Simulations massives de Dynamique des Dislocations : fiabilité et performances sur architectures parallèles et distribuées.

Arnaud Durocher

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Arnaud DUROCHER**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Simulations massives de dynamique des dislocations : fiabilité et performances sur architectures parallèles et distribuées.

Date de soutenance : 19 Décembre 2018

Devant la commission d'examen composée de :

Denis BARTHOU	Professeur, Bordeaux INP	Président du jury
Marc FIVEL	Directeur de Recherche, CNRS	Rapporteur
Jean-François MÉHAUT	Professeur, Université Grenoble Alpes	Rapporteur
Marc BLETRY	Directeur de Recherche, Université Paris XII	Examinateur
Olivier COULAUD	Directeur de Recherche, Inria	Directeur de Thèse
Laurent DUPUY	Ingénieur, CEA	Encadrant

– 2018 –

Résumé

La dynamique des dislocations modélise le comportement de défauts linéiques - les dislocations - présents dans la structure des matériaux cristallins. Il s'agit d'un maillon essentiel de la modélisation multi échelle des matériaux utilisé par exemple dans l'industrie du nucléaire pour caractériser le comportement mécanique et le vieillissement des matériaux sous irradiation. La capacité des dislocations à se multiplier, s'annihiler et interagir pose de nombreux défis informatiques, notamment sur la manière de stocker et traiter de manière efficace les données de la simulation. L'objectif de cette thèse est de répondre aux défis que posent les simulations massives de dynamique des dislocations dans un environnement parallèle et distribué au travers du logiciel Optidis. Dans cette thèse, je propose des améliorations au simulateur Optidis afin de permettre des simulations plus complexes en utilisant la puissance des super-calculateurs.

Mes contributions sont axées sur l'amélioration de la fiabilité et de la performance d'Optidis. La mise en place d'une nouvelle interface d'accès aux données a permis de dissocier l'implémentation des algorithmes de l'optimisation des performances. Cette structure de données permet de meilleures performances tout en améliorant la maintenabilité du code, même lorsque les données sont distribuées. Un nouvel algorithme de gestion des collisions entre dislocations et de formation des jonctions fiable et performant a été mis en place. Des techniques de détection de collision empruntées à la robotique et à l'animation 3D sont utilisées pour accélérer le calcul. S'appuyant sur l'utilisation de la nouvelle structure de données et un traitement des collisions plus élaboré, il permet une gestion de collisions fiable et autorise l'utilisation de pas de temps plus grands. La précision du résultat a été étudiée en se comparant au code NUMODIS, et la performance d'Optidis a été mesurée sur des simulations massives contenant plusieurs millions de segments de dislocations en utilisant plusieurs centaines de cœurs de calcul, démontrant que de telles simulations sont réalisables en un temps raisonnable.

Title

Large scale dislocation dynamics simulation : performance and reliability on parallel and distributed architectures.

Abstract

Dislocation dynamics simulations investigate the behavior of linear defects, called dislocations, in crystalline materials. It is an essential part multiscale modelling of the materials, used for instance in the nuclear industry to characterize the behavior and aging of materials under irradiation. The ability of dislocations to multiply, annihilate and interact presents many challenges, for instance in terms of storage and access to data. This thesis addresses some challenges of dislocation dynamics simulation on parallel and distributed computers. In this thesis, I improve the Optidis simulator to open the way to more complex simulations.

My contributions focuses mainly on improving the reliability and performance of Optidis. A new interface to access simulation data is proposed to dissociate its implementation form the physical algorithms. This data structure allows better performance as well as better code maintainability, even with distributed data. A new fast and reliable collision detection and handling algorithm has been implemented. Collision detection techniques from the robotics and 3D animation industries are used to speedup the detection process. With the use of the new data structure and a more reliable design, this algorithm enables more precise collision handling and the use of a larger simulation timestep. The precision of the results have been measured by comparing Optidis to Numodis. The performance of the code has been studied on larger scale simulations with millions of segments and hundreds of CPU cores, demonstrating that such simulations can now be achieved.

Keywords

Dislocations, high performance computing, simulation

Mots-clés

Dislocations, calcul intensif, simulation

Laboratoire d'accueil

CEA DEN/DMN/SRMA/LC2M; Inria Hiepacs

À ceux que j'aurais voulu voir lire ce manuscrit.

Remerciements

Je souhaite remercier tous ceux qui ont pu aider, de près ou de loin, à la rédaction de cette thèse. Ceux qui m'ont aidé dans le travail scientifique qui a abouti à ce manuscrit, mais aussi ceux qui m'ont soutenu tout au long de ces trois années, dans les bons moments, mais aussi dans les moins bons.

En premier lieu, je veux remercier mes encadrants Olivier COULAUD et Laurent DUPUY. Merci à eux deux pour leurs conseils précieux et pour m'avoir guidé vers le bon chemin pour la rédaction. Merci à Laurent d'avoir eu la patience d'expliquer les dislocations à l'informaticien que je suis.

