans le paragraphe 5.4, nous avons vu que les codes de simulation comme ExaSTAMP sont pour la plupart basés sur la méthode des cellules liées (décrite dans le paragraphe 3.4.2). Elle génère une grille structurée en cellules dont une présentation a été réalisée dans le paragraphe 5.1.2. Dans le cas des simulations hétérogènes, les grilles structurées ne sont pas adaptées (surcoûts de calculs et mémoires) aux spécificités des simulations hétérogènes. Dans le paragraphe 6.1.1, nous mettons en évidence sur un cas simple les limites de la méthode des cellules liées. Ensuite, pour pallier aux défauts de la grille structurée en cellules de taille fixe, la solution proposée est de s'appuyer sur une grille à blocs structurés avec la méthode de raffinement de maillage adaptatif en octree afin d'adapter localement et temporellement la grille.

Étude des limites de la méthode des cellules liées

La grille structurée est formée par des cellules dont la forme est un parallélépipède rectangulaire et dont les dimensions sont proches d'un cube dont l'arrête est égale au Rayon de coupure (rCut). Rappelons que le rCut est le maximum des rayons de coupure des potentiels utilisés. Les particules de la simulation sont alors associées aux cellules. Cette méthode permet de construire les listes de voisins avec un algorithme d'une complexité linéaire en fonction du nombre de particules. Néanmoins nous allons démontrer que cette méthode admet des limites pour les simulations dont la majeure partie des cellules ne contiennent pas ou très peu d'atomes. Pour plus de simplicité d'écriture dans la suite du manuscrit, le code d'origine ExaSTAMP est nommé *ExaSTAMP legacy*.

Dans le cas d'*ExaSTAMP legacy*, les atomes appartenant à une cellule sont stockés dans une structure de tableaux. Ce type de structure offre la possibilité de traiter chaque champ d'information avec des instructions Simd. Une cellule correspond donc à un bloc d'atomes sur lequel sera appliqué un ensemble d'opérateurs : recherche de voisins, potentiel, schéma numérique. Ainsi les différents opérateurs appliqués aux atomes s'effectuent en parcourant les cellules de la grille. La disposition des atomes à l'intérieur des cellules apporte une localité mémoire ainsi qu'une flexibilité concernant l'association des opérateurs et des données. Le coût d'application d'un opérateur sur une cellule sans atome est très faible vis-à-vis d'une cellule contenant au moins un atome. Cependant lorsque la majorité des cellules ne contiennent pas d'atomes, le nombre de cellules peut excéder le nombre d'atomes d'un ou plusieurs ordres de grandeur. Les cellules ne contenant pas d'atome induisent alors un surcoût supérieur ou équivalent au temps de calcul des cellules contenant au moins un atome.

Afin de mettre en évidence le surcoût des cellules sans aucun atome, nous allons faire varier pour une simulation donnée le nombre de cellules sans aucun atome. Pour cela, le cas test suivant est utilisé : un cristal parfait de cuivre composé de 13 500 atomes est généré à partir de la réplication d'une lattice cubique face centrée. Les interactions entre les atomes de cuivre sont modélisées par un potentiel Lj avec un rCut de 5.6 Å.

Le volume du cristal est un cube dont la valeur est notée V_{matire}. Autour du cristal, un nombre variable de cellules est ajouté afin de faire varier le volume du système noté V_{systme} tout en conservant le même nombre d'atomes et d'interactions.

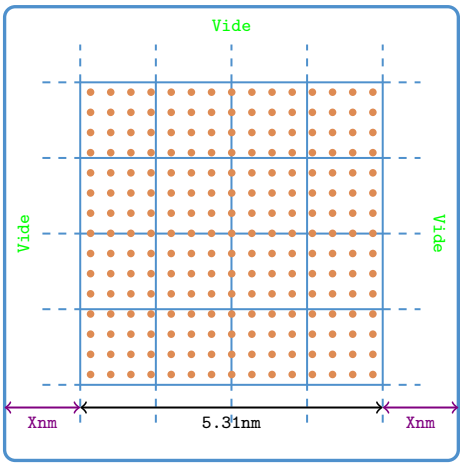
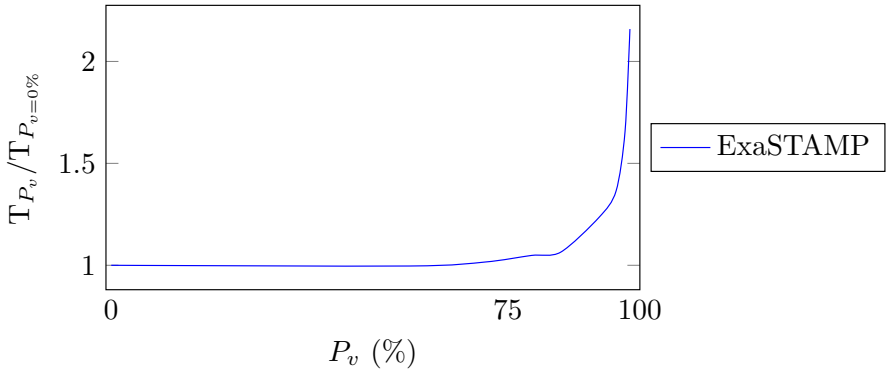


Figure 6.1 | Schéma explicatif d'une simulation générant un cristal parfait de cuivre de 13 500 atomes auquel un contour de cellules sans atome est ajouté. Le nombre de cellules ne contenant pas d'atome est piloté par la taille du domaine. Donc plus X_{nm} est grand, plus il y a de cellules ne contenant pas d'atomes suivant l'axe X.

Cette simulation est illustrée par la figure 6.1. Le principe de l'expérimentation est d'augmenter progressivement le nombre de cellules entourant le cristal en augmentant la taille système afin d'étudier l'impact des cellules ne contenant pas d'atome sur le temps de la simulation. Le pourcentage du domaine composé de cellules sans aucun atome est noté P_v. Il est défini par :

$$P_v = 100 * \frac{V_{systme} - V_{matire}}{V_{systme}} \%.$$

La simulation est réalisée sans parallélisme avec le code *ExaSTAMP legacy* sur un processeur Intel Xeon Skylake (Skl) représentatif des processeurs multicœurs composant les supercalculateurs. Le graphe 6.2 correspond à la durée de la simulation selon le pourcentage du domaine composé de cellules sans atome normalisé par le temps de la simulation sans cellule ne contenant pas d'atome. On peut constater qu'en dessous de 75% du domaine vide, l'impact des cellules sans aucun atome sur le temps de calcul est négligeable. L'impact des cellules sans aucun atome devient prédominant lorsque le nombre de cellules ne contenant pas d'atome correspond à plus de 90% du nombre de cellules.



Graphe 6.2 | Impact des cellules ne contenant pas d'atome sur les performances d'*ExaSTAMP legacy*. $T_{P_v=0\%}$ correspond au temps sans cellule ne contenant pas d'atome autour du bloc de cuivre.

Ce phénomène est présent dans de nombreuses simulations de Dm. Par exemple, lors de l'évolution d'un micro-jet, le nombre de cellules sans aucun atome finit par avoisiner les 95% lors

de la propagation de la nappe. Concernant l'impact d'une goutte sur une surface solide, lors du choc avec la paroi, des particules sont projetées rapidement dans la direction des axes formant la surface ce qui entraîne un agrandissement rapide du domaine de la simulation. Le nombre de cellules générées, et ne contenant pas ou peu d'atomes, dépasse au bout de quelques milliers de pas de temps les 99.8%, soit une cellule contenant des atomes pour 500 cellules sans aucun atome.

L'empreinte mémoire des cellules sans aucun atome de l'expérimentation 6.1 est alors désastreuse. En effet l'empreinte mémoire de *ExaSTAMP legacy* passe de 45Mo pour $P_v = 0\%$ à 258Mo pour $P_v = 98.12\%$ alors que le système n'est composé que de 13 500 atomes. Toutefois l'augmentation est moins importante pour d'autre code comme Lammps car il stocke moins d'éléments par cellule. Pour ce cas, Lammps passe de 45Mo pour $P_v = 0\%$ à 58Mo pour $P_v = 98.12\%$. Le temps lui augmente seulement de 10%. Les effets des cellules sans aucun atome sont visibles pour un plus grand pourcentage du domaine vide : 99%, dans ce cas le temps d'exécution est 4 fois plus long.

Pour cette expérimentation, un potentiel à faible coût est utilisé afin de mettre en évidence rapidement les surcoûts des calculs et de la mémoire d'une grille structurée. Avec des potentiels plus coûteux comme les potentiels Eam et Meam, la tendance est conservée mais les effets sont moins visibles avant 90% de cellules ne contenant pas d'atome.

En plus du surcoût des calculs dus aux appels des opérateurs sur des cellules ne contenant pas d'atome, les cellules sans charge de calcul impactent aussi la parallélisation intra-Nœud. En effet, l'ordonnanceur responsable de la répartition des calculs sur les threads (d'un Nœud ou d'un Socket) doit répartir un grand nombre de calculs dont la charge de calcul est extrêmement variable. Les problèmes de parallélisation par threads sont abordés dans le chapitre 7.

On peut aussi observer que lors de la parallélisation en mémoire distribuée (Mpi), les poids attribués aux cellules lors du partitionnement du domaine en sous-domaines sont faussés car ils sont généralement liés au nombre d'atomes. Une solution est d'ajouter un poids artificiel et constant aux cellules ne contenant aucun atome. La répartition de la charge est alors biaisée par un trop grand nombre de cellules vides combiné à des poids difficiles à ajuster. Ce point est développé dans de chapitre 8.

Raffinement de maillages adaptatifs et Dynamique Moléculaire

Dans le paragraphe 5.4.1, l'intégration d'une grille structurée dans le code *ExaSTAMP legacy* a déjà établi que la méthode des cellules liées permet une scalabilité quasi linéaire pour une parallélisation hybride Mpi+Tbb sur des dizaines de milliers de cœurs pour des simulations dont la densité d'atome est uniforme. Cependant dans le paragraphe 6.1.1 nous avons mis en évidence que la grille structurée illustrée par la figure 6.3-(a) n'est pas adaptée lorsque le nombre de cellules ne contenant pas d'atome est trop important.

Une première stratégie utilisée est d'ajuster le volume des cellules afin de trouver le compromis optimal entre le nombre de cellules et le surcoût induit par la recherche de voisins, voir figure 6.3-(b). Par exemple, pour la simulation d'un micro-jet à 200 millions d'atomes présenté lors du paragraphe 5.4.1 (portant sur les limites d'*ExaSTAMP legacy*), les cellules ont été agrandies et le temps de simulation a alors été réduit par deux.

Une deuxième stratégie pour réduire l'impact des cellules ne contenant pas d'atomes est d'utiliser une grille structurée adaptée, illustrée par la figure 6.3-(c) et définie dans le paragraphe 5.1.2. Cette solution est assez efficace sur une large variété de simulations comme le cas particulier illustré par la figure 6.1. Il est possible de faire mieux car on peut observer encore des cellules sans aucun atome inutile pour construire les listes des atomes voisins. De plus, dans le cas de la figure 6.3-(d), la grille adaptée structurée équivaut à une grille structurée. Le temps de la simulation est alors

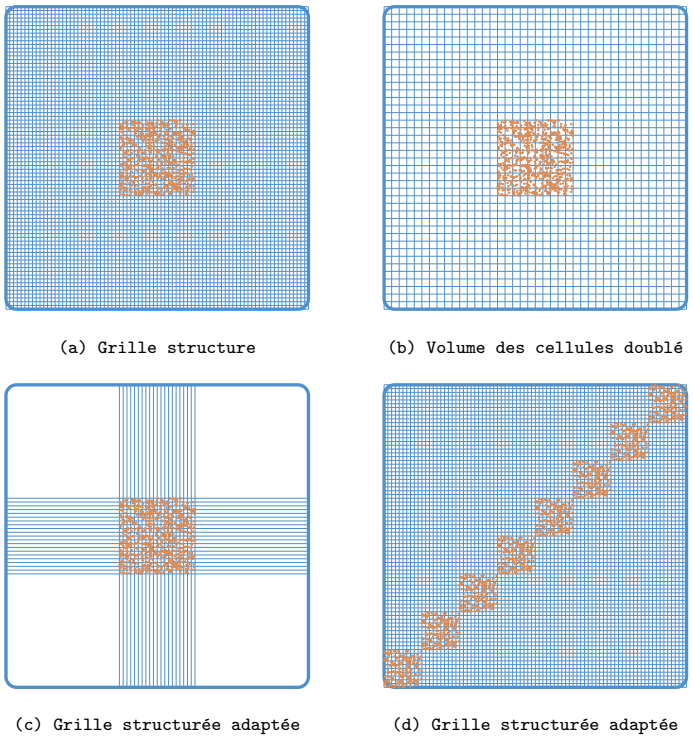


Figure 6.3 | Disposition d'atomes sur différentes grilles : (a) grille structurée - méthode des cellules liées. (b) grille structurée dont le volume des cellules est doublé. (c) grille adaptée structurée. (d) grille adaptée structurée équivalente à une grille structurée.

plus important que celui obtenu par la stratégie précédente. Cette stratégie n'est donc pas assez générique pour s'adapter à des simulations complexes et n'a pas été retenue.

Une troisième stratégie est d'exploiter une grille non structurée. Pour cela une table de connectivité est ajoutée. Dans ce cas, l'ensemble des cellules adjacentes à une cellule ne possède pas forcément la totalité des atomes nécessaires pour construire les listes des atomes voisins. Des règles géométriques sur la construction de la grille sont alors nécessaires. Cette stratégie peut produire une grande variété de grilles (selon la forme des cellules, leurs tailles, etc) réduisant le nombre de cellules ne contenant pas ou peu d'atomes mais au détriment d'un surcoût de stockage et de règles supplémentaires à définir lors de la construction de la grille. C'est pourquoi cette stratégie n'a pas été choisie non plus.

La dernière stratégie est d'introduire une grille à blocs structurés intégrant la méthode de raffinement de maillage adaptatif et la méthode de l'octree ou plus simplement la grille AMR. Le principe est de subdiviser récursivement les cellules en 8 cellules de même volume tant que le critère de raffinement est respecté, voir figure 6.4. Contrairement à la grille adaptée structurée, cette stratégie offre la possibilité de traiter n'importe quelle partie du domaine indépendamment des autres. Le principal défaut de la grille AMR est de dégrader la recherche de voisins dans certaines zones de la simulation. Néanmoins ces zones étant très peu denses, la charge de calcul dans ces zones est alors très faible.

Finalement, pour traiter un large panel de simulations dont la densité d'atome est non uniformément répartie, la grille AMR offre davantage de garanties grâce à sa flexibilité. D'ores et déjà, nous pouvons intuiter de nombreux avantages à l'utilisation d'une grille AMR :

- la grille est raffinée dynamiquement durant la simulation lui permettant de s'adapter aux déplacements des atomes;

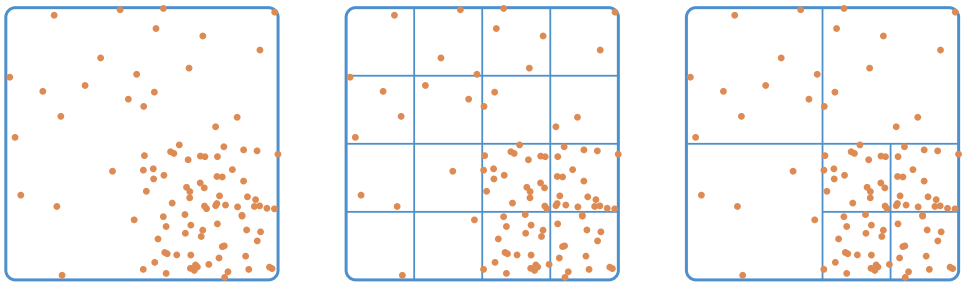


Figure 6.4 | Le déséquilibre entre les cellules (1 atome = 1 poids) avec une grille structurée est de 3.2 alors qu'il diminue à 1.4 pour une grille AMR.

- limitation de l'impact des cellules ne contenant pas d'atome lors de l'équilibrage de la charge entre les sous-domaines et de la distribution des tâches aux threads par un ordonnanceur;
- réduction de l'empreinte mémoire des simulations hétérogènes, comme par exemple les simulations d'impact de goutte qui nécessitaient un coût de stockage supérieur à la mémoire disponible des NŒUDS;
- intégration assez naturelle au sein de l'architecture logiciel d'*ExaSTAMP legacy*.

C'est pour toutes ces raisons que la grille AMR a été choisie comme solution au problème des cellules ne contenant pas ou peu d'atomes.

Impact d'une structure basée sur le Raffinement de Maillage Adaptatif (AMR)

L'utilisation d'une grille basée sur la méthode AMR en DM répond théoriquement aux besoins des simulations dont la densité d'atomes est fortement non uniforme. Afin de vérifier ses avantages, la première étape est de l'incorporer dans un logiciel existant : *ExaSTAMP legacy*. Pour cela, plusieurs questions sont posées : comment intégrer l'AMR dans l'architecture d'*ExaSTAMP legacy*? Comment concevoir la structure d'un octree (cellules feuilles, stockage des atomes) ? La méthode AMR apporte-t-elle une solution concrète au scénario énoncé dans la partie 6.1.1 ?

Conception et Développement

Le développement d'une grille AMR dans l'architecture d'*ExaSTAMP legacy*, illustrée dans la partie gauche de la figure 6.5, doit permettre de conserver les trois formes de parallélisme. Pour cela la grille AMR doit conserver certaines propriétés (contraintes) d'*ExaSTAMP legacy* comme le partitionnement du volume total de la simulation en sous-domaines de **tailles et de formes quelconques** (grille any) pour la répartition de charge. La grille associée à chaque sous-domaine est composée d'un ensemble de cellules vérifiant les **conditions de la méthode des cellules liées**. Les informations des cellules sont stockées dans des structures facilitant l'utilisation de la vectorisation. Pour cela deux approches sont possibles : un super-octree ou une forêt d'octrees.

Nous allons donc détailler ces deux approches afin de pouvoir sélectionner la plus adaptée aux besoins des simulations considérées dans le cadre de cette thèse.

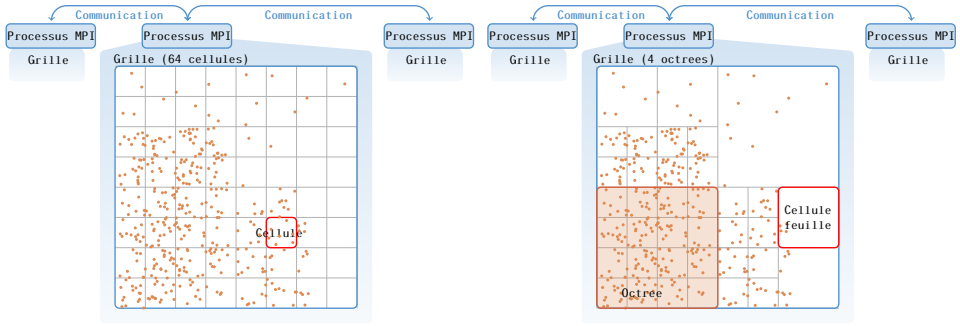


Figure 6.5 | Architecture d'*ExaSTAMP legacy* avec (droite) et sans (gauche) grille AMR.

Un super-octree

Une première approche d'intégration de la grille AMR dans l'architecture d'*ExaSTAMP legacy* est de décomposer la totalité du domaine de la simulation en un unique octree. Cet octree possède des dizaines de niveaux de raffinement. Toutes les cellules raffinées ont au moins un volume supérieur aux cellules générées par la méthode des cellules liées. Celles-ci sont nommées cellules de Verlet. Ainsi la répartition de la charge de calcul entre les sous-domaines par des méthodes de partitionnement prend en compte les poids associés aux cellules feuilles (feuilles de l'octree). Les sous-domaines correspondent à une sous-partie du super-octree. Cette sous-partie du super-octree peut alors être un octree déséquilibré, c'est-à-dire que l'arborescence de l'octree n'assure pas qu'un nœud de l'arbre soit le parent de 8 nœuds.

Lorsque l'algorithme de raffinement et d'agglomération est appliqué sur une grille AMR, la grille n'est pas nécessairement propriétaire des 8 cellules feuilles concaténées par l'algorithme d'agglomération car l'octree est déséquilibré. Ces cellules feuilles peuvent être traitées comme des cas particuliers (cellules sur les bords du domaine) limitant l'algorithme dans certaines zones du domaine. Une autre possibilité est d'équilibrer les octrees des sous-domaines en rapatriant les cellules feuilles nécessaires par de coûteuses communications MPI, dégradant la qualité de la répartition de la charge entre les sous-domaines. Afin d'éviter cette dégradation, la répartition peut être exécutée à chaque fois que la grille AMR doit être raffinée. Néanmoins, déclencher la répartition de la charge à chaque remaillage de la grille n'est pas envisageable car la fréquence de remaillage (peu coûteuse) doit être supérieure à la fréquence de répartition de la charge (très coûteuse).

À noter que certaines méthodes de partitionnement tiennent compte de l'arborescence en octree afin de conserver l'équilibre des octrees pour les niveaux de raffinement les plus élevés. Cependant cela réduit alors le nombre de méthodes de partitionnement utilisables, voir paragraphe 5.3.

Cette approche n'a finalement pas été exploitée à cause de la réduction du nombre de méthodes de partitionnement applicables et parce qu'elle paraissait moins prometteuse l'approche d'une forêt d'octrees.

Une forêt d'octrees

Afin de pallier le défaut du partitionnement des sous-domaines en sous-octrees déséquilibrés, nous avons opté pour une décomposition du domaine en une forêt d'octree. Cette stratégie, illustré par la partie droite de la figure 6.5 consiste à créer une grille cartésienne grossière de cellules avec un volume supérieur au volume des cellules de Verlet. Ces cellules grossières sont les cellules racines des octrees. Au lieu de répartir la charge en distribuant les cellules feuilles, les algorithmes de partitionnement tiennent compte des cellules racines des octrees.

Les cellules racines sont raffinées tant que le critère de raffinement est vérifié et tant que les cellules raffinées sont plus grandes que les cellules de Verlet. Les cellules qui ne sont pas des cellules feuilles sont nommées cellules intermédiaires. Le volume des cellules racines est défini à l'initialisation par le volume de la cellule de Verlet et le nombre de raffinements requis pour l'atteindre. Ce nombre correspond à la profondeur maximale commune à tous les octrees et sera notée D^{max}. Le paramètre D^{max} définit la taille des cellules racines, et donc par conséquent le nombre total d'octrees de la simulation et sa capacité à agglomérer les cellules ne contenant pas d'atome.

Le principal intérêt de cette stratégie est de créer implicitement un super-octree tout en assurant que les derniers niveaux de l'octree soient des octrees équilibrés. Les méthodes de partitionnement intégrées dans *ExaSTAMP legacy* via la libraire Zoltan (Sandia) sont utilisables sans condition supplémentaire et indépendamment de la fréquence de raffinement de la grille. Les poids associés à chaque octree dépendent des cellules feuilles, permettant ainsi une estimation plus précise de la charge de calcul que les poids utilisés précédemment par *ExaSTAMP legacy* lorsque le volume des cellules était artificiellement agrandi. De plus la décomposition de la grille Amr en octrees est structurée, la connectivité entre deux cellules feuilles de deux octrees adjacents n'a alors pas besoin d'être stockée lors de l'étape de raffinement des octrees.

En conclusion, l'approche basée sur la forêt d'octrees sera utilisée pour l'intégration de la grille Amr car elle répond aux contraintes d'ExaSTAMP tout en garantissant les bénéfices attendus par une grille Amr.

Élaboration de la structure d'un octree

Dans le but de vérifier les bénéfices de l'Amr pour les simulations dont la densité d'atomes est non uniforme, la grille Amr est développée dans le code ExaSTAMP dont l'architecture à été présentée dans le paragraphe 5.4.1.

Tout d'abord, l'utilisation d'une grille Amr basée sur la stratégie d'une forêt d'octrees permet de continuer à utiliser les spécialisations des grilles pré-existantes (rectilinéaire et any) en remplaçant le stockage en cellules par un stockage en octrees et de rajouter une structure *traversal* (liste d'index) pour les cellules feuilles. Ainsi le rôle de la grille consistant à appliquer des opérateurs sur des ensembles de cellules est conservé. Néanmoins la structure d'un octree diffère d'une cellule ou d'un bloc de cellules. Pour les grilles sans Amr, chaque cellule a une structure de tableaux pour stocker les informations des atomes alors que dans le cas d'un octree ce stockage peut être commun à toute les cellules d'un octree (nommée **stratégie de tri**) ou réparti entre les cellules feuilles (nommée **stratégie par copie**). La stratégie de stockage nécessite une conception différente de l'octree.

Description des stratégies

La stratégie par copie consiste à indifférencier les cellules au sein d'un octree en leur affectant à toutes les mêmes champs dont le stockage des atomes. L'arborescence de l'octree est construite par un double chaînage des cellules, voir figure 6.6. Une cellule a un champ contenant le niveau de raffinement, son numéro dans le nœud (index local de Morton), un champ pour l'index (ou le pointeur) déterminant sa cellule mère et 8 champs déterminant les cellules filles. La grille a accès aux cellules racines dont le champ "cellule mère" est NULL. Une cellule feuille a par définition tous ses champs "cellules filles" à NULL. Lorsqu'une cellule est raffinée, obtenue par l'algorithme 1, les informations des atomes sont copiées dans les cellules filles selon leurs positions cartésiennes. Bien que la copie soit coûteuse, distribuer les informations directement sur les cellules feuilles permet

d'appliquer des opérateurs tels que la réinitialisation d'un champs sans impacter les autres cellules feuilles.

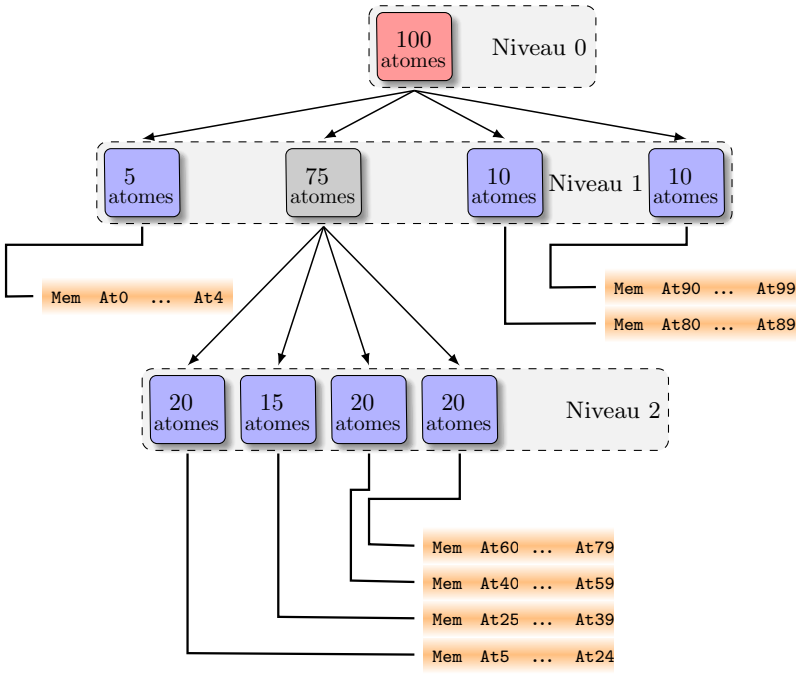


Figure 6.6 | Structure d'un octree avec une stratégie par copie

Algorithme 1 : Algorithme de raffinement par copie. creation() : crée une cellule en fonction des attributs de la cellule mère. calculIndexNouvelleCellule() : détermine l'index de la cellule fille accueillant l'atome

```
Données : Grille : grille d'octrees non raffinés, C() : critère de raffinement
Résultat : Grille d'octrees raffinés
Function Raffinement(Cellule, Critère, D^max)
    si Critère(Cellule) ≡ Vrai et Cellule.niveau < D^max alors
        pour mortonlocal ← 0 à 7 Par pas de 1 faire
            Cellule.cellulefille[mortonlocal].creation(mortonlocal)
        pour i ← 0 à Cellule.nombreAtomes Par pas de 1 faire
            index ← calculIndexNouvelleCellule(Cellule.atome[i])
            Cellule.cellulefille[index].ajout(Cellule.atome[i])
        // destruction de Cellule.atome
        pour mortonlocal ← 0 à 7 Par pas de 1 faire
            Raffinement(Cellule.cellulefille[mortonlocal], Critère, D^max)
pour tous Octree in Grille faire
    Raffinement(Octree, C, Grille.D^max)
```

La stratégie par tri consiste à différencier le stockage des atomes de la structure cellule. Pour cela la structure octree correspond à la cellule racine (niveau de raffinement 0) et contient le stockage des atomes. Les atomes sont triés en fonction de leur index de Morton. L'index de Morton d'un atome noté M_a est obtenu à partir de la position de l'atome notée P, la position de l'octree sur la

grille AMR notée O ainsi que de la largeur notée L_c^R, la longueur notée l_c^R et la hauteur notée H_c^R de la cellule raffinée R fois. La fonction d'encodage en index de Morton prend en entrée 3 entiers :

$$M_a = encode\_morton(m_x, m_y, m_z)$$

avec les entiers m_x m_y et m_z obtenus par :

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \left(\begin{bmatrix} L_c^0 \\ l_c^0 \\ H_c^0 \end{bmatrix} \times O - P \right) \times \begin{bmatrix} 1/L_c^{D^{max}} \\ 1/l_c^{D^{max}} \\ 1/H_c^{D^{max}} \end{bmatrix}$$

Comme détaillé dans le paragraphe 5.1.5, les cellules (feuilles, racines ou intermédiaires) sont associées à un index Morton correspondant à leurs chemins dans l'arborescence d'un octree. Chaque cellule pointe sur une partie de la structure de tableaux d'atomes stockée dans la cellule racine et connaît le nombre d'atomes qui lui est attribué. Pour plus de simplicité cette stratégie est illustrée par la figure 6.7 avec un tableau d'atomes au lieu d'une structure de tableaux. L'algorithme de raffinement utilisé est décrit par l'algorithme 2.

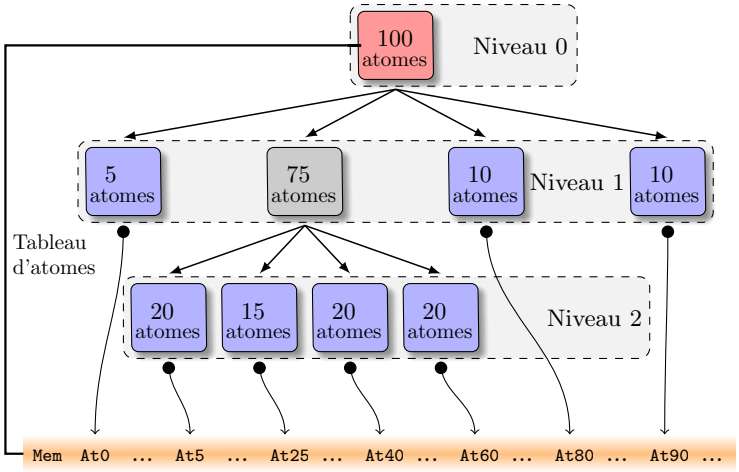


Figure 6.7 | Structure d'un octree avec une stratégie incorporant un tri

Algorithme 2 : Algorithme de raffinement par tri. encode_morton() calcule l'index de Morton en fonction du niveau de raffinement. indexAtomeIndexMorton(T,I) détermine la position du premier élément du tableau T pour un index I de Morton. TriAtome() calcule l'index de Morton pour chaque atome puis trie la structure de données. L'opérateur "A«B" correspond à la multiplication A par 2^B.

```
Données : Grille : grille d'octrees non raffinés, C : critère de raffinement
Résultat : Grille d'octrees raffinés
Fonction Raffinement(Cellule, Tab, Critère, D^max)
    si Critère(Cellule) ≡ Vrai et Cellule.niveau < D^max alors
        NR ← Cellule.niveau
        indexMorton ← encode_morton(Cellule.position,NR)
        deplacementBits ← 3*(D^max - NR+1)
        pour mortonlocal ← 0 à 7 Par pas de 1 faire
            Cellule.cellulefille[mortonlocal].creation(mortonlocal)
            indexMortonFille ← indexMorton + (mortonlocal « deplacementBits)
            Cellule.pointeurAtome ← indexAtomeIndexMorton(Tab,indexMortonFille)
            indexMortonFilleSuivant ← indexMortonFille + 1 « deplacementBits
            Cellule.nombreAtomes = indexMortonFilleSuivant - Cellule.pointeurAtome
        pour mortonlocal ← 0 à 7 Par pas de 1 faire
            Raffinement(Cellule.cellulefille[mortonlocal], Tab, Critère, D^max)
D^max ← Grille.D^max
pour tous Octree in Grille faire
    TriAtome(Octree)
    Tab ← Octree.atome.morton
    Raffinement(Octree.racine, Tab, C, D^max)
```

Les inconvénients de cette stratégie sont le surcoût de stockage inhérent à l'index de Morton utilisé lors du tri et le coût du tri en lui-même. Il existe plusieurs algorithmes de tri dont les plus avantageux ont une complexité en $N\log(N)$. Néanmoins, cette approche comporte de nombreux avantages par rapport à la stratégie par copie :

- une meilleure gestion de la mémoire car les allocations intermédiaires sont supprimées et le nombre de tableaux alloués est plus faible;
- les données sont contiguës en mémoire dans de plus grands tableaux limitant les surcoûts d'utilisation des instructions Simd (moins de créations et destructions de vecteurs);
- lorsqu'un atome change de cellule feuille tout en conservant le même octree, celui-ci n'est pas transféré dans une autre structure de stockage;
- lors de l'utilisation de la répartition de la charge entre les processus Mpi, les octrees sont répartis après avoir préalablement été alignés en mémoire dans des tampons (mémoire) d'envoi. Il est plus intéressant de copier peu de grands tableaux qu'un grand nombre de petits tableaux.

Bien que cette stratégie par tri soit théoriquement plus intéressante que la méthode par copie, le coût du tri peut être prédominant par rapport aux avantages. Ainsi, dans le but de choisir entre ces stratégies, elles sont évaluées dans un prototype de test.

Prototype de test

Pour valider le choix de conception d'un octree, un prototype de test a été développé en C++. Celui-ci reprend uniquement les portions d'un code de dynamique moléculaire impactées par les deux stratégies.

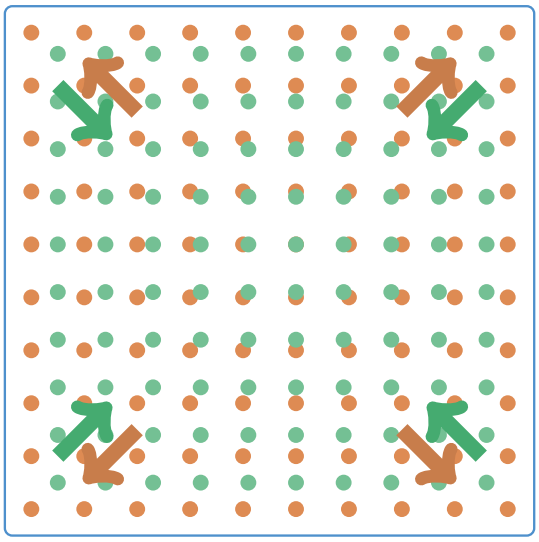


Figure 6.8 | Les atomes (rouges) sont répartis uniformément sur le domaine. A chaque itération, les atomes sont envoyés vers le centre (atomes verts) puis ils sont renvoyés à leurs positions initiales (atomes marrons).

Un cas test comprenant 250 000 atomes sur une grille $2D$ est créé. Les atomes sont initialement répartis uniformément sur un domaine composé de 100 octrees. La profondeur maximale des arbres est fixé à 3. Chaque cellule feuille contient environ 40 atomes. À chaque itération les atomes sont attirés vers le centre puis ils sont expulsés afin qu'ils récupèrent leurs positions initiales, voir figure 6.8. L'opération est répétée N fois. La valeur de ce déplacement permet de gérer le taux d'atomes transférés entre les octrees, pour la stratégie avec tri, et entre les cellules feuilles, pour la stratégie avec copie. Le but est d'estimer laquelle des deux stratégies est la moins coûteuse en temps pour ce cas assez simple. Le prototype mesure uniquement les parties impactées par la structure :

- la gestion des déplacements des atomes entre les octrees ou les cellules feuilles;
- le tri des tableaux selon l'index de Morton des atomes - *stratégie par tri uniquement*;
- le calcul des nouvelles positions des atomes (vectorisation), représentatif de l'application des opérateurs du schéma d'intégration;
- l'ajustement des pointeurs des cellules feuilles - *stratégie par tri uniquement*;
- le raffinement. Celui-ci incorpore la copie des tableaux dans le cas de la stratégie par copie alors que les tableaux sont préalablement triés pour la stratégie par tri afin de dissocier le coût du tri de l'ajustement des pointeurs des cellules feuilles.

Ce prototype et ce cas test vont nous permettre de choisir entre la stratégie par copie et la stratégie par tri.

Choix du tri

Avant de comparer les deux stratégies, il est important de se focaliser sur le choix du tri. Pour cela il faut chercher quel tri est optimal car le tri peut, en fonction du nombre d'atomes à trier et des permutations entre les éléments du tableau, devenir prépondérant. Tout d'abord en C++, la librairie standard propose un tri par défaut permettant de trier un tableau de structures selon un champ de la structure ou une fonction dépendant de ses champs. Néanmoins cette librairie ne permet pas explicitement de trier une structure de tableaux en fonction d'un tableau. Pour cela, une fonction de tri appliquant les permutations successives sur un ensemble de champs, via l'utilisation de *template variadic*, est utilisée. Quatre tris sont alors implémentés :

- Tri par Insertion : Complexité en N*N, néanmoins lorsque le tableau a subi très peu de permutation ou d'ajout, le tri par insertion a une complexité proche de N;
- Tri Rapide : Complexité en N*log(N);
- Tri par Tas : Complexité est en N*log(N);
- Tri par Introspection : Tri par défaut dans la librairie standard. Sa complexité est en N*log(N). Il mélange les trois précédents tris.

La stratégie par tri est testée avec les 4 tris dans le prototype de test décrit dans le paragraphe précédent selon diverses valeurs de déplacement des atomes. Les distances parcourues par les atomes font varier le taux d'atomes échangés entre les octrees/feuilles et désorganisant les structures de tableaux. La figure 6.9 expose les durées de simulation selon le tri utilisé en fonction du pourcentage d'atomes échangés. Lorsque les positions des atomes sont peu perturbées, l'algorithme de tri le plus intéressant est le tri par insertion car celui-ci ne parcourt qu'une seule fois chaque index de Morton et n'applique aucune permutation. Dès lors que l'ordre des éléments des tableaux est perturbé par l'ajout, la suppression ou le changement d'index de Morton, on observe que le tri par introspection est le plus rapide. À noter qu'en pratique, dans le cas de simulation dynamique, le nombre d'atomes échangé ne dépasse pas les 0.1%. Par la suite le tri par introspection sera le tri utilisé pour la stratégie par tri.

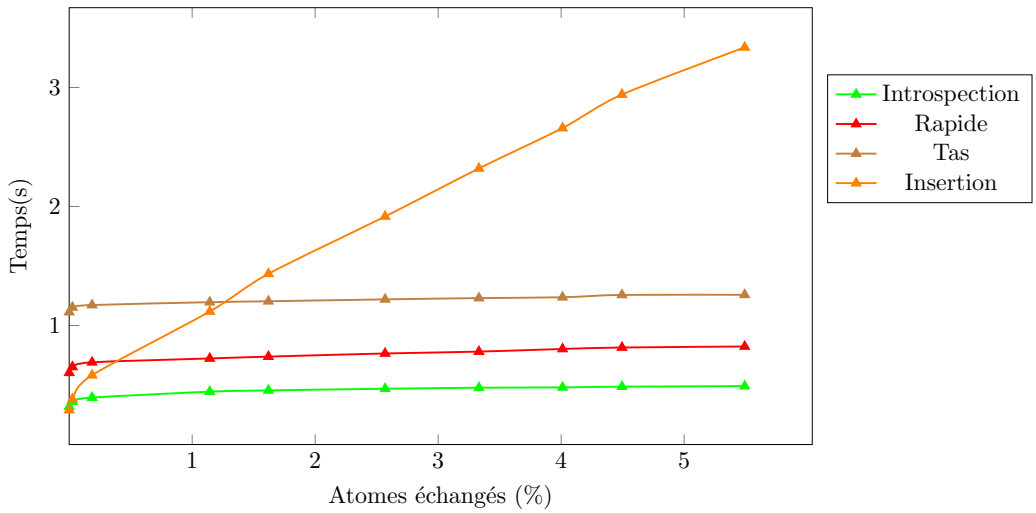
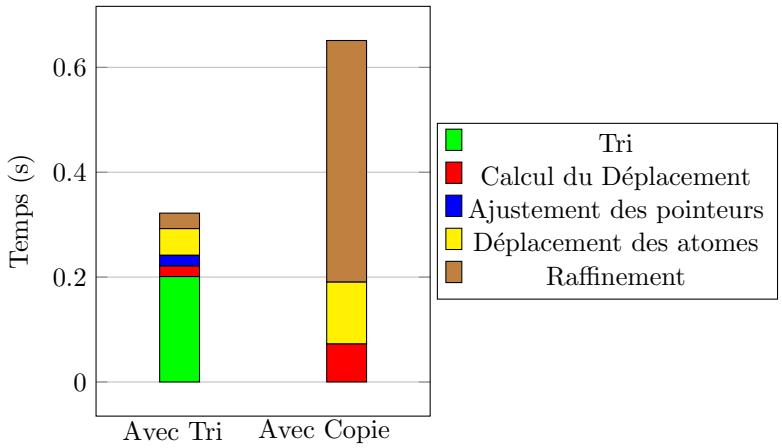


Figure 6.9 | Temps d'exécution en fonction du nombre d'atomes échangés entre les octrees selon le type de tri utilisé : introspection, rapide, tas et insertion.

Comparaison des stratégies

Maintenant que le choix du tri par introspection a été effectué pour la stratégie par tri, nous allons comparer les deux structures selon le scénario décrit pour le prototype de test.

L'histogramme 6.10 présente les résultats obtenus par le prototype de test sur un processeur Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz comportant les instructions vectorielles avx2. Le déplacement vers le centre puis vers l'extérieur du domaine est effectué 30 fois. Le raffinement est appliqué avant que les atomes ne soient expulsés vers l'extérieur. Les pointeurs sont ajustés quand les atomes sont attirés vers le centre du domaine ou transférés d'une cellule à une autre pour la stratégie avec copie. À noter que le temps pour rapatrier les atomes sur les nœuds racines des octrees n'est pas pris en compte.



Histogramme 6.10 | Temps en secondes en fonction de la stratégie de conception des octrees. La stratégie par copie n'intègre pas les étapes de tri et d'ajustement des pointeurs.

Le taux d'atomes échangés choisi entre les octrees pour la stratégie par tri est de 0.026% et de 0.82% entre les cellules feuilles pour la stratégie par copie. Ces taux sont représentatifs des simulations reproduisant l'évolution d'un micro-jet ou l'impact d'une nano-goutte sur une surface solide. De plus dans la section précédente, nous avons observé qu'en faisant varier le taux d'atomes échangés entre les octrees, le temps d'exécution de la stratégie par tri augmente très peu pour des cas inenvisageables (+23% pour 4.49% d'atomes échangés entre les octrees). Finalement, la stratégie par tri est deux fois plus rapide pour traiter ce problème.

La principale raison est le faible coût d'adaptation des pointeurs sur un tableau trié par rapport au temps de tri. Le temps de tri étant moitié moins coûteux que le raffinement de la stratégie de copie alors que celle-ci ne copie que 4 champs (positions, index) au lieu des 10 d'*ExaSTAMP legacy*. À noter que pour une répartition uniforme, l'empreinte mémoire est 4 fois plus faible dans le cas de la stratégie par tri car les données ne sont pas répliquées autant de fois qu'un octree est raffiné.

Lors de la compilation, le compilateur icc-2017 a été utilisée avec l'option de vectorisation *-mavx*. Les instructions Simd permettent théoriquement de traiter 4 doubles à la fois. En pratique, on constate que la vectorisation permet de traiter 3.6 fois plus rapidement la partie consistant à calculer le déplacement des atomes car elle est appliquée sur des tableaux composés de nombreux éléments.

De plus le déplacement des atomes d'un octree ou d'une cellule feuille à un ou une autre est 2.3 fois plus rapide. Ce gain s'explique par le nombre d'atomes échangés qui est plus faible.

Suite aux tests réalisés avec le prototype de test, la structure choisie pour la conception d'un octree est la structure par tri.

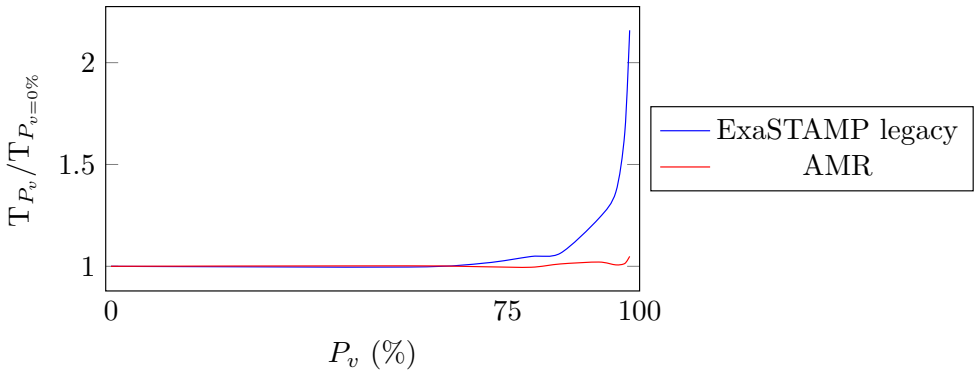
En conclusion, la grille Amr intégrée dans ExaSTAMP sera basée sur une forêt d'octrees. Les atomes associés à un octree sont stockés dans une structure de tableaux de la cellules racines favorisant la vectorisation. Les tableaux sont triés en fonction de l'index de Morton de chaque atome et les cellules feuilles pointent sur les éléments du stockage associés à leur index de Morton.

Étude de l'impact de la grille Amr sur les cellules ne contenant pas d'atome

Maintenant que la structure d'un octree est fixée, la grille Amr a été implémentée dans le code ExaSTAMP afin de vérifier si la méthode Amr réduit les surcoûts de calcul induits par les cellules ne contenant pas d'atome.

Dans le paragraphe paragraphe 6.1.1, les limites de la méthode des cellules liées ont été mises en évidence dans le code *ExaSTAMP legacy*. Pour cela un cristal parfait cubique (appelé bloc) est entouré par un nombre variable de cellules ne contenant pas d'atomes. Il est alors très simple de faire varier le nombre de cellules en dehors du bloc afin d'ajuster le pourcentage de cellules ne contenant pas d'atome pour mesurer l'impact de celles-ci.

Pour ce même cas test, le graphe 6.11 montre le temps d'exécution de la simulation en fonction du pourcentage de cellules sans aucun atome (P_v) normalisé par la durée de la simulation sans cellule supplémentaire. La courbe bleu est obtenue avec *ExaSTAMP legacy*. Elle permet d'avoir une référence par rapport au programme disponible au début de ma thèse. La courbe rouge correspond aux temps obtenus avec la grille Amr avec $D^{max} = 1$. Comme attendu, on observe que le coût engendré par les cellules supplémentaires sur le temps d'exécution de la simulation est atténué par l'utilisation des octrees.