Je remercie les membres du jury, Marc FIVEL et Jean-François MÉHAUT d'avoir pris le temps de rapporter le manuscrit, et Denis BARTHOU et Marc BLETRY d'avoir accepté d'examiner mes travaux.

Je tiens à remercier aussi les membres de la maison de la simulation pour les conseils et la bonne ambiance. Merci à Édouard, Julien, Mat(t)hieu, Maxime, Nathalie, Michel, Olivier, Pascal, Pierre, Thomas, Valérie, Yacine, et tous ceux qu'il serait trop long de lister ici. Merci surtout aux MVPs Ksander et Momo.

Merci à toute l'équipe du LC2M pour l'accueil chaleureux à Saclay, et à l'équipe Hiepacs de m'avoir accueilli pendant mes déplacements à Bordeaux.

Merci à ma famille : à mes parents et à ma sœur qui m'ont toujours soutenu, à Fatma qui m'a toujours supporté, à mes grands-parents qui ont été présents pour m'aider à m'installer. À Newton aussi, d'avoir arrêté d'éteindre mon PC.

Merci aux amis qui se reconnaitront, qu'ils soient à Paris, à Bordeaux, ailleurs, ou plus loin (genre au Chili).

Je remercie enfin tous ceux qui méritent d'être remerciés et qui n'apparaissent pas dans ces quelques lignes.

Table des matières

Introduction générale

La compréhension du comportement mécanique des matériaux utilisés dans des domaines comme l'aéronautique, l'automobile, l'électronique ou l'énergie nucléaire est un enjeu important en termes d'innovation, de rentabilité, de fiabilité voire de sécurité. Si le recours à l'expérience et à des modèles macroscopiques et phénoménologiques reste majoritairement utilisé, les progrès de la science des matériaux permettent aujourd'hui de mieux cerner les mécanismes physiques sous-jacents [65, 49, 94]. Ces mécanismes sont nombreux, parfois contradictoires et très sensibles à la composition chimique, la cristallographie, et de façon générale, à la microstructure du matériau, ainsi qu'aux sollicitations extérieures (température, pression, chimie, etc...). Ils partagent néanmoins la particularité de se jouer presque tous à l'échelle atomique sur des temps caractéristiques extrêmement courts. Il est dès lors extrêmement difficile de les relier quantitativement au comportement du matériau à l'échelle macroscopique sur sa durée de vie complète.

L'essor des outils informatiques offre aux métallurgistes la possibilité de bâtir, d'agréger leur savoir, d'enrichir leurs modèles, de les confronter à l'expérience et d'établir ce lien entre les échelles microscopiques et macroscopiques. Il serait ici vain d'effectuer le calcul atomistique d'un échantillon macroscopique et représentatif de matière compte tenu du nombre d'atomes qu'il faudrait considérer et du nombre de pas de temps nécessaires [1]. Une approche plus raisonnable est de diviser ce problème en différentes échelles spatiales et temporelles caractéristiques du matériau, de développer des modèles et des codes de calculs adaptés et de faire les faire dialoguer. Cette démarche dite *multi échelle* ou *micro-macro* a ainsi été théorisée et s'est fortement développée à partir des années 1990 conjointement avec le développement des moyens de calcul [75].

Le travail présenté dans cette thèse porte sur le développement d'un code de simulation de dynamique des dislocations (DD), qui est une composante de la démarche multi échelle précédente. Cette méthode vise à décrire les processus de déformation plastique des matériaux cristallins à l'échelle du grain [2] de façon à établir sa loi de comportement, c'est-à-dire le lien entre la déformation et l'état de contrainte [34]. Le principe de cette méthode est de suivre, de façon explicite et dans le temps, le comportement de l'ensemble des défauts linéiques, appelés dislocations, contenus dans un grain et qui sont les vecteurs de la déformation plastique irréversible d'un matériau, donc de la plupart de ses propriétés mécaniques [61]. La modélisation des dislocations sous la forme de lignes, discrétisées le plus souvent par une succession de segments, évoluant dans un espace tridimensionnel permet une réduction drastique du coût d'un calcul par rapport à une simulation atomistique complète. Selon une estimation détaillée dans la thèse d'Arnaud Etcheverry [40], le nombre d'objets à considérer est très fortement inférieur (gain d'un facteur

1. Le pas de temps caractéristique de ce type de calcul est de l'ordre de la femtoseconde.
2. La très grande majorité des métaux et alliages sont un agglomérat de cristaux, de quelques microns de côté, qui sont appelés *grains*

10^5 typiquement sur le nombre total de degrés de liberté) tandis que le pas de temps admissible est sensiblement plus grand (gain d'un facteur 50 à 10^5 selon l'application envisagée). Il est dès lors possible de réinvestir ces avantages numériques dans la simulation de systèmes plus grands et représentatifs en temps et en espace. C'est tout l'enjeu de mon travail.

La modélisation de la dynamique des dislocations repose avant tout sur la théorie des dislocations dont la genèse remonte à 1934 avec les publications quasi simultanées des travaux d'Orowan [88], Polanyi [95] et Taylor [108], dont les références principales sont les livres écrits par Friedel dès 1964 [49], Hull & Bacon [61] et Anderson, Hirth & Lothe [6]. Reconnaissant les limites des approches purement analytiques de ces pionniers, les métallurgistes se sont emparés dès 1966 des outils de simulation numérique avec notamment les travaux de Foreman et Makin [43, 44] sur le comportement de dislocations en deux dimensions, qui servent encore aujourd'hui d'étalonnage pour les codes actuels. La genèse des codes tridimensionnels remonte quant à elle à 1993 avec le code MicroMégas/Tridis et la publication des travaux de Kubin, Canova *et. al.* [23]. Plusieurs codes tridimensionnels ont depuis vu le jour dans différents pays dont l'Allemagne [119] et les États-Unis [18], chacun apportant de nouvelles fonctionnalités physiques, algorithmiques ou numériques. Fruits de cette vigueur, deux ouvrages de référence ont été publiés récemment sur les principes des simulations par dynamique des dislocations à savoir le livre de Cai & Bulatov [18] et le livre de Kubin [72]. Le lecteur intéressé trouvera notamment dans ce dernier ouvrage un historique détaillé de cette méthode.

Le code de simulation Optidis sur lequel a porté mon travail tire naturellement parti des avancées des codes précédents. Co développé depuis 2010 par des chercheurs du CEA, de l'INRIA et du CNRS, il est lui-même le descendant du code Numodis, lancé en 2007, et dont il emprunte encore de nombreuses lignes. Une spécificité importante d'Optidis est de s'inscrire dans le cadre des problématiques des matériaux pour l'énergie nucléaire. Il a ainsi pour cahier des charges de pouvoir traiter de façon explicite les défauts cristallins nanométriques générés par l'irradiation neutronique. Cet objectif implique une discrétisation plus fine des dislocations que pour des applications plus conventionnelles, et donc un nombre de degrés de liberté plus important pour un volume simulé donné. S'appuyant sur des grandeurs caractéristiques mesurées expérimentalement, ce nombre s'inscrit dans une plage allant de 10^6 à 10^9 degrés de liberté [40]. Ce constat a donc, dès le départ, motivé le développement du code Optidis en s'appuyant sur le calcul intensif qui permet, par l'utilisation des architectures parallèles et distribuées, d'augmenter la puissance de calcul disponible et simuler un plus grand nombre d'objets. L'adaptation des codes pour les supercalculateurs nécessite de traiter les problématiques informatiques liées à la distribution des données et des calculs sur les nombreux processeurs qui composent ces machines.

Au-delà du nombre relativement élevé de segments de dislocations à traiter, la nature et le comportement dynamique des dislocations au cours du temps posent plusieurs grands défis numériques :

Le premier défi concerne le calcul de la force élastique, appelée force de Peach-Koehler, ressentie par chaque segment, qui dépend directement de la position de l'ensemble des autres segments de dislocation. Cette interaction à longue distance empêche dès lors l'utilisation de rayons de coupure et constitue pour l'ensemble des codes recensés le principal coût de calcul. Des méthodes d'approximation de champ lointain sont utilisées pour réduire le coût de calcul des forces d'interactions, notamment la Méthode des Multipôles Rapide (FMM - *Fast Multipole Method*) [56, 76] utilisée par exemple dans les codes Optidis [16] et ParaDis [7]. Cette méthode hiérarchique a pour avantage, à l'inverse des méthodes plus classiques comme celles

utilisées par exemple dans MicroMégas/Tridis [104], de s'adapter aux architectures distribuées en réduisant le volume de messages à échanger.

Le second défi réside dans le caractère extrêmement dynamique des dislocations. Comme je le présenterai dans la première partie de ce manuscrit, les dislocations ont la faculté de se multiplier, de s'annihiler, et de former des enchevêtrements aussi denses qu'éphémères [61]. Il n'est ainsi pas rare de voir expérimentalement la densité de dislocation dans un matériau être multipliée par un facteur 100 après seulement quelques pourcents de déformation ! Le nombre de segments à considérer à chaque pas de temps est donc extrêmement fluctuant (quoique la plupart du temps en hausse). Cette dynamicité soulève de nombreuses problématiques informatiques, notamment pour le stockage efficace de ce type de données, et pour leur répartition équilibrée sur plusieurs processeurs dans le cadre du calcul intensif. Bien qu'elles n'aient été traitées que de façon marginale dans le cadre de la dynamique des dislocations, à l'exception des travaux d'Arnaud Etcheverry sur le code Optidis [40], la multiplication actuelle des cœurs de calcul sur des architectures de plus en plus parallèles rend indispensable une telle réflexion et constitue un axe important de ma thèse.