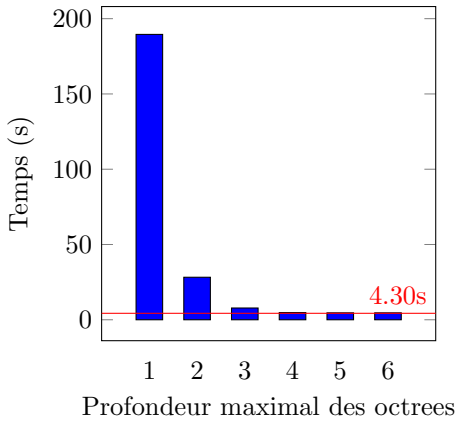


Graphe 6.11 | Impact des cellules ne contenant pas d'atome sur les performances d'*ExaSTAMP legacy* avec une grille rectilinéaire et ExaSTAMP avec une grille Amr. $T_{P_{v=0\%}}$ correspond au temps sans cellules ne contenant pas d'atome autour du bloc de cuivre.

Afin d'envisager des simulations comme l'impact d'une nano-goutte d'étain sur une surface solide, on effectue un cas extrême de cette simulation pour $P_v = 99.9981\%$. Grâce à ce cas test, nous démontrons que la solution basée sur la méthode Amr permet de réaliser des simulations de Dm sans que le maillage n'impacte les performances. Un bloc d'atome de 5.31nm^3 est généré alors que le domaine de simulation a un volume de 200nm^3, ce qui équivaut bien au cas extrême souhaité. La simulation est exécutée uniquement avec la grille Amr car la grille sans Amr est trop coûteuse en mémoire pour être utilisée par *ExaSTAMP legacy*.

Les temps d'exécution pour différentes valeurs de D^{max} sont présentés dans l'histogramme 6.12. À noter que le temps d'exécution de référence pour cette simulation (uniquement le bloc d'atome) est de 4.30 secondes (courbe rouge). On observe qu'à partir d'une certaine taille des octrees, pour $D^{max} = 4$ en l'occurrence, il est possible de rendre négligeable l'effet des espaces ne contenant pas

d'atome du domaine, du moins uniquement avec la vectorisation activée car le but de cette étude est d'évaluer les surcoûts de calcul des cellules ne contenant pas d'atome. Concernant la parallélisation en mémoire partagée, nous étudions l'impact des cellules ne contenant pas d'atomes sur le temps d'exécution dans le chapitre 7.



Histogramme 6.12 | Scénario extrême, le domaine est composé à 99.9981 % de cellules ne contenant pas d'atome. Ce scénario est testé avec une forêt d'octrees pour différents niveaux de profondeur.

Ces résultats préliminaires mettent en évidence que la méthode de Raffinement de Maillage Adaptatif est une solution très prometteuse pour gérer les cellules vides ou peu remplies. Dans la suite de ce chapitre nous allons décrire l'algorithme effectuant la recherche de cellules voisines et étudier les différentes optimisations possibles grâce à la grille AMR et ses bénéfices sur les temps de simulation.

Les listes de voisins

Jusqu'à présent la méthode des listes de Verlet n'a pas encore été abordée. *ExaSTAMP legacy* applique la combinaison entre la méthode des cellules liées et la liste de Verlet. Dans *ExaSTAMP legacy*, la construction des listes des atomes voisins consiste à stocker pour chaque atome un ensemble d'atomes voisins définis par les index des cellules contenant les atomes, les positions dans la structure de tableaux et les types des atomes (étain, cuivre, ...) : STD::VECTOR<STD::TUPLE<INT,INT,INT». Les atomes ne sont donc pas échangés entre les cellules tant que les listes de Verlet ne sont pas mises à jour car l'ordre des atomes dans les cellules ne doit pas être modifié.

Concernant la grille AMR, les listes de voisins sont stockés dans la structure de la cellule racine de l'octree et chaque élément de la liste de voisins est défini par un entier pour le type de l'atome, un entier pour l'octree contenant l'atome et un entier pour la position de l'atome dans l'octree. À noter que les listes de voisins sont stockées dans les cellules racines pour que les cellules feuilles mais aussi que les cellules intermédiaires puissent y accéder.

L'algorithme de recherche des atomes voisins est basé sur deux opérateurs : A et B. L'opérateur A consiste à calculer la distance entre deux atomes au sein d'une cellule afin de sélectionner les paires à stocker dans les listes de Verlet. Les comparaisons sont traitées avec des instructions SIMD. L'opérateur B effectue les comparaisons entre les atomes de deux cellules différentes. Afin de limiter le nombre de tests, un calcul préliminaire pour chaque atome détermine si l'atome peut interagir avec l'un des atomes d'une autre cellule. L'opérateur B inclut donc une opération de masquage lors de l'utilisation des instructions SIMD effectuant les tests de distance.

Dans le contexte d'une grille Amr, les opérateurs A et B sont utilisés pour trouver respectivement les paires d'atomes de voisins au sein des cellules feuilles et les paires d'atomes voisins entre deux cellules feuilles adjacentes. La figure 6.13 illustre graphiquement l'algorithme utilisé afin de créer dynamiquement la connectivité de chaque cellule feuille à chaque raffinement des cellules. Puisque la stratégie employée lors de l'implémentation de la grille Amr s'appuie sur une forêt d'octrees, l'algorithme tient compte de deux cas de figures : si la cellule est une cellule racine (couleur orange) ou si c'est une cellule raffinée (couleur violet).

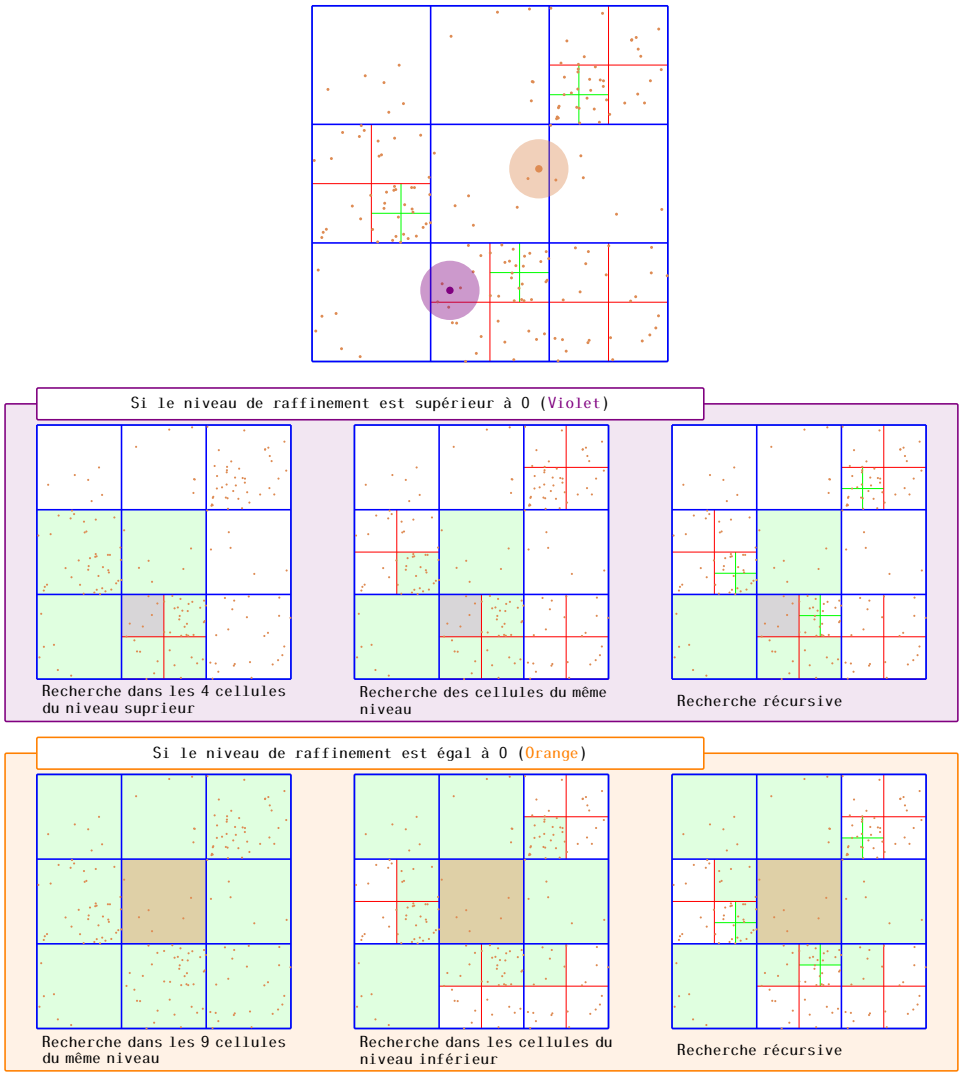


figure 6.13 | Construction des connectivités des cellules feuilles contenant la particule violette et la cellule feuille contenant la particule orange. Les zones coloriées en vert correspondent aux cellules éligibles à être dans la liste des cellules voisines. Les cellules bleues sont des cellules de niveaux o ou octree, les rouges pour le niveau 1 et les vertes pour le niveau 2.

Ainsi dans le cas de l'atome orange d'une cellule racine, les cellules feuilles voisines sont incluses dans les 26 octrees (8 en $2D$) adjacents. Ensuite l'algorithme va parcourir les cellules raffinées en ne conservant que les cellules feuilles adjacentes à la cellule racine. Lorsque l'atome considéré est dans une cellule raffinée comme c'est le cas pour l'atome violet, l'ensemble des cellules feuilles voisines est inclus dans les 8 (4 en $2D$) cellules intermédiaires du niveau de raffinement inférieur. Ensuite on parcourt toutes les cellules raffinée provenant de ces 8 cellules intermédiaires.

Finalement, étant donnée que la reconstruction des listes de Verlet est basée sur le déplacement

des atomes les plus rapides, celui-ci est un indicateur pertinent concernant le déclenchement de l'algorithme de raffinement des octrees. Lorsque la simulation est homogène et peu dynamique (cristal parfait), les atomes vibrent autour de leurs positions initiales et les listes de Verlet ne sont jamais mises à jour alors que dans le cas d'une simulation extrêmement dynamique les listes de Verlet peuvent être mises à jour tous les 6 à 10 pas de temps lorsque le rayon de Verlet est correctement paramétré. Pour l'ensemble des tests effectués dans cette thèse, le rayon de Verlet est fixé à 1Å.

Étude de l'optimisation de l'utilisation des caches mémoires

Maintenant que nous avons présenté notre approche de la conception de la méthode AMR pour des codes de DM, nous allons étudier l'impact de la grille AMR sur l'utilisation des caches mémoires. Compte tenu des avancées des processeurs à architecture UMA et NUMA, la gestion des accès mémoires et la localité des données sont des notions importantes et une source d'optimisation des codes de simulation numérique. En introduisant la grille AMR et sa décomposition en une forêt d'octrees, deux sources d'optimisation des accès mémoire sont alors mises en place.

Rappelons qu'au sein de chaque octree, les atomes sont triés selon leur index de Morton. Les cellules feuilles ont des numéros d'index de Morton basés sur leurs positions cartésiennes et leurs niveaux de raffinement. Les cellules racines ont un niveau de raffinement égal à 0. Les cellules feuilles sont des cellules raffinées au maximum D^{max} fois, D^{max} correspondant à la profondeur maximale commune à tous les octrees de la simulation. Une cellule racine peut être une cellule feuille.

Lorsque les cellules feuilles d'un octree sont parcourues afin de construire les listes de voisins ou d'effectuer le calcul des interactions, les données des atomes sont chargées dans des registres et dans les caches mémoire. Pour ce type d'opération, pour chaque atome un voisinage d'atomes est nécessaire. Le voisinage est défini par la distance RCUT à partir de laquelle les interactions à courte portée sont nulles. La valeur de RCUT dépend de la nature des atomes et du potentiel utilisé. Puisque le voisinage d'atomes est chargé pour chaque atome, les informations de chaque atome sont alors chargées plusieurs fois. Les cellules feuilles sont parcourues dans l'ordre de l'index de Morton (courbe en Z) afin de choisir les accès à la mémoire maximisant la réutilisation des données stockées dans les caches mémoire. Cette courbe de Lebesgue a la propriété de reprendre l'arborescence de l'octree consistant à parcourir successivement les cellules issues d'un même raffinement formant un cube.

Dans le but de conserver au maximum les informations dans les caches mémoire, nous utilisons aussi l'algorithme de blocage de cache. Il consiste à découper une structure de données pour qu'elle tienne dans l'un des niveaux de cache mémoire. Le principe est de gérer les accès mémoire afin de maximiser les chargements d'un sous-ensemble de données d'une structure de donnée plus importante. En réutilisant ces données dans la mémoire cache, cela permet d'éviter le goulot d'étranglement de la bande passante mémoire induite par certaines applications dont l'intensité arithmétique est faible (peu de calculs par accès mémoire). Par exemple lors de l'application du stencil du laplacien $3D$ sur une grille structurée, la technique de blocage de cache consiste à réarranger la boucle de calcul afin qu'elle parcourt des blocs (cube). La grille AMR et son découpage en une forêt d'octrees crée naturellement des blocs dont la taille est ajustable au moyen de D^{max}.

Afin de mesurer l'impact de l'algorithme de blocage de cache, un cristal parfait de 500 000 atomes de cuivre est généré à partir d'une maille cubique à face centrée. Celui-ci est simulé avec un potentiel LJ par EXASTAMP et sa grille AMR pour $D^{max} = 0, 1, 2, 3$. Le Tableau 6.1 nous donne le nombre de défauts de cache mesuré par le logiciel Perf (*perf stat -e cache-misses*) pour chaque valeur de D^{max}. Deux prises sont effectuées afin de dissocier l'initialisation de la boucle de calcul. Le nombre

de défauts de cache diminue entre $D^{max} = 0$ et $D^{max} = 2$ puis augmente car la taille des blocs de cellules feuilles est trop grande. La taille de bloc «idéale» est donc obtenue pour $D^{max} = 2$. Dans ce cas la simulation de 500 000 atomes possède 216 octrees soit $64 * 216 = 13,824$ cellules feuilles. Ainsi chaque cellule feuille comporte environ 36.16 atomes. Chaque atome a un coût mémoire de 93Bytes. De plus pour les opérateurs de recherche de voisins et du calcul des interactions, chaque octree a besoin de la première couche de cellules voisines, donc $(4+2)*6*6 = 216$ cellules feuilles. Finalement, le bloc mémoire chargé est de $216 * 36,14 * 93 = 725$KB. Celui-ci tient alors dans le cache L2. Lorsque D^{max} passe à 3, le bloc mémoire chargé est de 3336KB expliquant pourquoi le nombre de de défauts de cache augmente.

	D^{max}=0	D^{max}=1	D^{max}=2	D^{max}=3
1 itération	56 703 238	33 635 667	27 697 747	28 162 813
10 itérations	256 048 230	149 061 210	123 128 919	127 096 476
Différence	199 344 992	115 425 543	95 431 172	98 933 663

Table 6.1 | Nombre de "cache-misses" rapporté par l'outil perf en fonction de la profondeur maximale des octrees. Le test a été réalisé sur un Intel Xeon Skylake avec les caches mémoire suivant : L1i : 32KB, l1d : 32KB, L2 : 1 024KB et L3 : 33 792KB

Étude de l'influence des instructions vectorielles

Dans cette partie nous discutons de la stratégie mise en place afin de privilégier l'utilisation des instructions Simd. L'utilisation d'instructions Simd est une source d'optimisation des codes de simulation car elle permet de traiter davantage d'éléments par opération même si la vectorisation ralentit la fréquence des processeurs. De plus la tendance actuelle est à l'augmentation du nombre d'éléments traités par opération avec par exemple les instructions AVX512. Pour cela, nous nous intéressons exclusivement au calcul de l'énergie potentielle et de son gradient (paragraphe 3.4.3) pour chaque atome. Dans un premier temps nous comparons l'apport d'un tampon (tableau temporaire) de vectorisation versus un traitement séquentiel sans instruction Simd. L'utilisation d'un tampon de vectorisation consiste à aligner les données dans une structure favorisant l'utilisation d'instruction Simd. Dans un deuxième temps nous introduisons la méthode des blocs de Verlet afin de créer des blocs d'atomes contigus favorisant l'utilisation d'instructions Simd au détriment d'un surplus de calcul. Finalement, nous effectuons une comparaison complète de ces stratégies sur les architectures Intel Xeon Sandy Bridge, Haswell, Skylake (SKL) et Intel Xeon Phi Knights Landing (Knl) possédant au moins les jeux d'instructions Simd sse4.1 et avx-256bits pour différentes fréquences de processeur. Les processeurs Knl et Skl possèdent aussi chacun un type d'instruction d'avx-512bits. De plus, deux potentiels sont aussi utilisés : un potentiel de paires (Lennard Jones) et un potentiel Eam: Sutton Chen. Les deux potentiels mettent en valeur deux cas de figures représentatifs de nombreux autres potentiels, le potentiel Lj pour les potentiels très peu coûteux et le potentiel Eam Shutten Chen pour les potentiels coûteux.

Ainsi en variant la charge de calcul et le processeur multicœurs utilisé, les stratégies sont évaluées dans de nombreuses configurations et les observations réalisées couvrent donc un plus grand éventail de possibilités.

Impact de la vectorisation sur le temps d'exécution d'une simulation

Les durées des simulations de DM dépendent principalement du calcul des interactions modélisées par des potentiels. A contrario les opérateurs de construction des listes des atomes voisins ou du schéma numérique sont généralement faibles par rapport au calcul du potentiel. Les potentiels de pairs, comme le Lennard Jones (LJ), ont des propriétés de symétrie, c'est-à-dire que la force exercée par l'atome i sur l'atome j, notée $f_{i\to j}$ et égale à la force opposée de l'atome j sur l'atome i. Le calcul du potentiel entre l'atome i et j est calculé une seule fois et sa contribution est répercutée pour les deux atomes. Cette propriété permet de sauver de précieux cycles de calcul mais aussi de stockage car les listes de voisins sont partiellement construites. Des potentiels plus complexes comme les potentiels Embedded Atom Model (EAM) ne sont pas symétriques au sens propre du terme car $f_{i\to j} \neq -f_{j\to i}$, mais ils incorporent aussi de nombreux calculs intermédiaires effectués en double. Comme pour les potentiels de pairs, ceux-ci ne sont effectués qu'une seule fois et reportés pour les deux atomes en question. Lors de l'implémentation de la grille AMR dans EXASTAMP, uniquement les versions symétriques des potentiels ont été conservées.

Algorithme 3 : Algorithme du calcul du potentiel avec un tampon de vectorisation. appliqueCondition() est une fonction éliminant du tampon toutes interactions au-delà du rayon de coupure.

```
Données : TabCells : tableau des cellules feuilles, OpPotVec() : Opérateur vectoriel du
          potentiel, rCut : rayon de Verlet
Résultat : Calcule l'énergie potentielle
pour tous Cellules dans TabCells faire
    pour tous atomes dans Cellule faire
        ListeVoisins ← Cellules.listeVoisin(atomes)
        nombreDeVoisins ← ListeVoisins.taille
        tampon ← Cellules.alignementDonnées(ListeVoisins)
        appliqueCondition(rCut,tampon,nombreDeVoisins)
        OpPotVec(tampon,nombreDeVoisins)
        Cellules.stockResultats(atomes,tampon,nombreDeVoisins)
```

Lors du calcul des interactions d'un atome, *EXASTAMP legacy* aligne les différences relatives entre l'atome et ses atomes voisins dans une structure de tableaux nommée tampon de vectorisation. Cette stratégie permet d'appliquer les instructions SIMD (intel Intrinsics) en traitant plusieurs interactions à la fois, voir l'algorithme 3. Cependant lors des premières comparaisons en séquentiels entre *EXASTAMP legacy*, MINIMD et LAMMPS sur une simulation homogène et un potentiel LJ, nous nous sommes aperçu que l'opérateur du potentiel LJ est plus lent sur SKL et KNL. Ceci s'explique par le fait que les opérateurs responsables du calcul du potentiel LJ dans les codes LAMMPS et MINIMD n'utilisent pas d'instructions SIMD (vérification avec le compilateur Intel 2017 et l'option vec-report=5). Ils ne répliquent pas les données dans un espace de stockage dédié au calcul vectoriel. Bien que les interactions soient calculées en théorie plus rapidement, le coût supplémentaire de recopie des données est trop important. Les opérateurs ont donc été adaptés, voir l'algorithme 4, pour les potentiels LJ et EAM Sutton Chen pour la grille AMR.

Algorithme 4 : Algorithme du calcul du potentiel en séquentiel.

```
Données : TabCells : tableau des cellules feuilles, OpPotSeq() : Opérateur séquentiel du
          potentiel, rCut : rayon de Verlet
Résultat : Calcule l'énergie potentielle
pour tous Cellules dans TabCells faire
    pour tous atome dans Cellule faire
        ListeVoisins ← Cellules.listeVoisin(atome)
        pour tous voisins dans ListeVoisins faire
            distance ← atomes.position - voisins.position
            si distance < rCut alors
                Resultat ← OpPotSeq(distance)
                Cellules.stockResultats(Resultat)
```

Le tableau 6.2 rapporte les temps de calculs du potentiel LJ et EAM pour des simulations homogènes de cuivre sur différentes architectures de processeurs multicœurs. On observe dans la table 6.2 que les opérateurs de calcul utilisant un tampon de vectorisation sont plus rapides que les opérateurs sans tampon pour des potentiels coûteux alors que la tendance s'inverse pour des potentiels peu coûteux. On observe donc que l'utilisation des instructions SIMD est soumise à la condition que le coût de calcul soit supérieur au coût d'alignement des données.

Architecture - Potentiel	Avec tampon	Sans tampon	*ExaSTAMP legacy*
Skylake - LJ	191.93	94.95	204.12
Knights Landing - LJ	677.33	377.64	721.64
Skylake - EAM	81.83	113.3	94.13
Knights Landing - EAM	274.81	501.86	300.93

Table 6.2 | Comparaison de temps en secondes du calcul de l'énergie potentiel pour un potentiel LJ et un potentiel EAM Shutten Chen selon l'utilisation ou non d'un tampon de vectorisation. Les durées de simulations pour les potentiels LJ et EAM sont respectivement obtenues sur un cristal parfait de cuivre de 500 000 et 108 000 atomes pour 200 et 128 pas de temps.

La méthode des "blocs de Verlet"

Dans le paragraphe précédent nous avons mis en évidence que le surcoût d'alignement des positions relatives dans un tampon afin d'utiliser des instructions SIMD pour calculer les interactions est plus élevé que d'effectuer les opérations séquentiellement. Cependant il n'est pas nécessaire que les données soient alignées en mémoire dans un tampon de vectorisation (tableaux explicitement définis dans le code) pour pouvoir utiliser des instructions SIMD. En effet, les positions relatives des atomes sont chargées dans le tampon principalement afin d'éviter de calculer des interactions nulles, c'est-à-dire au-delà du Rayon de coupure. Notre approche est de profiter de la structure de tableaux utilisée pour stocker les atomes afin de calculer toutes les interactions avec des instructions vectorielles même si elles sont nulles.

Pour cela, une solution est de décomposer les tableaux en blocs de la taille d'un registre SIMD. Au lieu d'établir une liste de voisins pour chaque atome, on construit une liste de blocs d'atomes dont au moins un atome est a une distance inférieur au rayon de Verlet, cette méthode sera nommée

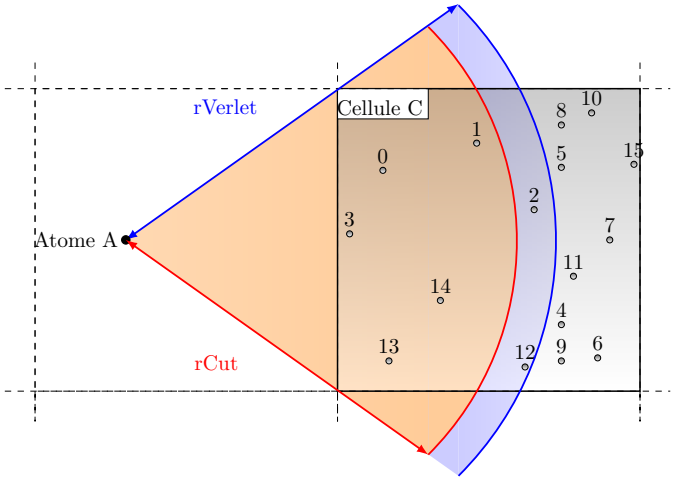


figure 6.14 | Soit un atome A, on cherche à déterminer l'ensemble des voisins de l'atome A inclus dans la cellule C.

Bloc de Verlet. La figure 6.14 illustre l'interaction entre un atome A et les atomes d'une cellule C. La figure 6.15 décrit comment la liste de blocs d'atomes dans le voisinage de l'atome A est construite. On a bien l'assurance que les atomes 0,1,2,3,13,14,12 de la cellule C validant le critère du rayon de Verlet sont dans les blocs 0 et 3.

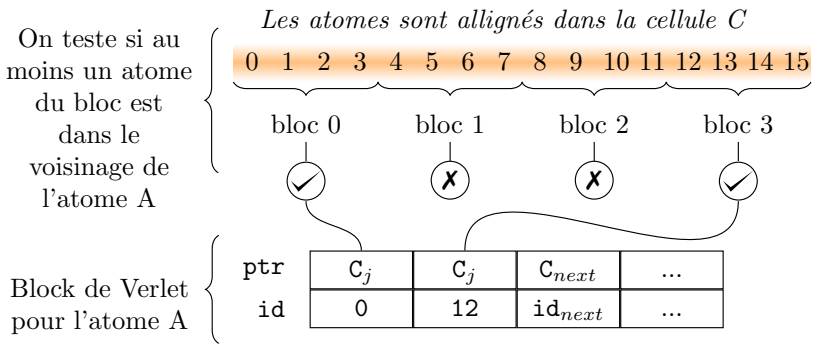


figure 6.15 | Recherche des blocs d'atomes contenant au moins un voisin.

Lorsqu'un bloc d'atomes est chargé dans un registre SIMD, un masque est appliqué afin de retenir uniquement les calculs sur les interactions dans le voisinage de l'atome, voir figure 6.16. Par ce mécanisme, davantage de calculs sont effectués. Cependant cette stratégie est théoriquement analogue à celle traitant les interactions séquentiellement mais en traitant directement les blocs. Le calcul du potentiel est alors effectué comme décrit par l'algorithme 5.

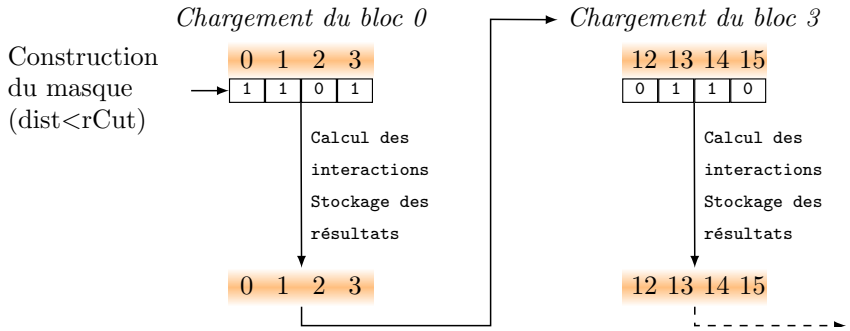


figure 6.16 | Représentation du calcul des interactions avec un potentiel sur des blocs d'atomes. Un masque est utilisé afin de mettre à 0 les atomes ne validant pas le critère du rayon de coupure du potentiel (rCut).

Algorithme 5 : Algorithme du calcul du potentiel avec un tampon de vectorisation. Les opérations distanceRelative() et <rCut sont traitées par des instructions Simd

```
Données : TabCells : tableau des cellules feuilles, OpPotVec() : Opérateur vectoriel du
          potentiel, rCut : rayon de coupure
Résultat : Calcule l'énergie potentielle
pour tous Cellules dans TabCells faire
    pour tous atomes dans Cellules faire
        ListeBlocs ← Cellules.listeVoisin(atomes)
        pour tous blocs dans ListeBlocs faire
            distances ← distanceRelative(atomes,blocs)
            si distance < rCut alors
                Resultats ← OpPotVec(distances,blocs.taille)
                Cellules.stockResultats(atomes,Resultats,blocs.taille)
```

En pratique, puisque les atomes sont triés selon leur index de Morton à l'intérieur des octrees, si un atome A d'un bloc N appartient à la liste de voisins d'un atome B, alors il y a de fortes chances que d'autres atomes du bloc N appartiennent à la liste des voisins de l'atome B. Via cette stratégie on évite la recopie d'information dans un tampon. Dans le pire des cas cette méthode est équivalente à la méthode séquentielle, et dans le meilleur des cas l'opérateur est accéléré par la taille d'un registre vectoriel.

L'une des principales limites de la méthode des blocs de Verlet est qu'elle s'applique aux potentiels nécessitant uniquement les informations incluses dans le voisinage des atomes. Cette méthode s'applique donc pour des potentiels de pairs ou pour les potentiels Eam. Cependant, cette méthode ne s'applique pas au potentiel Meam (détaillé dans le paragraphe 3.4.6) car il nécessite le voisinage des atomes inclus dans son voisinage. De plus le potentiel Meam a un coût arithmétique environ 10 fois plus élevé que les potentiels Eam, c'est pourquoi l'opérateur associé est écrit en utilisant des instructions Simd via un tampon de vectorisation. Une autre limite de la méthode des blocs de Verlet sera abordée lors du chapitre concernant la parallélisation intra-Nœud, chapitre 8.

Évaluation des stratégies

Maintenant que nous avons vu les trois techniques d'accès aux données utilisées par les opérateurs de calcul du potentiel, nous allons les évaluer sur une simulation homogène et statique en séquentiel. Elles sont évaluées sur les architectures de processeur Intel Xeon Sandy, Haswell, Skylake et Intel

Xeon Phi Knights Landing. Pour cela un cristal parfait de 108 000 atomes est généré à partir de la réplication d'une lattice à face cubique centrée de cuivre répliquée 30 fois selon les 3 axes. Les simulations sont réalisées avec le potentiel Lennard Jones, figure 6.17 et avec le potentiel Eam Sutten Chen, figure 6.18, pendant 128 pas de temps ($\Delta_t = 1ps$). Le rayon de Verlet est suffisamment grand (1Å) pour éviter la reconstruction des listes de voisins.

Les trois techniques sont évaluées pour 3 ou 4 types d'instructions Simd utilisés selon le processeur utilisé. Le compilateur utilisé est le Intel icc-17.0.6.256. On observe alors que les courbes obtenus dans la figure 6.17 et dans la figure 6.18 corroborent les observations effectuées pour le Tableau 6.2. Il semblerait que le tampon de vectorisation permet d'optimiser le temps de calcul lorsque le gain apporté par la vectorisation surpasse le coût de copie pour aligner les données en mémoire. De plus dans le cas du potentiel Eam, on remarque que le gain apporté par la vectorisation SSE4.1 ne supplante pas le surcoût d'alignement des données dans le tampon de vectorisation.

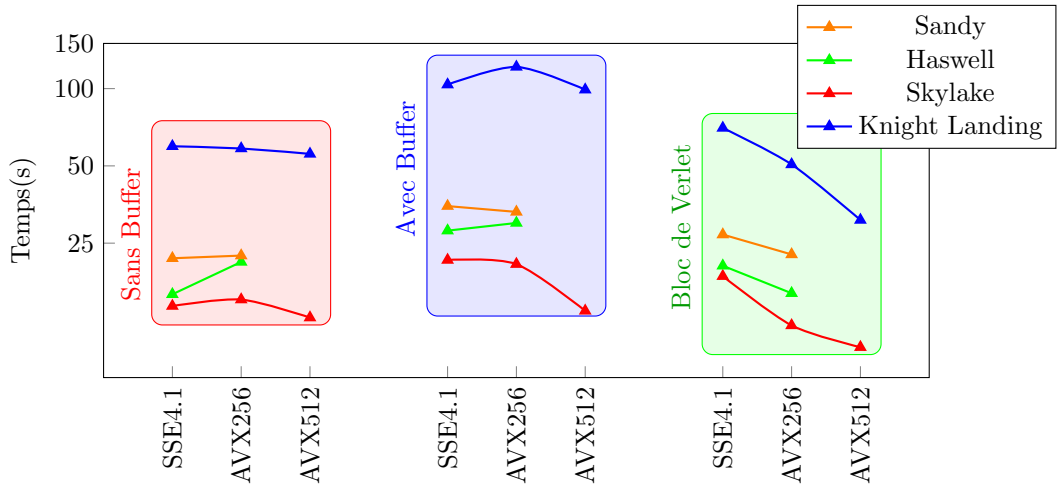


figure 6.17 | Temps effectif du calcul du potentiel Lennard Jones (échelle log10 en temps) selon le type d'architecture et le type vectorisation utilisé.

Concernant la méthode des blocs de Verlet, elle apporte dans les deux cas une solution prometteuse. Pour le potentiel Lennard Jones, en effectuant davantage de calcul tout en supprimant le coût de copie, la méthode des blocs de Verlet est particulièrement intéressante sur les machines possédant des instructions Simd traitant des blocs d'atomes plus larges. Les gains obtenus sur Skl et Knl avec de l'avx512 sont respectivement de 23.3% et de 44.7% par rapport à la stratégie séquentielle. En pratique, on observe que les blocs sont généralement remplis à plus de 50% d'atomes respectant le critère du rayon de coupure. Le surcoût d'utilisation des registres Simd et d'instructions de masquage est donc rentabilisé par le parallélisme de données.

On remarque que dans le cas du potentiel Eam le gain entre la méthode des blocs de Verlet et la stratégie avec un tampon de vecorisation est plus faible que dans le cas du potentiel Lj, d'environ 24.1% au lieu de 27.8% sur Skl et d'environ 44.4% au lieu de 68.9% sur Knl, alors que la méthode sans tampon est encore moins avantageuse. Cette différence s'explique d'une part car la méthode avec tampon traite uniquement les atomes dans les voisinages des atomes sans inclure d'instruction de masquage et d'autre part car le coût de copie des atomes dans le tampon est plus faible par rapport au coût du potentiel. On peut supposer que pour des potentiels plus coûteux que le potentiel Sutten Chen l'utilisation de la stratégie avec tampon sera plus bénéfique que la méthode des blocs de Verlet.

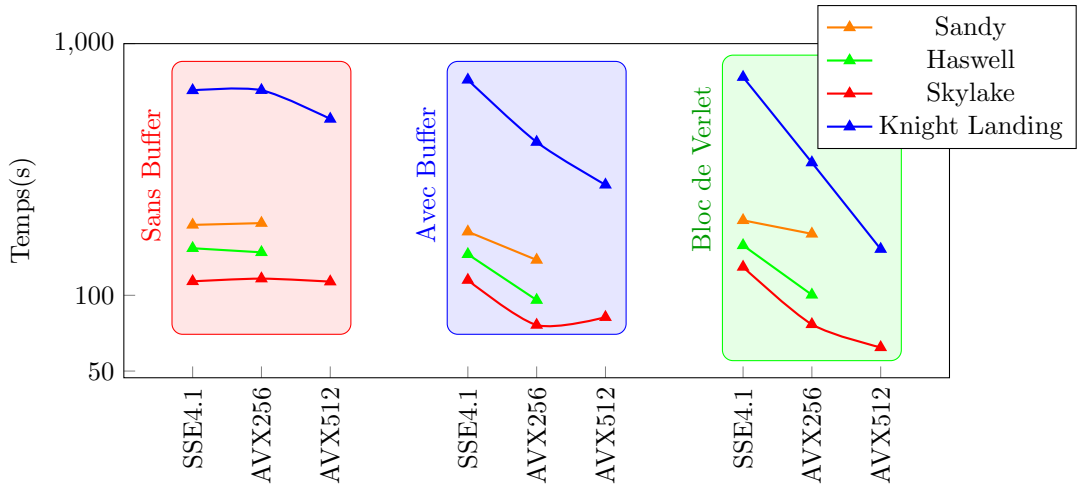


figure 6.18 | Temps effectif du calcul du potentiel EAM (échelle log en temps) selon le type d'architecture et le type de vectorisation utilisé.

Un autre intérêt de la méthode des blocs de Verlet est de réduire l'empreinte mémoire des listes de Verlet. En pratique, on observe que les listes de voisins sont plus chères à stocker que les atomes. Le nombre dépend du rayon de coupure du potentiel utilisé. Le Tableau 6.3 rapporte l'empreinte mémoire totale de la simulation décrite au début de ce paragraphe lorsque la méthode utilisée est la méthode des blocs de Verlet. Pour cette méthode, la taille des blocs utilisée dépend de la taille des registres vectoriels utilisés. Pour les intructions SSE4.1, le nombre d'éléments par blocs est de deux, de quatre pour les instructions AVX-256bits et de huit pour les instructions AVX 512-bits. Lorsque les blocs sont de taille 1, la méthode des blocs de Verlet équivaut à la méthode des listes de Verlet. On remarque que pour les simulations avec le potentiel Lennard Jones et Eam, la méthode des blocs de Verlet permet de réduire sur Knl et Skl respectivement 36.21% et 50.8% de l'empreinte mémoire totale de la simulation.

Taille des blocs	1	2	4	8
LJ	165.4	139.7	118.5	105.5
EAM	283.5	197.9	165.2	139.3

Table 6.3 | Empreinte mémoire en (MB) de la simulation en fonction de la taille des blocs de Verlet.

Finalement, pour les potentiels Lj et Eam, on supposera par la suite que la méthode des blocs de Verlet fournit les meilleurs résultats et la meilleure empreinte mémoire. Pour un potentiel Meam, on choisira plutôt la technique basée sur un tampon de vectorisation. Dans cette étude, le compilateur utilisé est celui d'intel. Si par exemple le compilateur gcc est utilisé pour un potentiel Meam, les temps sont comparables (+15% sur haswell) à ceux obtenus sur intel.

Dans le cadre de cette thèse, des tests préliminaires de comparaison ont aussi été effectués sur les processeurs ARM (thunderx2) avec le compilateur gcc. Les résultats sont exposés dans l'annexe 11.1.1. Néanmoins comme le processeurs thunderx2 ne profite guère de la vectorisation, on remarque que les résultats sont moins intéressant que pour un Skl, ce qui confirme l'impact bénéfique de la vectorisation sur les performances.

Conclusions

Dans ce chapitre, nous avons conçu et développé une grille AMR destinée aux application de DM. Elle est implantée dans le code EXASTAMP. Cette grille AMR permet d'adapter dynamiquement le maillage et ainsi de mieux gérer les déplacements très dynamiques des atomes. Nous avons observé que l'AMR permet de pallier les défauts de la méthode des cellules liées. Nous avons étudié un algorithme de blocage de cache (ou blocage d'antémémoire) afin de tenir compte de la mémoire hiérarchique des processeurs. Nous avons aussi étudié l'impact des instructions SIMD sur les opérateurs de calcul des potentiels. Nous avons introduit la méthode des blocs de Verlet réduisant l'empreinte mémoire de la simulation et vectorisant les opérateurs des potentiels. Cette première phase nous a permis d'obtenir une version séquentielle, vectorisée, et optimisée de la grille AMR.

Bien que les parallélisations intra-NŒUD et inter-NŒUDS aient été prise en compte dans la conception de l'intégration de la grille AMR dans EXASTAMP, ces deux points n'ont pas encore été abordés. Le chapitre suivant portera donc sur l'élaboration de diverses stratégies de parallélisation intra-NŒUD sur processeur multicœurs. Celles-ci tiendront compte des optimisations déjà effectuées dans ce chapitre.

7 Élaboration d'une parallélisation intra-Nœud adaptée aux architecture multicœurs

L'élaboration d'une grille AMR au sein de l'architecture d'ExaSTAMP a été réalisée en séquentiel et optimisée avec des instructions SIMD dans le chapitre 6. L'objectif de ce chapitre est de concevoir une parallélisation en mémoire partagée, nommée parallélisation intra-NŒUD, adaptée et optimisée pour la grille AMR.

Évaluer correctement le comportement de la parallélisation intra-NŒUD n'est pas chose aisée. En effet, selon la charge de calcul, le type de processeur ou la répartition des atomes, les résultats peuvent varier considérablement. Afin de se placer dans un cadre d'évaluation proche des phénomènes que le CEA souhaite étudier (voir paragraphe 3.5) nous proposons dans le paragraphe 7.1 deux simulations représentatives, à une plus faible échelle, de ces phénomènes, exécutables sur un NŒUD de calcul. Les simulations sont effectuées avec les potentiels LJ et MEAM. De plus, deux types de processeurs multicœurs récents sont utilisés pour obtenir les mesures de performance : l'Intel Xeon Skylake (SKL) et l'Intel Xeon Phi Knights Landing (KNL).

Deux premières stratégies de parallélisation intra-NŒUD sont introduites dans la partie 7.2. Ces stratégies sont basées sur différents découpages de la boucle de calcul itérant sur les cellules feuilles. Elles sont évaluées sur une première simulation test dont la densité atomique est parfaitement équirépartie (cf. paragraphe 7.2.2). L'objectif est de s'assurer des bonnes performances d'ExaSTAMP AMR pour des simulations plus "classiques". Ensuite, les deux stratégies sont évaluées sur les deux simulations représentatives du développement d'un micro-jet d'étain et de l'impact d'une nanogoutte d'étain sur une surface solide (cf. paragraphe 7.2.3). Les performances obtenues pour la parallélisation intra-NŒUD sur la grille AMR sont comparées aux résultats produits par les codes *ExaSTAMP legacy* et LAMMPS.

Une dernière stratégie de parallélisation est introduite dans la partie 7.3. Son but est de limiter l'utilisation des mutex (verrous) lors de phases multithreads qui ralentissent les simulations lors du calcul des potentiels. Elle est basée sur la méthode par vagues et elle s'appuie sur une parallélisation à base de tâches, voir paragraphe 7.3.3. Finalement, les performances de cette dernière stratégie sont étudiées dans le paragraphe 7.3.4 sur des cas homogènes et hétérogènes.

Mise en place de simulations "hétérogènes et dynamiques"

Dans ce chapitre, les expérimentations sont effectuées sur deux simulations correspondant aux simulations cibles mais à une échelle réduite afin qu'elles puissent être traitées sur un seul NŒUD d'un supercalculateur. Les deux simulations sont décrites et illustrées dans les paragraphes 7.1.1 et 7.1.2.

Éjecta de matière (micro-jet)

figure 7.1 | Développement d'un micro-jet suite à la propagation d'une onde de choc de gauche à droite.

La première simulation, nommée micro-jet, est issue d'une simulation numérique plus complexe reproduisant le développement d'un micro-jet d'étain. Cette simulation a été préalablement réalisée par le code *ExaSTAMP legacy* avec le potentiel EAM vniitf. Le nombre d'atomes est trop important pour pouvoir être exécuté sur un seul NŒUD de calcul. Afin d'obtenir un cas plus petit, une "lamelle" du domaine de la simulation a été extraite (illustrée par la figure 7.1). Cette lamelle (3D) a été prélevée dans le sens de la propagation (axe Ox) de la nappe afin de restituer au mieux les différences de densité. Cette simulation correspond alors bien aux capacités de mémoire et de calcul d'un NŒUD (SKL ou KNL). Des conditions aux limites périodiques sont appliquées sur les bords (axe Oy et Oz) afin de conserver la forme du micro-jet pendant quelques centaines de pas de temps. À noter qu'une fine couche de vide est ajoutée pour que les atomes sur les bords ne soient pas trop proches des atomes répliqués dans les zones fantômes. Ceci pour éviter des problèmes de stabilité numérique lors des premières itérations sur ce cas tronqué.

La particularité de cette simulation est d'être composée d'atomes se déplaçant particulièrement rapidement pour une simulation de dynamique moléculaire. Les listes de Verlet sont alors très régulièrement mises à jour. Cette simulation est alors dite dynamique et nous permet d'évaluer la capacité de la grille AMR à s'adapter au fur et à mesure que les atomes se déplacent. À noter que la méthode des cellules liées génère dans ce cas environ 95% de cellules contenant très peu d'atomes.

Impact d'une nano-goutte d'étain sur une surface solide (Impact)

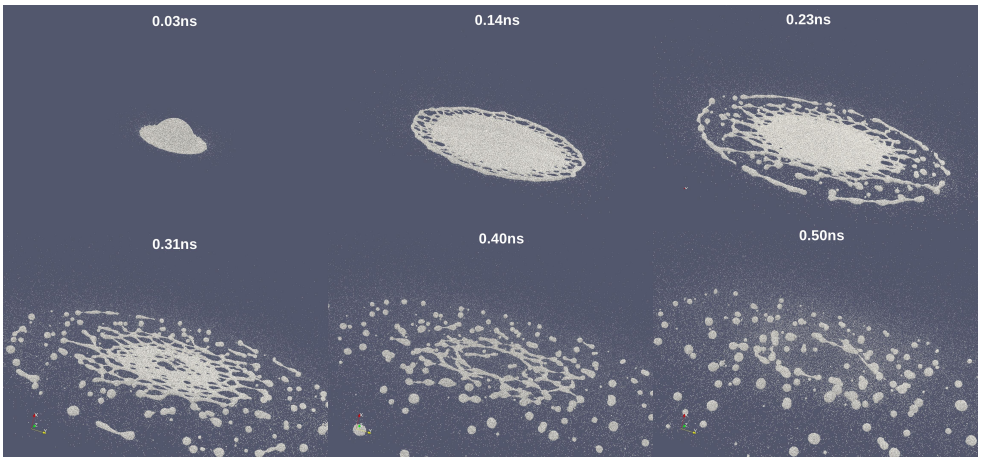


figure 7.2 | Impact d'une sphère d'étain contre un mur.

La deuxième simulation, nommée impact, consiste à miniaturiser une nano-goutte d'étain et de la projeter contre une surface solide. La figure 7.2 illustre les différentes étapes de cette simulation. Ce cas-test est exécuté par ExaSTAMP Amr car contrairement à la simulation précédente, c'est un exemple de simulation qu'il était jusqu'à présent quasi-impossible de réaliser par *ExaSTAMP legacy* sur des processeurs actuels (Knl ou Skl) avec 192GO de ram. La seule possibilité était jusqu'à présent de simuler moins d'atomes et donc un domaine beaucoup plus petit, ce qui ne permettait pas de reproduire les phénomènes physiques qui nous intéressent.

Pour réaliser cette simulation, une maille face cubique centrée est répliquée selon les 3 axes dont uniquement les atomes inclus dans une sphère d'un certain rayon sont conservés pour obtenir la forme d'une goutte. La goutte est composée de 3 889 370 atomes d'étain. Le potentiel Meam (rCut=4.17 Å) est utilisé pour modéliser les interactions entre les atomes d'étain. Ensuite le système est thermalisé à 800 degrés en appliquant le schéma d'intégration *langevin splitting* pendant 62 500 pas de temps. À la fin de la phase de thermalisation, les atomes sont projetés vers une surface solide. Les conditions aux limites de la simulation sont libres dans toutes les directions en dehors de celle modélisant la surface solide par un potentiel répulsif (mur). Après l'impact, les atomes se propagent principalement selon les axes Oy et Oz et on observe l'apparition de cavités et de bourrelets aux extrémités de la goutte. Le domaine de la simulation est agrandi au fur et à mesure que les atomes se déplacent. Finalement, on retrouve les étapes caractéristiques de impact d'une nano-goutte d'étain sur une surface solide déjà observées à une échelle plus importante.