Le troisième défi tient aux nombreuses collisions que connaissent les dislocations du fait de leur mouvement. Leurs conséquences mécaniques peuvent être importantes et il faut donc les traiter rigoureusement sous peine d'aboutir à des structures irréalistes physiquement. Le traitement de cette question dans le cadre de simulations de dynamique des dislocations m'a ainsi amené à réfléchir aux algorithmes de détection et de traitement de ces collisions de façon à garantir la validité physique des données de la simulation dans un contexte parallèle et distribué. Ce point n'a reçu, là encore, qu'une attention limitée dans le cadre de la dynamique des dislocations alors que l'augmentation du nombre de segments simulés accroit la probabilité de ce type d'évènement. Cette thèse est, là-encore, une contribution à cette réflexion.

Le présent manuscrit s'organise de la façon suivante :

Dans un premier temps, je présente le domaine de la dynamique des dislocations en définissant ce que sont les dislocations, leur rôle dans la plasticité des matériaux et comment elles peuvent être simulées. Je place la simulation de dynamique des dislocations dans le contexte de la modélisation multi échelle des matériaux et expose les problématiques liées aux simulations massives et aux architectures distribuées.

Je présente dans un deuxième temps une réponse à la problématique du stockage des données dynamiques. Après avoir précisé les conséquences de la dynamicité des données sur les simulations de dynamique des dislocations et étudié les différentes approches de la littérature, je propose une structure de données pour stocker les informations liées au réseau de dislocations pensée pour les architectures distribuées. Son objectif est d'améliorer la fiabilité et la performance des accès aux données.

La détection et le traitement des collisions entre les dislocations font l'objet de la troisième partie. Après avoir précisé les problématiques soulevées par le traitement des collisions dans la dynamique des dislocations, j'étudie les différentes réponses apportées par la communauté de la dynamique des dislocations, mais aussi d'autres communautés comme celle de la robotique ou de l'animation 3D. J'en ai déduit différents algorithmes, que j'ai ensuite comparés afin de choisir le plus adapté, que j'ai alors intégré dans Optidis.

L'apport de mes contributions a enfin été mesuré en validant physiquement les résultats sur des simulations simples et en mesurant la performance sur des cas-tests plus complexes sur

plusieurs centaines de cœurs de calcul.

Partie I

Simulation de dynamique des dislocations

Introduction

La théorie des dislocations s'intéresse au rôle joué par les dislocations sur les mécanismes de déformation plastique des matériaux. Les simulations de dynamique des dislocations sont utilisées dans le cadre de la modélisation multi échelle des matériaux à l'échelle *mésoscopique*. Elles font le lien entre l'échelle atomique et l'échelle macroscopique en fournissant des lois de comportement pour la mécanique des milieux continus basées sur les mécanismes atomistiques pertinents.

Plusieurs codes de dynamique des dislocations sont aujourd'hui utilisés pour effectuer des études sur la plasticité. La complexité grandissante des études à mener nécessite toutefois une puissance de calcul de plus en plus importante. Les codes de dynamique des dislocations doivent ainsi faire appel au calcul intensif pour effectuer les simulations massives nécessaires à la modélisation de volumes suffisamment grands pour être représentatifs. L'utilisation des architectures parallèles et distribuées permet d'accélérer les calculs, mais nécessite une adaptation des codes.

I.1 Les dislocations

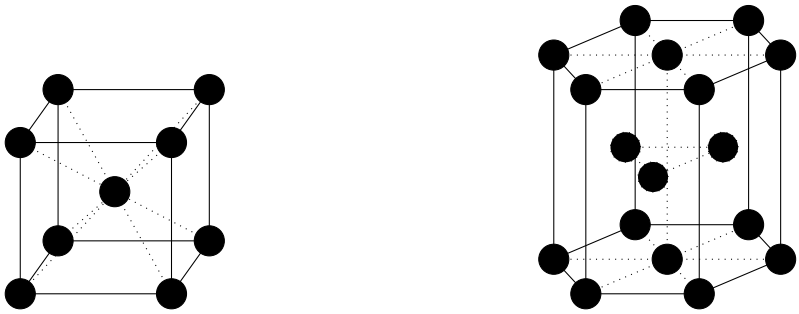
Les dislocations sont des défauts de la structure cristalline à l'origine des déformations plastiques (irréversibles) des matériaux cristallins tels que les métaux [22]. Elles évoluent dans la matière et jouent un rôle important pour de nombreuses propriétés mécaniques des matériaux.

I.1.1 Les dislocations : des défauts linéiques de la structure cristalline

La matière cristallisée est le plus souvent constituée d'une multitude de grains cristallins dont la taille est typiquement comprise entre 5 µm et 200 µm, à l'image de l'éprouvette de Zirconium de la figure I.1. Dans cette figure, le microscope optique à lumière polarisée permet de colorer les grains en fonction de leur orientation cristallographique.

FIGURE I.1 – Une éprouvette de Zirconium observée au microscope optique à lumière polarisée [74].