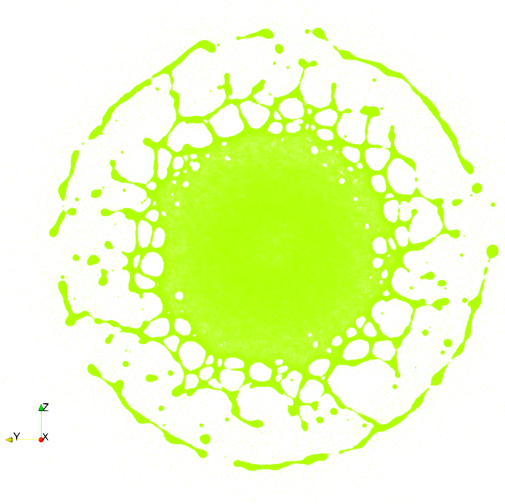


figure 7.3 | Impact d'une nano-goutte d'étain sur une surface solide à 300 000 ps.

Dans ce chapitre, les stratégies de parallélisation sont évaluées à partir de la "protection" (Sauvegarde) correspondant au pas de temps 300 000. Cette étape est illustrée par la figure 7.3. Les bornes du domaine de la simulation sont alors comprises entre (-36.817,-331.353,-331.35) et (257.719,331.353,331.353) nanomètre. L'espace ne contenant pas ou peu d'atomes représente environ 99,8% du volume total de la simulation.

Choix d'une parallélisation "itératives"

Dans ce paragraphe nous allons expliquer comment la parallélisation en mémoire partagée a été développée. La parallélisation des boucles de calcul est effectuée via l'utilisation de l'interface Open Multi-Processing (OpenMP). OpenMP est préféré à Tbb car il a l'intérêt d'être peu intrusif dans

le code grâce à l'utilisation de directive \#pragma omp devant les boucles de calcul. De plus, la communauté d'utilisateurs d'OpenMP est bien plus importante, offrant ainsi davantage de pérennité au code. La plupart des opérateurs de calculs dont les temps de traitement sont négligeables au vue de la durée totale de la simulation (schéma numériques, application des conditions de murs, etc) sont trivialement parallélisés entre les octrees. Cette stratégie est nommée *boucle octrees*. Elle est illustrée par l'image de gauche de la figure 7.4. Les boucles de calcul au sein de la grille sont alors écrites ainsi :

```
1  #pragma omp parallel for
2  for(ssize_t it = 0 ; it < octrees.size() ; it++)
3    Operator(octrees[it]);
```

L'impact de la parallélisation en mémoire partagée sur un code de simulation numérique est généralement mesuré par son accélération ou son efficacité. L'accélération d'un code parallèle est définie comme étant le temps de la simulation à 1 thread divisé par le temps de la simulation avec N threads. Donc par exemple, si le temps de simulation d'un code sur 3 threads est divisé par deux par rapport à sa version séquentielle, alors son accélération est de $1/0.5 = 2$ pour 3 threads. Plus l'accélération est élevée, plus le gain apporté par la parallélisation en mémoire partagée est important par rapport à la version séquentielle du code.

L'efficacité est obtenue en effectuant le ratio entre le temps mesuré à N threads et le temps idéal à N threads. Le temps idéal à N threads étant le temps séquentiel divisé par N. Donc si le temps de simulation est divisé par 2 pour trois threads, l'efficacité du code est alors de 66.6% pour trois threads.

On dit qu'un code a une scalabilité parfaite si le nombre de ressources allouées à la simulation (threads,cœurs) est proportionnel au gain, c'est-à-dire qu'en allouant deux fois plus de ressources à la simulation, celle-ci est deux fois plus rapide. Si un code a une scalabilité parfaite alors l'accélération est égale au nombre de threads et l'efficacité est de 100%. Dans la suite du manuscrit, afin de simplifier la démarche durant les études, on prendra pour acquis que la plupart des portions du logiciel ExaSTAMP Amr ont une efficacité de 100% car les mesures de performances de ces portions de code via l'outil Maqao [44] ont montré qu'elles sont très parallèles pour un très grand nombre de threads et que leurs temps sont négligeables. Par contre, cette constatation n'est pas valable pour les opérateurs suivant :

- le calcul du potentiel;
- la construction des listes de voisins;
- le raffinement de la grille.

Dans les paragraphes suivants, nous allons donc nous focaliser sur le développement d'une version efficace de ces opérateurs.

Parallélisation du calcul de l'énergie potentielle

La charge de calcul d'un code de Dm traitant uniquement les interactions à courte portée est principalement localisée dans l'opérateur construisant les listes des atomes voisins ainsi que dans l'opérateur calculant l'énergie potentielle. La scalabilité de tels codes de Dm est donc étroitement corrélée à ces opérateurs.

En dehors de ces deux opérateurs comme par exemple les opérateurs effectuant la réduction (somme) d'un champs comme l'énergie potentielle, la parallélisation est toujours effectuée entre

les octrees (stratégie *boucle octrees*). Dans ce cas les cellules feuilles d'un octree sont traitées en séquentiel. Nous avons choisi d'appliquer aussi cette stratégie pour l'opérateur effectuant le calcul de l'énergie potentielle car elle est facile à mettre en œuvre pour la grille AMR. La stratégie *boucle octrees* est aussi étudiée pour l'opérateur du calcul de l'énergie potentielle car elle permet de fixer le nombre de cellules feuilles (bloc) traitées séquentiellement, et donc d'appliquer l'algorithme de *cache blocking*.

Une deuxième stratégie de parallélisation peut lui être opposée : elle consiste à appliquer les opérateurs de construction des listes de voisins et du calcul du potentiel directement sur la structure des cellules feuilles. Le parcours des cellules feuilles est nommé *boucle feuilles*.

En séquentiel, l'unique thread traite les cellules feuilles dans le même ordre pour les deux stratégies *boucle octrees* et *boucle feuilles*. Cependant l'ordre de traitement diffère lorsque plusieurs threads sont utilisés en parallèle. En effet, lorsque les itérations de la boucle de calcul sont attribuées aux threads, la stratégie *boucle feuilles* distribue des blocs d'itérations correspondant aux cellules feuilles sans tenir compte du découpage en octrees. Alors que la stratégie *boucle octrees* distribue des blocs de cellules feuilles formant des octrees. Par conséquent la répartition des cellules feuilles entre les threads est différente ce qui impacte la parallélisation des calculs et les optimisations effectuées pour maintenir les données dans les caches mémoires. À noter qu'un des objectifs important de la stratégie *boucle octrees* est de forcer la taille des blocs de cellules feuilles en pilotant le paramètre D^{max} pour améliorer la réutilisation des caches mémoires.

La figure 7.4 montre les deux parcours *boucle octrees* (gauche) et *boucle feuilles* (droite) sur une grille AMR rectilinéaire avec D^{max}=1.

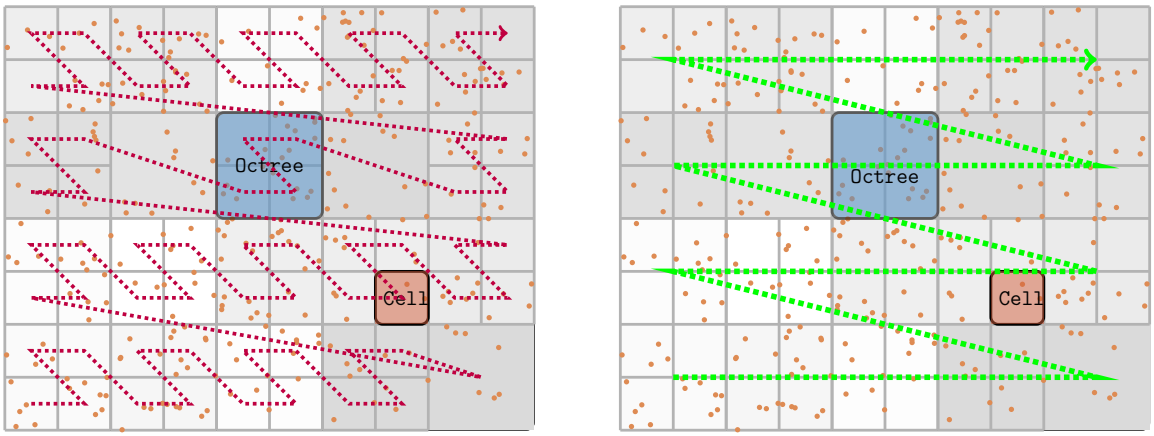


figure 7.4 | L'image de gauche décrit le parcours *boucle feuilles* (flèche voilette) alors que l'image de droite retranscrit le parcours *boucle octrees* (flèche).

L'un des points fort de la parallélisation en mémoire partagée est que chaque thread a accès à la totalité de la mémoire. Néanmoins lorsque deux threads modifient la même donnée, le résultat obtenu diffère selon l'ordre d'accès des threads. Par exemple si un thread a chargé la donnée avant que l'autre thread l'ait modifié, il risque d'écraser cette modification en stockant un nouveau résultat. Pour remédier aux accès concurrents, la solution est de laisser à un thread le soin de verrouiller l'accès à certaines données. Cela empêche ainsi les autres threads d'accéder ou de modifier une donnée tant que le premier thread n'a pas fini ses opérations. Ces verrous sont nommés mutex et sont intégrés dans de nombreuses librairies comme les STD::MUTEX de la librairie standard, les TBB::SPIN_MUTEX de TBB ou les OMP_LOCK_T de OPENMP. Les mutex utilisés dans EXASTAMP sont implémentés par la classe spin_mutex suivante :

```
class spin_mutex {
private:
    std::atomic_flag _lock = ATOMIC_FLAG_INIT;
    std::atomic<std::size_t> _spin_pred{0};
public:
    spin_mutex() {}
    ~spin_mutex() {}
    inline bool try_lock() { return !_lock.test_and_set(std::memory_order_acquire);}
    inline void lock() {
        std::size_t spin_count{0};
        while (!try_lock()) {
            if (spin_count < _spin_pred * 2)  continue;
            std::this_thread::sleep_for(std::chrono::microseconds(1));
            _spin_pred += (spin_count - _spin_pred) / 8;
          }
    }
    inline void unlock() {  _lock.clear(std::memory_order_release);}
};
```

Le principe est que le premier thread passant dans la fonction lock() d'un spin_mutex verrouille un std::atomic_flag. Si un deuxième thread tente de verrouiller ce même spin_mutex, il échoue au moment du try_lock() et retente sa chance tant que le premier thread n'est pas passé dans la fonction unlock(). Afin de limiter le nombre de tests, les try_lock sont effectués à intervalles de temps décroissant.

Or, lors du calcul des potentiels LJ, EAM et MEAM, une partie du calcul de l'interaction entre deux atomes est stockée pour chaque atome atome dans une variable *Ep*. Si deux atomes voisins sont traités par deux threads en même temps ceux-ci entrent en conflit lors de l'écriture des variables *Ep*. C'est pour cela qu'un mutex est ajouté pour chaque atome lors du calcul du potentiel. Afin de continuer à utiliser la méthode des blocs de Verlet, un mutex est ajouté aux blocs d'atomes.

Le découpage des boucles de calcul est décidé par la politique d'ordonnancement d'OPENMP. Il est possible de la fixer en ajoutant la directive *schedule()*, cette directive prend en compte 4 cas de figure :

- statique. Chaque thread reçoit environ le même nombre d'itérations. Lorsque la taille de bloc d'itérations est fixée, les blocs sont distribués sur les threads dans un ordre circulaire.
- dynamique. Par défaut la taille des blocs d'itérations est fixée à 1, chaque thread traite un bloc d'itérations puis en demande un autre. Si la boucle est parcourue plusieurs fois, les blocs d'itérations reçus par les threads ne seront donc pas forcément les mêmes.
- guidée. Comme pour le cas dynamique, les threads reçoivent les blocs d'itérations au fur et à mesure qu'ils finissent de traiter leurs blocs. Néanmoins, cette ordonnancement diffère du cas dynamique par sa décomposition de la boucle en blocs d'itérations. La taille des blocs est proportionnelle au nombre d'itérations non assignées divisé par le nombre de threads. Ainsi les premiers blocs auront beaucoup d'itérations alors que les derniers auront la taille définie par l'utilisateur ou par défaut 1.
- auto. Délègue le choix au runtime OPENMP entre les trois précédentes stratégies d'ordonnancement.

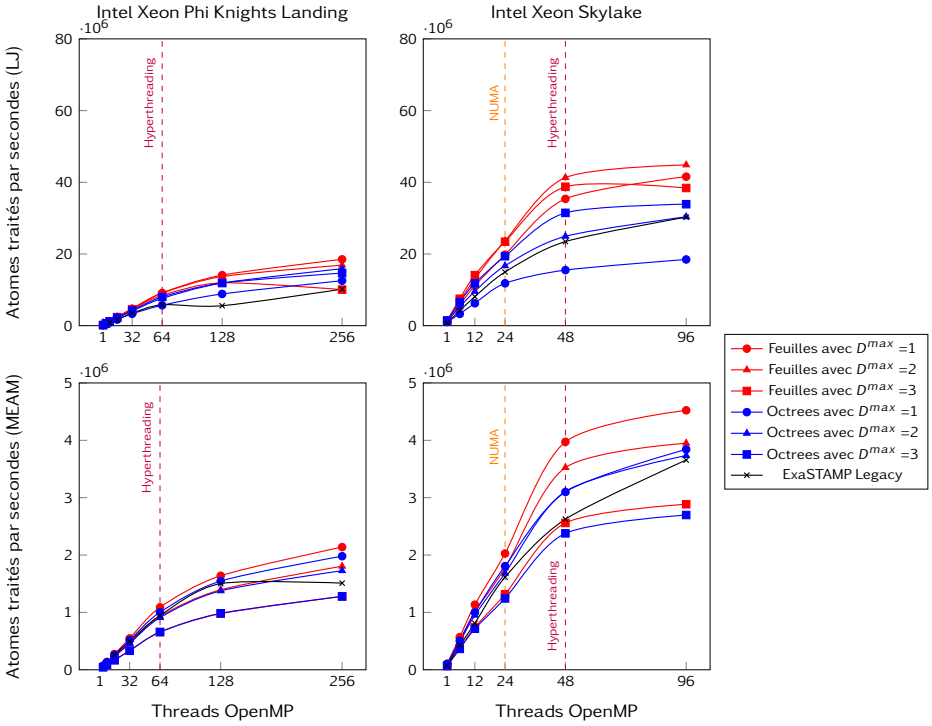
Évaluation sur des cas test "homogènes et statiques"

L'utilisation d'une grille AMR a pour but de réduire la durée des simulations hétérogènes et dynamiques. Dans cette partie nous montrons que la méthode AMR n'impacte pas les performances

des simulations homogènes et statiques. Pour cela, deux scénarios sont élaborés en faisant varier la charge de calcul par l'utilisation soit d'un potentiel LJ, soit d'un potentiel MEAM.

Rappelons que le potentiel LJ a l'intérêt de pouvoir étudier l'impact de la grille AMR sur l'ensemble du code car celui-ci ne masque pas les défauts des autres portions du code contrairement au potentiel MEAM. De plus, il est représentatif de la charge de calcul d'un grand nombre de potentiels et il est présent dans la plupart des logiciels de DM. Le potentiel MEAM est aussi utilisé car c'est l'un des potentiels les plus précis pour modéliser les interactions entre les atomes d'étain.

Pour ce cas, la mesure utilisée est le nombre d'atomes traités par seconde. Cette mesure est très importante et significative pour les physiciens. C'est pour cela que nous l'utilisons pour l'étude des résultats des performances pour ce paragraphe.



Graphe 7.5 | Nombre d'atomes traités par seconde sur une simulation homogène et statique avec un potentiel LJ et MEAM sur les architecture Intel Xeon Phi Knights Landing et Intel Xeon Skylake. La grille AMR utilise les stratégies *boucle octrees* et *boucle feuilles* avec $D^{max} = 1, 2, 3$.

Les résultats des graphes 7.5 sont obtenus en simulant avec *ExaSTAMP legacy* et ExaSTAMP avec AMR un cristal parfait de 6 912 000 d'atomes de cuivre pour le potentiel LJ (RCUT=5.68) et de 2 000 000 d'atomes d'étain (RCUT=4.17Å). Les politiques d'ordonnancement optimales sont respectivement fixées à dynamique et guidée pour les stratégies de parallélisation *boucle octrees* (en bleu) et *boucle feuilles* (en rouge). *ExaSTAMP legacy* (en noir) et sa grille classique ne spécifie pas de politique d'ordonnancement.

Pour les deux simulation, D^{max} est fixé à 1 (○), 2 (△), 3 (□) et la totalité des octrees sont complets. La profondeur des octrees modifie alors le parcours des cellules feuilles pour la stratégie *boucle feuilles* et pilote le nombre de cellules feuilles traitées séquentiellement pour la stratégie *boucle octrees*. L'objectif de la stratégie *boucle octrees* est de chercher la taille de bloc optimale maximisant la réutilisation des caches par l'algorithme de *cache blocking*. Globalement, on observe que pour ces simulations la stratégie *boucle feuilles* offre de meilleurs résultats que la stratégie *boucle octrees*. Le traitement des atomes sur SKL est alors environ 2 fois plus rapide que sur KNL. De plus,

l'hyperthreading apporte un gain dans tous les cas et les effets NUMA sont peu visibles. Finalement, le choix de D^{max} n'est pas anodin car bien que la densité soit homogène et que les octrees soient complets, le paramètre D^{max} peut faire varier jusqu'à 50% les temps de calcul lorsque toutes les ressources d'un processeur sont utilisées.

Pour les cas utilisant le potentiel LJ (les graphes du haut), on remarque que selon la stratégie de parallélisation utilisée, le D^{max} optimal n'est pas le même. Par exemple sur SKL il est de 2 pour la stratégie *boucle feuilles* alors qu'il est de 3 pour la stratégie *boucle octrees*. De plus le D^{max} optimal varie selon l'architecture cible. Pour la stratégie *boucle feuilles*, le D^{max} optimal est obtenu avec 2 sur SKL alors qu'il est de 1 sur KNL. On remarquera aussi que le D^{max} optimal a été décrémenté de 1 sur KNL pour la stratégie *boucle octrees*. Les différences entre les deux processeurs s'expliquent par le fait que les processeurs n'ont d'une part pas le même nombre de threads et d'autre part ils ne traitent pas les opérateurs à la même vitesse (fréquence/vectorisation). Donc il n'y a pas que la granularité de la tâche qui entre en compte pour choisir les paramètres optimaux, il faut aussi considérer le processeur et le nombre de threads.

Au final, du point de vue de l'accélération, le code avec l'AMR et sans l'AMR sont respectivement de 35.5 et 37.0 sur SKL pour 48 cœurs et de 119.6 et 95.5 KNL pour 64 cœurs. Ce qui est relativement proche alors que la grille AMR est plus rapide que *ExaSTAMP legacy* de 1.48 sur SKL et 1.82 sur KNL. Ces gains sont principalement obtenus grâce aux optimisations réalisées sur la vectorisation via la méthode des blocs de verlet et sur la réutilisation des caches mémoires via l'algorithme de *cache blocking*. En effet, si l'on exécute les codes avec un thread, l'accélération de la grille AMR par rapport à *ExaSTAMP legacy* est de 1.54 sur SKL et de 1.44 sur KNL ce qui illustre parfaitement qu'en plus de ne pas dégrader les performances dans un cas défavorable à l'AMR, notre solution améliore les performances du code.

Bien que les résultats obtenus avec la grille AMR soient toujours supérieurs (sauf pour la stratégie *boucle octrees* avec $D^{max} = 1$ sur SKL) à ceux d'*ExaSTAMP legacy*, les résultats optimaux sont obtenus empiriquement (choix de D^{max}). De plus, en dehors des opérateurs effectuant la recherche des voisins et du potentiel, les autres sections ne dégradent pas la scalabilité du code ExaSTAMP avec l'AMR car elles sont toujours négligeables. Dans notre cas, pour chaque simulation, le temps du potentiel est supérieur à 80% du temps total. De plus, en séquentiel, l'ajout des mutex dégrade fortement les performances en divisant le nombre d'atomes traités par seconde par 1.8 sur SKL et 2.46 sur KNL. Nous verrons dans la partie 7.3 comment les éviter.

Dans le cas d'un potentiel MEAM avec un coût de calcul supérieur (graphes du bas), l'écart entre les deux stratégies est plus faible et elles suivent les mêmes tendances en fonction de D^{max} pour les deux architectures cibles. De plus la stratégie *boucle feuilles* est plus rapide que la stratégie *boucle octrees* et *ExaSTAMP legacy* pour les deux architectures. L'opérateur du potentiel MEAM a une intensité arithmétique largement plus importante que celle du potentiel LJ. Il est moins sensible aux effets de cache et donc moins sensible au parcours des éléments. Il est donc plus intéressant de créer des petits blocs d'itérations afin de limiter l'inactivité des threads à la fin de la boucle de calcul. On remarque que la sensibilité des résultats en fonction des stratégies de parcours est plus faible que dans le cas avec un potentiel LJ.

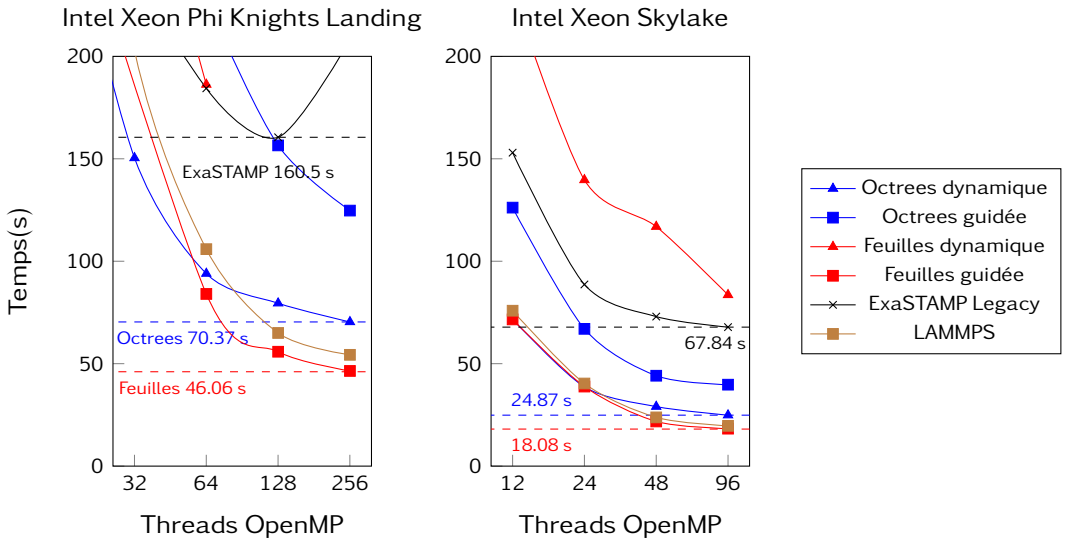
Finalement, ces simulations homogènes et statiques nous ont permis d'observer l'impact bénéfique de la parallélisation en mémoire partagée sur la grille AMR selon les stratégies de parcours *boucle octrees* et *boucle feuilles*. Pour ces scénarios, D^{max} a une grande influence sur la durée de la simulation. Dans le cas d'un potentiel peu coûteux il faudra aussi tenir compte des stratégies de parcours alors que celles-ci semblent peu impactantes avec un potentiel plus coûteux. Mais l'observation la plus intéressante et qu'ExaSTAMP AMR a de très bonnes performances sur un cas test qui ne lui est pas favorable.

Évaluation sur des scénarios "hétérogènes et dynamiques"

Le principe de l'Amr est d'adapter localement la taille des cellules de la grille à la densité d'atomes. Les deux simulations décrites dans le paragraphe 7.1 sont des cas dont la densité d'atomes est fortement hétérogène et dont le déplacement des atomes à chaque pas de temps est important (dynamique). À noter que la simulation micro-jet possède une densité moins hétérogène que la simulation d'impact d'une nano-goutte mais ses atomes se déplacent plus rapidement.

Micro-jet

Le développement d'un micro-jet décrit dans le paragraphe 7.1.1 est composé d'environ 3.5 million d'atomes d'étain. Les graphes 7.6 et 7.7 correspondent aux temps d'exécutions des simulations pour 100 pas de temps en utilisant respectivement un potentiel Lj (rCut=10Å) et un potentiel Meam (rCut=4.17Å). Comme il a été observé dans la partie précédente, le paramètre D^{max} influence la durée des simulations, celui-ci est alors fixé à son optimal pour ce cas test (obtenu après plusieurs tests). Dans ce paragraphe, notre attention se porte sur la compréhension de la politique de distribution des cellules feuilles sur les threads. Afin de ne pas surcharger les figures, les politiques d'ordonnancement statique et auto ne sont pas affichées car la politique statique ne fournit jamais les meilleurs résultats et la politique auto choisit la politique guidée.



Graphe 7.6 | Temps d'exécution d'ExaSTAMP et de Lammps pour la simulation micro-jet avec un potentiel Lj pour les méthodes de parallélisation OpenMP par *boucle octrees* et *boucle feuilles* selon les politiques d'ordonnancement dynamique et guidée. Le choix de D^{max} optimal est de 2.

Le potentiel Lj, graphe 7.6, est utilisé pour pour mettre d'éventuel surcoût de la structure. Les codes Lammps (package KOKKOS / OpenMP) et *ExaSTAMP legacy* sont utilisés comme points de repère. On remarque que la solution basée sur la grille Amr combinée avec la stratégie *boucle feuilles* et une politique d'ordonnancement guidée est plus rapide que les solutions proposées par *ExaSTAMP legacy* et Lammps pour les deux architectures cibles. Quant à la stratégie *boucle octrees*, elle est plus efficace avec la politique dynamique qu'avec la politique guidée car les octrees forment des blocs favorisant la réutilisation des caches mémoire dans les zones denses de la simulation. À contrario, distribuer les cellules feuilles à l'unité (politique dynamique) dégrade fortement les performances de la stratégie *boucle feuilles*.

Finalement l'accélération apportée par la grille AMR pour cette simulation est d'environ 3.5 par rapport à *ExaSTAMP legacy* car les cellules vides dégradent le parallélisme et engendrent des surcoûts de calculs. Par exemple entre 12 et 96 threads sur SKL, la durée de la simulation est seulement réduit par 2.25 pour *ExaSTAMP legacy* alors qu'elle est réduite par 3.85 pour LAMMPS et de 3.93 pour AMR.

On remarque aussi que le gain apporté par la grille AMR est plus faible comparé à LAMMPS (+9% sur SKL et +19% sur KNL). En effet, LAMMPS souffre moins des défauts liés à l'utilisation d'une grille structurée. Cela s'explique car les atomes sont stockés dans une unique structure de tableaux. Donc contrairement à *ExaSTAMP legacy* ou ExaSTAMP AMR, les atomes ne sont pas stockés dans les cellules, donc le coût de stockage des cellules est beaucoup plus faible. De plus, contrairement à *ExaSTAMP legacy*, LAMMPS parallélise (OpenMP) les calculs des interactions en itérant sur les atomes au lieu des cellules, et donc il ne parcourt pas de cellules ne contenant pas d'atome. Néanmoins tout comme *ExaSTAMP legacy* et ExaSTAMP avec AMR, LAMMPS est ralenti par l'utilisation d'instruction atomic : \#pragma omp atomic. Les temps observés pour LAMMPS à un thread sans instruction atomic et à deux threads avec instructions atomic sont respectivement de 452.9 secondes et 391.8 secondes sur SKL et de 2763 secondes et 3160 secondes sur KNL.

De plus, que ce soit sur SKL ou KNL la grille AMR passe moins de temps à construire les blocs de voisins et à calculer les interactions que les autres codes. Bien que le coût du raffinement soit d'environ 5% du temps total, le gain (séquentiel et parallèle) apporté sur l'opérateur construisant les listes des atomes voisins et l'opérateur du calcul de l'énergie potentielle contrebalance le coût du raffinement.

Logiciel	AMR (*)	AMR (**)	LAMMPS	*ExaSTAMP legacy*
Mémoire	3.2GB	8.0GB	6.4GB	48.0GB

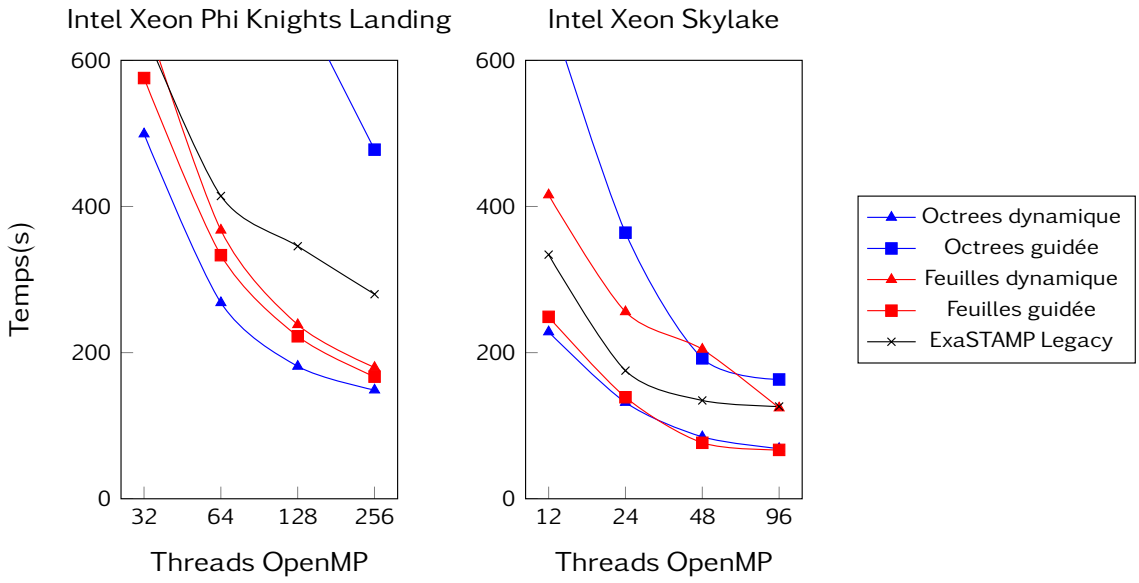
Table 7.1 | Empreinte mémoire pour différents codes pour la simulation micro-jet sur KNL avec 256 threads OpenMP. (**) Listes de Verlet. (*) Blocs de Verlet.

Le tableau 7.1 répertorie les empreintes mémoire des différents codes après 100 itérations en temps. On observe que l'empreinte d'*ExaSTAMP legacy* est 15 fois plus importante que pour la grille AMR à cause des cellules ne contenant pas d'atome et aux listes de Verlet plus coûteuses.

Concernant cette simulation avec un potentiel plus coûteux (MEAM), voir graphe 7.7, on observe que la méthode AMR réduit toujours l'impact des cellules ne contenant pas d'atome. Comme pour le potentiel LJ, les deux politiques de distribution des itérations engendrent pour les deux stratégies de parcours des résultats différents. On observe encore que la politique dynamique et la politique guidée sont respectivement les meilleures pour les stratégies *boucle octrees* et la *boucle feuilles*. De plus, comme le coût des calculs des interactions est plus important et masque davantage le coût des cellules sans atome, l'écart de temps entre la grille *ExaSTAMP legacy* et la grille AMR est plus faible.

Impact d'une nano-goutte d'étain sur une surface solide

La seconde simulation consiste à reprendre la simulation décrite dans le paragraphe 7.1.2 à 300 ps et sur 100 pas de temps. Comme dans le cas précédent, les potentiels LJ et MEAM sont employés. Les stratégies *boucle octrees* et *boucle feuilles* sont étudiées selon la politique d'ordonnancement



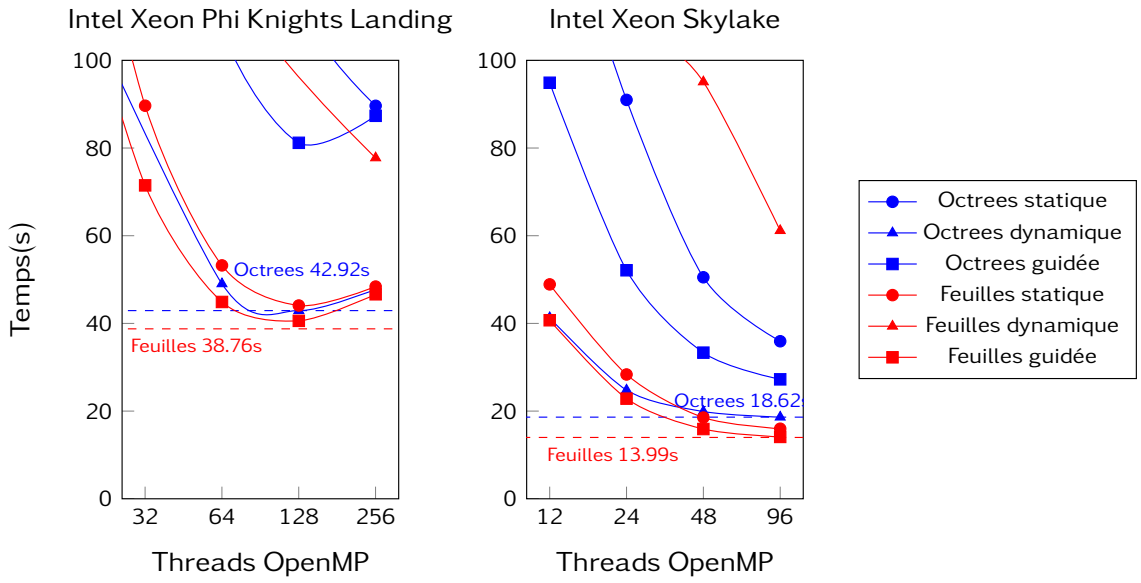
Graphe 7.7 | Temps d'exécution d'ExaSTAMP de la simulation micro-jet avec un potentiel Meam pour les méthodes de parallélisation OpenMP par *boucle octrees* et *boucle feuilles* selon les politiques d'ordonnancement statique, dynamique, auto et guidée. Le choix de D^{max} optimal est de 2.

dynamique, la politique statique et la politique guidée. La politique auto correspond dans ce cas à la politique guidée. Ce deuxième scénario a pour objectif de valider nos premières observations sur un autre cas hétérogène et dynamique. De plus, les codes *ExaSTAMP legacy* et Lammps ne peuvent exécuter une telle simulation à cause du trop grand nombre de cellules générées au sein d'un unique domaine.

Le rayon de coupure utilisé pour le potentiel Lj est diminué de 10 à 6 car le potentiel répulsif utilisé pour créer le mur (la surface solide) dépend au maximum des rayons de coupure. Le nombre de cellules de Verlet générées par la méthode des cellules liées aurait été d'environ 747 millions soit 181 012 octrees pour $D^{max} = 4$ (adapté à la taille du domaine). Le nombre de cellules feuilles est alors de 777 279 dont 179 996 sont des cellules racines. Le nombre d'octrees vides ou presque (composé d'une cellule racine) représentent donc 99.438% des octrees pour $0,86\%$ des atomes. Néanmoins si l'on s'intéresse à la boucle de calcul, les cellules racines ne représentent plus que 22.771% des cellules feuilles. La boucle de calcul est donc moins polluée par les cellules ne contenant pas ou peu d'atomes.

Le graphe 7.8 présente la durée des simulations en employant un potentiel Lj selon le type de politique d'ordonnancement des calculs en fonction du nombre de threads. On observe que, comme pour le micro-jet, les politiques dynamique et guidée sont respectivement optimales pour les stratégies *boucle octrees* et *boucle feuilles*. De plus, pour la stratégie *boucle feuilles*, la politique statique est proche de la politique guidée. Une explication est que les cellules feuilles ont environ la même charge de calcul sachant que les itérations des cellules feuilles obtenues après 4 raffinement composent $68,7\%$ de la boucle de calcul pour 93.17% du nombre d'atomes. Finalement les empreintes mémoires pour $D^{max} = 2, 3, 4$ sont respectivement de 18.6, 9.9, $8.7GO$. Cela montre à quel point il est nécessaire d'utiliser l'Amr.

Pour cette simulation, le temps de raffinement des octrees représente environ 20% du temps total de la simulation. Dans notre cas, l'algorithme de raffinement a été appliqué à chaque fois que les listes de Verlet ont été reconstruites. Une solution pour réduire son coût est d'ajuster les



Graphe 7.8 | Temps d'exécution d'ExaSTAMP Amr pour la simulation de l'impact d'une nano-goutte d'étain sur une surface solide avec un potentiel Lj. Les méthodes de parallélisation OpenMP sont soit *boucle octrees*, soit *boucle feuilles*. Elles sont utilisées avec les politiques d'ordonnancement statique, dynamique et guidée. Le choix de D^{max} est optimal est de 4.

pointeurs des cellules feuilles sur le stockage des atomes au lieu de raffiner un octree. Cette solution n'est pas utilisée car elle ajoute un autre degré de liberté difficile à appréhender. À noter que dans ce cas, cette solution permet de réduire de 13.99 secondes à 11.97 secondes sur Skl (le temps liée au raffinement est réduit par deux).

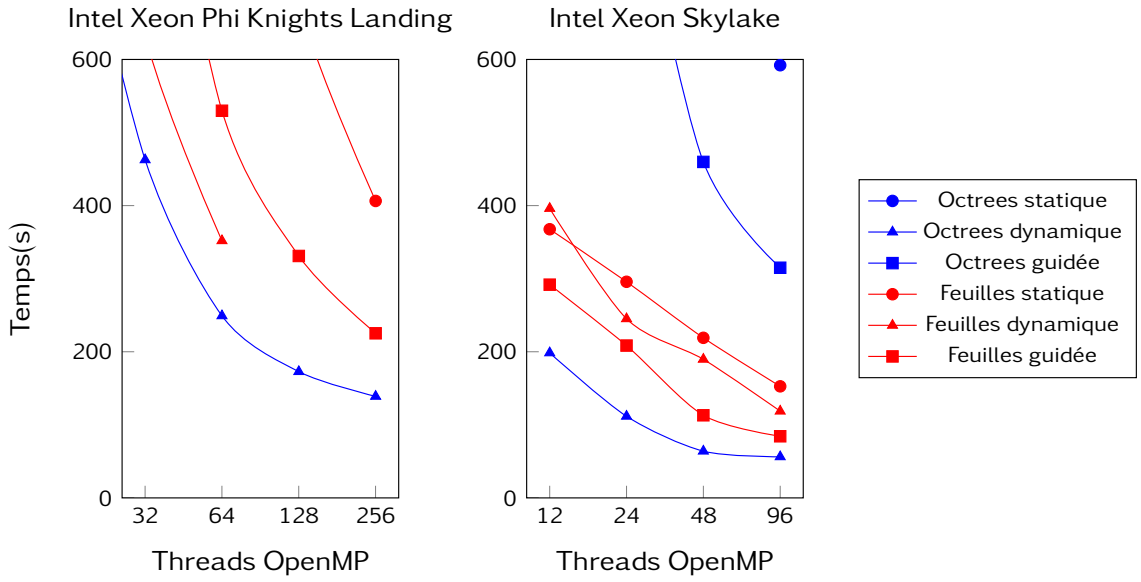


Figure 7.9 | Temps d'exécution d'ExaSTAMP Amr pour la simulation de l'impact d'une nano-goutte d'étain sur une surface solide avec un potentiel Meam pour les méthodes de parallélisation OpenMP par *boucle octrees* et *boucle feuilles* selon les politiques d'ordonnancement statique, dynamique et guidée. Le choix de D^{max} optimal est de 3 pour le Knl et de 4 pour le Skl.

Le graphe 7.9 est obtenu en utilisant le potentiel Meam (rCut= 4.17) au lieu du potentiel Lj. On observe, comme pour le micro-jet, que la tendance s'inverse par rapport au potentiel Lj. La stratégie *boucle octrees* avec la politique dynamique est la plus prometteuse.

Conclusion

Dans cette partie, les deux stratégies de parcours *boucle feuilles* et *boucle octrees* ont été évaluées sur trois scénarios différents. Ces études ont permis d'acquérir les enseignements suivant :

- Le paramètre D^{max} influence la durée des simulations en fonction du nombre de threads. Celui-ci tient compte du rayon de coupure et de l'intensité arithmétique du potentiel.
- Lorsque la simulation est homogène et statique, on observe que la stratégie de parcours *boucle feuilles* offre de meilleurs résultats que la stratégie *boucle octrees*.
- Lorsque la simulation est hétérogène et dynamique, la stratégie *boucle feuilles* avec la politique d'ordonnancement guidée est à privilégier pour un potentiel peu coûteux alors que la stratégie *boucle octrees* avec la politique dynamique est à prioriser pour les potentiels coûteux.
- La méthode Amr réduit drastiquement l'empreinte mémoire des cellules vides.
- Le coût de l'algorithme de raffinement et d'agglomération est négligeable par rapport à la durée de la simulation.
- L'utilisation des mutex impacte les performances du code et cela quelque soit la stratégie choisie.

Pour la suite du manuscrit, les stratégies de parcours *boucle feuilles* et *boucle octrees* sont respectivement utilisées avec les politiques guidée et dynamique.

La méthode par vagues

Au sein de l'opérateur effectuant le calcul de l'énergie potentielle, les calculs intermédiaires sont ajoutés à plusieurs variables afin de préserver des cycles de calcul. Pour éviter d'appliquer des verrous en utilisant de coûteux mutex, l'idée est de mettre en œuvre une méthode par vagues [92, 88, 89] qui classe les calculs afin d'éviter les accès concurrents à une même donnée par plusieurs threads.

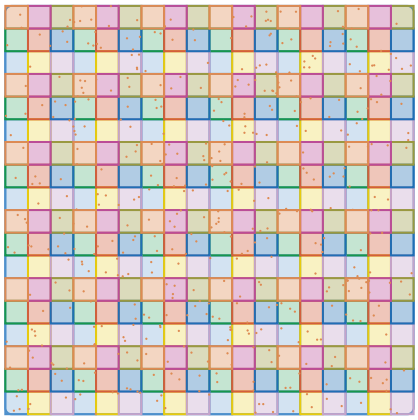


Figure 7.10 | Méthode par vagues en 2D.

La méthode par vagues est appliquée en DM en classant les cellules de Verlet en 27 vagues (9 en 2D), elles sont illustrées par la figure 7.10. Toutes les cellules d'une même vague peuvent être traitées en simultanée sans avoir besoin d'utiliser de mutex. En effet, on sait que les cellules de Verlet ont une largeur supérieure au maximum des RCUT, les cellules peuvent donc uniquement interagir avec les 26 cellules adjacentes. La classification assure que deux cellules d'une même vague n'ont pas de cellules adjacentes en commun, elles n'entrent donc jamais en conflit. Les vagues sont alors exécutées les unes après les autres. La méthode par vague peut prendre en compte des blocs de cellules de Verlet ce qui permet de réduire le nombre de vagues. Cependant, utiliser des blocs de cellules est une optimisation à double tranchant. Bien que les blocs améliorent la réutilisation des caches, chaque vague doit contenir un certain nombre de blocs afin de distribuer équitablement les calculs entre les threads.

Méthode par vagues combinée avec l'AMR

Dans le contexte d'un code de DM utilisant une grille AMR basée sur une forêt d'octrees formant une grille cartésienne de dimension 3, on sait que si le paramètre D^{max} est supérieur à 1 alors cela implique que la largeur minimale d'un octree est au moins supérieure à $2 * \max(\text{RCUT})$. Ainsi, si deux threads traitent deux octrees non adjacents mais sont tous les deux adjacents à un troisième octree, ils ne peuvent pas mettre à jour les informations d'un même atome contenu dans ce troisième octree. Cette situation est illustrée sur la figure 7.11.

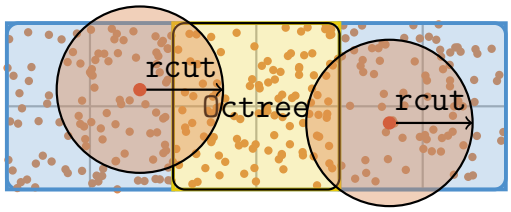


Figure 7.11 | Si deux atomes appartiennent à deux octrees bleus différents alors ces deux atomes ne peuvent interagir entre eux car la largeur d'un octree est supérieur à deux fois le maximum des rayons de coupures.

Dans le cas d'une grille AMR avec D^{max} >=1, la méthode par vagues consiste à classer les octrees selon 8 vagues en $3D$. La méthode est illustrée par la figure 7.12.

Cette stratégie est analogue à l'utilisation de blocs de cellules. Le défaut de cette adaptation de la méthode par vagues est qu'elle dépend du nombre d'octrees par vague. En effet, si le nombre d'octrees est inférieur au nombre de threads alors des threads seront inactifs pendant toute la durée du calcul de l'énergie potentielle. Pour les simulations cibles : micro-jet et l'impact d'une nano-goutte d'étain sur une surface solide, nous avons mesuré que le nombre d'octrees est largement supérieur à 8 fois le nombre de threads.

Limite de la méthode par vagues

La méthode par vagues a été implémentée dans le code EXASTAMP avec la grille AMR. Les octrees sont traités séquentiellement et classés selon 8 vagues. La boucle de calcul consiste en une double boucle imbriquée. La première boucle parcourt les vagues et la deuxième boucle parcourt les octrees de la vague. Cette dernière est parallélisée selon une politique d'ordonnancement dynamique (stratégie *boucle octrees*). Lors de nos premières expérimentations sur les scénarios homogènes et hétérogènes, nous avons observé que la barrière de synchronisation entre deux vagues engendre des moments d'inactivités pour certain threads. Les traces de l'opérateur du potentiel LJ pour

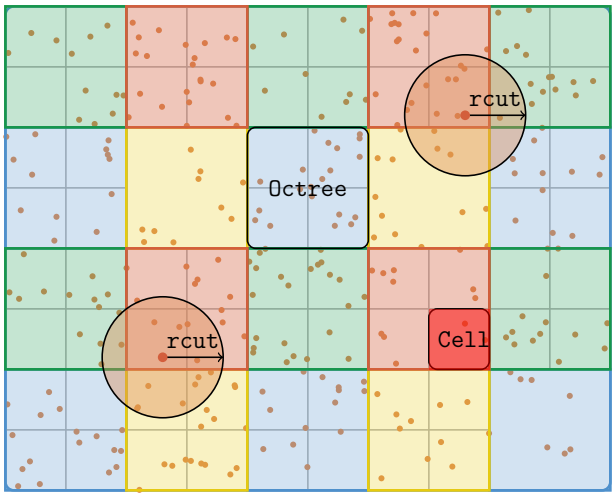


Figure 7.12 | Les octrees composant un sous domaine de simulation sont classés selon 4 vagues: bleu, jaune, vert et rouge. Si deux atomes sont dans deux octrees d'une même catégorie, ces deux atomes ne peuvent interagir entre eux. Le nombre minimal de vagues étant de 2^N, N étant la dimension.

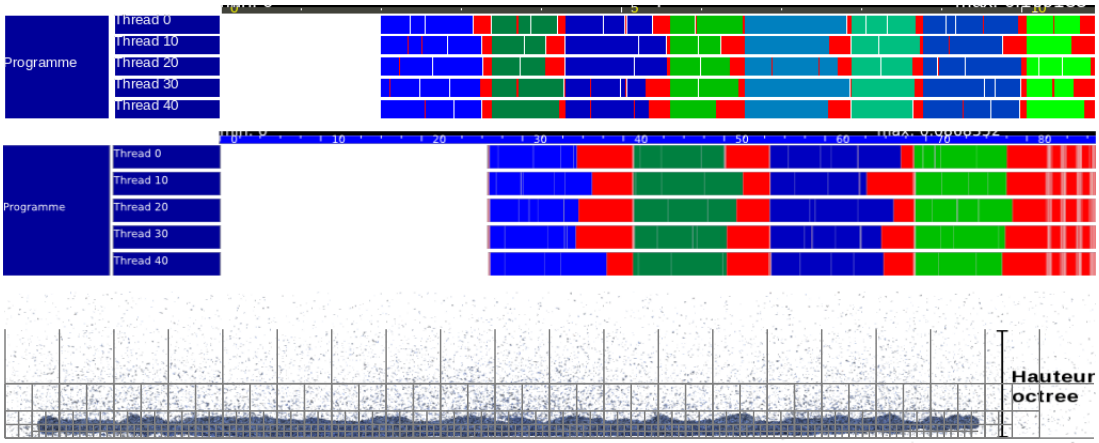


Figure 7.13 | Trace Vite sur Skl de la méthode par vagues pour la simulation micro-jet (haut) avec $D^{max} = 2$ et la simulation impact (milieu) avec $D^{max} = 4$. L'image en bas représente le scénario d'impact d'un nano-goutte selon les axes x et z.

les deux scénarios hétérogènes sont obtenus par le logiciel VITE [34], voir figure 7.13. Les traces correspondent à la 100 ième itération sur Skl. Elles n'incluent pas la reconstruction des listes de voisins. Une couleur correspond à une vague et le rouge représente les phases d'inactivités des threads. Pour plus de lisibilité seuls les threads 0, 10, 20, 30 et 40 sont répertoriés car nous avons remarqué que les autres threads suivaient la même tendance.

Dans les deux cas, on observe distinctement que les barrières créent de longues périodes d'inactivité des threads car aucun thread ne peut commencer un octrees de la vague N+1 tant que tous les octrees de la vague N n'ont pas été traités. Ce phénomène est négligeable pour les scénarios homogènes. Cependant l'inactivité des threads est d'autant plus longue que la répartition des atomes dans le domaines est déséquilibrée. Le temps d'inactivité des threads pour la simulation Impact représente alors 33.9% du temps d'exécution total des threads.

De plus, on remarque pour la simulation Impact que les threads donnent l'impression d'attendre alors que tous les octrees sont déjà traités. En réalité, la hauteur des octrees dépasse l'épaisseur de la goutte aplatie, voir la dernière partie de la figure 7.13, et donc la moitié des vagues traitent

très peu d'atomes. C'est pourquoi la charge de calcul par vague est déséquilibrée. Visuellement, on observe que les threads attendent sans raison apparente, mais en zoomant on s'aperçoit qu'un grand nombre d'octrees correspondant au nuage d'atomes très peu dense au dessus de la goutte sont traités et n'ont presque pas de charge de calcul.

Choix d'une parallélisation à base de tâches : utilisation d'un graphe de dépendances tâches.

Nous avons observé que les barrières entre les vagues entraînent l'inactivité des threads. Cependant pour un octree de la vague N, si tous les octrees adjacents des vagues inférieures ont été traités, il ne sera en concurrence avec aucun autre octree de la vague N ou de n'importe quel octree d'une vague inférieure. Afin de débloquer les octrees le plus rapidement possible, une parallélisation à base de tâches est nécessaire, le traitement d'un octree est alors une tâche et des dépendances entre les octrees sont alors ajoutées. Un Graphe orienté acyclique (Dag), nommé graphe de dépendances, est utilisé. Les sommets du graphe sont les tâches et les arcs orientés sont des dépendances. La figure 7.14 illustre le graphe de dépendances en $2D$.

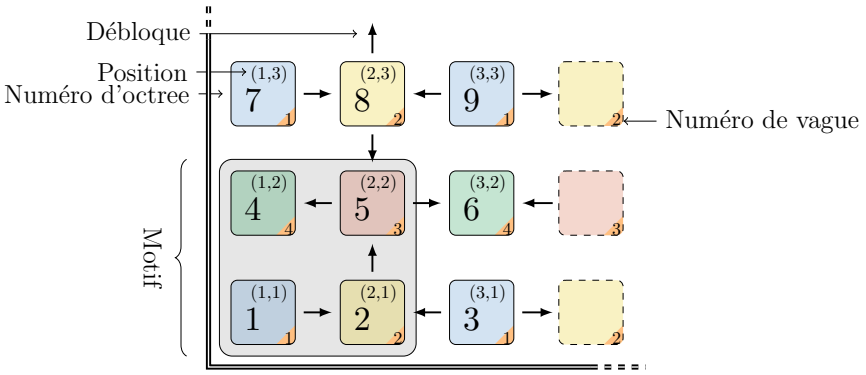


Figure 7.14 | Graphe de dépendances entre les octrees. Les vagues sont ordonnées dans l'ordre bleu, jaune, rouge puis vert. Les octrees de la vague bleu (1,3) débloquent les octrees de la vague jaune (2) etc. Le motifs se répètent sur toute la grille.

Les vagues sont ordonnées de 0 à 7. Les octrees de la vague 0 n'ont aucune dépendance d'entrée, ils correspondent aux sources du Dag (degré entrant égal à 0). Chaque octree de la vague 1 possède alors dans son voisinage deux octrees adjacents appartenant à la vague 0. Deux dépendances d'entrée sont alors créées. Chaque octree de la vague 2 possède deux octrees adjacents de la vague 1 et quatre de la vague 0. Néanmoins par transitivité, seules les dépendances avec la vague 1 sont conservées car elles n'auraient pas pu être débloquées si les octrees de la vague 0 n'avaient pas été préalablement exécutés. Finalement, les vagues de 1 à 7 ont par transitivité deux dépendances d'entrée et les octrees de la vagues 7 sont les puits du Dag (degré sortant égal à 0).

Les traces pour les simulations micro-jet et Impact sont générées de nouveau avec la méthode par vagues incluant un graphe de dépendances par le logiciel VITE, voir figure 7.15. Tout d'abord on observe que les vagues se chevauchent et que le taux d'inactivité des threads passe de 33.9% à 22.6% pour la simulation effectuant l'impact d'une nano-goutte. La durée de la simulation est alors réduit de 6.24%. De plus, on remarque que le thread 0 est le thread maître et qu'il distribue les tâches sur les autres. À noter que pour $D^{max} = 3$ le taux d'inactivité des threads est passé de 59.7% à 49.6% soit environ le même gain que pour $D^{max} = 4$.

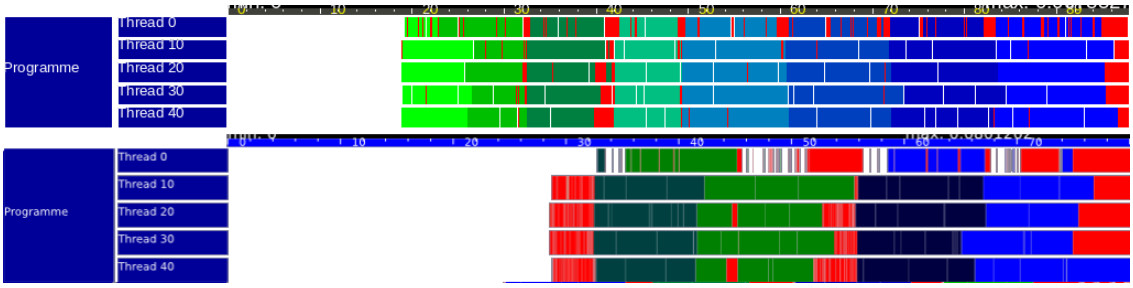


Figure 7.15 | Trace Vite sur Skl de la méthode par vagues pour le scénario microjetting (haut) avec $D^{max} = 2$ et le scénario impact de nanogoutte (bas) avec $D^{max} = 4$.

Étude de la méthode par vagues

Comme pour les stratégies de parcours *boucle feuilles* et *boucle octrees*, la méthode par vagues est étudiée sur les mêmes cas tests.

Simulations "homogènes et statiques"

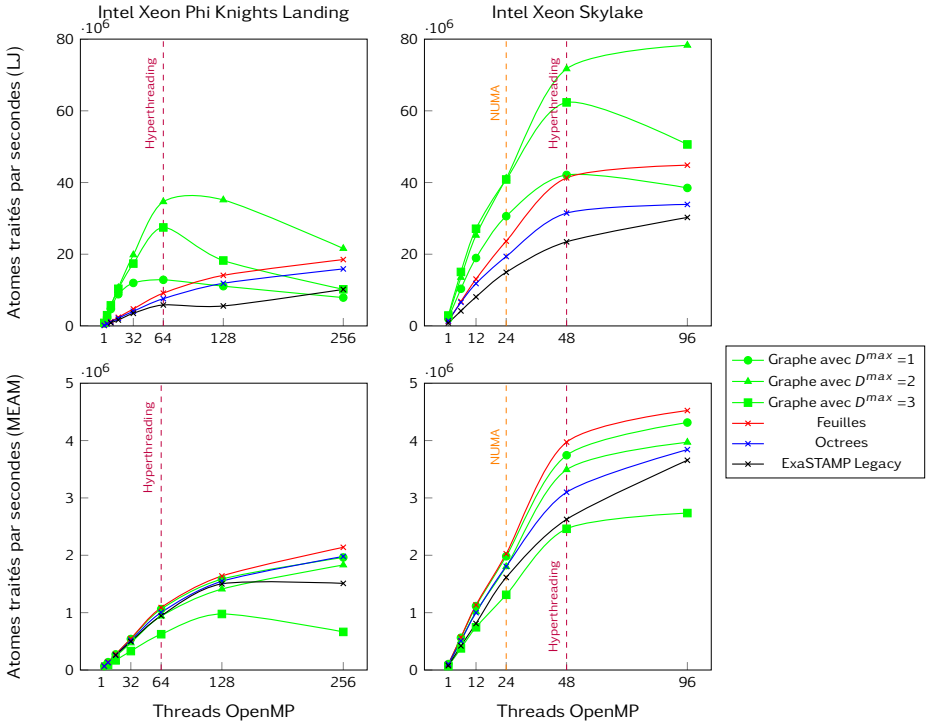


Figure 7.16 | Nombre d'atomes traités par seconde sur des simulations homogènes et statiques avec un potentiel Lj et Meam pour les architecture Intel Xeon Phi Knights Landing et Intel Xeon Skylake. La grille Amr utilise les stratégies *boucle octrees*, *boucle feuilles* et le graphe de dépendances avec $D^{max} = 1, 2, 3$.

Les résultats obtenus avec la méthode par vagues sont rapportés par la figure 7.16 pour les simulations homogènes décrites dans le paragraphe 7.2.2. Contrairement aux autres méthodes "itératives", la méthode par vagues avec tâches semble ne pas tirer partie de l'hyperthreading sur les deux architectures cibles. Pour le potentiel Lj la suppression des mutex apporte un gain non négligeable car ExaSTAMP Amr traite jusqu'à 3.45 fois plus d'atomes par seconde qu'*ExaSTAMP legacy* sur Knl et 2.58 sur Skl. Néanmoins, les gains pour le potentiel Meam ne supplantent pas

la stratégie *boucle feuilles* mais sont équivalents. Comme pour les stratégies "itératives", nous observons que le paramètre D^{max} impacte fortement les temps d'exécution des simulations.

De plus pour le potentiel LJ, on remarque que les résultats a 32 (KNL) ou 24 (SKL) threads sont égaux pour $D^{max} = 2$ et 3 alors que ceux-ci divergent respectivement pour 64 et 48 threads. La raison est que le nombre d'octrees pour cette simulation avec $D^{max} = 3$ n'est plus suffisant pour alimenter toutes les vagues. Cela explique pourquoi l'hyperthreading n'apporte pas de gain. On observe aussi très légèrement ce phénomène pour le potentiel MEAM.

Simulations "hétérogènes et dynamiques"

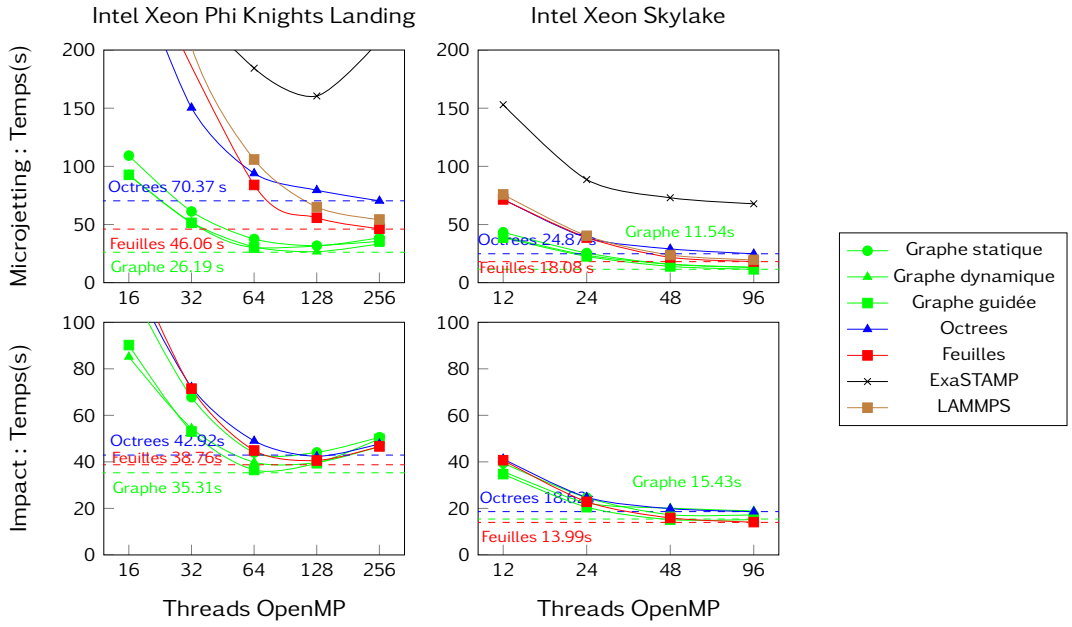


Figure 7.17 | Temps d'exécution d'EXASTAMP AMR pour la simulation micro-jet et d'impact d'une nano-goutte pour les méthodes de parallélisation OPENMP par graphe, octrees et feuilles. Les politiques d'ordonnancement utilisées sont statique, dynamique et guidée. Les choix optimaux du paramètre D^{max} sont respectivement de 2 et 4 pour le micro-jet et l'impact d'une nano-goutte. Les temps de simulations de LAMMPS et *ExaSTAMP legacy* sont ajoutés uniquement pour la simulation scénario micro-jet. Le potentiel utilisé pour modéliser les interactions étain-étain est le potentiel LJ.

En limitant l'impact des mutex, la méthode par vagues réduit le temps de calcul de l'opérateur du potentiel, tout en conservant une bonne scalabilité dans un cas équilibré. La figure 7.17 fournit les temps d'exécution des deux scénarios déséquilibrés pour les trois stratégies avec le potentiel LJ.

Pour la simulation micro-jet, la méthode par vagues est 1.64 fois plus rapide que LAMMPS sur KNL et 1.71 sur SKL. Puisque la méthode par vagues est basée sur une parallélisation par tâche, les politiques d'attribution des octrees n'influencent pas l'opérateur calculant l'énergie potentielle. Seul l'opérateur construisant des voisins est influencé et, pour cette simulation, cet opérateur correspond à environ $15 - 20\%$ du temps total de la simulation.

L'apport de la méthode par vagues est moins probant pour le scénario d'impact d'une nano-goutte. En effet, bien qu'un léger gain de 10.0% est observable sur KNL, la méthode par vagues est 10.3% plus lente que la stratégie *boucle feuilles* sur SKL. Ce cas test permet de mettre en avant l'un des défauts d'une parallélisation à base de tâches. En effet, il a été observé que la trace était polluée par des micro-tâches engendrant des surcoûts liés à la création de tâches pour peu de calculs. Ces micro-tâches ajoutent des dépendances au graphe de dépendances et elles augmentent l'inactivité des threads de plus de 20% sur SKL.

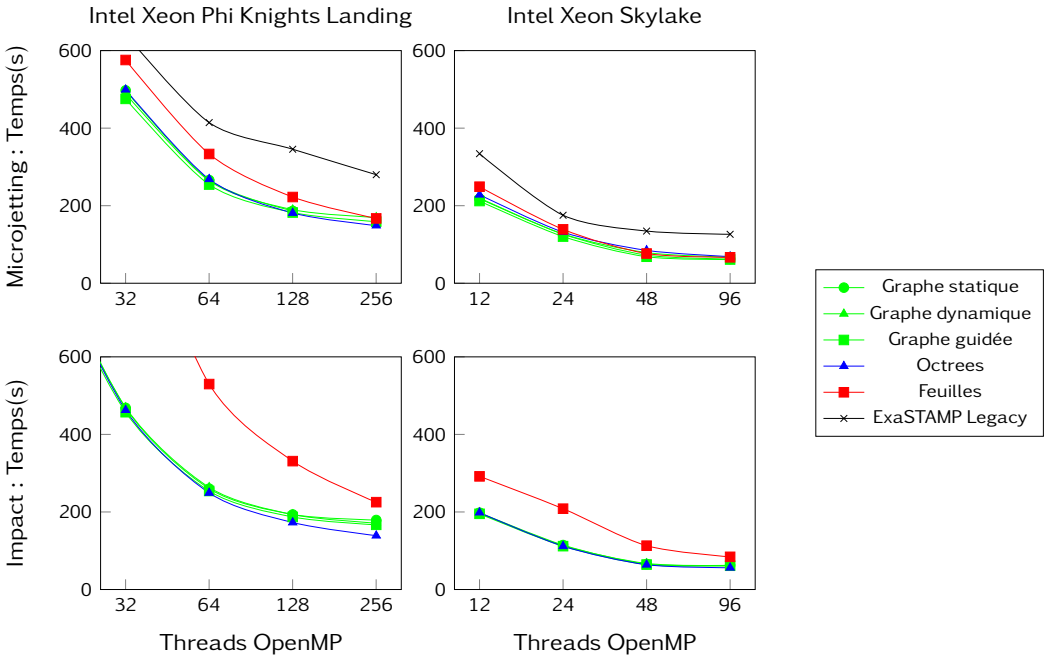


Figure 7.18 | Temps d'exécution d'ExaSTAMP Amr des simulations micro-jet et impact d'une nano-goutte pour les méthodes de parallélisation OpenMP par graphe, octrees et feuilles. Les politiques d'ordonnancement utilisées sont statique, dynamique et guidée. Les choix de D^{max} optimaux sont respectivement de 2 et 4 pour le micro-jet et l'impact d'une nano-goutte. Les temps de simulation de Lammps et *ExaSTAMP legacy* sont ajoutés uniquement pour la simulation micro-jet. Le potentiel utilisé pour modéliser les interactions étain-étain est le potentiel Meam.

En se rapportant à la figure 7.18 correspondant aux temps de simulation avec un potentiel Meam, la méthode par vagues a, comme pour le cas homogène, peu d'impact sur les temps de simulation lorsque le potentiel a une intensité arithmétique importante. Néanmoins, la méthode par vagues est très proche du parcours *boucle octrees*. Donc, bien qu'elle ne soit pas optimale, la méthode par vagues offre de bon de résultats pour ces deux simulations quel que soit le potentiel utilisé.

En conclusion, nous avons observé que la méthode par vagues basée sur un Dag offre des résultats supérieurs pour des potentiels peu coûteux sur des simulations homogènes et hétérogènes. Cette méthode n'est pas la plus intéressante pour des potentiels plus coûteux, cependant elle est généralement proche de l'optimal alors que la stratégie optimale varie d'une simulation à l'autre. De plus, les simulations considérées sont composées d'un seul type d'atomes, lorsque plusieurs types de potentiels avec différents coûts sont utilisés, on peut formuler l'hypothèse que la méthode par vagues est le meilleur compromis.

Conclusions

Dans ce chapitre, 3 stratégies de parallélisation en mémoire partagée ont été développées afin de répartir au mieux la charge de calcul entre les threads. Celles-ci portent sur la portion de code dédiée au calcul des interactions. Les deux premières parallélisations sont basées sur des parcours différents des cellules : *boucle feuilles* et *boucle octrees*. La dernière stratégie est basée sur une parallélisation par tâches et un graphe de dépendances (Dag).

Les stratégies ont été évaluées selon plusieurs simulations homogènes et hétérogènes, statiques et dynamiques, avec un potentiel peu coûteux : Lj et un potentiel coûteux : Meam. Tout d'abord nous avons observé que les stratégies conservent l'apport des optimisations séquentielles. Le paramètre

D^{max} influence les performances des trois stratégies. Dans un cas test non favorable à la grille AMR (cas homogène), celle-ci obtient quand même de bons résultats.

Pour un potentiel peu coûteux tel que le potentiel LJ, on observe que la méthode par vagues propose dans tous les cas étudiés la meilleure alternative en raison de la suppression des mutex. Néanmoins, celle-ci nécessite un grand nombre d'octrees par vague. Dans le cas d'un potentiel plus coûteux tel que le potentiel MEAM, la meilleure stratégie de parallélisation intra-NŒUD n'est pas toujours la même, pour des simulations dynamiques la méthode à privilégier est *boucle octrees* contrairement aux simulations homogènes pour lesquels la stratégie *boucle feuilles* est à favoriser.

La prochaine étape consiste à concevoir le dernier niveau de parallélisme afin de distribuer les calculs entre les NŒUDS d'un supercalculateur. Cette étape est décrite et évaluée dans le chapitre 8.

8 Élaboration d'une parallélisation inter-Nœuds basée sur le partitionnement du domaine

La parallélisation intra-Nœud en OpenMP pour la grille Amr a été élaborée dans le chapitre 7. L'objectif de ce chapitre est de paralléliser efficacement les calculs entre les Nœuds, nommée parallélisation inter-Nœuds, d'un supercalculateur. La méthode proposée est basée sur le partitionnement spatial du domaine.

Dans ce cadre, nous allons étudier le système de gestion des messages Mpi actuellement disponible dans *ExaSTAMP legacy*. Ce système est basé sur les développements effectués par E. Cieren [29]. Nous ferons évoluer le système afin qu'il soit plus performant et adapté aux grilles Amr. Pour cela nous commençons par étudier dans le paragraphe 8.1 les limites actuelles d'*ExaSTAMP legacy* lors de la mise à jour des zones fantômes et nous proposons une optimisation de la mise à jour des zones fantômes entre deux grilles Amr (cf. paragraphe 8.2).

Ensuite, après avoir introduit dans le paragraphe 8.3 les méthodes de partitionnement dynamique de domaine pour les grilles Amr, nous étudierons l'impact des méthodes incluses dans la librairie Zoltan afin de déduire laquelle est la plus intéressante. Finalement dans ce chapitre nous allons mettre en place la dernière brique nécessaire pour effectuer dans le chapitre 9 la simulation d'un micro-jet d'étain ou l'impact d'une nano-goutte d'étain sur une surface solide sur des centaines de Nœuds.

Limites actuelles des communications Mpi dans ExaSTAMP Legacy

Dans *ExaSTAMP legacy*, la propagation des informations entre les sous-domaines est assurée par un modèle d'échange de données via la bibliothèque Mpi, cf. paragraphe 4.2.1. Chaque sous-domaine est attribué à un processus Mpi. Les sous-domaines ont des identifiants uniques correspondant à leurs rangs Mpi. Ainsi la propagation d'une information telle que le nombre d'atomes, l'énergie potentielle du système ou la température est assurée soit par des communications collectives, soit par des communications points à points.

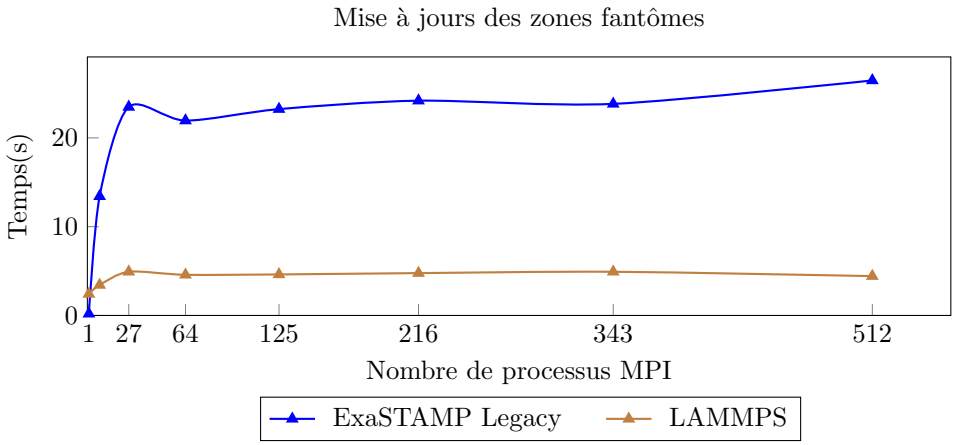
Rappelons que la méthode de partitionnement spatial de domaine consiste à décomposer le domaine de la simulation en sous-domaines disjoints. Les sous-domaines sont alors entourés d'une zone fantôme afin de reproduire le voisinage des atomes sur les bords des sous-domaines pour le calcul des interactions. Dans *ExaSTAMP legacy*, la zone fantôme d'un sous-domaine est une couche de cellules. Cette notion de zone fantôme est assez bien connue et on la retrouve dans de nombreux autres types de simulation comme par exemple dans le code d'hydrodynamique Enzo.

ExaSTAMP legacy propose un système nommé *messageCenter* traitant le transfert des atomes entre les sous-domaines [29]. Au cours de la simulation, la transmission des informations des atomes intervient lorsqu'un atome change de sous-domaine, lorsque les informations sur les zones fantômes sont mises à jour ou lors du déclenchement d'une répartition de la charge.

Le développeur fournit au *messageCenter* un vecteur de la librairie standard composé de Tuples contenant les informations des atomes. L'utilisation d'une structure en tuple permet de sélectionner les informations par atome comme les positions ou le type. Ensuite les tuples sont sérialisées dans les tampons d'envoi du *messageCenter*. Les communications sont alors assurées par des communications point à point afin d'éviter que deux sous-domaines non adjacents ne communiquent des messages vides.

La mise en œuvre du transfert des atomes dans les zones fantômes est partiellement conservée avec la méthode de l'Amr. Bien que l'utilisation de tuples soit flexible car elle permet de gérer n'importe quelle type de donnée, elle ne tire pas profit de la structure de tableaux utilisés pour stocker les atomes dans les cellules (*ExaSTAMP legacy*) ou les octrees (ExaSTAMP Amr). En effet, les informations des atomes (positions, identifiant et type) sont copiés dans des tuples (std::tuple<double,double,double,uint64_t,uint8_t>) puis les tuples sont recopiés dans des tampons. Ensuite, une fois que les sous-domaines reçoivent les tableaux de tuples dans les tampons de réception, ils sont recopiés un à un dans les structures de tableaux des cellules. C'est pourquoi cette méthode ne profite absolument pas du fait que les informations soient contiguës en mémoire et qu'ils peuvent être copiées dans des tampons par paquets.

Des tests préliminaires allant jusqu'à 512 Knl ont été menés avec *ExaSTAMP legacy* et le code Lammps. Les tests ont montré que le système basé sur le *messageCenter* d'*ExaSTAMP legacy* est au moins 5 fois plus lent que Lammps pour effectuer la mise à jour des zones fantômes, cf. le graphe 8.1.



Graphe 8.1 | Temps d'exécution de la mise à jour des zones fantômes pour les codes *ExaSTAMP legacy* et Lammps. Pour cela on simule un cristal parfait de cuivre de taille variable mais dont chaque processus Mpi possèdent 4 millions d'atomes de cuivre. Chaque processus Mpi opère sur un Knl de 64 cœurs avec 256 threads OpenMP.

Dans la partie suivante nous proposons une optimisation de la mise à jour des zones fantômes en transférant les informations contiguës en mémoire directement dans les tampons.

Mise à jour des zones fantômes pour des grilles AMR

Dans ce paragraphe nous étudions différentes optimisations possibles de la mise à jour des zones fantômes pour des configurations simples dont le nombre d'atomes échangés est facilement pilotable,

c'est-à-dire des simulations homogènes et statiques. L'objectif est d'étudier le comportement du système. Le domaine de simulation est alors décomposé en sous-domaines de même volume et dont la taille de la surface d'échange est la même. Ces sous-domaines possèdent chacun une grille rectilinéaire, cf. paragraphe 5.1.2.

Une première idée est d'éviter d'aligner les données dans des tampons intermédiaires comme *ExaSTAMP legacy* car le cas test a montré que le coût est principalement localisé dans le remplissage des tampons d'*ExaSTAMP legacy*. Pour cela les tableaux d'informations des atomes stockés dans les octrees sur le bord de chaque sous-domaine sont directement transférés dans les octrees des zones fantômes. Les communications entre les sous-domaines sont alors assurées par des communications point à point via des messages Mpi asynchrones. Le nombre de messages Mpi est alors d'environ 5 fois le nombre de cellules feuilles sur le bord des sous-domaines et les messages contiennent très peu d'éléments.

De plus, comme les atomes sont triés en fonction de leurs coordonnées cartésiennes via un index de Morton, il est possible de réduire le nombre de messages Mpi en copiant les données de plusieurs cellules feuilles à la fois lorsque les index de Morton des cellules feuilles sont consécutifs. Après des expérimentations préliminaires, cette idée a été mise de côté car le gain par rapport à *ExaSTAMP legacy* était faible et bien en deçà des performances de Lammps à cause du trop grand nombre de messages Mpi.

Une deuxième idée est d'utiliser des tampons d'envoi et de réception comme *ExaSTAMP legacy* mais en copiant les portions contiguës contenant les informations des atomes. Cette idée d'optimisation est développée dans le paragraphe suivant, son implémentation est détaillée dans le paragraphe 8.2.2 et elle est validée dans le paragraphe 8.2.4.

Optimisation de la mise à jour des zones fantômes

Lors de la conception des octrees, nous avons déjà mis en avant que pour chaque octree, les atomes sont triés dans une structure de tableaux. L'ordre des éléments des tableaux est perturbé (permutation, insertion, suppression) seulement lorsque les listes de Verlet sont mises à jour. Les tableaux sont alors triés de nouveau.

Notre optimisation de la mise à jour des zones fantômes est réalisée en deux étapes. Elle tient compte de deux points importants : la fréquence de mise à jour des listes de Verlet et l'alignement des données à transférer.

Les atomes appartenant à une cellule feuille sont délimités par deux repères (début et fin) sur la structure de tableaux inclus dans l'octree. Tant que les listes de Verlet ne sont pas mises à jour, les deux repères ne sont pas modifiés. Lors de la première étape, on peut alors définir, pour un certain nombre d'itérations, quelles sont les portions d'atomes à transférer entre les sous-domaines. Sachant que chaque sous-domaine connaît sa connectivité avec ses sous-domaines adjacents, la première étape consiste donc à diffuser, par des communications points à points, le nombre d'atomes à envoyer / réceptionner par cellule feuille. Si une cellule feuille est sur le bord mais elle ne contient pas d'atome, aucune information n'est envoyée. Par défaut, le sous-domaine de réception sait qu'il ne recevra pas d'information de cette cellule feuille.

La deuxième étape consiste à regrouper les données dans des tampons d'envoi, envoyer les données et les extraire des tampons de réception. Cette étape est représentée entre un sous-domaine A et un sous-domaine B par le schéma 8.2. L'algorithme de transfert entre deux sous-domaines est effectué en 4 phases :

- Phase 1 : sélection des cellules feuilles à envoyer;

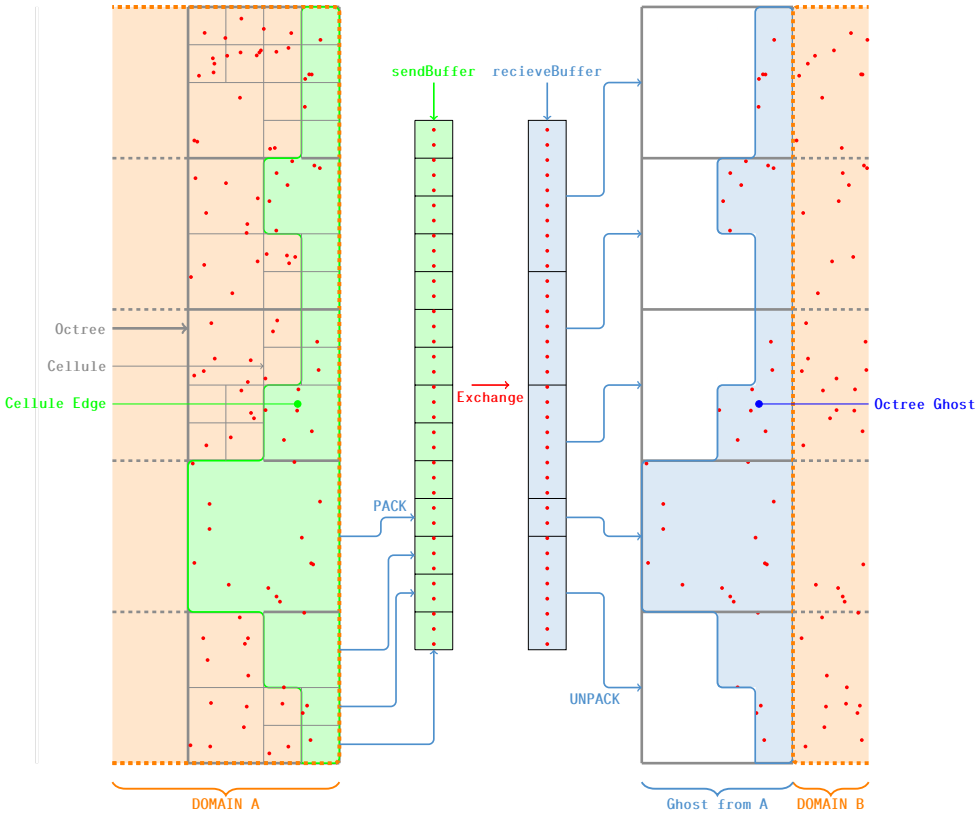


Schéma 8.2 | Mise à jour d'une partie des octrees fantômes du domaine B par des atomes provenant du domaine A. Afin de minimiser le transfert de particules, uniquement les données des cellules feuilles sur le bord du domaine A sont sélectionnées (en vert).

- Phase 2 : copie (std::copy) des tableaux d'atomes dans un tampon d'envoi;
- Phase 3 : envois des données du tampon d'envoi dans le tampon de réception;
- Phase 4 : copie (std::copy) des données dans les octrees fantômes, à noter que les octrees dans les zones fantômes ne sont pas raffinés.

L'étape 1 permet de préparer plusieurs points de l'étape 2. Elle permet notamment de présélectionner les cellules feuilles à envoyer pour chaque sous-domaine. Les tampons d'envoi / réception ont été préalablement alloués en fonction du nombre d'atomes à envoyer / recevoir. Lors de la copie, les tableaux à copier ont une position déjà prédéterminée dans les tampons d'envoi. De même que la copie des données du tampon de récéption dans les octrees fantômes qui est aussi effectuée par blocs de données déjà prédéterminés. Les copies des blocs mémoires contigus sont réalisées en parallèle avec une parallélisation OpenMP. Pour la copie, aucun mutex n'est alors nécessaire car il n'y a aucun chevauchement entre les données à copier.

Cette optimisation de la mise à jour des informations des zones fantômes repose sur deux étapes cruciales. La première étape permet de préparer la réplication des atomes. Elle est effectuée à chaque fois que les listes de Verlet sont mises à jour. La deuxième étape consiste à répliquer à chaque itération les atomes sur les sous-domaines voisins. Dans le paragraphe suivant nous détaillons comment cette optimisation est mise en œuvre dans ExaSTAMP Amr.

Mise en place de l'optimisation de la mises à jour des zones fantômes.

La mise en place du nouveau système de la mise à jour des zones fantômes intervient principalement au niveau de la grille Amr. Les squellettes des fonctions effectuant la mise des zones fantômes sont alors inchangés par rapport à *ExaSTAMP legacy*.

Lors de la mise à jour des zones fantômes, et compte tenu des simulations de dynamique moléculaire ciblées et des potentiels utilisés, uniquement les informations concernant les positions cartésiennes, l'identifiant et le type des atomes sont à mettre à jour. Afin de tenir compte d'éventuelles informations supplémentaires, le système de transfert est basé sur des *variadic templates* en C++.

Après avoir fourni au système les types des éléments à envoyer, le développeur doit fournir les pointeurs sur les tableaux à copier dans le tampon et le nombre d'éléments par tableau. Pour cela, on définit une structure notée *infoLeaf* pour chaque cellule feuille au bord du sous domaine. Cette structure contient la position cartésienne de l'octree propriétaire de la cellule sur la grille (3 entiers), la position du premier atome (1 entier) et le nombre d'atomes dans la cellule feuille (1 entier). À partir de ces informations un dernier entier est calculé, il correspond à la position dans le tampon d'envoi où les tableaux sont copiés en parallèle.

Une fois ces informations transmises aux sous-domaines voisins, on alloue les tampons de réception et d'envoi utilisés pour envoyer les données des atomes par messages Mpi. À noter que l'identification de octrees fantômes recevant telle ou telle portion mémoire du tampon de réception est effectuée à partir des coordonnées cartésiennes de l'octree contenues dans *infoLeaf*.

Les tampons sont des tableaux de *char* (1 Bytes) permettant de moduler la taille des tampons pour n'importe quels types d'informations. Chaque tampon est alors dédié à un message Mpi. Les tampons sont alloués par un identifiant noté *msgId*. Le listing 7.1 décrit la phase de copie d'un octree dans un tampon. La fonction packCopy est alors appelée par la fonction pack. Les entiers shift, octreeShiftBegin et octreeShiftEnd correspondent respectivement à la position où les données sont copiées dans le tampon, le premier et le dernier élément des tableaux à copier.

```
1  struct pass { template<typename ...T> pass(T...) {} };
2
3  template <class... T>
4  void packCopy(const char* p_buffer, size_t shift, size_t begin, size_t end ,int
       nbElemPerMsg, T*... p_send)
5  {
6      size_t offset = 0;
7      pass{(
8          std::copy( p_send + begin, p_send + end, (T*) (&p_buffer[shift*sizeof(T) +
               offset]) ),
9          offset += nbElemPerMsg*sizeof(T)
10         ,1)...};
11 };
12 ...
13 ghostComm.pack(msgId, shift, octreeShiftBegin, octreeShiftEnd,
14   ptr_octree->rx.data(), // Tableaux à copier
15   ptr_octree->ry.data(),
16   ptr_octree->rz.data(),
17   ptr_octree->id.data(),
18   ptr_octree->ti.data());
```

Listing 8.1 | Portion du code effectuant la copie des tableaux dans un tampon.

L'extraction des informations du tampon de réception, voir le listing 7.2, est effectuée similairement à la phase de sérialisation des informations. Cette phase est déclenchée une fois que tous les messages Mpi sont terminés. Les informations sont alors directement copiées dans les octrees fantômes. Dans la grille Amr, les octrees fantômes (*ptr_octrees_ghost*) sont remplis en parallèle.

```
{
template <class... T>
void unpackCopy(const char* p_buffer, size_t shift, size_t size, int nbElemPerMsg, T
    *... p_recv)
{
    assert( nbElemPerMsg >= size );
    size_t offset = 0;
    pass{(
        std::copy( (const T*)(&p_buffer[shift*sizeof(T) + offset]), (const T*)(&
            p_buffer[(shift+size)*sizeof(T) + offset]), p_recv) ,
        offset += nbElemPerMsg*sizeof(T)
        ,1)...};
};
...
ghostComm.unpack( msgId ,octreeShift, nbElemOctree,
  ptr_ghost_octree->rx.data(), // Tableaux à remplir
  ptr_ghost_octree->ry.data(),
  ptr_ghost_octree->rz.data(),
  ptr_ghost_octree->id.data(),
  ptr_ghost_octree->ti.data());
};
```

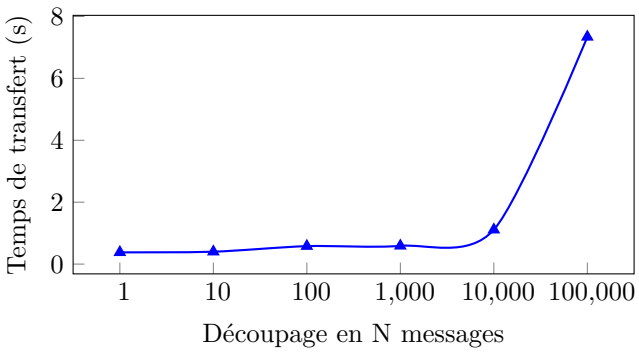
Listing 8.2 | Portion du code effectuant l'extraction des informations du tampon.

Les performances de cette stratégie sont évaluées dans le paragraphe 8.2.4. Cependant, lors des premières expérimentations de validation sur des simulations composées de milliards d'atomes avec des centaines de processus Mpi, la taille mémoire des messages Mpi posait problème et ne pouvait être gérée par le réseau d'interconnexion. La solution choisie a été de découper les messages Mpi.

Étude du découpage des messages Mpi.

La découpe des messages Mpi permet d'éviter que la taille des messages Mpi ne soit trop importante. Cependant cette solution dégrade-t-elle les performances ? En combien de messages un message Mpi doit-il être découpé ? Afin d'étudier jusqu'à quel point le découpage des messages Mpi peut impacter le temps des communications Mpi, nous allons essayer différents découpages. Pour cela un cristal parfait de cuivre de 4 millions d'atomes est simulé sur 100 pas de temps. Cette simulation est réalisée sur un Skl avec deux processus Mpi et 24 threads OpenMP par processus Mpi. Chaque processus Mpi traite 2 millions d'atomes. Les deux sous-domaines s'échangent 482 944 atomes à chaque itération.

Le graphe 8.3 décrit le temps des communications en fonction du nombre de messages Mpi. Ainsi 1 message contient 482 944 atomes alors que 96 588 messages contiennent 5 atomes chacun (ou presque). On observe que la courbe est strictement croissante, donc plus il y a de messages Mpi plus le temps de transfert est long. Néanmoins pour cette simulation, le temps des communications Mpi croît significativement qu'à partir d'un découpage en 10 000 messages Mpi, soit à peine 50 atomes par message. Or le but de ce découpage était de pallier la trop grande taille des messages (dizaine de millions d'atomes par message) en répartissant les données à envoyer dans une vingtaine



Graphe 8.3 | Temps de transfert des atomes dans les zones fantômes entre deux sous-domaines en fonction du nombre de messages Mpi.

de messages. Ce qui est bien inférieur à 10 000. C'est pourquoi il est raisonnable d'affirmer que cette partie ne dégradera pas le temps des communications.

Validation et comparaisons avec les codes Lammps et ExaSTAMP legacy

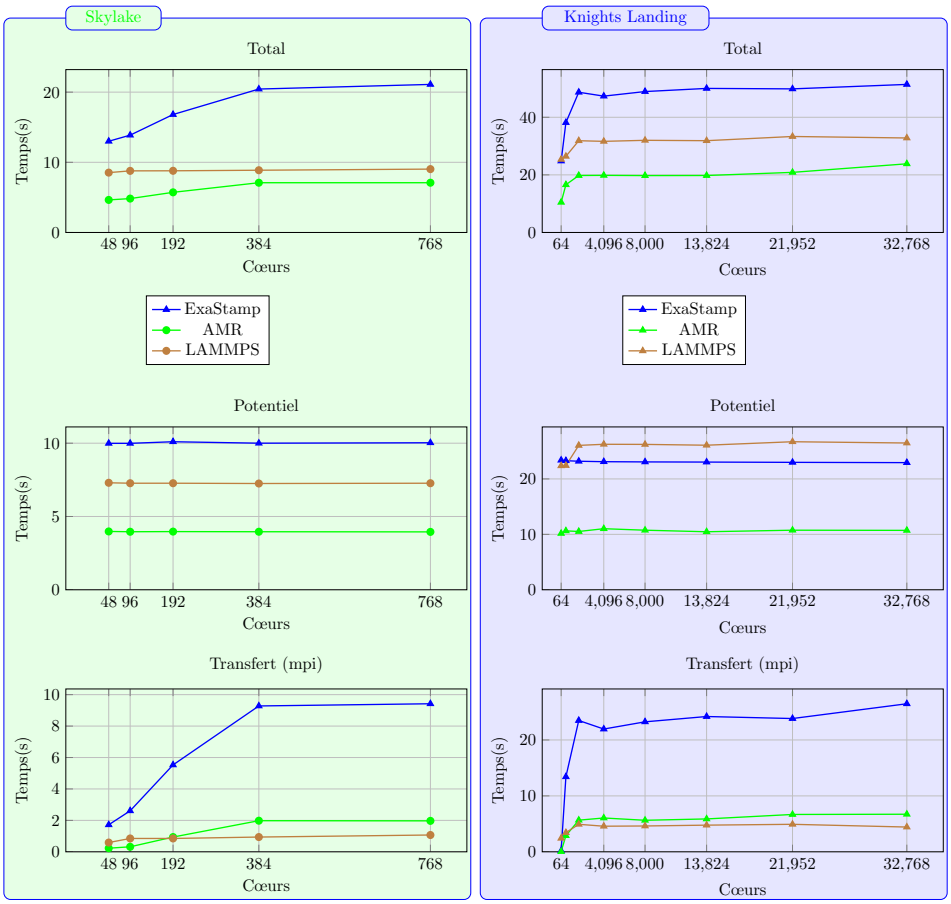
Dans ce paragraphe nous comparons notre implémentation de la mise à jour des zones fantômes avec les codes *ExaSTAMP legacy* et Lammps sur des cas homogènes et statiques. Pour cela, un cristal parfait de 4 millions d'atomes de cuivre est attribué à chaque sous-domaine. Le principe est d'affecter un sous-domaine par Knl (64 cœurs) ou un sous-domaine par socket de chaque Skl (24 cœurs/socket). C'est pourquoi lorsque l'on augmente le nombre de sous-domaines, on augmente le nombre d'atomes simulés. Cela permet d'étudier le comportement des communications alors que la charge de calcul par sous-domaine est constante et ainsi de valider que les communications Mpi n'augmentent pas en fonction du nombre de processus Mpi.

Les résultats des simulations sont obtenus entre 1 à 512 Knl et 1 à 20 Skl. Ils sont respectivement représentés dans la partie bleue et verte des graphes 8.4. Les simulations sont réalisées avec un potentiel Lj. La simulation est alors composée de 2 milliards et 48 millions d'atomes sur 512 Knl et tous les codes utilisent le maximum de threads OpenMP disponibles. Les temps concernant la portion du code traitant le calcul des interactions (potentiel) et la mise à jour des zones fantômes (transfert) sont dissociés du temps total de la simulation.

On peut tout d'abord observer que pour tous les codes, le temps total de la simulation augmente principalement au début puis se stabilise au bout de 8 Nœuds. Cette augmentation s'explique par l'augmentation du temps de transfert alors que le temps de calcul des interactions reste quasi inchangé pour tous les codes. Ce phénomène est dû à l'apparition de nouvelles communications Mpi. À partit du moment où chaque sous-domaine est entouré de 26 sous-domaines, comme c'est le cas à partir de 8 Nœuds avec des conditions aux limites périodiques, le temps de transfert se stabilise pour tous les codes.

Concernant les temps de transfert, comme annoncé dans le paragraphe 8.1, le code *ExaSTAMP legacy* est respectivement 8.8 et 5.9 fois plus lent que Lammps sur Skl et Knl. Cet écart est respectivement réduit à 1.9 et 1.5 pour ExaSTAMP avec la grille Amr.

Finalement dans cette partie, nous avons optimisé la mise à jour des zones fantômes en copiant en parallèle des informations contiguës en mémoire dans des tampons. Bien que sur ce cas Lammps reste environ 1.51 fois plus rapide que notre stratégie, le coût de l'opérateur calculant l'énergie reste prédominant ce qui explique pourquoi ExaSTAMP Amr est finalement plus rapide que Lammps. De



Graphe 8.4 | Temps de simulation d'un cristal parfait de cuivre en fonction du nombre de cœurs pour différentes portions des codes LAMMPS, EXASTAMP avec et sans AMR. Les codes sont exécutés avec respectivement 48 et 256 threads OPENMP par processus MPI sur SKL et KNL.

plus ce code optimisé est environ 4 fois plus rapide qu'*EXASTAMP legacy* sur ce cas homogène et statique.