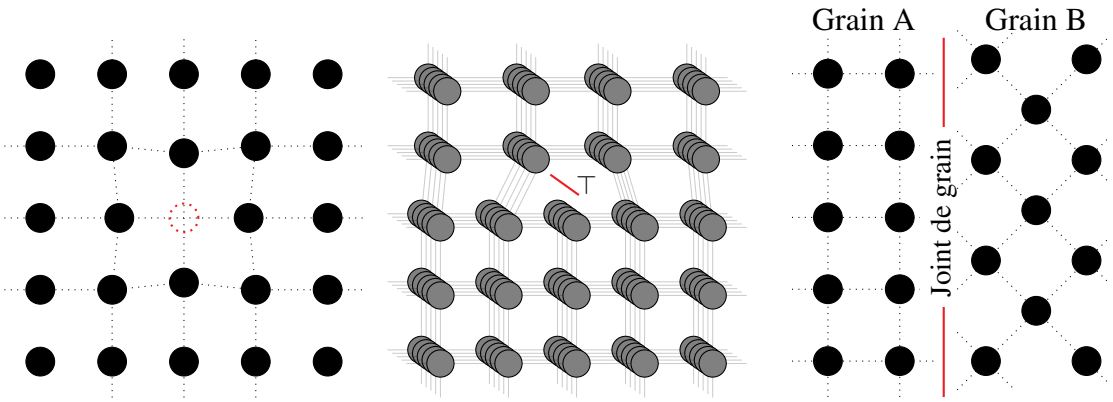
Chaque grain possède une structure cristalline dans laquelle les atomes sont arrangés de manière périodique et régulière. Un motif, appelé maille élémentaire (fig. I.2), se répète plusieurs fois pour former le réseau cristallin. Il existe plusieurs types de mailles élémentaires, comme par exemple les structures cubique centrée (BCC - *Body-Centered Cubic*) ou hexagonale compacte (HCP - *Hexagonal Close-Packed*) présentées en figure I.2.



(a) Structure cubique centrée. (b) Structure hexagonale compacte.

FIGURE I.2 – Exemples de mailles élémentaires.

Le réseau cristallin peut présenter des défauts qui se manifestent par un mauvais alignement des atomes. La figure I.3 illustre différents exemples de défauts présents dans les cristaux. Un atome en excès (interstitiel) ou en défaut (lacunaire) formera un défaut ponctuel (figure I.3a), alors que deux cristaux orientés différemment formeront un défaut surfacique, aussi appelé joint de grain (figure I.3c). Les dislocations sont les défauts linéaires dans le réseau cristallin (Fig. I.3b), dont les champs de contrainte et déformation ont été théorisés dès 1907 par VOLTERRA [114], bien avant l'utilisation des dislocations pour modéliser la plasticité.



(a) Défaut ponctuel : lacune. (b) Défaut linéique : dislocation coin (⊤). (c) Défaut surfacique : joint de grain.

FIGURE I.3 – Défauts dans les cristaux.

Les dislocations, sur la figure I.3b, forment des lignes parcourant le centre des trapèzes que forment les plans successifs, marqué par ⊤ . Ces lignes sont observables au Microscope Électronique en Transmission (MET), comme on peut le voir sur la figure I.4 sous la forme de lignes noires et fines.

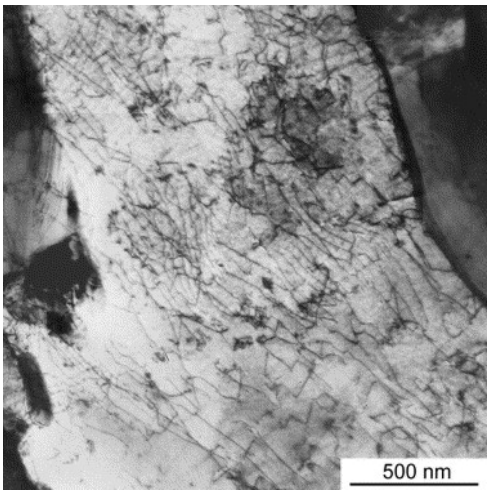


FIGURE I.4 – Des dislocations observées au MET dans du fer [68].

I.1.2 Rôle des dislocations dans la plasticité

Les dislocations sont à l'origine de la déformation plastique des cristaux. Elles jouent un rôle primordial dans plusieurs propriétés mécaniques des matériaux. Nous présentons ici les phénomènes d'écrouissage et de durcissement par irradiation.

L'étude des propriétés mécaniques des matériaux est basée sur l'observation de leur réaction sous une sollicitation appliquée. Sous contrainte, ils se déforment plus ou moins facilement et selon une amplitude variée. De manière générale, les matériaux se déforment en deux phases, élastique et plastique [22], comme l'illustre la courbe contrainte/déformation de la figure I.5.