Équilibrage dynamique de la charge pour la grille AMR

Dans la partie précédente, une optimisation de la mise à jour des zones fantômes a été proposée pour des simulations homogènes. Dans le cas des simulations dont la densité est non uniforme comme l'évolution d'un micro-jet ou l'impact d'une nano-goutte d'étain sur une surface solide, la charge de calcul est inégalement répartie entre les sous-domaines par la méthode de partitionnement de domaine classique. Dans cette deuxième partie nous proposons d'étendre notre stratégie aux grilles dites any déjà présentes dans *EXASTAMP legacy*, c'est-à-dire que les sous-domaines n'ont pas de forme prédéfinie. L'objectif est d'utiliser les méthodes de partitionnement de domaine décrites dans le chapitre 5 pour équilibrer la charge de calcul. Pour cela, la mise en place de la grille AMR avec une topologie any (sous-domaine de forme quelconque) est discutée dans le paragraphe 8.3.1 ainsi que l'adaptation de la méthode de parallélisation intra-NŒUD basée sur la méthode par vagues et un graphe de dépendances de tâches.

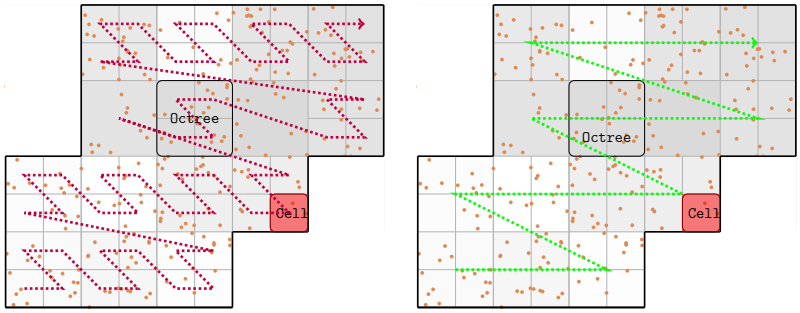
Après avoir introduit les méthodes de partitionnement, une première étude concernant l'impact des stratégies de parallélisation OPENMP en fonction du nombre de processus MPI est réalisée

dans le paragraphe 8.3.4. Finalement, une deuxième étude est menée sur l'impact des différentes méthodes de partitionnement dans les paragraphes 8.3.5 et 8.3.6.

Grille Amr et topologie ANY

La topologie any ou grille any a été introduite dans ExaSTAMP afin d'utiliser les méthodes de partitionnement générant des domaines de formes quelconques. Ce type de grille a été déclinée pour la grille Amr pour les mêmes raisons. La principale différence entre une grille rectilinéaire est qu'il faut vérifier si la cellule à laquelle on accède appartient ou non à la grille.

Pour éviter de vérifier si une cellule appartient à une grille à chaque fois, nous avons montré dans le paragraphe 5.4.1 qu'ExaSTAMP s'appuie sur une structure nommée traversal dans laquelle est stockée un ensemble d'index de cellules. Ce système permet de parcourir un ensemble de cellules comme par exemple les cellules fantômes sans avoir à vérifier si les cellules appartiennent ou non à la grille any. Les traversals sont reconstruits à chaque fois que la répartition de la charge modifie l'ensemble des cellules attribuées à un sous-domaine. Ce système est conservé pour la grille Amr.



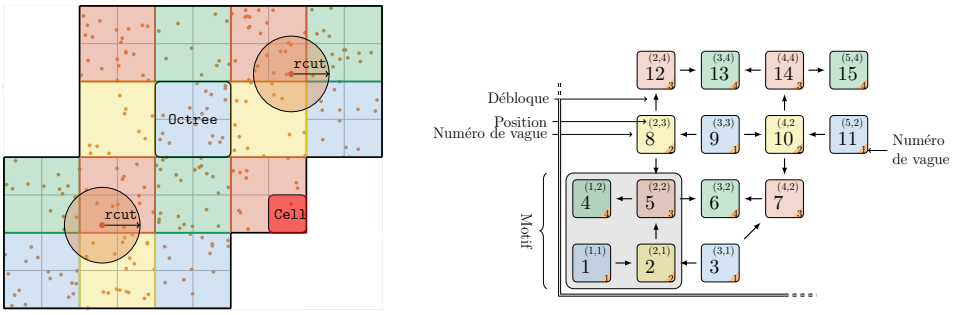
Boucle Feuilles Boucle Octrees

Figure 8.5 | Représentation des stratégies de parallélisation intra-Nœud pour une grille Amr avec une topologie any. La stratégie *boucle octrees* est représentée par la schéma de gauche. Comme pour une grille rectilinéaire, les octrees sont parcourus dans le sens d'un des axes de la simulation. La stratégie *boucle feuilles*, schéma de droite, parcourt les octrees comme pour le stratégie *boucle octrees* puis parcourt les cellules feuilles selon l'index de Morton.

Avec l'utilisation des traversals, les différentes stratégies de parallélisation intra-Nœud (*boucle octrees*,*boucle feuilles* et Graphe) utilisées pour calculer l'énergie potentielle sont préservées. Les stratégies «itératives» sont représentées par la figure 8.5. Pour la stratégie *boucle feuilles*, un traversal parcourant les cellules feuilles est ajouté. Le traversal est mis à jour lorsque les atomes migrent d'une cellule feuille à une autre. Les cellules feuilles ne contenant pas d'atome sont quant à elles exclues du traversal.

Concernant le graphe de dépendances de tâches (Dag) associé à la méthode par vagues, cf. image 8.6, celui-ci doit tenir compte des octrees manquant sur les bords du domaine. Les octrees sont toujours classées selon 8 vagues (4 en 2D) en fonction de leurs coordonnées cartésiennes sur la grille et les vagues sont toujours ordonnées de 1 à 8. Par exemple, dans l'image de droite dans la figure 8.5, les vagues sont traitées dans l'ordre suivant : bleue, jaune, rouge puis finalement verte.

Néanmoins puisqu'il manque des octrees, le motif des dépendances entre les octrees n'est plus juste car le motif a été conçu en supprimant des dépendances par transitivité. Par exemple, si l'on observe l'octree correspondant à la tâche numéro 7 et appartenant à la vague rouge, celui-ci devrait avoir deux dépendances provenant de deux octrees de la vague jaune. Cependant l'octree qui devrait se trouver juste en dessous n'appartient pas au sous-domaine. On ne peut donc pas appliquer le motif.



Méthode par vagues — Graphe de dépendances de tâches

Figure 8.6 | Représentation des stratégies de parallélisation intra-Nœud pour une grille Amr avec une topologie any. Le graphe de dépendances de tâches (schéma de droite) associé à la méthode par vague (schéma de gauche) est adapté afin de prendre en compte les octrees manquant.

Une première solution pour conserver le motif de base (comme pour une grille rectilinéaire) est de rajouter des tâches sans calcul correspondant aux octrees dans la zone fantôme afin d'ajouter des dépendances. Le problème avec cette solution est qu'elle alourdit le graphe de dépendances et dégrade donc l'exécution des tâches.

Une autre solution est d'affecter pour chaque octree des dépendances entre lui-même et tous les octrees appartenant aux vagues inférieures présents dans son voisinage. Toutefois l'utilisation excessive de dépendances dans un graphe de tâches (Dag) n'est pas non plus recommandée.

Finalement, nous avons choisi d'adapter le graphe de dépendances de tâches tout en minimisant le nombre de dépendances entre les tâches. Dans le cas où l'on cherche à ajouter une dépendance entre un octree A et un octree B n'appartenant pas à la grille. Il faut alors déterminer quelles sont les dépendances entre l'octree A et les tâches associés aux octrees débloquant la tâche de l'octree B. On remonte alors le motif des dépendances jusqu'à atteindre les tâches de la vague 0. De plus, il faut vérifier que les nouvelles dépendances sont effectuées entre des octrees adjacents . Par exemple pour le graphe de dépendance de tâches de la figure 8.5 a été adaptée. Une dépendance entre l'octree de la tâche numéro 7 de la vague rouge et l'octree de la tâche 3 de la vague bleue est alors ajoutée.

Afin de tenir compte d'un nombre variable de dépendances pour chaque tâche, le calcul des dépendances est préalablement effectué dans une classe nommée *coloringOctree* à chaque fois que la grille gagne ou perd des octrees lors de la répartition de la charge. De plus, comme la solution permet d'établir un graphe de dépendances de tâches tout en minimisant le nombre de dépendances, on en profite pour enlever toutes les tâches traitant des octrees ne contenant pas d'atome. Dans ce cas, la classe *coloringOctree* est mise à jour à chaque fois qu'un atome migre d'un octree à un autre, c'est-à-dire à chaque fois que les listes de Verlet sont mises à jour. Le listing suivant montre comment, pour n'importe quel type de potentiel, les tâches dont les informations sont stockées dans la structure coloringOctree cWave sont exécutées.

```
1 template<typename F> void gridAMR::launchGraph(F potential) {
2   #pragma omp parallel
3   #pragma omp master  /* the master thread launches tasks*/
4   for(size_t i = 0; i < cWave.size() ; i++)  /* cWave is pre-sorted according to the
        wave number */
5     OpenMPTask(cWave.numberOfDependencies(i),
6       cWave.valueOfDependencies(i), cWave.getIndex(i),
7       cWave.arrayOfDependencies(i), potential);
8 }
```

Listing 8.3 | Lancement des tâches avec dépendances avec l'interface OpenMP.

Les tâches sont lancées explicitement avec l'interface OpenMP. De plus, puisque le nombre de dépendances entre les tâches change à chaque itération, on utilise alors un commutateur en fonction du nombre de dépendances. La fonction *OpenMPTask* du listing suivant fournit l'implémentation d'une tâche OpenMP avec des dépendances allant jusqu'à 3^3-1=26. *uTask* correspond à l'index de la tâche à exécuter et *pTask* correspond à l'ensemble des tâches devant être impérativement exécutées avant la tâche *uTask*.

```
#define DEPS_EVAL(X) tmp_depGraph[pTask[X]]
#define DEPS1 DEPS_EVAL(0)
...
#define DEPS26 DEPS25, DEPS_EVAL(25)
#define EVAL_N(X) DEPS ## X
#define EVAL_PRAGMA(x) _Pragma (#x)
#define DO_PRAGMA(x) EVAL_PRAGMA(x)
#define OMP_TASK_DEPEND(N,OUT) DO_PRAGMA(omp task depend(in: EVAL_N(N)) depend(out:
    tmp_depGraph[OUT]))
#define CASE(N, ...) case N:\
  {\
    OMP_TASK_DEPEND(N,uTask)\
    potential(idOctree);\
  } break;

template<typename F>
inline void TMPL_AMRGrid::OpenMPTask(std::size_t nbDep, std::size_t uTask, int
    idOctree, std::size_t * pTask, F potential) {
#if _OPENMP >= 201511 // version openmp 4.5
  int * tmp_depGraph = depGraph;
  switch(nbDep)
  {
    case 0:
    {
      #pragma omp task depend(out: tmp_depGraph[uTask])
      potential(idOctree);
    } break;
    CASE(1);
    ...
    CASE(26);
    default:break;
  }
#endif
 }
```

Listing 8.4 | Tâche OpenMP avec un nombre de dépendances variable.

À noter que si l'on utilisait Tbb pour construire le graphe de dépendances, il suffirait de définir une tâche par la classe node_t en associant un octree et l'opérateur effectuant le calcul de l'énergie potentielle. Les dépendances entre les tâches seraient alors créées avec la fonctionmake_edge et elles pourraient être facilement détruites par la fonction remove_edge.

Une autre modification importante apportée à la grille Amr par la topologie any se situe lors de la mise à jour des zones fantômes. En effet, le système de transfert des atomes dans les zones fantômes établi dans le paragraphe 8.2.1 a été adapté afin d'accéder directement au traversal des cellules feuilles situées sur le bord du domaine. Le système prend maintenant en compte un nombre dynamique de sous-domaines adjacents. À noter que si deux sous-domaines adjacents ne

s'échangent pas d'atome, alors ces deux sous-domaines n'effectuent pas de communications Mpi tant que les listes de Verlet ne sont pas reconstruites.

Finalement, la conception d'une grille Amr avec une topologie any est simplifiée grâce à l'utilisation des traversals. De plus, si l'on considère les deux simulations décrites dans le paragraphe 7.1 avec un seul processus Mpi, 48 threads OpenMP, le potentiel Lj et le graphe de dépendances, l'utilisation de la grille Amr avec la topologie any ne ralentit pas le temps de la simulation (<1%) et n'augmente pas l'empreinte mémoire de la simulation (<1%) par rapport à la grille rectilinéaire.

Méthode de partitionnement de domaine.

Les sous-domaines peuvent désormais avoir n'importe quelle forme géométrique. L'étape suivante est d'utiliser des méthodes de partitionnement de domaine afin de répartir dynamiquement et équitablement la charge de calcul entre les sous-domaines. Pour cela nous nous sommes basés sur les développements déjà effectués par E. Cieren durant sa thèse [29] pour une grille any sans Amr. Son implémentation est construite autour de l'Api C de la librairie de partitionnement Zoltan développée par le laboratoire national Sandia.

Dans *ExaSTAMP legacy*, le développeur doit fournir un ensemble de champs décrivant les positions des cellules sur une grille cartésienne, leurs index (locaux ou globaux). Le développeur doit aussi spécifier si le processus Mpi/sous-domaine est le propriétaire de la cellule. Il doit fournir les poids associés pour toutes les cellules et bien sûr la méthode de partitionnement à utiliser. La librairie Zoltan retourne alors le sous-domaine de destination de chaque cellule. Ensuite c'est à *ExaSTAMP legacy* de faire migrer les atomes dans les bons sous-domaines en utilisant des messages Mpi. La cohérence des envois et des réceptions des atomes est alors assurée par le système *messageCenter*.

Comme nous l'avons vu dans le paragraphe 8.1, le système *messageCenter* copie des éléments un à un dans les tampons d'envoi. Dans le paragraphe 8.2.1, nous avons introduit un système optimisant la mise à jour des zones fantômes. Celui-ci est principalement basé sur la copie en parallèle de données contiguës (tableaux) dans les tampons. Pour cela il suffit de fournir au système la structure des données à envoyer, les pointeurs pour chaque tableau et le nombre d'éléments à copier. Nous avions utilisé ce système uniquement pour faire migrer les informations des cellules feuilles sur le bord des sous-domaines mais il n'y a aucune contre-indication à faire migrer n'importe quelle cellule. C'est pourquoi, pour la grille Amr, nous utilisons ce système. En pratique, comme la répartition de charge n'est utilisée que quelques fois dans une simulation (de l'ordre d'une fois toute les 10 000 itérations), cette optimisation a finalement peu d'impact sur le temps des simulations.

L'un des intérêts à utiliser la grille Amr au lieu d'une grille structurée (*ExaSTAMP legacy*) réside dans la réduction du nombre d'éléments (cellules/octrees) à fournir à Zoltan mais aussi dans la qualité des poids associés. En effet, lorsque la densité d'atomes est fortement non uniforme, une grande partie du domaine peut se retrouver avec très peu d'atomes. Lorsque la grille est structurée, la méthode de partitionnement doit alors prendre en compte des cellules sans atome. Si le poids associé a une cellule sans atome est égal à 0, alors la répartition de charge ne prend pas en compte les surcoûts liés à cette cellule et dans ce cas un sous-domaine peut recevoir la quasi-totalité des cellules vides. La stratégie choisie par *ExaSTAMP legacy* est d'ajouter des poids artificiels aux cellules sans atome. Cependant la valeur de ce poids par rapport aux poids des cellules contenant des atomes est difficile à déterminer. Dans le cas d'une grille Amr, les cellules ne contenant pas d'atome sont concaténées en une cellule racine (octree), réduisant ainsi la dégradation de l'équilibrage de la charge de calcul entre les sous-domaines par les cellules ne contenant pas d'atome. De plus, comme la grille Amr fournit uniquement les poids associés aux octrees, le nombre d'éléments à équilibrer

par la méthode partitionnement est donc plus faible (divisé par $2^{3*D^{max}}$) ce qui accélère le temps d'exécution de la méthode de partitionnement.

Un aspect très important lors de la répartition de la charge est le choix du poids que l'on associe à chaque octree pour la grille Amr ou à chaque cellule pour *ExaSTAMP legacy*. Dans *ExaSTAMP legacy*, le poids associé à une cellule j est égal à :

$$W_j = \sum_{ta \in T} \sum_{ta \in T} \Big[nbAtomes[ta] * nbAtomes[tb] * \Phi(ta, tb) \Big] + \alpha.$$

Où T est l'ensemble des types d'atomes, α est le poids artificiel constant (normalement égal à 0) pour chaque cellule, *nbAtomes* correspond à un tableau contenant le nombre d'atomes d'un type inclus dans la cellule et Φ est la fonction de coût du potentiel utilisée entre deux types d'atomes (par exemple 1 pour un Lj, 10 pour Eam et 100 pour un Meam). Donc si la simulation n'a qu'un seul type d'atome (t), on a la relation suivante :

$$W_j = nbAtome[t]^2 * \Phi(t, t) + \alpha.$$

Dans un premier temps, l'approche naïve utilisée par la grille Amr a été de réutiliser la même formule pour calculer les poids des octrees. Cependant, cette formule a été mise au point pour une grille structurée. Elle part donc du principe que la cellule est environ de la taille du rayon de coupure, et donc quasiment tous les atomes à l'intérieur d'une cellule interagissent entre eux. Dans ce cas, il est raisonnable de formuler l'hypothèse que le coût du calcul des interactions dans la cellule est corrélé avec le calcul du poids. Dans le cas d'un octree, sa taille est bien plus grande que le rayon de coupure, les atomes inclus dans un octree interagissent donc très peu entre eux. Au final, lorsque l'on utilise cette formule pour des simulations dont la densité d'atomes est non uniforme, les poids associés aux octrees des zones denses écrasent totalement les poids des octrees des zones moins denses.

Dans un deuxième temps cette formule a été adaptée à l'Amr. Pour cela on applique la formule à toutes les cellules feuilles de l'octree et le poids associé à l'octree est égal à la somme de ces poids. Cette technique aurait pu être suffisante. En effet, les cellules non raffinées, c'est-à-dire dont la taille est largement plus grande que le rayon de coupure, contiennent en théorie très peu d'atomes. Donc, même si les contributions des cellules non raffinées au poids de l'octree sont fausses, elles influencent très peu le poids de l'octree. Cependant, le nombre d'atomes dans les cellules feuilles dépend du critère de raffinement utilisé. Le problème est qu'il n'y a initialement aucun lien entre la quantité de calcul et le critère de raffinement. Ainsi si l'on utilisait cette formule pour calculer le poids d'un octree, le critère de raffinement impacterait directement la répartition de la charge entre les sous-domaines. Pour que le critère ne fausse pas les poids outre mesure, les cellules doivent être raffinées lorsqu'elles contiennent quelques atomes. Comme le but de l'Amr est justement d'agglomérer les cellules contenant peu d'atomes, ce poids n'est pas utilisé.

Finalement dans un dernier temps nous avons cherché un poids représentatif de la charge de calcul des interactions sans que celui-ci dépende du critère de raffinement. La solution choisie a été de sommer tout simplement le nombre d'atomes voisins pour chaque atome de l'octree. Cette solution est d'une part plus précise que la formulation d'*ExaSTAMP legacy* qui est une approximation du nombre d'interactions et d'autre part elle ne dépend pas du critère de raffinement. Cependant elle nécessite que les listes des atomes voisins soient construites. Dans le cas où la simulation ne contient qu'un type d'atome (t) on obtient la formulation suivante :

$$W_j = \sum_{a \in A_j} \Big[nbVoisins(a, t) * \Phi(t, t) \Big].$$

avec A_j l'ensemble des atomes inclus dans l'octree j, $nbVoisin(a,t)$ le nombre d'atomes voisins du type d'atome t pour un atome a. Cette formulation s'étend alors facilement pour des simulations utilisant plusieurs types d'atomes.

$$W_j = \sum_{a \in A_j} \left[\sum_{t \in T} \left[nbVoisins(a,t) * \Phi\Big(type(a), t\Big) \right] \right].$$

Cette formulation est légèrement modifiée pour tenir compte des spécificités du potentiel MEAM car pour chaque atome, on parcourt les voisins des voisins. Ce qui est environ égal à voisins * voisins. C'est pourquoi on obtient dans ce cas (simulation avec un type d'atome) :

$$W_j = \sum_{a \in A_j} \left[nbVoisins(a,t)^2 * \Phi(t,t) \right].$$

Dorénavant, ces poids sont utilisés pour toutes les simulations effectuées dans les études de ce chapitre. La charge de calcul d'un sous-domaine i est notée L_i. Elle est définie par :

$$L_i = \sum_{j \in E_i} W_j.$$

Où E_i est l'ensemble des octrees du sous-domaine i. Afin d'évaluer la qualité des méthodes de partitionnement, on utilise généralement la métrique de déséquilibre [105] entre la charge des sous-domaines notée λ. Elle est définie par l'équation 8.1 :

$$\lambda = \frac{L_{max}}{\bar{L}} - 1. \tag{8.1}$$

Où L_{max} correspond à la charge maximale de calcul des sous-domaines et $\bar{L}$ correspond à la charge moyenne de calcul des sous-domaines.

Un deuxième point important à définir pour les méthodes de partitionnement de graphe est la valeur du poids de chaque arc du graphe. Le but de ce poids est de modéliser le volume des communications MPI. Dans notre cas, nous utilisons ces poids pour équilibrer le nombre d'atomes à transférer durant la mise à jour des zones fantômes. Pour cela, le poids associé à l'arc reliant un octree O_R et un octree O_P est défini par la formule suivante :

$$Arc(O_R, O_P) = \#A_R + \#A_P + 1.$$

Où $\#$ est la cardinalité d'un ensemble et A_x l'ensemble des atomes de l'octree x. Pour cette formule, nous ajoutons par défaut 1 à la somme des atomes inclus dans les deux octrees car pour appliquer les méthodes de partitionnement de graphe, les poids des arcs doivent être strictement supérieurs à 0.

Maintenant que nous avons présenté la mise en place de la répartition de la charge de calcul entre les sous-domaines, nous allons étudier son impact pour les méthodes incluses dans la librairie Zoltan avec les méthodes géométriques RCB, RIB et SFC. Concernant les méthodes de partitionnement de graphe, nous utilisons celle implantée par défaut dans Zoltan : Partitionning Hypergraph and Graph (PHG) [42]. Comme pour Metis, la méthode PHG est basée sur les trois phases suivantes : agglomération, initialisation du partitionnement et raffinement. Des comparaisons avec PARMETIS sont effectuées par S. Rajamanickam et E.G. Boman [112].

Observations qualitatives des simulations dynamiques et hétérogènes

Avant de réaliser les études portant sur l'influence des méthodes de partitionnement de domaine, nous allons tout d'abord discuter des deux simulations décrites dans le paragraphe 7.1. Afin d'épurer les résultats, les simulations sont uniquement exécutées sur un cluster de 8 Skls mais la quasi-totalité des conclusions sont aussi valables pour des clusters de Knls. Dans ce paragraphe nous concentrons nos observations sur la répartition de la charge de calcul selon les méthodes de partitionnement.

Répartition de la charge de calcul :

En Dm, le nombre d'instructions de calcul effectuées lors de l'opérateur du calcul de l'énergie potentielle est directement corrélé au nombre d'interactions entre un atome et ses atomes voisins. Dans le paragraphe 8.3.4, le poids associé à chaque octree est égal à la somme du nombre d'atomes voisins pour chaque atome de l'octree. Par extension ce nombre correspond au nombre d'interactions traitées par l'opérateur du calcul de l'énergie potentielle pour un octree.

Dans le but de caractériser visuellement les méthodes de partitionnement de domaine, nous observons tout d'abord la distribution de la charge à l'aide du logiciel ParaView. Pour cela, nous allons dresser les profils des simulations en prenant en compte le nombre d'atomes voisins pour chaque atome. À noter que le potentiel utilisé a un rCut égal à 4.17Å. Cette valeur du paramètre rCut correspond alors au rayon de coupure du potentiel Meam que l'on utilise pour modéliser les interactions entre les atomes d'étain.

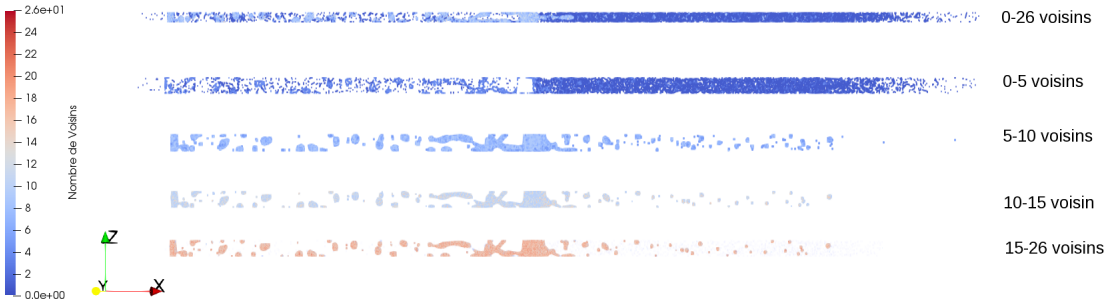


Figure 8.7 | Simulation d'un micro-jet au temps T=750 000Δ_t colorée selon le nombre d'atomes voisins pour chaque atome dans un rayon de 4.17.

La figure 8.7 dresse le profil pour la simulation micro-jet. Tout d'abord on observe que sur la tranche de 0 à 26 atomes voisins, on remarque qu'on ne distingue que des atomes dont le nombre de voisins est inférieur à 12. Cela s'explique par le fait que les atomes à l'interface entre le vide et le matériau ne possèdent que la moitié de leurs voisinages par rapport à un atome dans le matériau.

En observant la tranche d'atomes possédant entre 0 à 5 atomes, on remarque que la moitié de droite du micro-jet contient très peu de charge de calcul. En effet, les atomes ont été propulsés et n'interagissent plus avec d'autres atomes. Finalement pour cette simulation, la charge de calcul (plus de 15 atomes voisins) se concentre principalement sur quelques blocs de matière. Ainsi, les méthodes de partitionnement de domaine doivent faire en sorte de partitionner ces gros blocs de matière.

Concernant la simulation reproduisant l'impact d'une nano-goutte d'étain sur une surface solide, elle est illustrée par la figure 8.8. On remarque qu'une fois qu'on a filtré les atomes dont le nombre d'atomes voisins est inférieur à 5, on constate que la charge de calcul est principalement située dans la zone d'impact de la nano-goutte avec la surface libre. Une part non négligeable de la charge se situe aussi dans le bourrelet (cercle) et dans des petits éjectas entre la zone d'impact et le bourrelet.

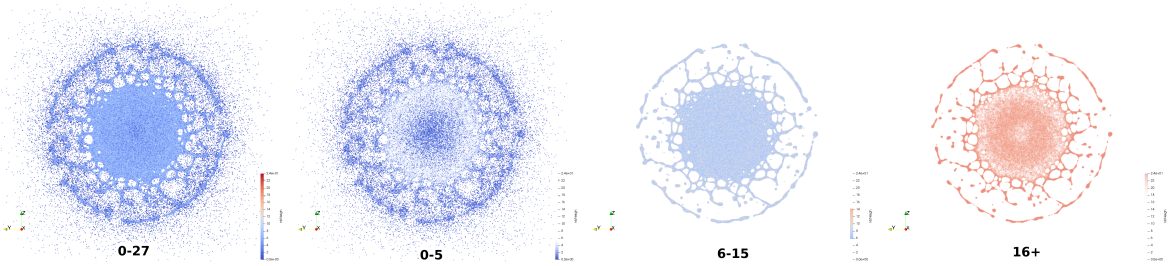


Figure 8.8 | Simulation de l'impact d'une nano-goutte d'étain sur une surface solide au pas de temps T=300 000Δ_t colorée selon le nombre d'atomes voisins pour chaque atome dans un rayon de 4.17. Ensuite la simulation est décomposée selon des tranches dépendant du nombre d'atomes voisins.

Contrairement à la simulation micro-jet, les méthodes partitionnement de domaine vont devoir prendre en compte un grand bloc d'atomes (centre), un moyen (bourrelet) et plein de petits blocs d'atomes entre les deux.

Étude qualitative du fonctionnement des méthodes de partitionnement :

Les méthodes de partitionnement de domaine RCB, RIB, SFC et PHG sont appliquées pour les deux simulations testées afin de décomposer le domaine de la simulation en 20 sous-domaines (20 processus Mpi).

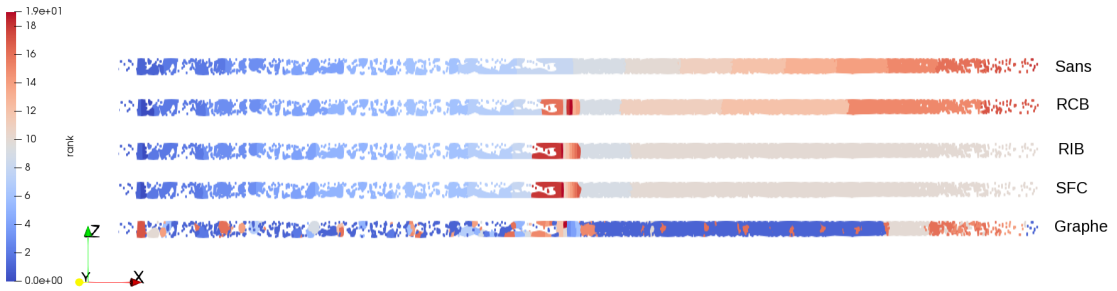


Figure 8.9 | Simulation micro-jet au pas de temps T=750 000Δ_t obtenue avec différentes méthodes de partitionnement. Les atomes sont colorés selon le numéro du processus Mpi associé au sous-domaine.

Pour la simulation micro-jet, cf. figure 8.9, on remarque tout d'abord que, contrairement aux méthodes de partitionnement géométriques, la méthode PHG associe plusieurs sous-domaines non connexes à un processus Mpi. Même si cela permet probablement de minimiser la métrique λ, augmenter le nombre de sous-domaines par processus Mpi va augmenter la taille des zones fantômes. On s'attend donc à ce que cette méthode ne soit pas la plus prometteuse pour cette simulation. Concernant les méthodes SFC et RIB, elles fournissent un partitionnement très proche si ce n'est que les interfaces entre les domaines sont légèrement différentes.

Pour la simulation de l'impact d'une nano-goutte d'étain sur une surface solide on constate que comme pour la simulation précédente, la méthode PHG affecte plusieurs sous-domaines à un processus Mpi, voir figure 8.10. Sur ce cas, puisque la hauteur des octrees est supérieure à l'épaisseur de la goutte, la plupart des atomes sont inclus dans des octrees formant un plan (donc 2D). La visualisation des méthodes est donc plus simple. On peut alors facilement observer le partitionnement du domaine par les méthodes de partitionnement géométriques. Par exemple la méthode RCB crée bien des sous-domaines de formes parallélépipèdes à base rectangulaire. La méthode RIB, quant à elle, divise récursivement les sous-domaines par un plan orthogonale à

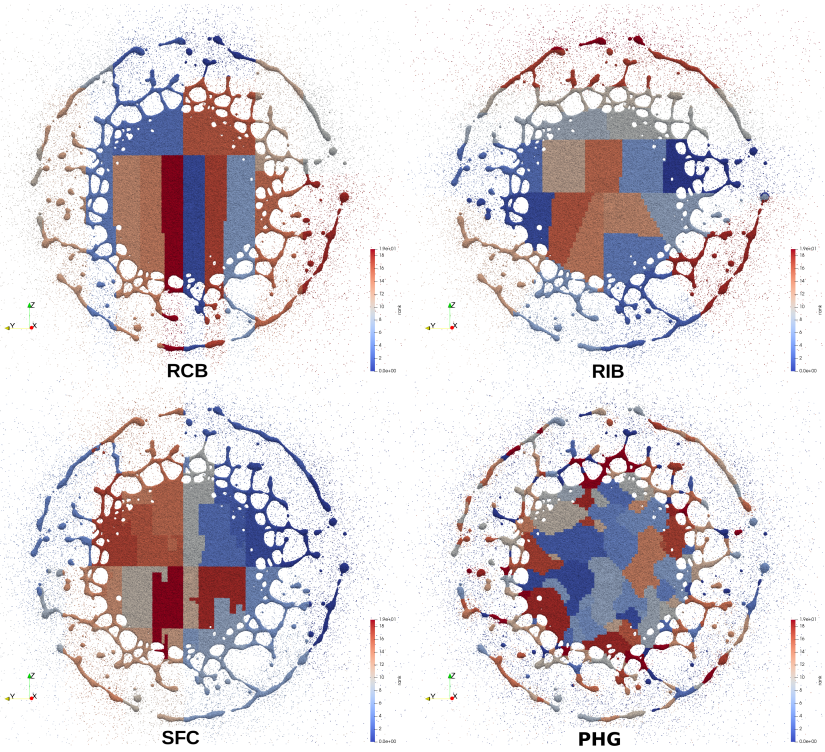


Figure 8.10 | Image de la simulation de l'impact d'une nano-goutte d'étain sur une surface solide au pas de temps T=300 000Δ_t obtenue avec différentes méthodes de partitionnement. Les atomes sont colorés selon le numéro du processus Mpi associé.

fl'axe principal d'inertie, ce qui crée des sous-domaines en forme de parallélépipèdes. Le principal problème des méthodes géométriques est qu'elles créent des sous-domaines sans tenir compte du nombre d'atomes à l'interface entre deux sous-domaines. Alors que la méthode PHG cherche par exemple à découper le bourrelet lorsque l'épaisseur de celui-ci est faible, ce qui minimise le nombre d'atomes à l'interface.

Étude de l'influence des stratégies de parallélisation OpenMP

Dans le paragraphe 7.3.4 nous avons vu que le choix de la parallélisation OpenMP varie en fonction du potentiel utilisé et de la simulation considérée. À cela s'ajoute qu'au fur et à mesure que le nombre de sous-domaines augmente, ces derniers contiennent de moins en moins d'octrees. Cette diminution d'octrees impacte alors le choix de la parallélisation intra-Nœud optimale et le choix du paramètre D^{max}. En effet, on rappelle qu'avec peu d'octrees, la parallélisation intra-Nœud n'est pas efficace sur un grand nombre de cœurs de calcul. Cette étude est alors découpée en deux parties. Une première partie est dédiée au potentiel peu coûteux Lj alors que la deuxième partie est réalisée avec un potentiel plus coûteux, le potentiel Meam. À noter que l'étude des méthodes de partitionnement est menée dans le paragraphe suivant, c'est pourquoi, pour plus de simplicité, uniquement la méthode RCB est utilisée. De plus, l'ensemble des stratégies de parallélisation OpenMP ont été testée avec les politiques statique, dynamique et guidée. Les résultats sont disponibles en annexe 11.2.

On observe que la meilleure stratégie de parallélisation intra-Nœud sur un Nœud pour un potentiel Lj, voir figure 8.11, est la stratégie basée sur la méthode par vagues avec un graphe de dépendance de tâches. On remarque que pour les deux simulations avec D^{max} =2 ou 4, les temps

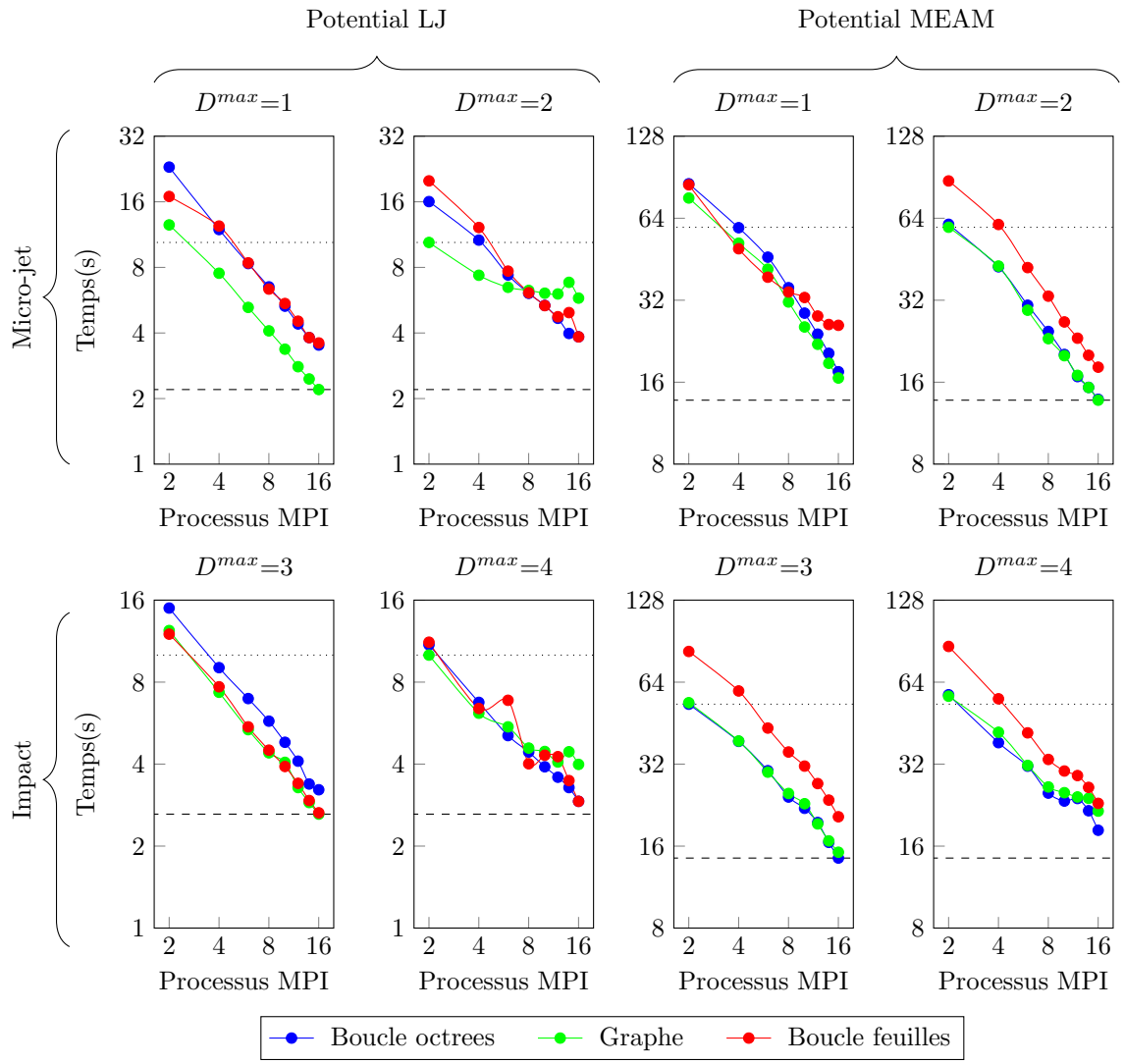


Figure 8.11 | Temps des simulations reproduisant l'évolution d'un micro-jet et l'impact d'une nano-goutte d'étain sur une surface solide. Les simulations sont exécutées sur 1 à 8 SKL. Deux processus MPI avec 48 threads OPENMP chacun sont associés pour chaque SKL. Les simulations sont réalisées avec plusieurs valeurs de D^{max} et méthodes de parallélisation OPENMP. La méthode de partitionnement de domaine utilisée est la méthode RCB. Les barres en pointillés minces et larges représentent respectivement le meilleur temps sur 1 et 8 SKL.

d'exécution pour la méthode par vagues ne diminuent presque plus à partir de 6 processus MPI. La raison est que le nombre d'octrees par vague doit au moins être supérieur au nombre threads, ce qui n'est plus le cas.

À partir d'un certain nombre de processus MPI, les autres stratégies sont plus intéressantes que la méthode par vagues. L'idée dans ce cas là est de réduire le paramètre D^{max} afin d'augmenter le nombre d'octrees par sous-domaine.

Finalement il vaut mieux choisir une valeur du paramètre D^{max} pour laquelle l'algorithme de *cache blocking* n'est pas forcément optimal mais dont le nombre d'octrees exécutables en parallèle est suffisant.

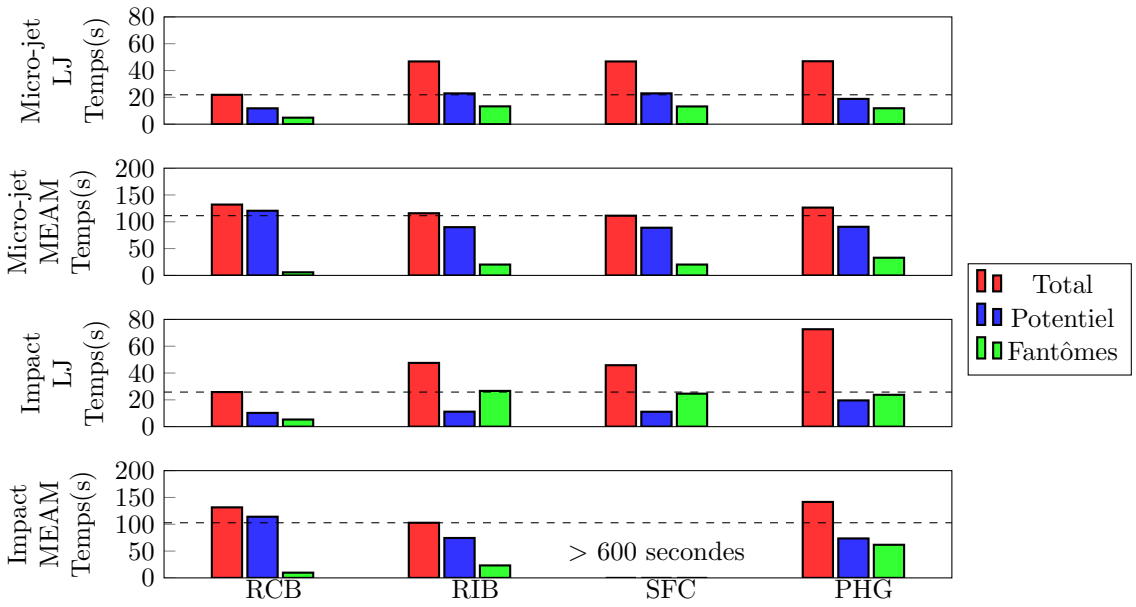
Concernant la stratégie *boucle feuilles*, celle-ci dépend très peu du paramètre D^{max} alors qu'on peut observer que la stratégie *boucle octrees* y est très sensible. À noter que pour la simulation Impact, pour 16 processus MPI, la meilleure stratégie n'est plus la stratégie graphe mais la stratégie *boucle feuilles* même si la différence est très faible.

Pour le potentiel Meam, on constate dans les 4 graphes de droite de la figure 8.11 que la stratégie *boucle feuilles* est la plus mauvaise. Les stratégies *boucle octrees* et celle basée sur la méthode par vagues sont alors presque équivalentes pour les deux simulations, c'est pourquoi on privilégiera la stratégie la moins contraignante : *boucle octrees*.

En conclusion de cette étude, lorsque la simulation est basée sur le potentiel Lj, ou d'une manière plus générale sur des potentiels peu coûteux, nous privilégions l'utilisation de la méthode par vagues combinée avec un graphe de dépendance de tâches en faisant varier la valeur du paramètre D^{max}. Quant aux potentiels plus coûteux tels que le potentiel Meam, nous favoriserons la stratégie *boucle octrees*.

Étude de l'influence des méthode de partitionnement

La stratégie de parallélisation intra-Nœud a été étudiée en fonction du nombre de processus Mpi, cf. paragraphe 8.3.4. Nous allons maintenant étudier l'influence des méthodes de partitionnement de domaine sur ces mêmes cas-tests. Pour cela nous utilisons les méthodes de partitionnement incluses par défaut dans la librairie Zoltan. Les simulations sont effectuées sur 8 Skls avec 2 processus Mpi par Skl et 48 threads OpenMP par processus Mpi. Nous avons choisi d'utiliser 8 Skls pour deux raisons. La première raison est que dans le cas d'un potentiel Meam, ces simulations sont représentatives des ressources qu'on allouerait pour ce nombre d'atomes. La deuxième raison est que dans le cas du potentiel Lj, on peut étudier comment se comporte notre système Amr lorsque la charge totale de calcul est très faible.

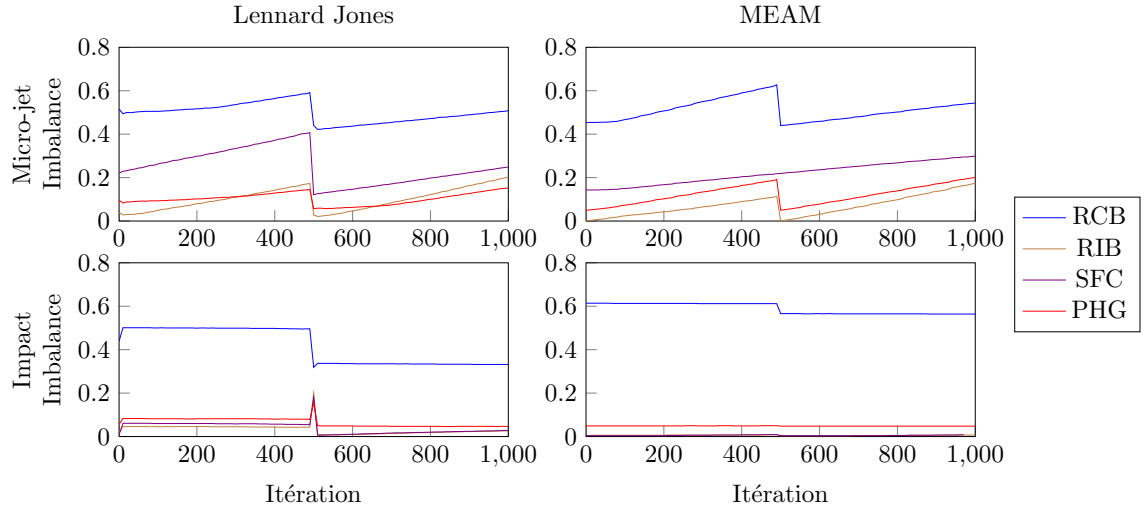


Histogramme 8.12 | Temps total (en rouge) en secondes de différentes simulations. Pour cela, plusieurs méthodes de partitionnement sont utilisées. Pour chaque simulation, le temps du processus le plus lent est reporté pour les portions du code effectuant le calcul de l'énergie potentielle (en bleu) et la mise à jour des zones fantômes (en vert).

Le temps d'exécution total est le temps qu'on cherche à réduire car c'est ce qui intéresse les utilisateurs (physiciens). Cependant pour mieux comprendre l'impact des méthodes de partitionnement, nous allons nous concentrer sur les deux opérateurs les plus consommateurs en temps, c'est-à-dire le calcul de l'énergie potentielle et la mise à jour des zones fantômes. Les temps des simulations micro-jet et impact pour les potentiels Lj et Meam sont présentés dans les histogrammes 8.12. On peut tout d'abord observer que pour le potentiel Lj, la méthode de partitionnement RCB est la plus

adaptée pour ces simulations car elle permet de minimiser au mieux les temps d'exécution des deux opérateurs les plus coûteux. De plus, comme prévu lors de l'étude qualitative des partitionnement des simulations, la méthode PHG est la plus lente à réaliser la mise à jour des zones fantômes.

Lorsque le potentiel Meam est utilisé, la méthode de partitionnement RIB devient plus intéressante que la méthode RCB. Elle est équivalente à la méthode SFC pour le cas micro-jet. Cela s'explique par un meilleur équilibrage de la charge de calcul pour l'opérateur calculant l'énergie potentielle. En effet, si l'on observe la mesure de déséquilibre λ pour les deux simulations (voir graphes 8.13) on remarque que le déséquilibre de calcul pour le potentiel Meam est bien plus important pour la méthode RCB que pour les autres.