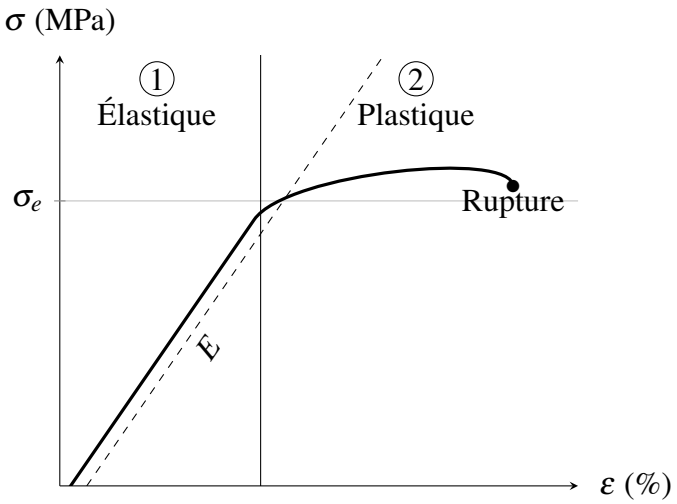


FIGURE I.5 – Phases de déformation lors de l'application d'une contrainte en traction. Une courbe de traction donne la contrainte appliquée σ (MPa) en fonction de la déformation imposée ε (%) au matériau.

La première phase de déformation élastique ① a lieu à faible déformation. Le matériau est alors déformé de manière réversible tel un ressort. La déformation s'effectue de manière homogène en modifiant l'espace entre les atomes, sans modifier la topologie du réseau cristallin. Dans ce régime, le matériau se déforme selon la loi élastique de HOOKE.

Pour des contraintes et des déformations plus élevées ② une déformation irréversible, appelée *déformation plastique* se produit. La limite d'élasticité σ_e marque la séparation entre les deux phases.

Écrouissage

Lorsqu'un matériau est déformé plastiquement, ses propriétés physiques sont durablement modifiées. Par exemple, la limite élastique peut augmenter comme le montre la figure I.6.

Une première traction est effectuée ① sur le matériau au-delà de la limite élastique. La contrainte est alors relaxée et l'échantillon reprend une position au repos légèrement plus allongée.

Lorsqu'une sollicitation est de nouveau appliquée, on remarque que la limite élastique est plus grande et correspond à la contrainte maximale de la première traction.

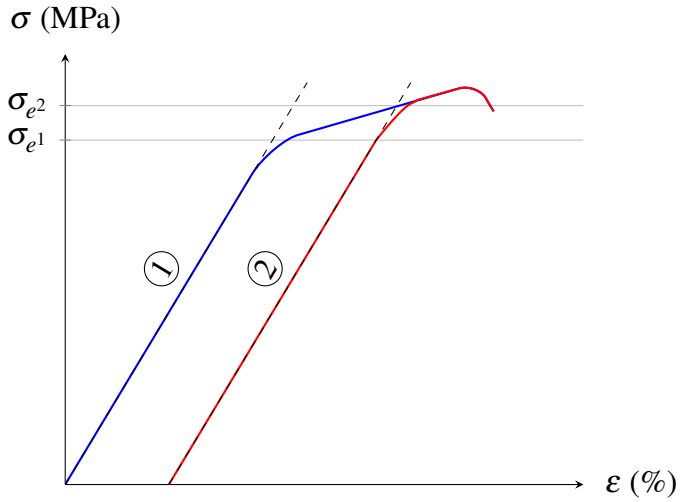


FIGURE I.6 – Écrouissage : après une déformation plastique la limite élastique du matériau est repoussée.

Ce phénomène est directement lié à la densification du réseau de dislocation lors d'une déformation plastique [80]. Les jonctions qui forment un obstacle au déplacement des dislocation sont plus nombreuses, et la déformation plastique apparaît donc pour des contraintes plus élevées. La relation de Taylor [108, 86] relie la limite élastique σ_e à la densité de dislocations :

$$\sigma_e(\rho) = \sigma_i + \alpha\mu b\sqrt{\rho} \tag{I.1}$$

La limite élastique du matériau évolue selon la racine carrée de la densité ρ de dislocations. Dans cette formule σ_i et μ sont des constantes qui dépendent du matériau, et α dépend de la microstructure considérée. C'est sur cette base que les dislocations permettent d'étudier les mécanismes de l'écrouissage [80, 19]

Irradiation

Les effets de l'irradiation sur les matériaux sont multiples, mais sont tous étroitement liés à la plasticité. Les matériaux soumis à une dose importante de radiations sont très souvent plus durs,

mais aussi moins ductiles. Le durcissement augmente la limite d'élasticité du matériau, alors que la baisse de ductilité réduit la déformation maximale avant rupture. La courbe I.7 présente une comparaison des courbes contrainte/déformation d'un matériau irradié et non irradié.

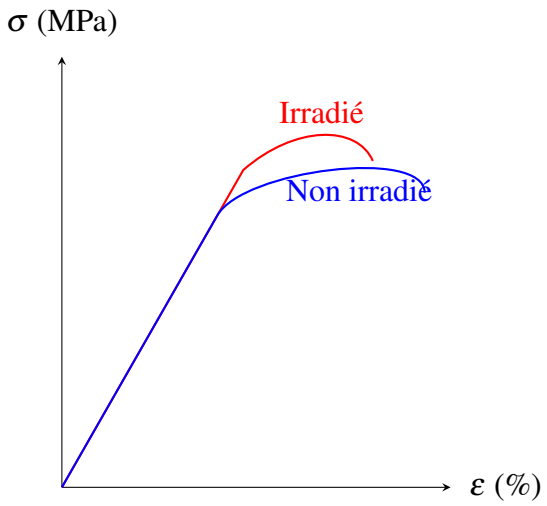


FIGURE I.7 – Comparaison des courbes contrainte déformation d'un matériau irradié et non irradié.

Lors de l'irradiation, la structure des cristaux est modifiée, et les défauts qui se forment peuvent être modélisés par des boucles d'irradiation. Ces boucles d'irradiation que l'on peut voir dans la figure I.8 forment un obstacle à la progression des dislocations, générant alors des phénomènes de durcissement.

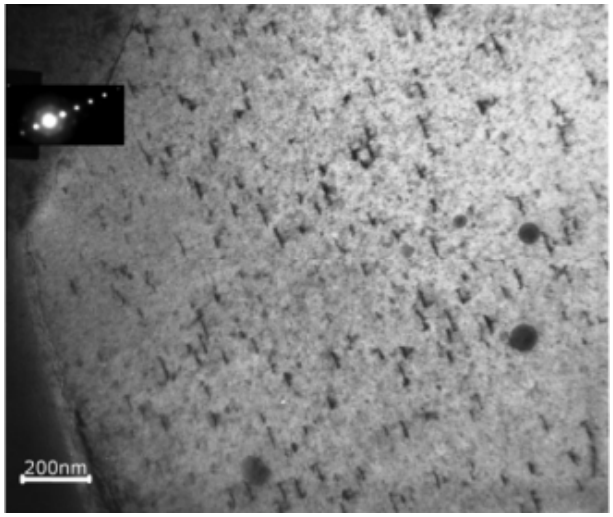


FIGURE I.8 – Des boucles d'irradiation dans le zirconium observées au MET [111].

I.1.3 Théorie des dislocations

Les concepts fondamentaux de la théorie des dislocations sont décrits dans plusieurs ouvrage de référence dont les livres de HULL et BACON [61] et de HIRTH et LOTHE [6] qui définissent les propriétés physiques et le comportement des dislocations.

I.1.3.1 Définitions

Vecteur de Burgers

Une ligne de dislocation est caractérisée par son vecteur directeur $\boldsymbol{\xi}$ et son vecteur de Burgers $\boldsymbol{b}$. Ce vecteur a été introduit par BURGERS [20] et décrit la déformation du réseau induite par la dislocation. Il se définit comme la différence entre le circuit de Volterra [114] dans un cristal parfait et celui autour de la dislocation, comme l'illustre la figure I.9. Un circuit est dessiné autour de la dislocation (ici ξ vers la feuille) dans le sens horaire (convention RH - *Right Hand*) (fig. I.9 gauche). Le même circuit (même déplacement d'atomes) est dessiné dans un cristal parfait (fig. I.9 droite). Dans la convention FS/RH [6] utilisée dans ce manuscrit, le vecteur de Burgers relie la fin de ce circuit à son début : $\boldsymbol{b} = [\boldsymbol{FS}]$.

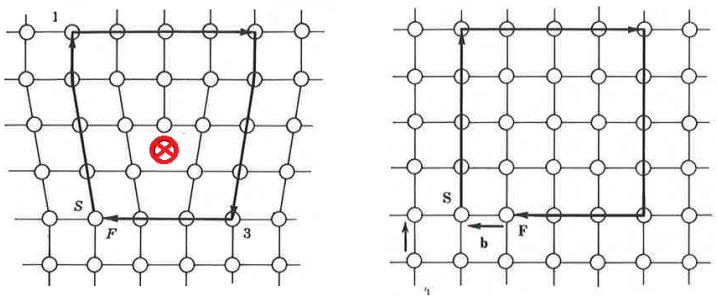


FIGURE I.9 – Mise en évidence du vecteur de Burgers [6]. Le vecteur de Burgers est la différence entre le circuit de Volterra dans le cristal déformé (gauche) et dans le cristal parfait (droite).

Les dislocations forment un réseau de lignes enchevêtrées qui peut avoir une géométrie complexe. La figure I.10 présente un exemple de réseau de dislocations. Les points où les lignes de dislocations se regroupent sont appelés nœuds physiques (•). Chaque ligne possède son vecteur de Burgers (b_i) qui reste constant entre les nœuds physiques.

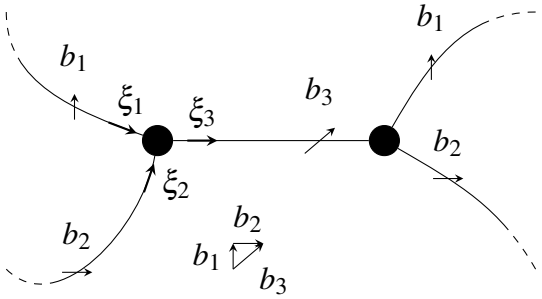


FIGURE I.10 – Un réseau de dislocations.