Graphe 8.13 | Mesure λ des simulations selon le scénario d'un micro-jet et de l'impact d'une nano-goutte d'étain sur une surface solide. Pour cela plusieurs méthodes de partitionnement de domaines sont utilisées. Les simulations sont réalisées soit avec un potentiel Lj soit avec un potentiel Meam.

Dans le cas d'un potentiel Meam cette mesure de la charge semble pertinente en raison du grand nombre de calculs effectué par rapport au nombre d'accès mémoires ou autres effets annexes. Par exemple, pour la simulation Impact avec la méthode RCB, λ_{RCB} est environ égal à 0.6 (+60%) et le temps moyen mit par les processus Mpi pour effectuer le calcul du potentiel est de 68.16 secondes alors que le temps maximal est de 114.07 secondes. Soit un pourcentage de déséquilibre du temps mesuré ($(\frac{Temps_{max}}{Temps_{moyen}} - 1) * 100$) de +67%, ce qui est très proche de la prédiction faites par λ_{RCB}. Quant aux temps de calcul obtenus pour cette même simulation avec la méthode de partitionnement RIB, le pourcentage de déséquilibre du temps mesuré est de +7.5% pour λ_{RIB} environ égale à 0.009 (0.9%).

En revanche, dans le cas d'un potentiel très peu couteux comme le potentiel Lj, on remarque que cette métrique n'est pas adéquate pour quantifier le déséquilibre réel de la charge de calcul. Cela est dû aux poids assignés aux octrees qui ne sont pas représentatifs de la vraie charge car d'autres effets que le nombre d'interactions sont à prendre en compte tels que les accès mémoires. Cependant, même si ces poids ne sont 100% fiables, lors des tests de comparaison avec Lammps j'ai pu mesurer que pour la simulation micro-jet avec le potentiel Lj, le temps de la simulation est divisé par 5.64 entre 1 et 8 Skls pour Lammps alors qu'il est divisé par 6.21 pour la grille Amr. Finalement ce poids reste un bon estimateur pour répartir la charge.

Pour toutes les méthodes de partitionnement appliquées à la simulation du micro-jet avec un potentiel Meam, on constate que la valeur de λ ne cesse d'augmenter (linéairement). Cela est dû au déplacement vers la droite de la nappe d'atomes de la simulation. En mesurant la pente de la

mesure λ en fonction des itérations en temps, il est donc possible de déterminer à partir de combien d'itérations il est plus intéressant de partitionner de nouveau le domaine. En pratique cela dépend aussi du temps qu'il faut pour répartir de nouveau la charge entre les sous-domaines.

Simulation	micro-jet		Impact	
Potentiel	Lj	Meam	Lj	Meam
Recursive Coordinate Bisection	520 045 293	490 929 955	532 770 042	663 095 870
Recursive Inertial Bisection	705 880 713	653 704 668	560 035 614	1 838 264 641
Space Filling Curve	653 669 415	613 131 465	742 160 430	1 824 132 496
Partitionning Hypergraph and Graph	835 889 634	686 147 390	571 415 305	1 006 365 915

Table 8.1 | Nombre d'atomes échangés pendant la mise à jour des zones fantômes selon la méthode de partitionnement utilisée et la simulation considérée. Les simulations sont réalisées sur 1 000 pas de temps. ($\Delta_t = 1ps$). Pour cela 8 Skl sont utilisés avec 16 processus Mpi. Chaque processus Mpi a 48 hyperthreads pour 24 cœurs.

Concernant le temps de mise à jour des zones fantômes, si l'on mesure le nombre d'atomes qui ont été recopiés, voir le tableau 8.1, on remarque que ce temps (cf histogrammes 8.12) est lié à ce nombre d'atomes. En effet, on observe par exemple que la méthode RCB est celle qui effectue le plus rapidement cette portion du code et qu'elle copie beaucoup moins d'atomes que les autres. Cette méthode géométrique surpasse toutes les autres et cela pour toutes les simulations. Bien que les méthodes géométriques ne tiennent pas compte du nombre d'atomes à transférer, la forme en parallélépipède rectangulaire des sous-domaines issus de la méthode RCB permet de réduire naturellement le nombre d'octrees dans les zones fantômes et implicitement le nombre d'atomes.

À contrario des méthodes géométriques, la méthode de partitionnement PHG copie davantage d'atomes que les autres alors qu'elle devrait réduire le nombre d'atomes dans les zones fantômes, cf. paragraphe 5.3.2. Comme nous l'avons observé dans le paragraphe 8.3.3, cette méthode associe plusieurs sous-domaines par processus Mpi. Par conséquent le volume des zones fantômes est bien plus important. C'est pourquoi nous allons tester une autre méthode de partitionnement de graphe qui n'associe pas plusieurs sous-domaines non-connexes à un processus Mpi.

Partitionnement du domaine avec ParMetis

Les méthodes de partitionnement de graphe ont la propriété de tenir compte des arcs reliant les sommets d'un graphe. Cependant, la méthode de graphe PHG présente dans la librairie Zoltan affecte à chaque processus plusieurs sous-domaines non-connexes. Cela a pour conséquence d'augmenter le volume des zones fantômes. Cette augmentation alourdit les communications Mpi mais aussi le nombre d'interactions à calculer alors que le principal avantage des méthodes de partitionnement de graphes est de tenir compte de la connectivité entre les octrees. Une autre solution est alors d'interfacer Zoltan avec la méthode de partitionnement ParMetis. Dans ce paragraphe nous allons étudier l'influence de la méthode proposée par ParMetis d'une part sur sa capacité à équilibrer la charge de calcul et d'autre part à réduire les communications Mpi durant la mise à jour des zones fantômes.

Étude qualitative de l'impact d'une nano-goutte d'étain sur une surface solide

Avant d'étudier l'impact de ParMetis sur les temps de simulation, nous effectuons d'abord des observations qualitatives sur le cas de l'impact d'une nano-goutte d'étain sur une surface solide.

D'autres tests ont aussi été effectués sur la simulation d'un micro-jet mais nous nous focalisons uniquement sur le cas de l'impact d'une nano-goutte.

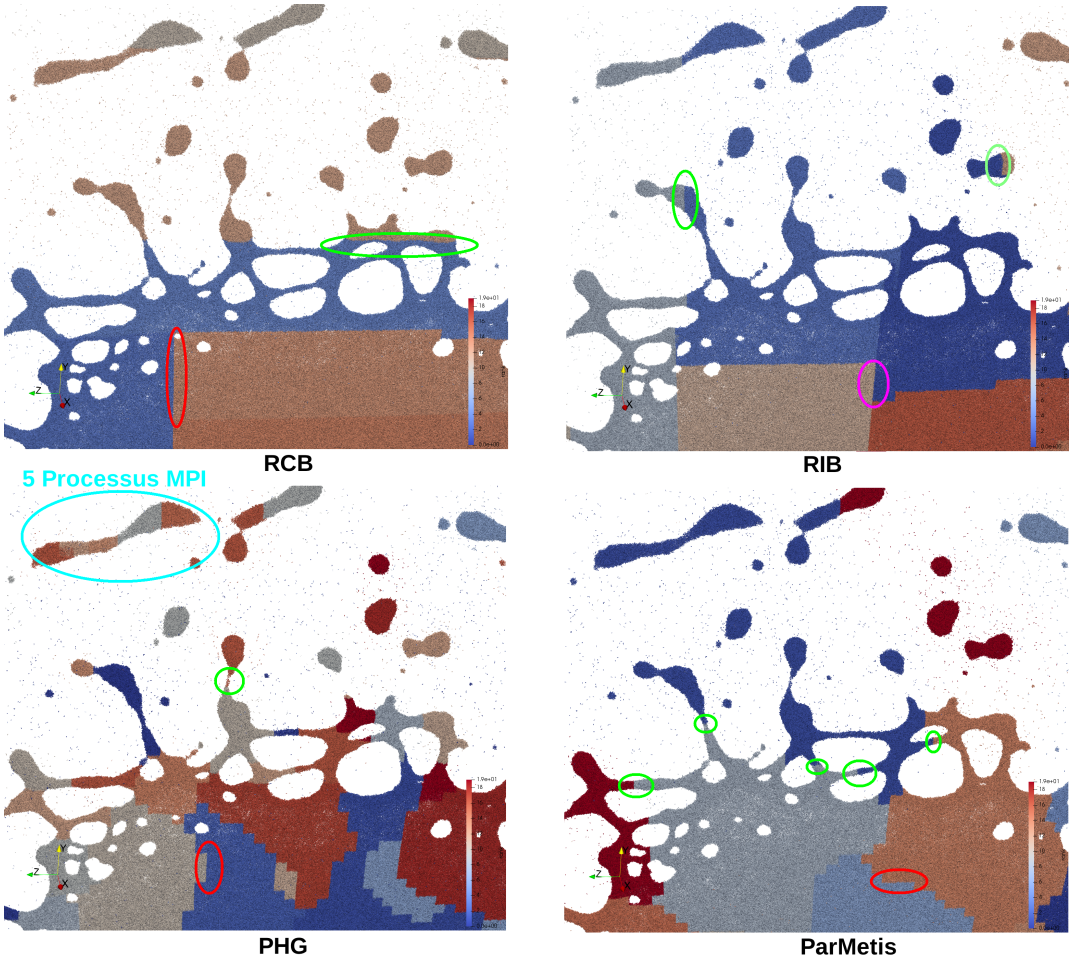


Figure 8.14 | Images zoomées de l'impact d'une nano-goutte d'étain sur une surface solide pour les méthodes de partitionnement de domaine Recursive Coordinate Bisection (RCB), Recursive Inertial Bisection (RIB), Partitionning Hypergraph and Graph (PHG) et ParMetis.

Tout d'abord, nous observons sur la figure 8.14 que ParMetis n'affecte pas plusieurs sous-domaines par processus Mpi. On remarque aussi que la méthode RIB créent des interfaces entre les sous-domaines en «diagonale» (ovale violet) ce qui augmente le nombre d'octrees à l'interface par rapport à un découpage selon un axe parallèle aux axes (Oy) ou (Oz) (ovales rouges). De plus on constate que la méthode PHG peut associer jusqu'à 5 processus Mpi pour un petit ensemble d'atomes (ovale bleu clair).

De plus, si l'on observe les interfaces entre les sous-domaines au niveau des petits éjectas de matière (ovales verts) on remarque que les méthodes géométriques partitionnent le domaine sans tenir compte de l'épaisseur du matériau alors que la méthode ParMetis va chercher à créer les interfaces les plus petites possible. On peut alors faire l'hypothèse que cette méthode est bien plus prometteuse que les précédentes.

Étude de l'impact de la méthode ParMetis

On souhaite tout d'abord vérifier si la méthode de partitionnement de graphe permet bel et bien de réduire le nombre d'atomes à recopier dans les zones fantômes. Le nombre d'atomes recopiés

en utilisant ParMetis et le ratio entre ce nombre et celui obtenu en utilisant la méthode RCB sont fournis dans le tableau 8.2. Dans tous les cas on constate que la méthode ParMetis surpasse les autres méthodes.

Simulation	micro-jet		Impact	
Potentiel	Lj	Meam	Lj	Meam
ParMetis	499 623 894	457 918 392	385 902 162	628 217 344
versus RCB (%)	96.07	93.27	72.43	94.58

Table 8.2 | Nombre d'atomes échangés pendant la mise à jour des zones fantômes pour la méthode de partitionnement ParMetis selon la simulation considérée. Les simulations sont réalisées sur 1 000 pas de temps. ($\Delta_t = 1ps$). Pour cela 8 Skls sont utilisés avec 16 processus Mpi. Chaque processus Mpi a 48 hyperthreads pour 24 cœurs.

Les résultats illustrés par les histogrammes 8.15 montrent que l'utilisation de la méthode ParMetis permet de réduire le temps de mise à jour des zones fantômes tout en conservant une bonne répartition de la charge pour l'opérateur calculant l'énergie potentielle. En effet, si l'on observe la métrique $\lambda_{ParMetis}$ sur les graphes 8.15 on remarque que celle-ci est très proche de λ_{RIB}.

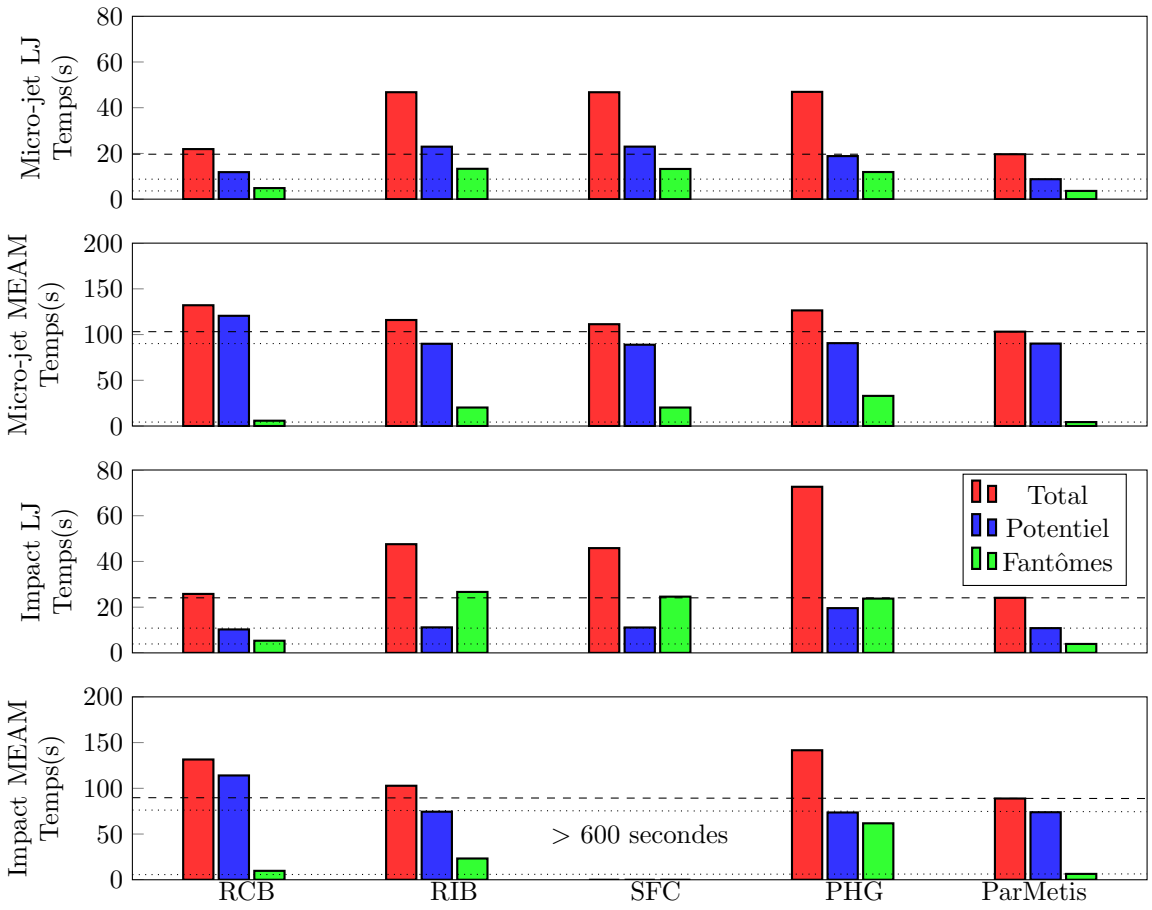


Figure 8.15 | Temps total (en rouge) en secondes de différentes simulations. Pour cela, plusieurs méthodes de partitionnement sont utilisées. Pour chaque simulation, le temps du processus le plus lent est reporté pour les portions du code effectuant le calcul de l'énergie potentielle (en bleu) et la mise à jour des zones fantômes (en vert).

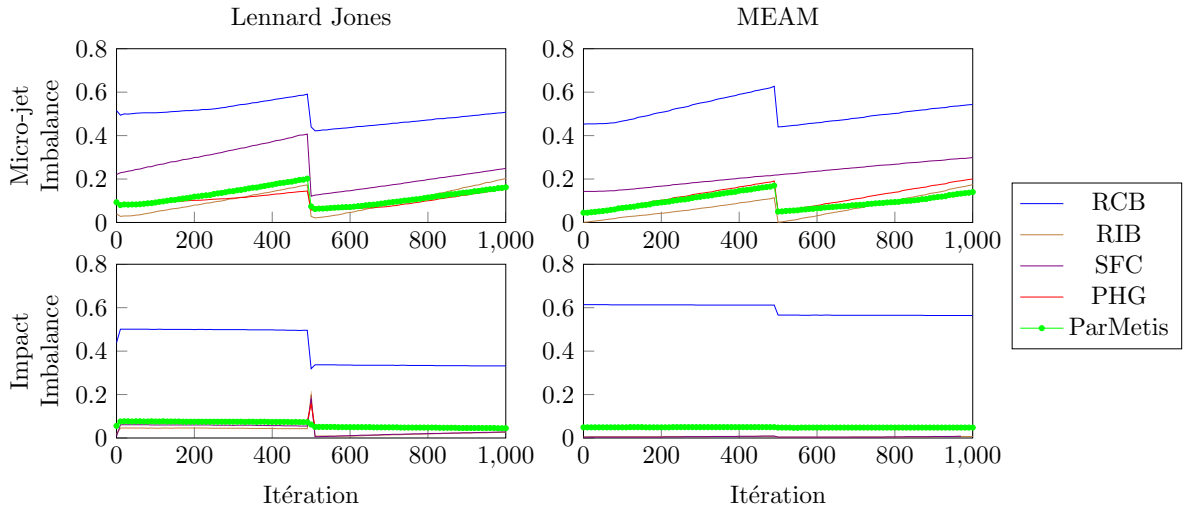


Figure 8.16 | Mesure de la métrique λ pour les simulations micro-jet et impact. Pour cela plusieurs méthodes de partitionnement de domaines sont utilisées. Les simulations sont réalisées soit avec un potentiel LJ soit avec un potentiel MEAM.

Les partitionnements obtenus via la méthode PARMETIS sont plutôt prometteurs. Il serait donc intéressant d'étudier d'autres méthodes de partitionnement de graphe comme PT-Scotch (cf. paragraphe 5.3.2). À noter que Zoltan propose lui aussi un partitionnement géométrique basé sur l'arborescence d'un octree nommée "reftree". Pour cela le développeur doit fournir à la librairie Zoltan les éléments de la grille grossière (cellules racines) mais aussi les arborescences des octrees, c'est-à-dire l'ensemble des cellules feuilles et intermédiaires. Le développeur peut notamment spécifier l'ordre de parcours des éléments grossiers (octrees) de la grille. Ces méthodes n'ont pas été évaluées car généralement PT-Scotch offre des résultats équivalents à PARMETIS et la méthode "reftree" est une méthode géométrique qui ne cherche donc pas à réduire le nombre d'atomes à recopier dans les zones fantômes.

Finalement, le partitionnement obtenu avec PARMETIS surpasse les autres méthodes de partitionnement pour toutes les simulations effectuées. Cependant, si l'on regarde attentivement le temps de l'opérateur effectuant le calcul de l'énergie potentielle, on se rend compte que pour le potentiel MEAM ce temps est très légèrement plus faible pour la méthode RIB et SFC que pour la méthode PARMETIS. Donc si la charge de calcul du potentiel est plus important alors que les zones de fantômes ne s'agrandissent pas, il est possible que la méthode PARMETIS ne soit plus la meilleure. De plus, bien que ces simulations soit représentatives des simulations envisagées, elles restent des cas-tests. Il est donc difficile de prévoir avec exactitude qu'elles seront les impacts des méthodes de partitionnement pour des centaines de processus MPI.

Des informations complémentaires sont obtenues en étudiant des cas de production. Les résultats sont présentés dans le chapitre 9.

Vers une recherche automatique de la stratégie OPENMP

Avant d'évaluer l'AMR sur des simulations de production, une dernière étape est utile pour limiter le nombre de simulations nécessaires pour trouver le meilleur jeu de paramètres. Pour cela, nous cherchons une heuristique permettant de choisir pour chaque sous-domaine quelle est la meilleure stratégie de parallélisation intra-NŒUD. Notre première idée est de choisir automatiquement (option *auto*) l'une des trois stratégies en fonction du nombre d'octrees sur chaque sous-domaines.

En partant du constat que pour les expérimentations effectuées lors de l'*étude de l'influence des stratégies OpenMP* (1 Nœud), la meilleure stratégie est généralement la méthode par vagues, puis la méthode *boucle octrees* et enfin la méthode *boucle feuilles*. De plus, pour ces stratégies et pour D^{max} fixé, le nombre minimum d'éléments (octrees/cellules feuilles) exécutables en parallèle est différent. Par exemple, pour une grille contenant 800 octrees avec $D^{max} = 2$, la méthode par vagues peut traiter en simultané $\frac{nbOctrees}{nbVague} = 100$ éléments. La méthode *boucle octrees* peut exécuter jusqu'à 400 octrees en même temps et la méthode *boucle feuilles* peut exécuter $400 * 8^2$=25 600 cellules feuilles. Le principe de l'option *auto* est alors d'utiliser la meilleure méthode de parallélisation intra-Nœud en fonction du nombre de threads et du nombre d'éléments exécutables en parallèle en privilégiant les méthodes dans l'ordre suivant : vagues/graphe, *boucle octrees* puis *boucle feuilles*.

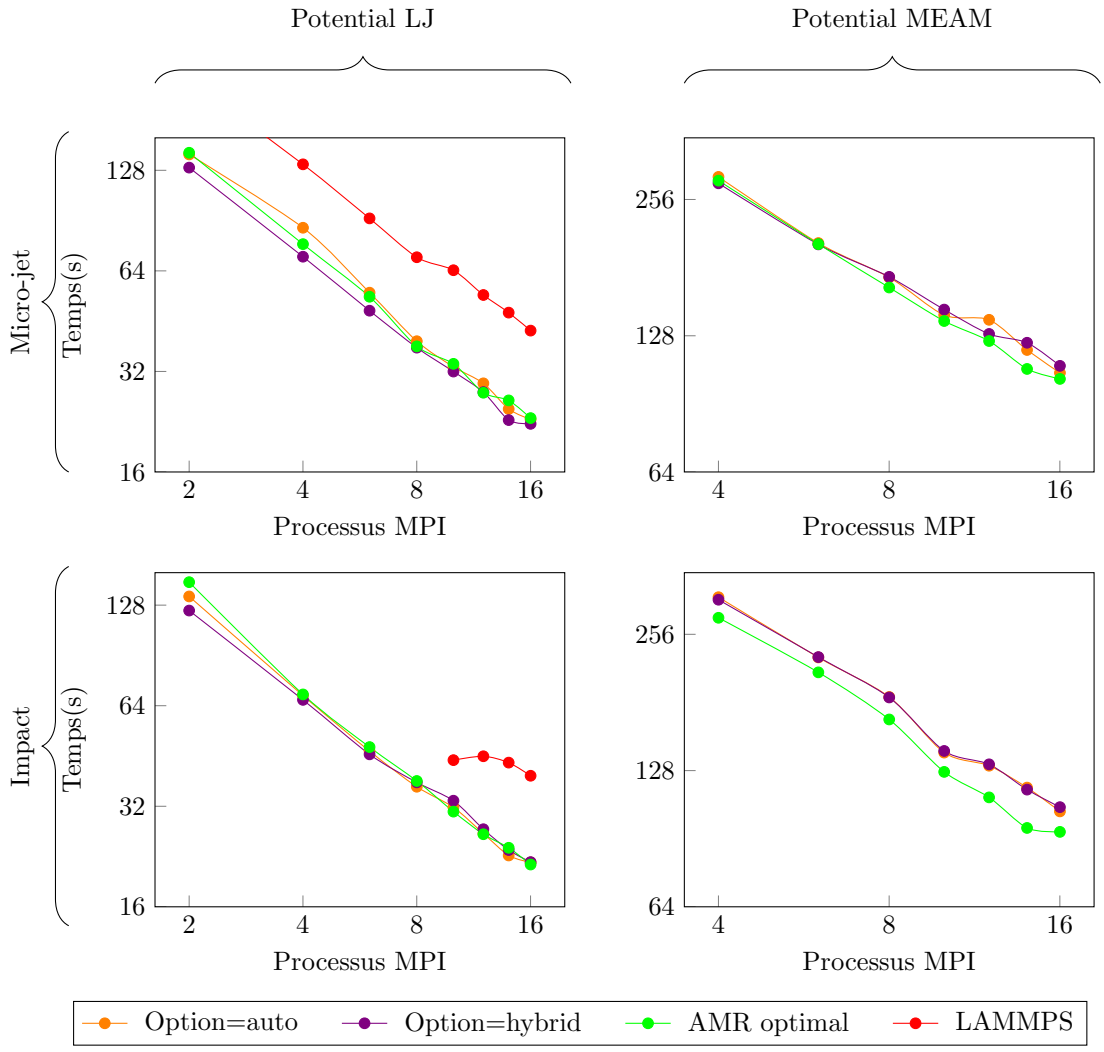


Figure 8.17 | Temps d'exécution des simulations micro-jet et impact pour les potentiels LJ et MEAM avec la grille AMR et LAMMPS (uniquement pour le LJ). Les simulations réalisées avec l'AMR, les stratégies *auto*, hybride et optimal (obtenue empiriquement) sont utilisées.

Les résultats obtenus (cf. graphes 8.17) montrent que l'option *auto* est légèrement moins bonne que la solution optimale pour un potentiel LJ. De plus, elle s'éloigne davantage pour le potentiel MEAM. La principale difficulté est de définir pour quel nombre d'octrees, l'option *auto* doit passer d'une méthode de parallélisation à l'autre. Dans notre cas nous avions par exemple fixé le passage de la méthode par vagues à *boucle octrees* lorsque le nombre d'octrees est inférieur à 8 (vagues)

fois le nombre de threads. Cependant nous n'avions pas prévu que les octrees possédant quelques atomes perturberaient ce passage. En faisant une étude paramétrique sur le critère choisissant la méthode de parallélisation intra-Nœud, on peut se ramener à des résultats en accord avec l'optimal, voire très légèrement meilleurs, mais ce critère n'est alors plus valable pour la deuxième simulation.

Une autre idée est de mixer la méthode par vagues avec la méthode *boucle octrees* (option *hybride*). Le but est de traiter les octrees contenant beaucoup d'atomes avec la méthode par vagues et laisser à la méthode *boucle octrees* le soin de traiter les octrees contenant peu d'atomes (par exemple inférieur à 10). Le but est alors d'alléger le graphe de dépendances des micro-tâches. Les résultats obtenus sont particulièrement intéressants pour le potentiel Lj car ils sont meilleurs que les résultats obtenus jusqu'à présent. Par contre on ne constate pas de gain pour un potentiel Meam.

En plus d'ExaSTAMP Amr, nous avons aussi utilisé le code Lammps sur ces simulations avec un potentiel Lj, cf. graphes 8.17. On constate alors que pour la simulation micro-jet, les "droites" (échelle log) sont presque parallèles et que Lammps est 1.83 fois plus lent sur un Skl que la grille Amr. Pour la simulation Impact, Lammps ne peut être exécuté avec moins de 5 Skls (10 processus Mpi). Le message d'erreur retourné par Lammps est : *Too many neighbor bins*). Pour un nombre suffisant de processus Mpi, Lammps stagne. Cela est principalement dû au temps des communications Mpi qui prennent environ 65% du temps total et à la construction des listes de voisins qui prennent 20% du temps total.

Finalement pour des tests réels nous privilégierons l'option *hybride* pour un potentiel Lj alors que pour le potentiel Meam nous utiliserons la méthode *boucle octrees*.

Conclusion

En conclusion, nous avons mis en place des optimisations permettant d'améliorer le temps de mise à jour des zones fantômes en rendant le système d'envoi des atomes entre les sous-domaines compatible et performant avec une parallélisation intra-Nœud. Nous avons aussi adapté la répartition de la charge entre les sous-domaines pour des grilles Amr, notamment en adaptant la grille any déjà présente dans *ExaSTAMP legacy*. Nous avons ensuite étudié pour la méthode RCB, l'impact des méthodes de parallélisation intra-Nœud en fonction du nombre de processus Mpi pour en déduire laquelle est la plus optimale et pour combien de processus Mpi.

Ensuite, des méthodes de partitionnement de domaines ont été testées et nous en avons déduit que pour limiter le temps de mise à jour des zones fantômes tout en conservant une bonne répartition de la charge, la meilleure solution était d'utiliser la méthode de partitionnement de graphe ParMetis. Nous avons aussi proposé un système "automatique" pour trouver la meilleure parallélisation intra-Nœud.

Maintenant, nous allons valider nos développements et l'ensemble de nos hypothèses dans le chapitre suivant sur des cas de production avec un très grand nombre d'atomes et de processeurs Knls.

9 Validation de la méthode AMR sur des simulations de production

L'objectif de ce chapitre est de valider notre solution basée sur la méthode Amr avec des cas de production. Nous montrons que notre solution améliore la répartition de la charge pour des simulations de Dm dont la densité d'atomes est fortement non uniforme et dont les atomes se déplacent très rapidement. Pour cela, nous étudions le comportement d'ExaSTAMP Amr sur deux simulations : le développement d'un micro-jet d'étain avec 1 milliard 249 millions d'atomes d'étain et l'impact d'une nano-goutte d'étain sur une surface solide de 516 millions d'atomes d'étain, cf. paragraphe 9.1. Ces simulations sont exécutées sur le supercalculateur TERA-1000-2 (classé 18 au top 500 [111]). La majorité des exécutions sont alors effectuées sur 256 Knls , soit 16 384 cœurs de calcul. Bien que cela ne soit pas facile sur une machine de production, quelques exécutions sont également réalisées sur 65 536 cœurs.

Notre évaluation d'ExaSTAMP Amr est divisée en 3 études. La première étude consiste à évaluer l'influence du critère de raffinement sur les temps d'exécution des simulations, cf. paragraphe 9.2. La deuxième étude a pour objectif de valider les hypothèses formulées dans les chapitres 7 et 8 quant au choix de la parallélisation intra-Nœud pour les deux simulations cibles, cf. paragraphe 9.3. La dernière étude a pour vocation de confirmer que la méthode de partitionnement de domaine ParMetis est la plus intéressante pour les deux simulations, cf. paragraphe 9.5.

Finalement, dans le but de confronter notre système Amr face à l'état de l'art, nous allons comparer, dans le paragraphe 9.5, ExaSTAMP Amr au code Lammps et *ExaSTAMP legacy* sur le plus gros cas test utilisable pour ces 3 codes : le développement d'un micro-jet de 589 millions d'atomes.

Présentation des expérimentations

La validation des contributions de cette thèse est effectuée sur deux simulations actuellement en cours de production sur le supercalculateur du Cea. Ces deux simulations reproduisent à une grande échelle le développement d'un micro-jet avec 1 milliard 249 millions d'atomes d'étain (voir figure 9.1) et l'impact d'une nano-goutte d'étain de 516 millions d'atomes sur une surface solide (voir figure 9.2). Les simulations sont respectivement reprises au pas temps 1 149 000 et 500 000 avec le potentiel Meam (rCut=4.17Å). À noter que la simulation de la nano-goutte est un cas particulièrement intéressant pour valider nos contributions. En effet, Cette simulation n'est pas réalisable avec le code *ExaSTAMP legacy* car la grille structurée en cellules de Verlet aurait théoriquement environ 103 milliards de cellules ce qui est trop important pour tenir en mémoire. Avec ExaSTAMP Amr, le nombre de cellules est ramené à environ 200 millions d'octrees.

Les simulations sont exécutées sur le supercalculateur TERA-1000-2 composé d'environ 8 000 Knls. Les Knls sont reliés entre eux par un réseau Bull eXascale Interconnect (Bxi). Chaque Knl

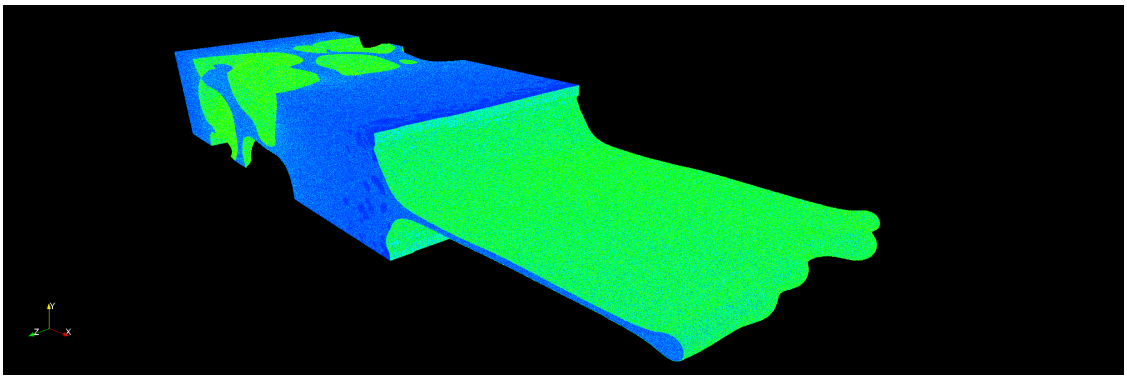


Figure 9.1 | Développement d'un micro-jet de 1 milliard 249 millions atomes d'étain obtenu avec le potentiel MEAM après 1 110 000 itérations en temps (Δt = 1fs).

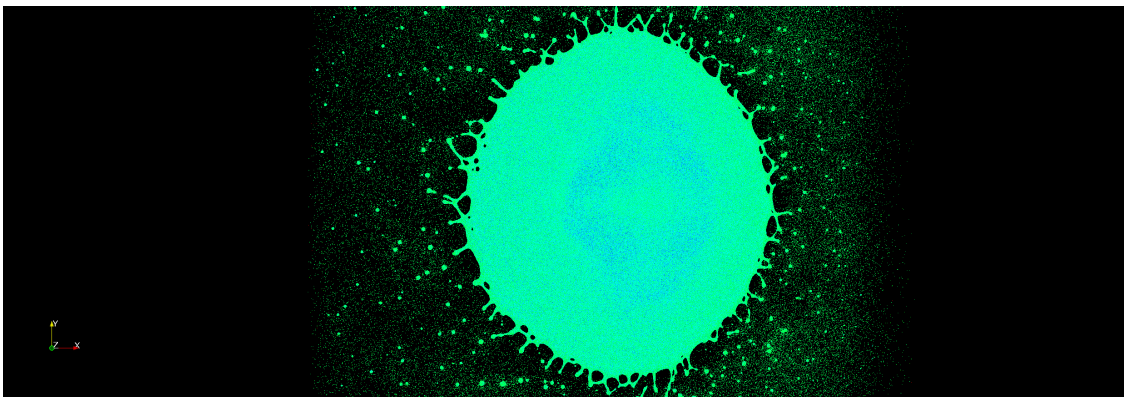


Figure 9.2 | Impact d'une nano-goutte d'étain de 516 millions atomes d'étain obtenu avec le potentiel MEAM après 490 000 itération en temps (Δt = 1fs).

possède 68 cœurs dont 4 cœurs sont dédiés au système. Chaque cœur peut exécuter 4 hyperthreads et possède des instructions SIMD AVX-512bits permettant d'appliquer simultanément une même opération à 8 doubles.

Dans le cadre de cette thèse, nous reprenons les conditions de production des deux simulations, c'est-à-dire entre 256 et 512 KNLS. Dans le but d'étudier plusieurs paramétrages possibles de la grille AMR dans un temps raisonnable, les simulations sont effectuées sur 1 000 pas de temps avec $\Delta t = 1fs$.

Étude de l'influence du critère de raffinement sur le temps d'exécution des simulations

Un critère important de l'algorithme AMR est défini pour déclencher ou non la subdivision d'une cellule en 8 cellules (3D). Le critère dépend de nombreux paramètres comme le nombre d'atomes, la densité d'atomes ou même le type des atomes. Rappelons aussi que le niveau de raffinement d'une cellule correspond à l'étage de l'octree où se trouve cette cellule. Par exemple une cellule dont le niveau de raffinement est 4 a été obtenue après 4 raffinements. Le niveau maximal de raffinement d'une cellule est limité par le paramètre D^{max}. En effet, si une cellule a son niveau de raffinement égal à D^{max} alors elle est dite de «Verlet», c'est-à-dire que l'arrête est égale à RVERLET.

Dans cette étude nous souhaitons comprendre son impact sur le temps d'exécution de simulations complexes dont la densité d'atomes est extrêmement hétérogène. Pour cela nous allons faire varier le critère de raffinement et observer le temps d'exécution des simulations décrites dans le paragraphe 9.1.

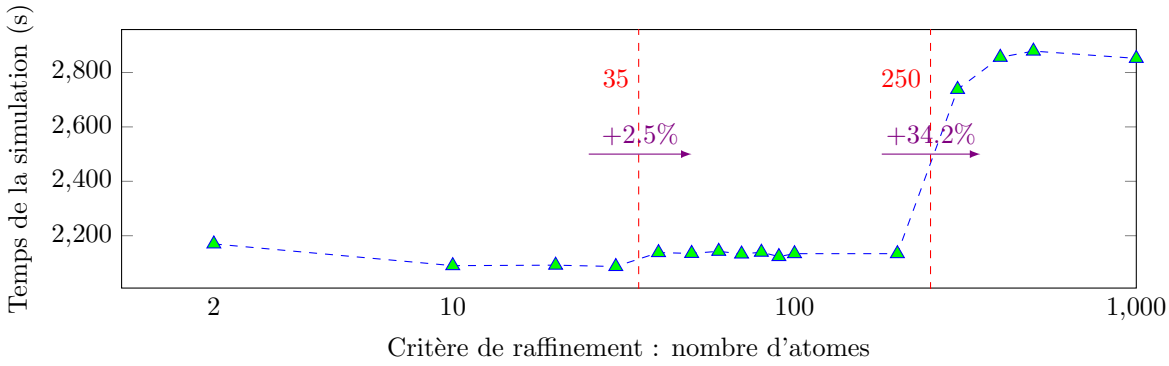
À noter que le critère de raffinement détermine le nombre d'atomes dans les cellules feuilles et par conséquent le nombre de cellules feuilles. La stratégie de parallélisation intra-Nœud *boucle feuilles* est alors fortement impactée par le critère de raffinement. C'est pourquoi, afin d'éviter un deuxième degré de liberté pouvant fausser nos interprétations, nous fixons la stratégie de parallélisation à *boucle octrees*.

L'objectif de l'Amr est d'adapter localement la taille des cellules de la grille en fonction de la densité d'atomes. Le critère de raffinement que nous avons retenu est le suivant : si la cellule contient au moins un certain nombre d'atomes fixé, alors elle est subdivisée. D'autres critères analogues tels que la densité d'atomes auraient pu aussi être utilisés. Pour limiter le périmètre d'étude, nous nous somme concentrés sur l'étude de l'impact du critère « nombre d'atomes ». De plus, pour les deux simulations cibles, nous avons constaté deux cas de figure :

1. Si l'espace ne contient pas d'atome, nous ne souhaitons pas que les octrees de la grille soient raffinés.

2. Si l'espace contient de la matière dont la densité d'atomes est élevée, nous souhaitons que les cellules soient le plus raffinées possible.

Le cas de figure d'un matériau « moyennement » dense n'existe pas, mis à part à l'interface entre les blocs de matière et le vide. C'est pourquoi il n'est pas nécessaire de chercher une autre solution telle que la densité d'atomes qui permet de tenir compte à la fois du nombre d'atomes et du volume de la cellule.



Graphe 9.3 | Temps d'exécution de la simulation d'un micro-jet de 1 milliard 249 millions d'atomes d'étain en fonction du critère de raffinement. Les simulations sont réalisée sur 1 000 pas de temps (Δt=1ps) avec un potentiel Meam.

Pour étudier l'impact du critère de raffinement, nous mesurons le temps d'exécution des simulations en faisant varier le nombre d'atomes à partir duquel une cellule est raffinée, cf. graphes 9.3 et 9.4. Pour la simulation micro-jet, on remarque alors 3 paliers différents. Ceux-ci correspondent aux différents seuils pour lesquels les cellules ne sont plus obtenues après N raffinement, avec $N \in \mathbb{N}$ et $0 <= N <= D^{max}$.

En effet, pour cette simulation dont le paramètre D^{max} est fixé à 2, on a mesuré que lorsque le critère de raffinement est supérieur à 250 atomes, la très grande majorité des octrees sont uniquement

composés d'une cellule racine. Le temps d'exécution de la simulation augmente alors de 34.2% pour deux raisons. Le temps de construction des listes d'atomes voisins est plus long car il y a davantage d'atomes dans les cellules et donc le nombre de tests (distance entre deux atomes < rCut) augmente. Le temps de recopie des atomes dans les zones fantômes est aussi plus long car on copie davantage d'atomes.

On remarque aussi une légère augmentation du temps d'exécution de 2.5% lorsque le critère est fixé à 35 atomes. Cette augmentation est liée aux cellules feuilles qui ne sont plus obtenues après deux raffinements mais seulement un. Cette augmentation est plus faible que pour le critère de raffinement précédent (fixé à 250 atomes) car les cellules agglomérées contiennent moins d'atomes. Ainsi le nombre de tests pour construire les listes d'atomes voisins dans une même cellule augmente avec une complexité en N^2, ce qui explique pourquoi l'augmentation du temps d'exécution est plus importante pour 250 atomes que pour 35 atomes. Finalement pour cette simulation on fixera le critère de raffinement aux alentours de 10 atomes.

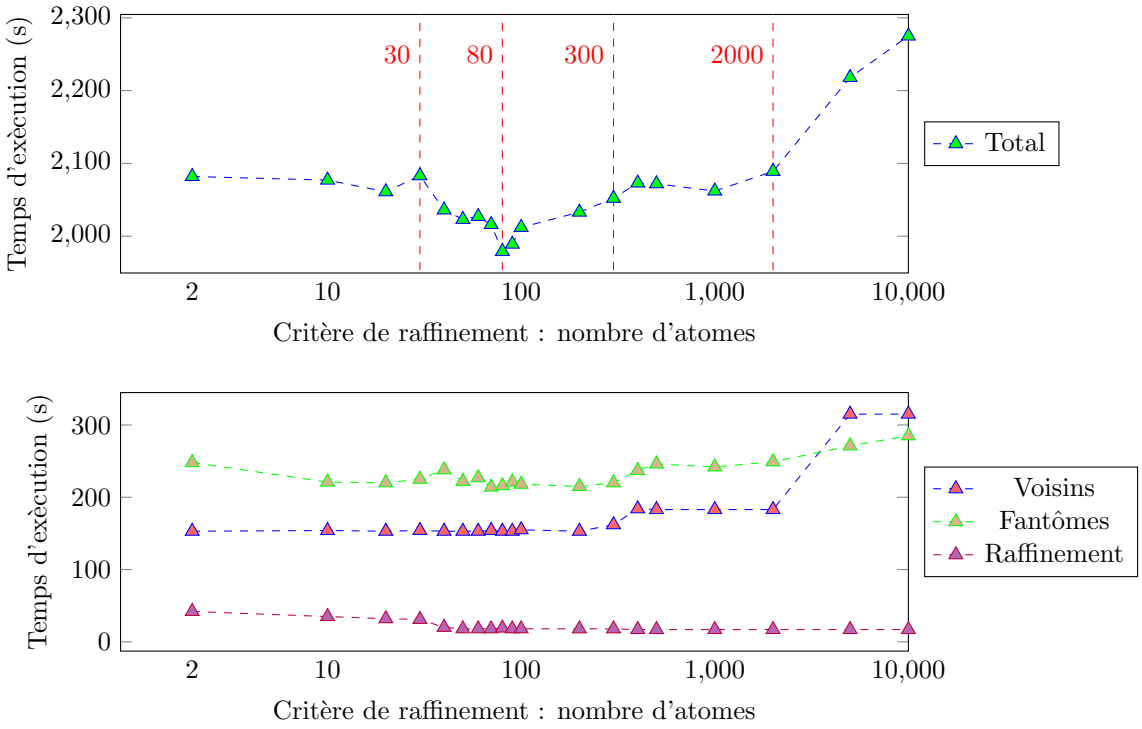


Figure 9.4 | Temps d'exécution de la simulation de l'impact d'une nano-goutte d'étain sur une surface solide de 1 milliard 249 millions d'atomes d'étain en fonction du critère de raffinement. Les simulations sont réalisée sur 1 000 pas de temps (Δt=1ps) avec un potentiel MEAM.

Concernant la simulation d'impact d'une nano-goutte d'étain, pour mieux comprendre l'influence du critère de raffinement on étudie également les temps correspondant aux opérateurs appliquant l'algorithme de raffinement, de construction des listes des atomes voisins et de la mise à jour des zones fantômes. Pour cette simulation, on remarque 4 comportements pour 4 intervalles distincts du critère de raffinement :

- Entre 2 et 30 atomes, le temps total de la simulation diminue très légèrement car le temps de l'opérateur mettant à jour les zones fantômes diminue. Cette diminution s'explique en partie par le fait que les octrees contenant très peu d'atomes sur le bord des sous-domaines sont de moins en moins raffinés finement. Ainsi, au lieu de copier dans le tampon d'envoi plusieurs

fois un très faible nombre d'atomes (moins de 5 atomes), ceux-ci sont copiés en une seule fois. De plus, comme il y a moins de cellules à raffiner, le temps de raffinement passe de 42 à 32 secondes.

- Entre 30 et 300 atomes, on constate que le temps diminue de 30 à 80 atomes puis il remonte légèrement jusqu'à 200 atomes. La diminution entre 30 et 40 atomes est corrélée avec la réduction du temps de l'opérateur raffinant la grille qui passe de 32 à 18 secondes. En effet, pour cette simulation, le paramètre D^{max} est fixé à 3. En raison du nombre de cellules plus important à raffiner, le temps de raffinement des cellules de niveau de raffinement 2 est théoriquement 8 fois plus élevé que celui pour les cellules dont le niveau de raffinement est de 1. Or comme pour la simulation micro-jet, à partir d'un critère fixé à 35 atomes, on constate que les cellules feuilles les plus raffinées ont un raffinement maximal de 2. Quant au temps de construction des listes des atomes voisins, il est constant.

- Entre 300 et 2000 atomes, les cellules feuilles les plus raffinées ont pour niveau de raffinement 1. Dans ce cas là, le temps de construction des listes des atomes voisins augmente de 153 secondes à 184 secondes et le temps de mise à jour des zones fantômes passe de 220 à 240 secondes. Comme pour la simulation micro-jet, ce phénomène est dû à l'augmentation du nombre d'atomes par cellule feuille.

- Lorsque le critère de raffinement est supérieur à 2000 atomes, les octrees sont composés d'une seule cellule racine, le temps de la simulation augmente alors de 9% entre la simulation utilisant le critère de raffinement fixé à 2000 atomes et le critère fixé à 10000 atomes.

En conclusion de cette étude, bien que le critère de raffinement puisse faire varier le temps de la simulation micro-jet jusqu'à $+36.7\%$ et le temps de la simulation de l'impact d'une nano-goutte d'étain de $+11.5\%$, les meilleurs résultats pour ces simulations avec la parallélisation intra-Nœud *boucle octrees* sont obtenus pour des critères (nombre d'atomes) de raffinement plutôt faible (entre 10 et 30 atomes).

Validation de l'influence de la parallélisation intra-nœud

Les études menées concernant l'élaboration d'une stratégie de parallélisation intra-Nœud dans le chapitre 7 et dans le paragraphe 8.3.4 nous ont permis d'émettre plusieurs hypothèses. Le choix de la parallélisation intra-Nœud dépend de 3 facteurs :

1. la distribution de la densité d'atomes de la simulation;
2. le coût du potentiel utilisé;
3. la valeur du paramètre D^{max}.

Nous avons déduit des expérimentations que, lorsque le nombre d'octrees par sous-domaines est largement supérieur au nombre de threads, la stratégie *boucle octrees* est la plus intéressante pour la simulation d'un micro-jet avec un potentiel Meam. De plus, nous avons observé que les temps de simulation entre la stratégie *boucle octrees*, l'option auto et la méthode par vagues sont très proches. Les temps d'exécution de la simulation d'un micro-jet de 1 milliards 249 millions d'atomes retranscrits dans le tableau 9.1 corroborent cette hypothèse.

Concernant la simulation reproduisant l'impact d'une nano-goutte d'étain sur une surface solide de 516 millions d'atomes, on constate que, comme dans l'étude effectuée dans le paragraphe 8.3.4,

Simulation	Développement d'un micro-jet	Impact d'une nano-goutte
Méthode par vagues	2365.6	2355.9
boucle octrees	2368.8	2034.9
boucle feuilles	6800.2	2299.1
Option auto	2363.8	2356.6
Option hybride	2393.3	2288.4

Table 9.1 | Temps d'exécution en seconde selon la méthode de parallélisation intra-Nœud utilisée et la simulation considérée. Les simulations sont réalisées sur 1000 pas de temps avec un potentiel Meam. Pour cela 256 Knl sont utilisés avec 256 processus Mpi. Chaque processus Mpi dispose de 256 hyperthreads pour 64 cœurs.

la stratégie *boucle octrees* surpasse la stratégie *boucle feuilles* (moins 264 secondes) et la méthode par vagues (moins 320 secondes). Concernant les options hybride et auto que nous avons étudiées dans le paragraphe 8.3.7, celles-ci ne permettent pas de réduire davantage le temps d'exécution que la stratégie *boucle octrees*.

À propos des simulations effectuées avec le potentiel Lj, on a observé que pour les deux simulations, les temps d'exécution les plus courts sont obtenus avec l'option hybride. En effet, l'option hybride permet de combiner l'avantage de la méthode par vagues avec stratégie *boucle octrees*. Les octrees contenant plus de 10 atomes sont alors traitées par la méthode par vagues afin de supprimer l'utilisation des mutex. Les octrees contenant moins de 10 atomes sont quant à eux traitées par la stratégie *boucle octrees* afin d'éviter les coûts de création d'un grand nombre de tâches pour traiter très peu d'atomes.

En conclusion, les temps d'exécution obtenus selon les stratégies de parallélisation intra-Nœud confirment les hypothèses évoquées dans les chapitres précédents. Pour l'étude suivante, nous utilisons la stratégie *boucle octrees* dans le cas où les simulations sont réalisées avec un potentiel Meam et l'option hybride lorsque la simulation est effectuée avec le potentiel Lj.

Étude de l'influence des méthodes de partitionnement de domaine

Dans ce paragraphe nous allons valider les hypothèses formulées dans les paragraphes 8.3.5 et 8.3.6 concernant le choix de la méthode de partitionnement de domaine et de son impact sur les temps d'exécution du code ExaSTAMP Amr. Pour cela nous évaluons les méthodes RCB, RIB, SFC, PHG et ParMetis sur 1 000 pas de temps (Δt =1fs) avec le potentiel Lj puis avec le potentiel Meam. Pour la simulation d'un micro-jet de 1 milliard 249 millions d'atomes d'étain, on ajoute les temps d'*ExaSTAMP legacy* comme point de repère afin de montrer les améliorations dues à l'utilisation de la méthode Amr. Rappelons également qu'*ExaSTAMP legacy* ne peut exécuter la simulation reproduisant l'impact d'une nano-goutte d'étain sur une surface solide à cause du trop grand nombre de cellules (plus de 100 milliards).

Pour pouvoir réaliser la simulation d'impact d'une nano-goutte avec ExaSTAMP Amr, le paramètre D^{max} doit être fixé à au moins 2 pour diviser par environ 64 le nombre de cellules. De plus, si l'on souhaite utiliser la méthode ParMetis, le paramètre D^{max} doit au moins être supérieur à 3 afin de limiter le nombre d'octrees. Lorsque le paramètre D^{max} d'ExaSTAMP Amr est fixé à 3, nous obtenons les meilleurs temps de simulation pour les autres méthodes de partitionnement.

À noter que le temps de l'opérateur effectuant la répartition de la charge (méthode de partitionnement et migration des atomes d'un domaine à l'autre) n'est pas mesuré dans ces exemples. En

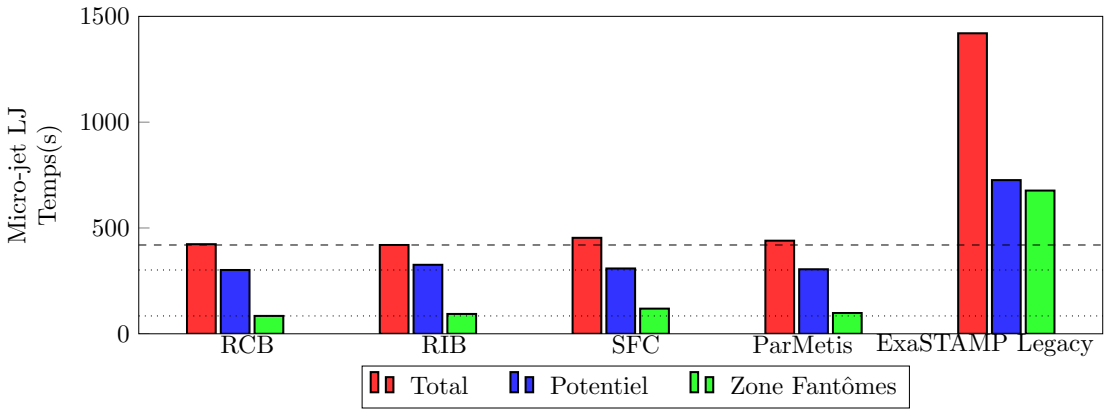
pratique ce temps est négligeable par rapport au temps mis par les opérateurs du potentiel ou de la mise à jour des zones fantômes. Cela se comprend car l'opérateur de répartition de la charge a une fréquence d'appel très faible (1 fois toutes les 10 000 ou 30 000 itérations) comparé à l'opérateur effectuant le calcul du potentiel qui est appelé à chaque itération. La suite de notre étude portant sur l'impact des méthodes de partitionnement pour les deux simulations cibles est alors divisée en deux parties : l'une porte sur les simulations réalisées avec le potentiel LJ et l'autre sur les simulations effectuées avec le potentiel MEAM.

Potentiel LJ :

Les mesures de temps de la simulation d'un micro-jet et de l'impact d'une nano-goutte sont respectivement présentées dans les histogrammes 9.5 et 9.6. Tout d'abord, on constate que la méthode de partitionnement de graphe PHG n'est pas utilisable car la bibliothèque Zoltan renvoie l'erreur suivante :

No space on proc ... - number of bytes requested = ... Memory error.

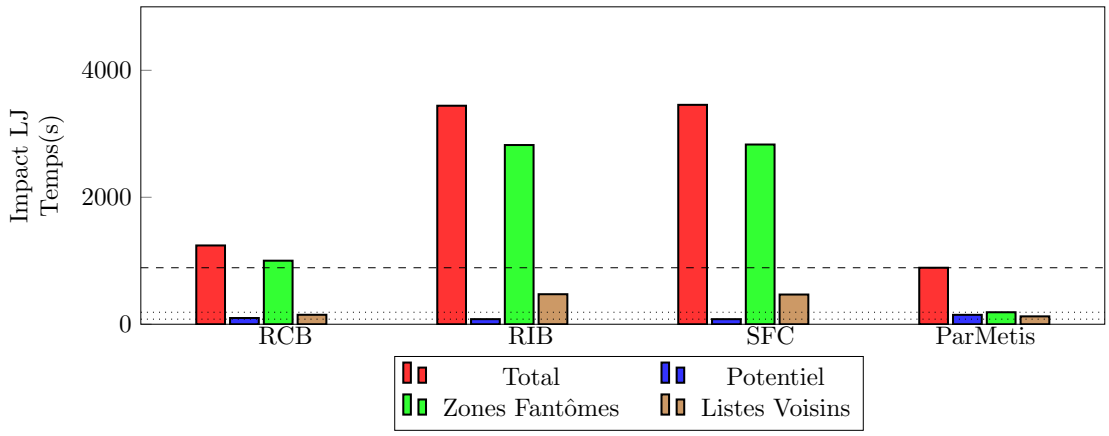
D'une manière générale cette méthode n'est techniquement pas applicable pour toutes les simulations présentes dans ce chapitre à cause de cette erreur.



Histogramme 9.5 | Temps d'exécution de la simulation reproduisant le développement d'un micro-jet d'environ 1 milliard 249 millions d'atomes (étain) selon plusieurs méthodes de partitionnement pendant 1 000 itérations en temps (Δt =1fs). Les temps pour les différentes portions du code correspondent aux sommes, pour chaque itération, des temps d'exécution sur le processeur le plus lent. Les simulations sont réalisées avec un potentiel LJ. Elles sont exécutées sur 256 KNL avec 1 processus MPI par KNL et 256 threads OPENMP par processus MPI.

Pour la simulation d'un micro-jet, on observe qu'*EXASTAMP legacy* est au moins trois plus lent qu'EXASTAMP AMR car le temps de l'opérateur de l'énergie potentielle est deux fois plus long. De plus, l'optimisation de la mise à jour des zones fantômes effectuée dans EXASTAMP AMR permet de diviser le temps de cet opérateur par 5.7 (SFC) voir 8.0 (RCB) par rapport à *EXASTAMP legacy*. À noter que la méthode de partitionnement RCB est utilisée par *EXASTAMP legacy* pour cette simulation. Finalement, pour ce cas, les méthodes RCB et RIB fournissent de meilleurs résultats que PARMETIS car le temps de l'opérateur du potentiel prédomine le temps de la mise à jour des zones fantômes.

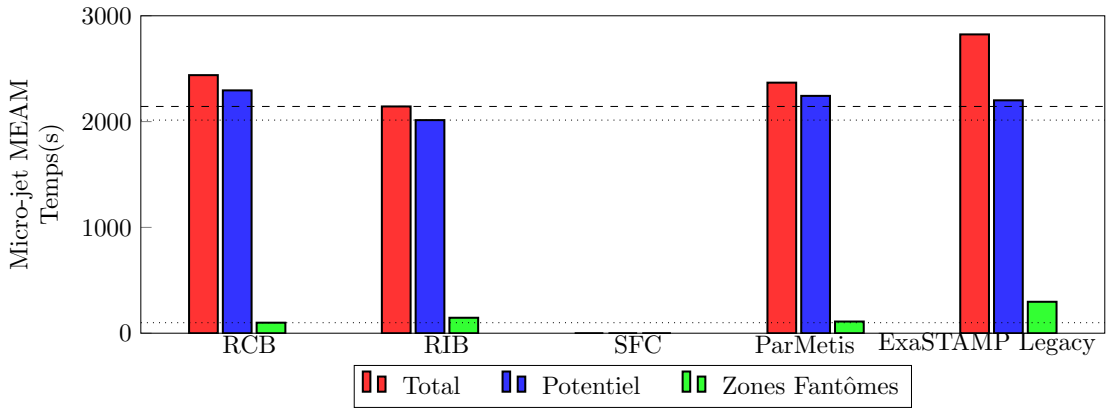
Pour la simulation de l'impact d'une nano-goutte, c'est le temps de l'opérateur de la mise à jour des zones fantômes qui prédomine largement sur celui du potentiel. À noter que pour cette simulation, le temps de mise à jour des listes des atomes voisins est, cette fois, plus long que le temps de l'opérateur du potentiel. De plus, on mesure que la méthode de partitionnement de graphe PARMETIS permet de réduire drastiquement le temps de mise à jour des zones fantômes. En effet,



Histogramme 9.6 | Temps d'exécution de la simulation reproduisant l'impact d'une nano-goutte d'étain sur une surface solide d'environ 516 millions d'atomes (étain) selon plusieurs méthodes de partitionnement pendant 1000 itérations en temps (Δt =1fs). Les temps pour les différentes portions du code correspondent aux sommes, pour chaque itération, des temps d'exécution sur le processeur le plus lent. Les simulations sont réalisées avec un potentiel LJ. Elles sont exécutées sur 256 KNL avec 1 processus MPI par KNL et 256 threads OPENMP par processus MPI.

cette portion du code est alors traitée 5.29 fois plus rapidement que la méthode RCB et 14.97 fois plus rapidement de la méthode SFC. On remarque aussi que le nombre d'atomes échangés pour les méthodes de partitionnement géométriques est au moins deux fois plus élevé que le nombre d'atomes échangés par PARMETIS, cf. tableau 9.2. Toutefois, la raison principale pour laquelle PARMETIS est plus rapide que les autres méthodes, c'est parce qu'elle équilibre également le nombre d'atomes échangés entre les sous-domaines. Finalement pour cette simulation, les hypothèses effectuées dans le paragraphe 8.3.6 concernant les avantages des méthodes de partitionnement de graphe comme PARMETIS se vérifient à une plus grande échelle.

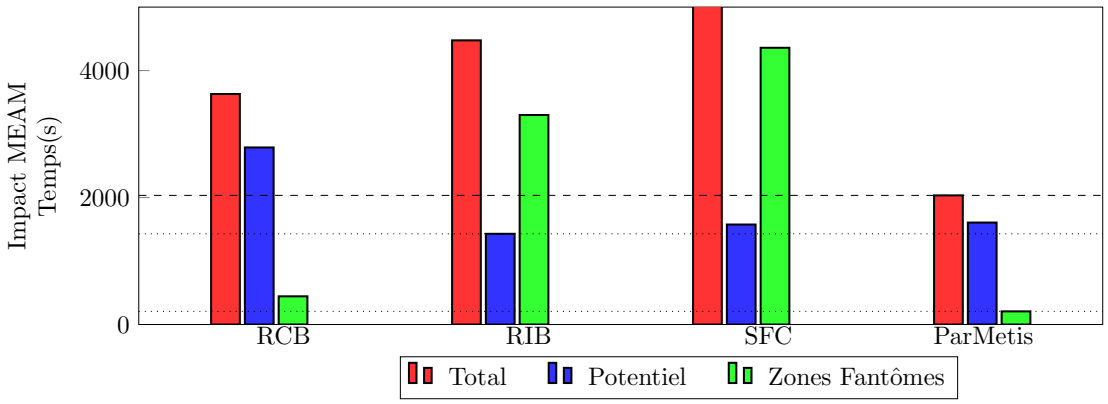
Potentiel MEAM :



Histogramme 9.7 | Temps d'exécution de la simulation reproduisant le développement d'un micro-jet d'environ 1 milliard 249 millions d'atomes (étain) selon plusieurs méthodes de partitionnement pendant 1000 itérations en temps (Δt =1fs). Les temps pour les différentes portions du code correspondent aux sommes, pour chaque itération, des temps d'exécution sur le processeur le plus lent. Les simulations sont réalisées avec un potentiel MEAM. Elles sont exécutées sur 256 KNL avec 1 processus MPI par KNL et 256 threads OPENMP par processus MPI.

Concernant les temps de la simulation d'un micro-jet avec le potentiel MEAM, on observe que le temps de l'opérateur du potentiel compose entre 90 et 95% du temps total des simulations effectuées

avec ExaSTAMP Amr et 78% pour *ExaSTAMP legacy* (RIB). Or, on sait que l'opérateur du potentiel Meam est plus coûteux que celui du potentiel Lj et que le temps de l'opérateur du potentiel Lj est déjà prédominant pour cette simulation. Cela explique pourquoi ParMetis n'est pas la méthode de partitionnement la plus efficace pour cette simulation. De plus, selon la méthode de partitionnement utilisée par ExaSTAMP Amr et *ExaSTAMP legacy*, nous avons mesuré que le gain de temps ($(\frac{T_{ExaSTAMP\ legacy}}{T_{ExaSTAMPAmr}} - 1) * 100$) pour l'opérateur du potentiel est toujours supérieur à 9.2% et peut atteindre jusqu'à 26% pour la méthode RCB. Finalement, pour la simulation d'un micro-jet avec un potentiel Meam, notre solution basée sur l'Amr est 1.32 fois plus rapide que le code *ExaSTAMP legacy*. À noter que cette simulation coûte plusieurs mois de calculs, c'est pourquoi réaliser cette simulation 1.32 fois plus rapidement permet d'une part de réduire le coût financier de telles simulations, mais surtout de pouvoir effectuer des simulations complexes avec toujours plus d'atomes.



Histogramme 9.8 | Temps d'exécution de la simulation reproduisant l'impact d'une nano-goutte d'étain sur une surface solide d'environ 516 millions d'atomes (étain) selon plusieurs méthodes de partitionnement pendant 1 000 itérations en temps (Δt =1fs). Les temps pour les différentes portions du code correspondent aux sommes, pour chaque itération, des temps d'exécution sur le processeur le plus lent. Les simulations sont réalisées avec un potentiel Meam. Elles sont exécutées sur 256 Knl avec 1 processus Mpi par Knl et 256 threads OpenMP par processus Mpi.

La dernière simulation étudiée est l'impact d'une nano-goutte d'étain sur une surface solide avec un potentiel Meam. Cette simulation est un parfait exemple pour lequel la méthode de partitionnement ParMetis permet de combiner une bonne répartition de la charge de calcul pour l'opérateur calculant l'énergie potentielle tout en minimisant le temps de l'opérateur effectuant la mise à jour des zones fantômes. En effet, la simulation réalisée avec la méthode de partitionnement de graphe ParMetis est 1.8 fois plus rapide que la simulation réalisée avec la méthode RCB et 2.2 fois plus rapide que la simulation réalisée avec la méthode RIB, cf. histogramme 9.8. Cela s'explique principalement par la différence de temps lors de la mise à jour des zones fantômes entre la simulation réalisée avec ParMetis et les autres simulations. Par exemple, ParMetis est 2.15 fois plus rapide que la méthode RCB, 15.94 fois plus rapide que RIB et 21.06 fois plus rapide que SFC. Le temps de l'opérateur du potentiel pour ParMetis est équivalent à celui de la méthode RIB. À partir des observations effectuées pour la simulation d'un micro-jet, on peut alors formuler l'hypothèse que si *ExaSTAMP legacy* pouvait exécuter cette simulation, le temps de la simulation serait probablement bien supérieur au temps obtenu avec ExaSTAMP Amr à cause du temps de mise à jour des zones fantômes.

Dans le but de montrer qu'ExaSTAMP Amr permet d'obtenir de très bon résultats sur un très grand nombre de cœurs, nous avons exécuté les deux simulations avec le potentiel Meam sur 16 384, 32 768 et 65 536 cœurs (256,512 et 1024 Knls), cf. tableau 9.3. Entre 16 384 et 65 536 cœurs, le temps de la simulation du micro-jet est divisé par 2.64. Quant au temps d'exécution de l'impact

Simulation	Développement d'un micro-jet		Impact d'une nano-goutte	
Potentiel	LJ	MEAM	LJ	MEAM
Recursive Coordinate Bisection	538.6	517.6	142.0	611.7
Recursive Inertial Bisection	428.6	395.4	164.6	555.4
Space Filling Curve	640.3	593.4	159.3	530.0
PARMETIS	447.4	419.5	122.9	381.8

Table 9.2 | Nombre d'atomes échangés (milliards) pendant la mise à jour des zones fantômes selon la méthode de partitionnement utilisée et la simulation considérée. Les simulations sont réalisées sur 1 000 pas de temps. ($\Delta t = 1ps$). Pour cela 256 KNL sont utilisées avec 256 processus MPI. Chaque processus MPI a 256 hyperthreads pour 64 cœurs.

Développement d'un micro-jet d'étain		
Nombre de cœurs	16 384	65 536
Total	2143	808
Potentiel	2014	693
Zones Fantômes	146	134
Listes d'atomes voisins	29	11
Raffinement	30	12
Impact d'une nano-goutte d'étain sur une surface solide		
Nombre de cœurs	16 384	65 536
Total	2034	997
Potentiel	1606	766
Zones Fantômes	207	135
Listes d'atomes voisins	154	73
Raffinement	34	16

Table 9.3 | Temps d'exécution en secondes selon le nombre de cœurs et la simulation considérée. Les simulations sont réalisées sur 1 000 pas de temps avec un potentiel MEAM. Pour cela entre 256 et 1024 KNLS sont utilisées avec 1 processus MPI par KNL. Chaque processus MPI a 256 hyperthreads pour 64 cœurs.

de la nano-goute d'étain, il est divisé par 2.04. Cette baisse s'explique principalement par le gain apporté lors du calcul de l'énergie potentielle qui représente plus de 80% des temps d'exécution des simulations. À noter que pour 1024 KNLS, ces simulations sont presque trop «petites».

En conclusion, lorsque le temps de l'opérateur du potentiel est prédominant comme c'est le cas pour la simulation d'un micro-jet, la solution à favoriser est la méthode RIB. Lorsque le temps de l'opérateur effectuant la mise à jour des zones fantômes est prédominant, il vaut mieux utiliser une méthode de partitionnement minimisant le temps de cet opérateur tout en conservant un « bon » équilibrage de la charge des calculs de l'énergie potentielle, c'est-à-dire dans notre cas celle de PARMETIS.

Comparaison de l'AMR avec l'état de l'art : LAMMPS

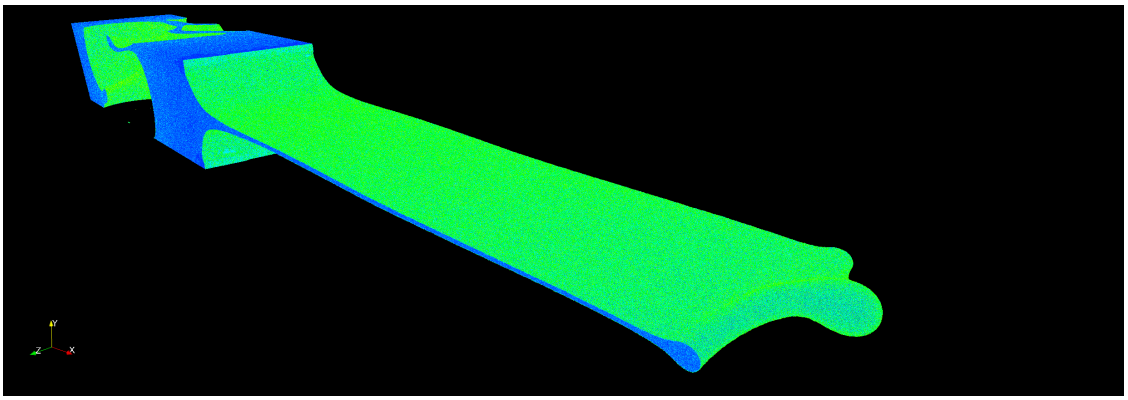
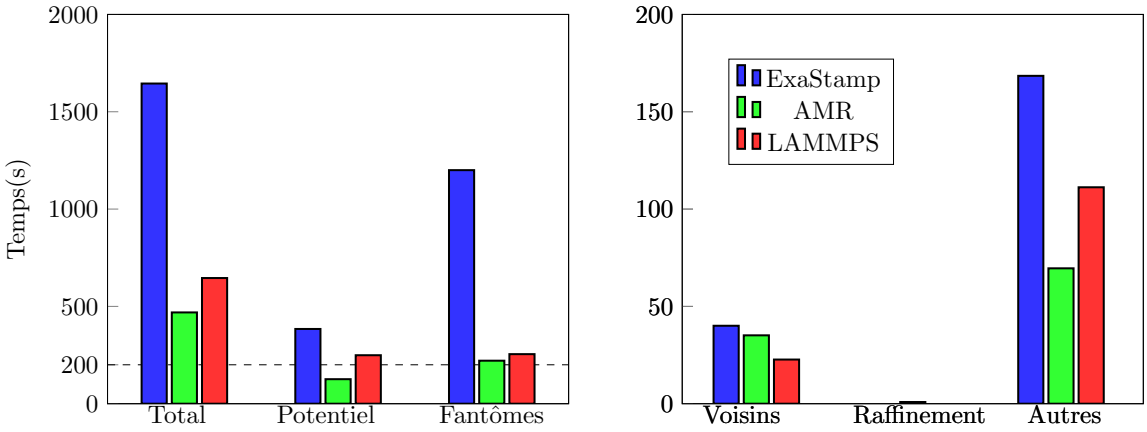


Figure 9.9 | Développement d'un micro-jet de 589 228 457 d'atomes d'étain obtenu avec le potentiel MEAM après 1 550 000 itération en temps (Δt = 1fs).

Nous effectuons nos comparaisons avec le code de référence LAMMPS en partant d'une sauvegarde d'un micro-jet d'environ 589 millions d'atomes d'étain au pas de temps 1 550 000 ($\Delta t = 1fs$), voir figure 9.9. Les interactions entre les atomes d'étain sont modélisées par le potentiel LJ (RCUT=10Å). Nous choisissons ce cas car les sauvegardes sur les simulations présentées dans le paragraphe 9.1 échouent avec le code LAMMPS. Pour les codes EXASTAMP AMR, *EXASTAMP legacy* et LAMMPS, les simulations sont alors réalisées sur 512 KNLS avec 1 processus MPI par KNL et 256 threads OPENMP par processus MPI.



Histogramme 9.10 | Temps d'exécution de simulation reproduisant l'impact d'une nano-goutte d'étain sur une surface solide d'environ 589 millions d'atomes d'étain selon plusieurs codes pendant 1 000 itérations en temps (Δt =1fs). Les temps pour les différentes portions du code correspondent aux sommes, pour chaque itération, des temps d'exécution sur le processeur le plus lent. Les simulations sont réalisées avec un potentiel LJ. Elles sont exécutées sur 512 KNL avec 1 processus MPI par KNL et 256 threads OPENMP par processus MPI.

La méthode de partitionnement de domaine RCB est utilisée par LAMMPS, *EXASTAMP legacy* et EXASTAMP AMR. Pour cette simulation, EXASTAMP AMR est exécuté avec l'option hybride. Les temps obtenus pour les portions du code prenant le plus de temps sont présentés dans l'histogramme 9.10. Concernant le temps total de la simulation, EXASTAMP AMR est le code le plus rapide pour réaliser cette simulation. EXASTAMP AMR est 1.38 fois plus rapide que LAMMPS et 3.51 fois plus

rapide que *ExaSTAMP legacy*. Cela est en partie dû au temps requis pour appliquer l'opérateur du calcul de l'énergie potentielle et l'opérateur effectuant la mise à jour des zones fantômes.

En effet, Lammps et *ExaSTAMP legacy* sont respectivement 2.06 et 3.05 plus lents qu'ExaSTAMP Amr pour appliquer l'opérateur effectuant le calcul de l'énergie potentielle. Cette observation résulte d'une part de l'utilisation de la méthode des blocs de Verlet optimisant la vectorisation des calculs et d'autre part de la suppression des mutex dans certaines phases multithreads. L'opérateur du potentiel d'*ExaSTAMP legacy* est aussi ralenti par l'utilisation d'un tampon de vectorisation et un grand nombre de cellules ne contenant pas d'atome. De plus, ExaSTAMP Amr effectue la mise à jour des zones fantômes plus rapidement que Lammps et bien plus rapidement que *ExaSTAMP legacy*. À noter que le temps cumulé de ces deux opérateurs pour tous les codes représentent au moins 73% du temps total de la simulation.

On notera aussi que le temps de raffinement de la grille Amr est très faible par rapport au temps de calcul total ($< 0.2\%$). Finalement cette étude nous a permis de montrer les avantages de la méthode Amr par rapport au code Lammps sur une simulation dont la densité d'atomes est non uniforme, dont les atomes se déplacent rapidement et dont le nombre d'atomes est très important.

Conclusion

Dans ce chapitre nous avons étudié l'influence de 3 paramètres importants de la méthode Amr implantée dans ExaSTAMP (critère de raffinement, stratégie de parallélisation intra-Nœud et la méthode de partitionnement) sur le temps d'exécution des simulations de production. Nous en avons déduit quels sont les paramétrages optimaux pour les deux simulations ciblées par cette thèse. Par ailleurs nous avons également confronté ExaSTAMP Amr à *ExaSTAMP legacy* sur les deux simulations cibles et à Lammps sur un dernier cas test représentatif d'un micro-jet.

Finalement, nous avons démontré l'intérêt de la méthode Amr en Dm pour réduire le temps des simulations dont la distribution de la densité d'atomes est fortement déséquilibrée. Elle est d'autant plus nécessaire pour des simulations comme l'impact d'une nano-goutte d'étain sur une surface solide car elles ne sont pas techniquement réalisables par la plupart des codes de Dm actuels.

10 Conclusion et Perspectives

Cette thèse a pour ambition d'utiliser le plus efficacement possible les supercalculateurs pour réaliser des simulations de Dm dont la distribution de la densité des atomes est extrêmement inégale. Pour cela, nous désirons mettre en œuvre des modèles de programmation efficaces en calcul haute performance permettant d'appréhender les architectures machine massivement multicœurs actuelles et futur. Nous souhaitons également étudier des structures de données adaptées aux contraintes imposées par les simulations de Dm tout en appliquant les 3 niveaux de parallélisation : vectorisation, intra-Nœud et inter-Nœuds.

Contributions

L'objectif de cette thèse est d'optimiser les performances ainsi que l'équilibrage de la charge de calcul pour les 3 niveaux de parallélisme dans le code de Dm ExaSTAMP. Dans cette thèse, nous avons développé ExaSTAMP Amr, un code de simulation numérique de dynamique moléculaire basé sur la méthode de raffinement de maillage adaptatif et la méthode de l'octree. L'un des objectifs de la structure Amr est de réduire le nombre de cellules ne contenant pas ou peu d'atomes, qui ralentissent le temps d'exécution des simulations. Pour cela, nous avons évalué diverses architectures possibles d'un octree avant de choisir la plus optimale. Une structure spécifique a été conçue et développée pour favoriser l'utilisation des instructions Simd avec une structure Amr.

Nous avons ensuite optimisé la réutilisation des données dans les caches mémoire via l'utilisation d'un algorithme de *cache blocking*. Nous nous sommes ensuite intéressés à l'optimisation de l'opérateur effectuant le calcul de l'énergie potentielle. Lorsque la simulation est basée sur des potentiels peu coûteux tels que le potentiel Lj ou le potentiel Eam Sutten-Chen, nous avons introduit et utilisé la méthode des blocs de Verlet. Cette méthode permet d'utiliser la vectorisation sans avoir à aligner les données en les copiant dans un tampon de vectorisation. Cela nous a permis de réaliser la simulation d'un cristal de cuivre de 108 000 atomes en séquentiel (Knl) avec le potentiel Eam Sutten Chen 1.97 fois plus rapidement qu'avec *ExaSTAMP legacy*. Toutefois pour des potentiels très coûteux tels que le potentiel Meam, on utilise un tampon de vectorisation. Le choix de la méthode se fait automatiquement lors de la lecture du jeu de données de l'utilisateur.

Concernant la parallélisation intra-Nœud avec ExaSTAMP Amr, nous avons élaboré plusieurs stratégies de parallélisation de l'opérateur calculant l'énergie potentielle. Nous avons notamment adapté la méthode par vagues à la grille Amr afin d'éviter l'utilisation de coûteux mutex. Nous avons également proposé une parallélisation des calculs à l'aide d'un graphe de dépendances de tâches (Dag) ce qui nous a permis d'exécuter des tâches appartenant à plusieurs vagues en même temps. Par exemple, cette stratégie permet en partie de réduire le temps d'exécution d'une simulation d'un

micro-jet de 3.5 millions d'atomes avec un potentiel LJ sur 1 KNL (64 cœurs) de 50.9% par rapport à LAMMPS et de 83.4% par rapport à *ExaSTAMP legacy*.

Après plusieurs expérimentations qui nous ont permis de déterminer le comportement des stratégies de parallélisation intra-NŒUD, nous avons déduit qu'il vaut mieux privilégier la méthode par vagues avec un graphe de dépendances de tâches (DAG) lorsque le potentiel est peu coûteux comme par exemple le LJ ou le EAM Sutten-Chen. Pour des potentiels coûteux tels que le potentiel MEAM, on utilisera plutôt la stratégie de parallélisation intra-NŒUD *boucle octrees*.

La dernière étape a été d'élaborer la parallélisation inter-NŒUDS adaptée à la structure AMR. Pour cela, nous avons adapté les développements déjà effectués dans *ExaSTAMP legacy* à la grille AMR pour construire des sous-domaines dont la forme est quelconque. L'objectif est d'appliquer des méthodes de partitionnement spatiales du domaine tel que la méthode RCB. Un système de gestion des zones fantômes adapté à l'AMR a également été développé. Il permet de recopier efficacement en parallèle (threads) uniquement les atomes sur le bords du domaine.

Concernant le choix de la méthode de partitionnement de domaine à utiliser, nous avons observé deux comportements lors de notre étude sur des simulations exécutées sur 256 KNLS. Lorsque le temps de calcul de l'énergie potentielle est largement prédominant, il est préférable d'utiliser une méthode géométrique. A contrario, lorsque le temps de la mise à jour des zones fantômes est largement prédominant, on utilisera la méthode de partitionnement de graphe PARMETIS.

Par ailleurs, pour les simulations effectuées sur 16 384 cœurs (256 KNLS), nous avons mesuré qu'EXASTAMP AMR permet de réaliser la simulation d'un micro-jet d'étain de 1 milliard 249 millions d'atomes avec un potentiel MEAM. EXASTAMP AMR est alors 1.31 fois plus rapide que *ExaSTAMP legacy*. Pour cette même simulation réalisée avec le potentiel LJ, EXASTAMP AMR est 3.38 fois plus rapide qu'*ExaSTAMP legacy*.

Nous avons aussi effectué des tests sur l'impact d'une nano-goutte d'étain sur une surface solide de 516 millions d'atomes, cette simulation n'était jusqu'alors techniquement pas réalisable avec *ExaSTAMP legacy*. Pour ce type de simulation, le temps de mise à jour des zones fantômes prédomine celui du calcul de l'énergie potentielle. Dans ce cas, nous avons montré que la méthode de partitionnement de graphe PARMETIS permet de réduire davantage le temps total de la simulation que les méthodes de partitionnement géométrique RCB, RIB et SFC.

Ces deux simulations ont également été effectuées sur 65 536 cœurs (1024 KNLS) et le temps d'exécution a été divisé par 2.65 pour la simulation d'un micro-jet et par 2.04 pour la simulation d'impact d'une nano-goutte d'étain. À noter que ces simulations sont presque trop petites pour autant de cœurs.

Nous avons également confronté notre solution au code de référence LAMMPS sur un micro-jet d'étain de 583 millions d'atomes avec un potentiel LJ. Nous avons alors pu constater que EXASTAMP AMR est 1.38 fois plus rapide que LAMMPS et 3.51 plus rapide qu'*ExaSTAMP legacy*.

Finalement l'intégration de la méthode de Raffinement de Maillage Adaptatif dans un code de DM permet de :

- réduire le nombre de cellules ne contenant pas d'atome;
- réduire l'empreinte mémoire des simulations;
- améliorer l'utilisation de la vectorisation via l'utilisation de la méthode des blocs de Verlet;
- supprimer les mutex lors du calcul de l'énergie potentielle via l'utilisation de la méthode par vagues;

- améliorer l'application des méthodes de partitionnement pour des simulations contenant beaucoup de cellules sans atome;
- réduire le temps d'exécution des simulations dont la densité est homogène;
- concevoir et développer un système efficace et parallèle (communications Mpi) pour la gestion des zones fantômes pour une structure Amr;
- réduire le temps de simulations complexes et coûteuses comme le développement d'un micro-jet de plus de 1 milliard d'atomes d'étain sur des dizaines de milliers de cœurs;
- réaliser des simulations jusqu'alors inconcevables (temps d'exécution trop long et coûts de stockage mémoire trop importants) comme la simulation d'impact d'une nano-goutte d'étain sur une surface solide de 516 millions d'atomes d'étain.

Perspectives

«Super-octree» :

La grille Amr a été initialement pensée comme une forêt d'octrees afin de faciliter de nombreux points tels que l'application des méthodes de partitionnement. Une autre idée, moins prometteuse, est d'utiliser un unique octree pour tout le domaine. Cependant, lors de nos expérimentations sur une nano-goutte, bien que notre méthode Amr permet de réduire drastiquement le nombre de cellules ne contenant pas d'atome, cette simulation est tellement extrême que le nombre d'octrees ne contenant pas d'atome n'est toujours pas négligeable. Si des simulations avec encore plus d'espaces sans atome venaient à être effectuées, l'utilisation d'un «super-octree» permettrait de supprimer totalement le coût des cellules ne contenant pas d'atome. Ce système doit pouvoir être activé ou désactivé au profit du système d'une forêt d'octree en fonction de l'évolution de la dynamique des atomes.

Méthodes de partitionnement de graphe :

L'utilisation de la méthode de partitionnement de graphe ParMetis a montré que ce type de partitionnement permet de réduire le temps de l'opérateur des zones fantômes. De plus, ParMetis permet de conserver une répartition de la charge de calcul de l'opérateur du potentiel proche des meilleures méthodes géométriques. Améliorer l'équilibrage du nombre d'atomes échangés entre les sous-domaines est une étape cruciale qui nous permettra assurément d'améliorer les performances des codes de Dm pour des simulations plus complexes que l'impact d'une nano-goutte d'étain. Ainsi des études sont à mener pour évaluer d'autres méthodes de partitionnement de graphe comme PT-Scotch et modéliser les arcs entres les sommets du graphe avec de nouveau poids. De plus, compte tenu qu'un sous-domaine communique avec l'ensemble des sous-domaines adjacents, au lieu d'utiliser un arc entre deux sous-domaines, une autre possibilité est d'utiliser des méthodes de partitionnement d'hypergraphes prenant en compte des arcs entre plusieurs sous-domaines.

Méthodes d'apprentissage :

Dans cette thèse nous nous sommes intéressés aux méthodes de partitionnement géométriques et de graphes. Nous avons notamment étudié leurs influences sur les différentes portions d'un code de Dm. Une idée est de tester plusieurs méthodes de partitionnement sur quelques pas de temps tout en mesurant l'impact des partitionnements sur le temps de chaque opérateur (zones

fantômes, énergie potentielle, etc). Le but est d'enrichir une base de données en stockant les partitionnements afin d'appliquer des méthodes d'apprentissage par renforcement pour en déduire un tout nouveau partitionnement du domaine. Une autre application des méthodes d'apprentissage plus facile à mettre en place est de prédire quel est le meilleur jeu de paramètres en fonction du type de simulation, du pas de temps, du nombre d'atomes et des ressources matérielles allouées à la simulation à partir des données des simulations déjà effectuées (méthodes supervisées).

Interactions à longue portée :

Dans ce manuscrit nous nous sommes exclusivement focalisés sur l'optimisation des codes de DM pour des simulations dont les interactions à longue portée sont négligeables. Nous avons de fortes raisons de penser que l'AMR permet aussi d'améliorer les temps d'exécution de simulations comprenant également des interactions à longue portée. En effet, l'une des méthodes les plus abouties pour traiter ces interactions sur un très grand nombre de cœurs est la méthode *fast multipole* [61]. Celle-ci est basée sur une décomposition du domaine en un octree. Cette méthode est applicable avec des tâches suivant un Graphe orienté acyclique et a déjà été étudiée avec un grand nombre de tâches en parallèle [3]. Il est donc possible d'établir un lien entre l'octree de la méthode *fast multipole* et celui de la méthodeAMR tout en combinant notre graphe de dépendances de tâches avec celui de la méthode *fast multipole* afin de limiter l'inactivité des threads.

11 Annexes

Chapitre architecture Amr

Comparaison Machines

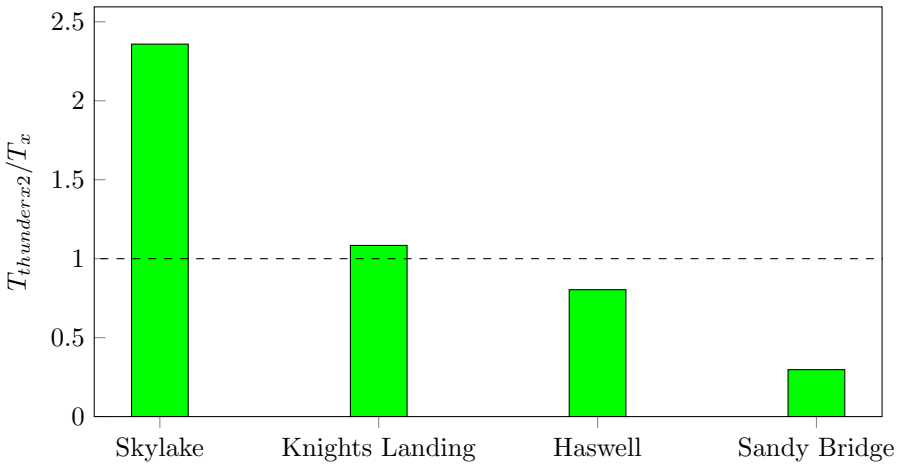


Figure 11.1 | Temps de simulations d'ExaSTAMP avec une grille Amr pour un cristal parfait avec un potentiel MEAM. Les temps sont rapportés pour différents processeurs normalisés par le temps du processeur thunderx2 (arm).

La figure 11.1 montre les temps d'exécution pour une même simulation sur différentes architectures de processeurs. La simulation consiste à simuler sur 100 pas de temps de cristal parfait d'étain avec le potentiel Meam. ExaSTAMP Amr est compilé avec les options suivantes pour chaque architecture :

- Thunderx2 : -Ofast -march=armv8.1-a -mcpu=thunderx2t99 -mtune=thunderx2t99 -funroll-loops;
- Skl : -O3 -xCore-AVX512;
- Knl : -O3 -xMIC-AVX512;
- Haswell :-O3 -mavx;
- Sandy Bridge :-O3 -mavx;

Les simulations sont alors exécuté avec la parallélisation suivante :

- Thunderx2 : 2 processus Mpi, 32 cœurs par processus Mpi et 2 threads OpenMP par cœurs;
- Skl : 2 processus Mpi, 24 cœurs par processus Mpi et 2 threads OpenMP par cœurs;
- Knl : 1 processus Mpi, 64 cœurs par processus Mpi et 4 threads OpenMP par cœurs;
- Haswell : 4 processus Mpi, 8 cœurs par processus Mpi et 2 threads OpenMP par cœurs;
- Sandy Bridge : 2 processus Mpi, 8 cœurs par processus Mpi et 1 threads OpenMP par cœurs;

Chapitre parallélisation Mpi

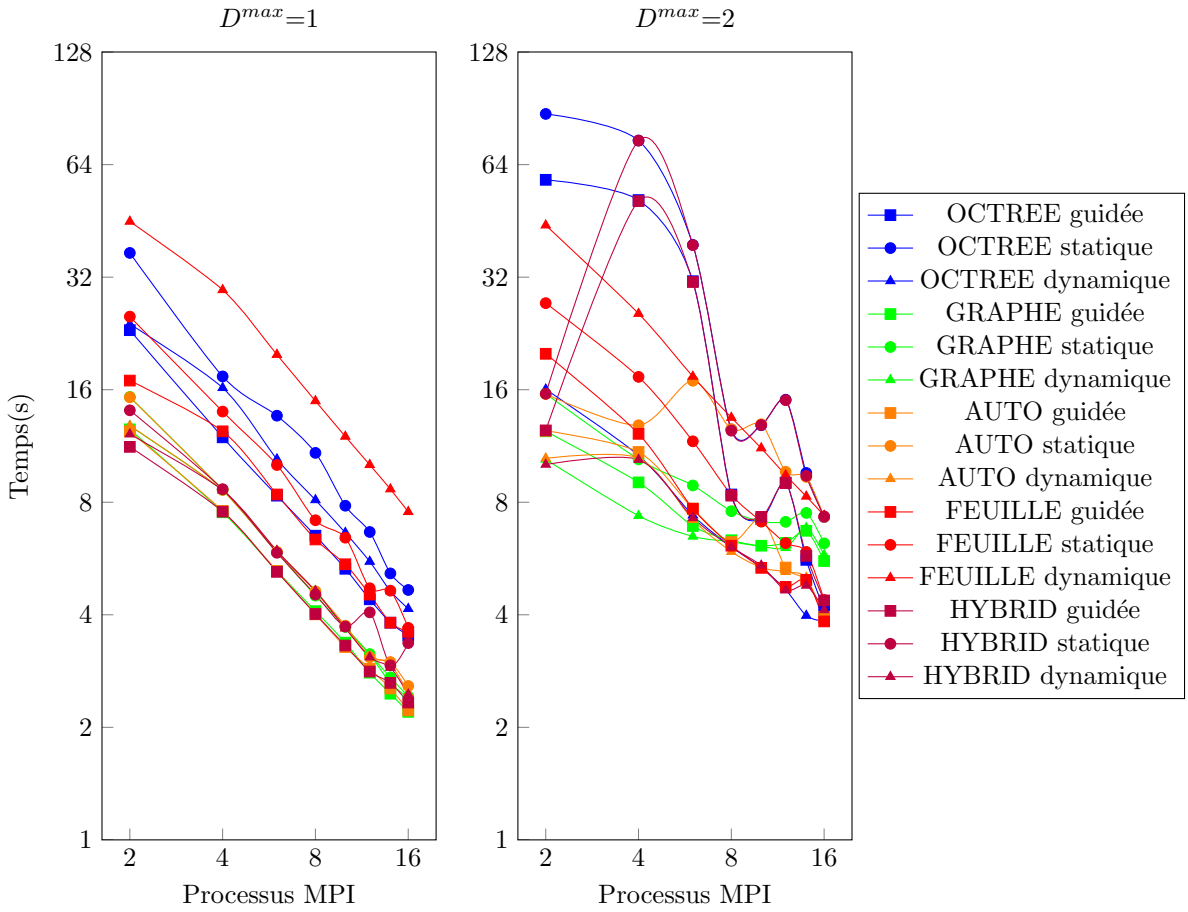


Figure 11.2 | Temps de simulations d'ExaSTAMP avec une grille Amr des simulations d'un micro-jet pour les méthodes de parallélisation OpenMP par graphe, octrees et feuilles selon les politiques d'ordonnancement statique, dynamique et guidée. Le potentiel utilisé pour modéliser les interactions étain-étain est le potentiel Lj.

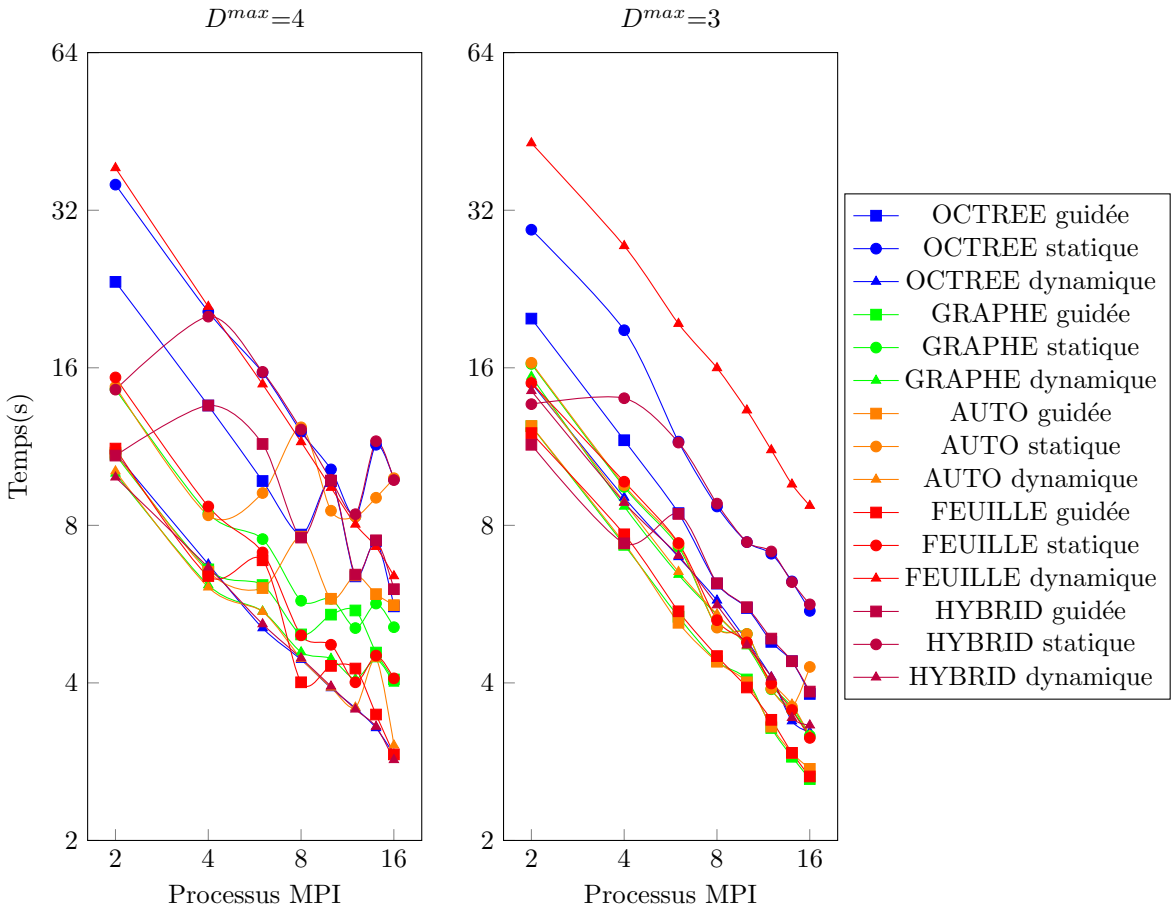


Figure 11.3 | Temps de simulations d'ExaSTAMP avec une grille Amr des scénarios impact d'une nano-goutte pour les méthodes de parallélisation OpenMP par graphe, octrees et feuilles selon les politiques d'ordonnancement statique, dynamique et guidée. Le potentiel utilisé pour modéliser les interactions étain-étain est le potentiel Lj.

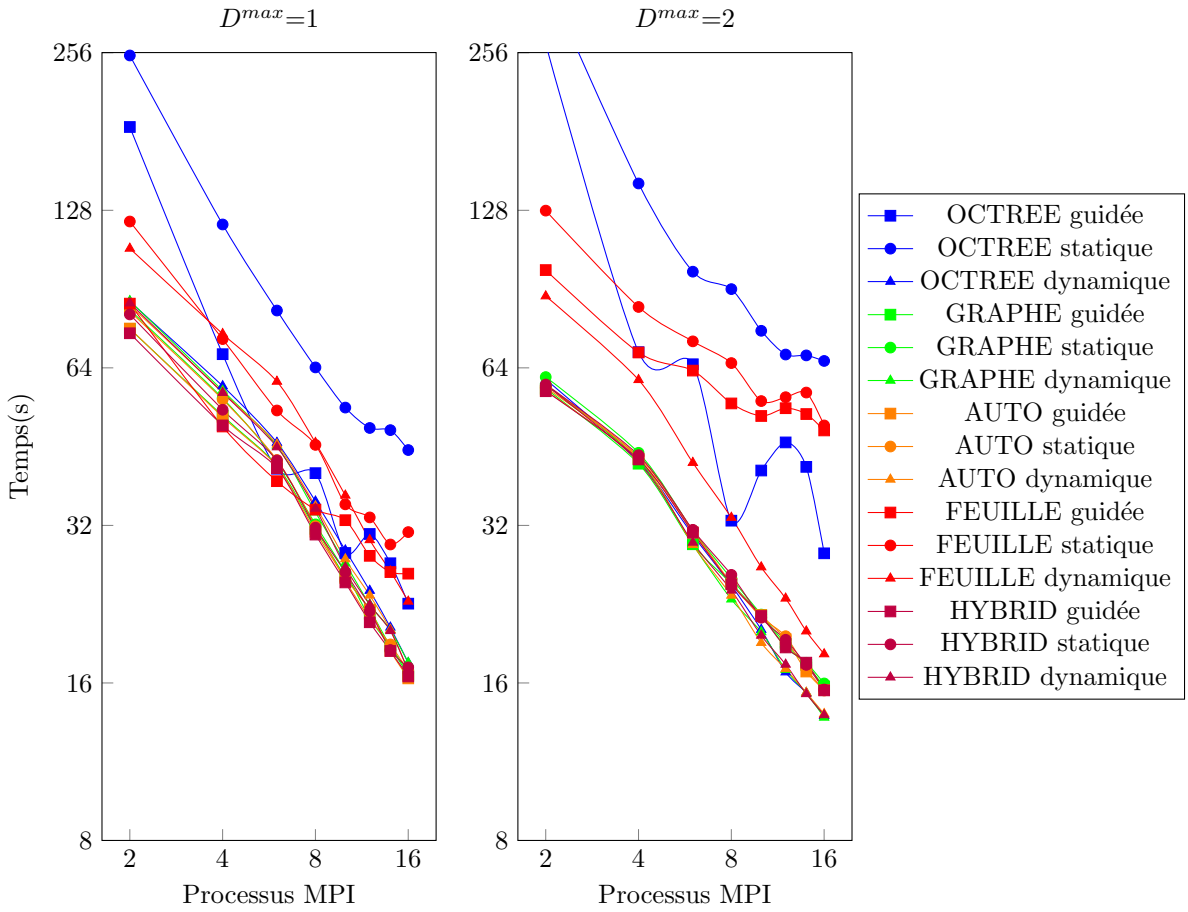


Figure 11.4 | Temps de simulations d'ExaSTAMP avec une grille Amr des simulations d'un micro-jet pour les méthodes de parallélisation OpenMP par graphe, octrees et feuilles selon les politiques d'ordonnancement statique, dynamique et guidée. Le potentiel utilisé pour modéliser les interactions étain-étain est le potentiel Meam.

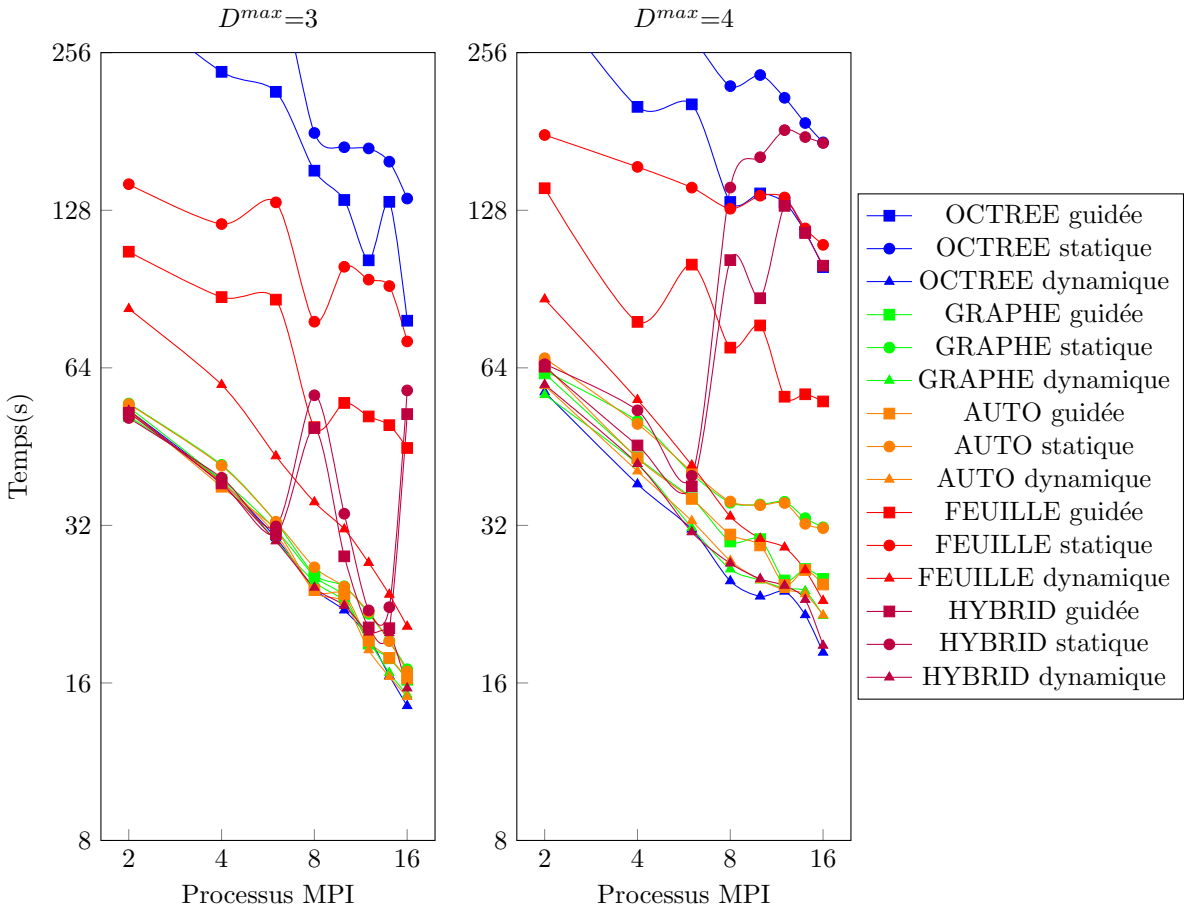


Figure 11.5 | Temps de simulations d'ExaSTAMP avec une grille Amr des scénarios impact d'une nano-goutte pour les méthodes de parallélisation OpenMP par graphe, octrees et feuilles selon les politiques d'ordonnancement statique, dynamique et guidée. Le potentiel utilisé pour modéliser les interactions étain-étain est le potentiel Meam.

Glossaire

Bull eXascale Interconnect Réseaux d'interconnexion entre les Nœuds d'un supercalculateur développé par l'entreprise Bull. 31, 135

Courbe de Peano La courbe de Peano est une fonction surjective et continue de l'intervalle unitaire sur le carré unitaire. 51

Entier de Morton L'entier de Morton indexe linéairement des données multidimensionnelles, préservant la localité des données à plusieurs niveaux. 47

Graphe orienté acyclique Graphe composé de sommet d'arc orienté contenant des puits et des sources. 38, 104, 150

GROningen MAchine for Chemical Simulations Logiciel de simulation en dynamique moléculaire développé initialement par l'Université de Groningen, et à présent maintenu et étendu par différentes organisations, notamment l'Université d'Uppsala, l'Université de Stockholm et l'Institut Max-Planck de recherche sur les polymères. 60

hypergraphe Sous ensemble de dimension inférieur comme par exemple une droite pour un plan. 50, 52, 149

hyperplan Graphe dont les arrêtes relies 2 ou davantage de sommets. 46, 50

Intel Thread Building Block Librairie pour la parallélisation en mémoire partagée. 37

Message Passing Interface Librairie utilisée pour la parallélisation en mémoire distribuée. 36

Open Multi-Processing Librairie pour la parallélisation en mémoire partagée. 37, 91

Posix Thread API permettant de manager des threads. 37, 53

Processeur à jeu d'instructions réduit Type de jeu d'instruction utilisé par certains processeurs. 32

Sauvegarde Fichier permettant à un code de réaliser une reprise d'une simulation déjà commencée.. 91

Tuple C'est un ensemble d'éléments comparable aux listes mais qui, une fois déclaré, ne peut plus être modifié. Les éléments peuvent être de n'importe quel type. 110

Unités Arithmétiques et Logiques Partie matérielle du processeur effectuant les calculs. 30, 32

Liste des acronymes

Api Interface de Programmation Applicative. 52, 56, 57, 120, *Glossary:* Interface de Programmation Applicative

Eam Embedded Atom Model. 17, 19–21, 24, 54, 57, 60, 66, 81–83, 85–87, 90, 94, 121, 147, 148

Gpu Processeur Graphique. 34, 37, 38, 59, 60

Meam Modified Embedded Atom Model. 17, 19, 21, 24, 54, 66, 85, 87, 89, 91, 94–101, 105–108, 121–123, 125, 127–129, 131–145, 147, 148, 151, 154, 155

Mpi Message Passing Interface. 7, 11, 36, 49, 50, 52–55, 57–60, 66, 69, 73, 109–111, 113–116, 120, 122, 124–132, 134, 140–145, 149, 152, *Glossary:* Message Passing Interface

Numa Non Uniform Memory Access. 30–32, 36, 80, 96

ram Random Access Memory. 30, 91

rCut Rayon de coupure. 17, 19, 20, 42, 47, 57, 64, 80, 83, 85, 91, 95, 97, 101, 102, 123, 135, 138, 145

rVerlet Rayon de Verlet. 17, 18

Simd Simple Instruction Multiple Données. 5, 33, 34, 39, 40, 56, 60, 63, 64, 73, 76, 78, 81–86, 88, 89, 136, 147

Tbb Intel Thread Building Block. 37, 38, 54–56, 66, 91, 93, 119, *Glossary:* Intel Thread Building Block

Alu Unités Arithmétiques et Logiques. 30–32, 34, 37, *Glossary:* Unités Arithmétiques et Logiques

Amr Raffinement de Maillage Adaptatif. 6, 7, 10, 11, 42, 44–47, 51, 52, 63, 67–70, 72, 77–80, 82, 88–102, 105–110, 112–121, 127, 128, 132–137, 140, 141, 143, 145–155

Bxi Bull eXascale Interconnect. 31, 135, *Glossary:* Bull eXascale Interconnect

Cea Commissariat à l'énergie atomique et aux énergies alternatives. 10, 13, 30, 54, 89, 135, *Glossary:* Commissariat à l'énergie atomique et aux énergies alternatives

Dag Graphe orienté acyclique. 38, 104, 107, 117, 118, 147, 148, 150, *Glossary:* Graphe orienté acyclique

DM Dynamique Moléculaire classique. 6, 9–11, 14–17, 20, 25, 27, 41, 42, 45, 47, 49, 52, 54, 58–61, 63, 65, 68, 77, 80, 82, 88, 92, 95, 102, 123, 135, 146–150

DRB Dual Recursive Bipartitioning. 53

EXASTAMP Exaflopique de Simulation Temporelle Atomistique et Moléculaire Parallèle. 7, 10, 11, 26, 54–59, 61, 63–66, 68–70, 76–78, 80, 82, 83, 88–93, 95–100, 102, 105–107, 109–113, 115–117, 120, 121, 134, 135, 140, 141, 143, 145–148, 151–155

GROMACS GROningen MAchine for Chemical Simulations. 60, *Glossary:* GROningen MAchine for Chemical Simulations

KNL Intel Xeon Phi Knights Landing. 10, 58, 81, 82, 86, 87, 89–91, 95, 96, 98, 100, 105, 106, 110, 115, 116, 123, 134–136, 140–145, 147, 148, 151, 152

LAMMPS Large-Scale Atomic/Molecular Massively Parallel Simulator. 7, 11, 54, 58–61, 66, 82, 89, 97–99, 106, 107, 110, 111, 115, 116, 128, 133–135, 145, 146, 148

LJ Lennard Jones. 20, 54, 55, 60, 64, 80–83, 86, 87, 89, 94–102, 105–108, 115, 120, 121, 125, 127–129, 131–134, 140–145, 147, 148, 152, 153

NPT Ensemble «T-p». 54

NVE Microcanonique. 14, 15, 54

NVT Canonique. 14, 15, 54

OPENMP Open Multi-Processing. 7, 11, 37–39, 54–56, 58–60, 91–94, 97–100, 106, 107, 109, 110, 112, 114, 116, 119, 120, 125–127, 132, 133, 141–143, 145, 152–155, *Glossary:* Open Multi-Processing

PHG Partitionning Hypergraph and Graph. 122, 124, 125, 128–130, 140, 141

PARMETIS ParMetis. 61, 122, 129–132, 134, 135, 140–144, 148, 149

PBE Partition Binaire de l'Espace. 46, 47, 50, 51

RCB Recursive Coordinate Bisection. 50, 57, 59, 61, 122, 124–131, 134, 140–145, 148

RIB Recursive Inertial Bisection. 50, 57, 61, 122, 124, 128–130, 132, 140, 141, 143, 144, 148

RISC Processeur à jeu d'instructions réduit. 32, 33, *Glossary:* Processeur à jeu d'instructions réduit

SFC Space Filling Curve. 50, 51, 57, 61, 122, 124, 128, 129, 132, 140–144, 148

SNAP Spectral Neighbor Analysis Potential. 24, 54

SISD Simple Instruction Simple Donnée. 34

SKL Intel Xeon Skylake. 65, 81, 82, 86, 87, 89–91, 95, 96, 98, 100, 103, 105, 106, 114–116, 123, 126–129, 131, 134, 151, 152

SMP Multiprocesseur Symétrique à mémoire Partagée. 30

SMT Simultaneous Multithreading. 37

UCT Unité Centrale de Traitement. 30, 34, 35, 37, 38, 59

pThread Posix Thread. 37, 53, *Glossary:* Posix Thread

Uma Uniform Memory Access. 30, 31, 80

AIMD Dynamique Moléculaire *ab initio*. 13

DEM Méthode par Éléments Discrets. 14

DPD Dynamique des Particules Dissipatives. 14

k-d-tree k-dimensional tree. 46, 47, *Glossary:* k-dimensional tree

octree Octree. 47, *Glossary:* Octree

Bibliographie

[1] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C Smith, Berk Hess, and Erik Lindahl. Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1:19–25, 2015.
➢ Cité à la page 60.

[2] SA Adelman and JD Doll. Generalized langevin equation approach for atom/solid-surface scattering: General formulation for classical scattering off harmonic solids. *The Journal of chemical physics*, 64(6):2375–2388, 1976.
➢ Cité à la page 15.

[3] Emmanuel Agullo, Bérenger Bramas, Olivier Coulaud, Eric Darve, Matthias Messner, and Toru Takahashi. Task-based fmm for multicore architectures. *SIAM Journal on Scientific Computing*, 36(1):C66–C93, 2014.
➢ Cité à la page 150.

[4] Michael P Allen and Dominic J Tildesley. *Computer simulation of liquids*. Oxford university press, 2017.
➢ Cité à la page 9.

[5] M.P. Allen and D.J. Tildesley. *Computer Simulation of Liquids*. Clarendon Press, 1987.
➢ Cités aux pages 10, 18, and 20.

[6] Ann S Almgren, John B Bell, Mike J Lijewski, Zarija Lukić, and Ethan Van Andel. Nyx: A massively parallel amr code for computational cosmology. *The Astrophysical Journal*, 765(1):39, 2013.
➢ Cité à la page 46.

[7] AS Almgren, VE Beckner, JB Bell, MS Day, LH Howell, CC Joggerst, MJ Lijewski, A Nonaka, M Singer, and M Zingale. Castro: A new compressible astrophysical solver. i. hydrodynamics and self-gravity. *The Astrophysical Journal*, 715(2):1221, 2010.
➢ Cité à la page 46.

[8] Hans C Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *The Journal of chemical physics*, 72(4):2384–2393, 1980.
➢ Cité à la page 14.

[9] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198, 2011.
➢ Cité à la page 38.

[10] Shiraz D Aziz and Sanjeev Chandra. Impact, recoil and splashing of molten metal droplets. *International journal of heat and mass transfer*, 43(16):2841–2857, 2000.
➢ Cité à la page 27.

[11] Stephen T Barnard and Horst D Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and experience*, 6(2):101–117, 1994.
➢ Cité à la page 52.

[12] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
➢ Cité à la page 47.

[13] Herman JC Berendsen, JPM van Postma, Wilfred F van Gunsteren, ARHJ DiNola, and JR Haak. Molecular dynamics with coupling to an external bath. *The Journal of chemical physics*, 81(8):3684–3690, 1984.
➢ Cité à la page 14.

[14] Herman JC Berendsen, David van der Spoel, and Rudi van Drunen. Gromacs: a message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91(1-3):43–56, 1995.
➢ Cité à la page 60.

[15] Marsha J Berger and Shahid H Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, (5):570–580, 1987.
➢ Cités aux pages 46 and 50.

[16] Marsha J Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of computational Physics*, 82(1):64–84, 1989.
➢ Cités aux pages 10 and 46.

[17] Marsha J Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of computational Physics*, 53(3):484–512, 1984.
➢ Cités aux pages 10 and 46.

[18] Robert D Blumofe, Christopher F Joerg, Bradley C Kuszmaul, Charles E Leiserson, Keith H Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *Journal of parallel and distributed computing*, 37(1):55–69, 1996.
➢ Cité à la page 38.

[19] Robert D Blumofe and Charles E Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM (JACM)*, 46(5):720–748, 1999.
➢ Cité à la page 38.

[20] Donald W Brenner. Empirical potential for hydrocarbons for use in simulating the chemical vapor deposition of diamond films. *Physical review B*, 42(15):9458, 1990.
➢ Cité à la page 24.

[21] W Michael Brown, Jan-Michael Y Carrillo, Nitin Gavhane, Foram M Thakkar, and Steven J Plimpton. Optimizing legacy molecular dynamics software with directive-based offload. *Computer Physics Communications*, 195:95–101, 2015.
➢ Cité à la page 58.

[22] Greg L Bryan, Michael L Norman, Brian W O'Shea, Tom Abel, John H Wise, Matthew J Turk, Daniel R Reynolds, David C Collins, Peng Wang, Samuel W Skillman, et al. Enzo: An adaptive mesh refinement code for astrophysics. *The Astrophysical Journal Supplement Series*, 211(2):19, 2014.
➢ Cité à la page 46.

[23] Richard A Buckingham. The classical equation of state of gaseous helium, neon and argon. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 168(933):264–283, 1938.
➢ Cité à la page 19.

[24] Thang Nguyen Bui and Curt Jones. A heuristic for reducing fill-in in sparse matrix factorization. Technical report, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA ..., 1993.
➢ Cité à la page 52.

[25] David R Butenhof. *Programming with POSIX threads*. Addison-Wesley Professional, 1997.
➢ Cité à la page 37.

[26] Herbert B Callen and Theodore A Welton. Irreversibility and generalized noise. *Physical Review*, 83(1):34, 1951.
➢ Cité à la page 15.

[27] Cédric Chevalier and François Pellegrini. Pt-scotch: A tool for efficient parallel graph ordering. *Parallel computing*, 34(6-8):318–331, 2008.
➢ Cité à la page 53.

[28] L Paul Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1-4):97–108, 1989.
➢ Cité à la page 44.

[29] Emmanuel Cieren. *Molecular Dynamics for Exascale Supercomputers*. PhD thesis, Bordeaux, 2015.
➢ Cités aux pages 10, 26, 56, 57, 109, 110, and 120.

[30] Emmanuel Cieren, Laurent Colombet, Samuel Pitoiset, and Raymond Namyst. Exastamp: A parallel framework for molecular dynamics on heterogeneous clusters. In *European Conference on Parallel Processing*, pages 121–132. Springer, 2014.
➢ Cités aux pages 10, 26, and 54.

[31] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The mpi message passing interface standard. In *Programming environments for massively parallel distributed systems*, pages 213–218. Springer, 1994.
➢ Cités aux pages 11 and 36.

[32] P Colella, DT Graves, TJ Ligocki, DF Martin, D Modiano, DB Serafini, and B Van Straalen. Chombo software package for amr applications design document. *Available at the Chombo website: http://seesar. lbl. gov-/ANAG/chombo/(September 2008)*, 2009.
➢ Cité à la page 46.

[33] Jason Cong and M Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi design. In *30th ACM/IEEE Design Automation Conference*, pages 755–760. IEEE, 1993.
➢ Cité à la page 53.

[34] K Coulomb, M Faverge, J Jazeix, O Lagrasse, J Marcoueille, P Noisette, A Redondy, and C Vuchener. Visual trace explorer (vite). Technical report, Technical report, 2009.
➢ Cité à la page 103.

[35] Leonardo Dagum and Ramesh Menon. Openmp: An industry-standard api for shared-memory programming. *Computing in Science & Engineering*, (1):46–55, 1998.
➢ Cités aux pages 11 and 37.

[36] Murray S Daw and M Io Baskes. Semiempirical, quantum mechanical calculation of hydrogen embrittlement in metals. *Physical review letters*, 50(17):1285, 1983.
➢ Cité à la page 20.

[37] Murray S Daw and Michael I Baskes. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Physical Review B*, 29(12):6443, 1984.
➢ Cités aux pages 20 and 21.

[38] Murray S Daw, Stephen M Foiles, and Michael I Baskes. The embedded-atom method: a review of theory and applications. *Materials Science Reports*, 9(7-8):251–310, 1993.
➢ Cité à la page 20.

[39] Mark S Day and John B Bell. Numerical simulation of laminar reacting flows with complex chemistry. *Combustion Theory and Modelling*, 4(4):535–556, 2000.
➢ Cité à la page 46.

[40] T De Rességuier, E Lescoute, A Sollier, G Prudhomme, and P Mercier. Microjetting from grooved surfaces in metallic samples subjected to laser driven shocks. *Journal of Applied Physics*, 115(4):043525, 2014.
➢ Cité à la page 25.

[41] Saïd Derradji, Thibaut Palfer-Sollier, Jean-Pierre Panziera, Axel Poudes, and François Wellenreiter Atos. The bxi interconnect architecture. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 18–25. IEEE, 2015.
➢ Cité à la page 31.

[42] Karen D Devine, Erik G Boman, Robert T Heaphy, Rob H Bisseling, and Umit V Catalyurek. Parallel hypergraph partitioning for scientific computing. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 10–pp. IEEE, 2006.
➢ Cité à la page 122.

[43] Estelle Dirand, Laurent Colombet, and Bruno Raffin. Tins: A task-based dynamic helper core strategy for in situ analytics. In *Asian Conference on Supercomputing Frontiers*, pages 159–178. Springer, 2018.
➢ Cité à la page 54.

[44] Lamia Djoudi, Denis Barthou, Patrick Carribault, Christophe Lemuet, Jean-Thomas Acquaviva, William Jalby, et al. Maqao: Modular assembler quality analyzer and optimizer for itanium 2. In *The 4th Workshop on EPIC architectures and compiler technology, San Jose*, volume 200, 2005.
➢ Cité à la page 92.

[45] Anshu Dubey, Katie Antypas, Murali K Ganapathy, Lynn B Reid, Katherine Riley, Dan Sheeler, Andrew Siegel, and Klaus Weide. Extensible component-based architecture for flash, a massively parallel, multiphysics simulation code. *Parallel Computing*, 35(10-11):512–522, 2009.
➢ Cité à la page 46.

[46] Ralph Duncan. A survey of parallel computer architectures. *Computer*, 23(2):5–16, 1990.
➢ Cité à la page 33.

[47] Olivier Durand and Laurent Soulard. Modeling from molecular dynamics simulations of ejecta production induced by shock-loaded metallic surfaces. *Journal of Dynamic Behavior of Materials*, 3(2):280–290, 2017.
➢ Cités aux pages 9, 25, 26, and 27.

[48] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202 – 3216, 2014. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.
➢ Cité à la page 58.

[49] Roger Espasa, Mateo Valero, and James E Smith. Vector architectures: past, present and future. In *Proceedings of the 12th international conference on Supercomputing*, pages 425–432. ACM, 1998.
➢ Cité à la page 33.

[50] Denis J Evans and Brad Lee Holian. The nose–hoover thermostat. *The Journal of chemical physics*, 83(8):4069–4074, 1985.
➢ Cité à la page 14.

[51] P Ewald. The calculations of optical and electrostatic lattice potentials. *Ann. Phys.*, 64:253–287, 1921.
➢ Cité à la page 17.

[52] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181. IEEE, 1982.
➢ Cité à la page 52.

[53] Michael J Flynn. Some computer organizations and their effectiveness. *IEEE transactions on computers*, 100(9):948–960, 1972.
➢ Cité à la page 33.

[54] SM Foiles, MI Baskes, and Murray S Daw. Embedded-atom-method functions for the fcc metals cu, ag, au, ni, pd, pt, and their alloys. *Physical review B*, 33(12):7983, 1986.
➢ Cité à la page 20.

[55] Pascal Jean Frey and Paul-Louis George. *Maillages: applications aux éléments finis*. Hermès Science Publications, 1999.
➢ Cités aux pages 42 and 43.

[56] Bruce Fryxell, Kevin Olson, Paul Ricker, FX Timmes, Michael Zingale, DQ Lamb, Peter MacNeice, Robert Rosner, JW Truran, and H Tufo. Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131(1):273, 2000.
➢ Cités aux pages 45 and 46.

[57] Haohuan Fu, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, Fangli Qiao, et al. The sunway taihulight supercomputer: system and applications. *Science China Information Sciences*, 59(7):072001, 2016.
➢ Cité à la page 9.

[58] J Willard Gibbs. *Elementary principles in statistical mechanics*. Courier Corporation, 2014.
➢ Cité à la page 14.

[59] Josiah Willard Gibbs. *Elementary principles in statistical mechanics developed with especial reference to the rational foundation of thermodynamics/by J. Willard Gibbs*. New York:: C. Scribner, 1902.
➢ Cité à la page 14.

[60] Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Massó, Thomas Radke, Edward Seidel, and John Shalf.

The cactus framework and toolkit: Design and applications. In *International Conference on High Performance Computing for Computational Science*, pages 197–227. Springer, 2002.
➢ Cité à la page 46.

[61] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.
➢ Cité à la page 150.

[62] Brian J Gribben. *Application of the multiblock method in computational aerodynamics*. PhD thesis, University of Glasgow, 1998.
➢ Cité à la page 44.

[63] François Hermeline. Triangulation automatique d'un polyèdre en dimension. *RAIRO. Analyse numérique*, 16(3):211–242, 1982.
➢ Cité à la page 44.

[64] Berk Hess, Carsten Kutzner, David Van Der Spoel, and Erik Lindahl. Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of chemical theory and computation*, 4(3):435–447, 2008.
➢ Cité à la page 60.

[65] Roger W Hockney and James W Eastwood. *Computer simulation using particles*. crc Press, 1988.
➢ Cité à la page 17.

[66] William G Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Physical review A*, 31(3):1695, 1985.
➢ Cité à la page 14.

[67] William G Hoover, Anthony JC Ladd, and Bill Moran. High-strain-rate plastic flow studied via nonequilibrium molecular dynamics. *Physical Review Letters*, 48(26):1818, 1982.
➢ Cité à la page 14.

[68] Reazul Hoque, Thomas Herault, George Bosilca, and Jack Dongarra. Dynamic task discovery in parsec: A data-flow task-based runtime. In *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, page 6. ACM, 2017.
➢ Cité à la page 36.

[69] John Edward Jones. On the determination of molecular fields.—ii. from the equation of state of a gas. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 106(738):463–477, 1924.
➢ Cité à la page 19.

[70] George Karypis. Multi-constraint mesh partitioning for contact/impact computations. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 56. ACM, 2003.
➢ Cité à la page 52.

[71] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(1):69–79, 1999.
➢ Cité à la page 52.

[72] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.
➢ Cité à la page 52.

[73] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.
➢ Cité à la page 52.

[74] George Karypis and Vipin Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, 1998.
➢ Cités aux pages 52 and 53.

[75] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3):285–300, 2000.
➢ Cité à la page 52.

[76] George Karypis, Kirk Schloegel, and Vipin Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. *Version 1.0, Dept. of Computer Science, University of Minnesota*, page 22, 1997.
➢ Cité à la page 52.

[77] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970.
➢ Cités aux pages 52 and 54.

[78] T Kiang. Random fragmentation in two and three dimensions. *Zeitschrift fur Astrophysik*, 64:433, 1966.
➢ Cité à la page 44.

[79] Joel Koplik and Rui Zhang. Nanodrop impact on solid surfaces. *Physics of Fluids*, 25(2):022003, 2013.
➢ Cités aux pages 9 and 26.

[80] Dominique LaSalle and George Karypis. Multithreaded graph partitioning. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 225–236. IEEE, 2013.
➢ Cité à la page 52.

[81] Der-Tsai Lee. Proximity and reachability in the plane. Technical report, ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB, 1978.
➢ Cité à la page 44.

[82] SY Liem, D Brown, and Julian HR Clarke. Molecular dynamics simulations on distributed memory machines. *Computer Physics Communications*, 67(2):261–267, 1991.
➢ Cité à la page 60.

[83] M Lijewski, A Nonaka, and J Bell. Boxlib, 2011.
➢ Cité à la page 46.

[84] Frank Löffler, Joshua Faber, Eloisa Bentivegna, Tanja Bode, Peter Diener, Roland Haas, Ian Hinder, Bruno C Mundim, Christian D Ott, Erik Schnetter, et al. The einstein toolkit: a community computational infrastructure for relativistic astrophysics. *Classical and Quantum Gravity*, 29(11):115001, 2012.
➢ Cité à la page 46.

[85] Rainald Löhner and Paresh Parikh. Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal for Numerical Methods in Fluids*, 8(10):1135–1149, 1988.
➢ Cité à la page 44.

[86] Fritz London. The general theory of molecular forces. *Transactions of the Faraday Society*, 33:8b–26, 1937.
➢ Cité à la page 20.

[87] Peter MacNeice, Kevin M Olson, Clark Mobarry, Rosalinda De Fainchtein, and Charles Packer. Paramesh: A parallel adaptive mesh refinement community toolkit. *Computer physics communications*, 126(3):330–354, 2000.
➢ Cité à la page 46.

[88] Chris M Mangiardi and R Meyer. Molecular-dynamics simulations using spatial decomposition and task-based parallelism. In *Mathematical and Computational Approaches in Advancing Modern Science and Engineering*, pages 133–140. Springer, 2016.
➢ Cité à la page 101.

[89] Chris M Mangiardi and Ralf Meyer. A hybrid algorithm for parallel molecular dynamics simulations. *Computer Physics Communications*, 219:196–208, 2017.
➢ Cité à la page 101.

[90] Michael McCool, James Reinders, and Arch Robison. *Structured parallel programming: patterns for efficient computation*. Elsevier, 2012.
➢ Cité à la page 37.

[91] EE Meshkov. Instability in shock-accelerated boundary separating two gasses. *Izv. Akad. Nauk SSSR, Mekh. Zhidk. Gasa*, 5:151–158, 1969.
➢ Cité à la page 25.

[92] Ralf Meyer. Efficient parallelization of short-range molecular dynamics simulations on many-core systems. *Physical Review E*, 88(5):053309, 2013.
➢ Cité à la page 101.

[93] Gary L Miller, S-H Teng, and Stephen A Vavasis. A unified geometric approach to graph separators. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 538–547. IEEE, 1991.
➢ Cité à la page 52.

[94] Francesco Miniati and Phillip Colella. Block structured adaptive mesh and time refinement for hybrid, hyperbolic+ n-body systems. *Journal of Computational Physics*, 227(1):400–430, 2007.
➢ Cité à la page 46.

[95] Francesco Miniati and Daniel F Martin. Constrained-transport magnetohydrodynamics with adaptive mesh refinement in charm. *The Astrophysical Journal Supplement Series*, 195(1):5, 2011.
➢ Cité à la page 46.

[96] Jamaludin Mohd-Yusof, Sriram Swaminarayan, and TC Germann. Co-design for molecular dynamics: An exascale proxy application. *Los Alamos National Laboratory, Tech. Rep. LA-UR*, pages 13–20839, 2013.
➢ Cité à la page 60.

[97] Gert Moliere. Theorie der streuung schneller geladener teilchen i. einzelstreuung am abgeschirmten coulomb-feld. *Zeitschrift für Naturforschung A*, 2(3):133–145, 1947.
➢ Cité à la page 19.

[98] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.
➢ Cité à la page 30.

[99] Philip M Morse. Diatomic molecules according to the wave mechanics. ii. vibrational levels. *Physical Review*, 34(1):57, 1929.
➢ Cité à la page 19.

[100] Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966.
➢ Cité à la page 47.

[101] Andrew Nonaka, AS Almgren, JB Bell, MJ Lijewski, CM Malone, and M Zingale. Maestro: An adaptive low mach number hydrodynamics algorithm for stellar flows. *The Astrophysical Journal Supplement Series*, 188(2):358, 2010.
➢ Cité à la page 46.

[102] Shūichi Nosé. A molecular dynamics method for simulations in the canonical ensemble. *Molecular physics*, 52(2):255–268, 1984.
➢ Cité à la page 14.

[103] Shuichi Nosé. A unified formulation of the constant temperature molecular dynamics methods. *The Journal of chemical physics*, 81(1):511–519, 1984.
➢ Cité à la page 14.

[104] Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1):157–160, 1890.
➢ Cité à la page 51.

[105] Olga Pearce, Todd Gamblin, Bronis R De Supinski, Martin Schulz, and Nancy M Amato. Quantifying the effectiveness of load balance algorithms. In *Proceedings of the 26th ACM international conference on Supercomputing*, pages 185–194. ACM, 2012.
➢ Cité à la page 122.

[106] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*, pages 493–498. Springer, 1996.
➢ Cité à la page 53.

[107] Gregory F Pfister. An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O*, 42:617–632, 2001.
➢ Cité à la page 31.

[108] John R Pilkington and Scott B Baden. Partitioning with spacefilling curves. 1994.
➢ Cité à la page 51.

[109] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.
➢ Cités aux pages 11, 49, and 58.

[110] Alex Pothen, Horst D Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM journal on matrix analysis and applications*, 11(3):430–452, 1990.
➢ Cité à la page 52.

[111] Top500 Project. Top500 – The List. `http://www.top500.org`.
➢ Cités aux pages 9, 29, 31, and 135.

[112] Sivasankaran Rajamanickam and Erik G Boman. An evaluation of the zoltan parallel graph and hypergraph partitioners. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2011.
➢ Cité à la page 122.

[113] Sivasankaran Rajamanickam, Erik G Boman, Karen Dragon Devine, Vitus Joseph Leung, Lee Ann Riesen, Umit Catalyurek, Doruk Bozdag, Cedric Chevalier, and Michael Wolf. Tutorial: The zoltan toolkit. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2011.
➢ Cité à la page 50.

[114] Dennis C Rapaport and Dennis C Rapaport Rapaport. *The art of molecular dynamics simulation*. Cambridge university press, 2004.
➢ Cité à la page 9.

[115] James Reinders. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. " O'Reilly Media, Inc.", 2007.
➢ Cité à la page 37.

[116] Lewis Fry Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 210:307–357, 1911.
➢ Cité à la page 46.

[117] Robert D Richtmyer. Taylor instability in shock acceleration of compressible fluids. *Communications on Pure and Applied Mathematics*, 13(2):297–319, 1960.
➢ Cité à la page 25.

[118] Richard M Russell. The cray-1 computer system. *Communications of the ACM*, 21(1):63–72, 1978.
➢ Cité à la page 33.

[119] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.
➢ Cité à la page 37.

[120] Kirk Schloegel, George Karypis, and Vipin Kumar. Parallel multilevel diffusion algorithms for repartitioning of adaptive meshes. *Rapport technique, University of Minnesota, Department of Computer Science and Army HPC Center*, 1997.
➢ Cité à la page 52.

[121] Kirk Schloegel, George Karypis, and Vipin Kumar. Wavefront diffusion and lmsr: Algorithms for dynamic repartitioning of adaptive meshes. *IEEE Transactions on Parallel and Distributed Systems*, 12(5):451–466, 2001.
➢ Cité à la page 54.

[122] T Schneider and E Stoll. Molecular-dynamics study of a three-dimensional one-component model for distortive phase transitions. *Physical Review B*, 17(3):1302, 1978.
➢ Cité à la page 15.

[123] Horst D Simon. Partitioning of unstructured problems for parallel processing. *Computing systems in engineering*, 2(2-3):135–148, 1991.
➢ Cité à la page 50.

[124] W Smith, TR Forester, and IT Todorov. The dl poly classic user manual. *STFC, STFC Daresbury Laboratory, Daresbury, Warrington, Cheshire, WA4 4AD, United Kingdom, version*, 1, 2012.
➢ Cité à la page 59.

[125] Mark Smotherman. Ibm stretch (7030)–aggressive uniprocessor parallelism. *historical reconstruction website, Clemson University. http://www. cs. clemson. edu/˜ mark/stretch. html*, 1999.
➢ Cité à la page 30.

[126] Frank H Stillinger and Thomas A Weber. Computer simulation of local order in condensed phases of silicon. *Physical review B*, 31(8):5262, 1985.
➢ Cité à la page 24.

[127] Frank H Stillinger and Thomas A Weber. Fluorination of the dimerized si (100) surface studied by molecular-dynamics simulation. *Physical review letters*, 62(18):2144, 1989.
➢ Cité à la page 24.

[128] George Gabriel Stokes. *On the effect of the internal friction of fluids on the motion of pendulums*, volume 9. Pitt Press Cambridge, 1851.
➢ Cité à la page 42.

[129] John E Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66, 2010.
➢ Cité à la page 37.

[130] AP Sutton and J Chen. Long-range finnis–sinclair potentials. *Philosophical Magazine Letters*, 61(3):139–146, 1990.
➢ Cités aux pages 20 and 21.

[131] William C Swope, Hans C Andersen, Peter H Berens, and Kent R Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, 76(1):637–649, 1982.
➢ Cité à la page 16.

[132] Valerie E Taylor and Bahram Nour-Omid. A study of the factorization fill-in for a parallel implementation of the finite element method. *International journal for numerical methods in engineering*, 37(22):3809–3823, 1994.
➢ Cité à la page 50.

[133] Jerry Tersoff. New empirical approach for the structure and energy of covalent systems. *Physical Review B*, 37(12):6991, 1988.
➢ Cité à la page 24.

[134] Romain Teyssier. Cosmological hydrodynamics with adaptive mesh refinement-a new high resolution code called ramses. *Astronomy & Astrophysics*, 385(1):337–364, 2002.
➢ Cité à la page 46.

[135] Aidan P Thompson, Laura P Swiler, Christian R Trott, Stephen M Foiles, and Garritt J Tucker. Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials. *Journal of Computational Physics*, 285:316–330, 2015.
➢ Cité à la page 24.

[136] MBBJM Tuckerman, Bruce J Berne, and Glenn J Martyna. Reversible multiple time scale molecular dynamics. *The Journal of chemical physics*, 97(3):1990–2001, 1992.
➢ Cité à la page 16.

[137] Dean M Tullsen, Susan J Eggers, and Henry M Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *ACM SIGARCH computer architecture news*, volume 23, pages 392–403. ACM, 1995.
➢ Cité à la page 37.

[138] P Vashishta, Rajiv K Kalia, José P Rino, and Ingvar Ebbsjö. Interaction potential for sio 2: a molecular-dynamics study of structural correlations. *Physical Review B*, 41(17):12197, 1990.
➢ Cité à la page 24.

[139] Loup Verlet. Computer" experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical review*, 159(1):98, 1967.
➢ Cités aux pages 15 and 17.

[140] John Von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.
➢ Cité à la page 34.

[141] Michael S Warren and John K Salmon. A parallel hashed oct-tree n-body algorithm. In *Supercomputing'93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 12–21. IEEE, 1993.
➢ Cité à la page 51.

[142] David F Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The computer journal*, 24(2):167–172, 1981.
➢ Cité à la page 44.

[143] Thomas A Weber and Frank H Stillinger. Dynamical branching during fluorination of the dimerized si (100) surface: A molecular dynamics study. *The Journal of chemical physics*, 92(10):6239–6245, 1990.
➢ Cité à la page 24.

[144] Sandra Wienke, Paul Springer, Christian Terboven, and Dieter an Mey. Openacc—first experiences with real-world applications. In *European Conference on Parallel Processing*, pages 859–870. Springer, 2012.
➢ Cité à la page 37.

[145] D Wolff and WG Rudd. Tabulated potentials in molecular dynamics simulations. *Computer physics communications*, 120(1):20–32, 1999.
➢ Cités aux pages 20 and 24.

[146] Arthur Mason Worthington. *A study of splashes*. Longmans, Green, and Company, 1908.
➢ Cité à la page 26.

[147] Hideki Yahagi and Yuzuru Yoshii. N-body code with adaptive mesh refinement. *The Astrophysical Journal*, 558(1):463, 2001.
➢ Cité à la page 46.

[148] Alexander L Yarin. Drop impact dynamics: splashing, spreading, receding, bouncing.... *Annu. Rev. Fluid Mech.*, 38:159–192, 2006.
➢ Cités aux pages 9 and 26.

[149] W Zhang, L Howell, A Almgren, A Burrows, and J Bell. Castro: A new compressible astrophysical solver. ii. gray radiation hydrodynamics. *The Astrophysical Journal Supplement Series*, 196(2):20, 2011.
➢ Cité à la page 46.

[150] James F Ziegler and Jochen P Biersack. The stopping and range of ions in matter. In *Treatise on Heavy-Ion Science*, pages 93–129. Springer, 1985.
➢ Cité à la page 19.

Dans le contexte de la dynamique moléculaire classique appliquée à la physique de la matière condensée, les chercheurs du CEA étudient des phénomènes physiques à une échelle atomique. Pour cela, il est primordial d'optimiser continuellement les codes de dynamique moléculaire sur les dernières architectures de supercalculateurs massivement parallèles pour permettre aux physiciens d'exploiter la puissance de calcul pour reproduire numériquement des phénomènes physiques toujours plus complexes. Cependant, les codes de simulations doivent être adaptés afin d'équilibrer la répartition de la charge de calcul entre les cœurs d'un supercalculateur.

Pour ce faire, dans cette thèse nous proposons d'incorporer la méthode de raffinement de maillage adaptatif dans le code de dynamique moléculaire ExaSTAMP. L'objectif est principalement d'optimiser la boucle de calcul effectuant le calcul des interactions entre particules grâce à des structures de données multi-threading et vectorisables. La structure permet également de réduire l'empreinte mémoire de la simulation. La conception de l'AMR est guidée par le besoin d'équilibrage de charge et d'adaptabilité soulevé par des ensembles de particules se déplaçant très rapidement au cours du temps.

Les résultats de cette thèse montrent que l'utilisation d'une structure AMR dans ExaSTAMP permet d'améliorer les performances de celui-ci. L'AMR permet notamment de multiplier par 1.31 la vitesse d'exécution de la simulation d'un choc violent entraînant un micro-jet d'étain de 1 milliard 249 millions d'atomes sur 256 KNLs. De plus, l'AMR permet de réaliser des simulations qui jusqu'à présent n'étaient pas concevables comme l'impact d'une nano-goutte d'étain sur une surface solide avec plus 500 millions d'atomes.

Équilibrage dynamique de charge sur supercalculateur exaflopique appliqué à la dynamique moléculaire

Thèse Cea/Dam/Dif – Université de Bordeaux

Raphaël Prat

Sous la direction de Raymond Namyst & Laurent Colombet

2016 – 2019

In the context of classical molecular dynamics applied to condensed matter physics, CEA researchers are studying complex phenomena at the atomic scale. To do this, it is essential to continuously optimize the molecular dynamics codes of recent massively parallel supercomputers to enable physicists to exploit their capacity to numerically reproduce more and more complex physical phenomena. Nevertheless, simulation codes must be adapted to balance the load between the cores of supercomputers.

To do this, in this thesis we propose to incorporate the Adaptive Mesh Refinement method into the ExaSTAMP molecular dynamics code. The main objective is to optimize the computation loop performing the calculation of particle interactions using multi-threaded and vectorizable data structures. The structure also reduces the memory footprint of the simulation. The design of the AMR is guided by the need for load balancing and adaptability raised by sets of particles moving dynamically over time.

The results of this thesis show that using an AMR structure in ExaSTAMP improves its performance. In particular, the AMR makes it possible to execute 1.31 times faster than before the simulation of a violent shock causing a tin microjet of 1 billion 249 million atoms on 256 KNLs. In addition, simulations that were not conceivable so far can be carried out thanks to AMR, such as the impact of a tin nanodroplet on a solid surface with more than 500 million atoms.