Le vecteur de Burgers répond à la loi des nœuds : la somme des vecteurs de Burgers des dislocations incidentes à chaque nœud physique est nulle. Cette propriété formalisée par l'équation I.2 est représentée par le triangle en bas de la figure I.10.

$$\sum_{\xi_i \; entrant} \boldsymbol{b_i} = \sum_{\xi_i \; sortant} \boldsymbol{b_i} \tag{I.2}$$

On distingue deux configurations particulières pour les dislocations selon l'orientation du vecteur de Burgers par rapport à la direction de la ligne. Lorsque le vecteur de Burgers est orthogonal à la ligne ($\boldsymbol{b} \perp \boldsymbol{\xi}$), on parle de dislocation *coin*. Cette situation est par exemple la situation illustrée en figure I.9. Si le vecteur de Burgers est colinéaire à la ligne ($\boldsymbol{b} \parallel \boldsymbol{\xi}$), on parle de dislocation *vis*. Dans les autres cas, la dislocation est dite *mixte*.

Glissement des dislocations

Les déformations plastiques se propagent par le déplacement successif des atomes du réseau, et induisent un déplacement des dislocations. La figure I.11 illustre le déplacement d'une dislocation dans un cristal. Le vecteur de Burgers représente le déplacement des atomes au cours du glissement de la dislocation.

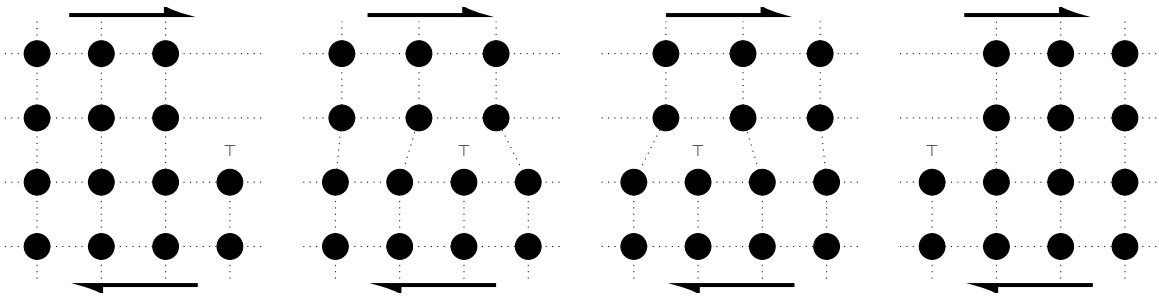


FIGURE I.11 – Déplacement d'une dislocation lors de la déformation plastique. Une dislocation ($\top$) se déplace lorsqu'une contrainte en cisaillement est appliquée au cristal. La partie haute du cristal glisse latéralement par rapport à la partie basse.

Les dislocations se déplacent principalement par *glissement* dans le plan contenant la ligne de dislocation et le vecteur de Burgers. La combinaison d'un plan de glissement et d'un vecteur de Burgers s'appelle un *système de glissement*. Pour chaque type de cristallographie, il existe un nombre restreint de systèmes de glissement qui permettent le déplacement des dislocations. De manière générale, les plans denses et les vecteurs de Burgers dans des directions denses forment des systèmes de glissement actifs qui permettent le déplacement des dislocations [6]. Ce type de dislocations sera qualifié de *glissile*. Les systèmes non actifs empêchent le mouvement des dislocations que l'on qualifiera alors de *sessiles*. Par exemple dans les cristaux cubiques face centrés, il existe 12 systèmes de glissement préférentiels dont on retrouve les plans de type $\{1\,1\,1\}$ et les Burgers de type $\frac{1}{2}\langle 1\,1\,0\rangle$ respectivement sur les faces et les arêtes du tétraèdre de Thompson (Fig. I.12). Chaque système de glissement possède ses propriétés intrinsèques qui dépendent du matériau considéré, et le mouvement des dislocations y sera plus ou moins facilité.

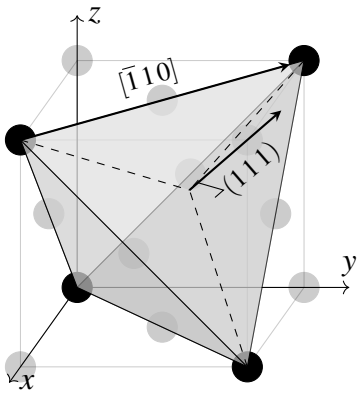


FIGURE I.12 – Tétraèdre de Thompson. Dans les matériaux FCC, les plans de glissement des systèmes préférentiels sont les faces $\{1\,1\,1\}$ du tétraèdre, et les vecteurs de Burgers $\frac{1}{2}\langle 1\,1\,0\rangle$ sont ses demi-arêtes.

D'autres phénomènes de déplacement des dislocations comme la *montée* existent pour des températures élevées. Si le glissement est un déplacement *conservatif* qui se produit sans transport de matière, la montée intervient lorsque les atomes en excès ou en défaut diffusent au sein du matériau et sont absorbés par les dislocations.

I.1.3.2 Force de PEACH-KOEHLER

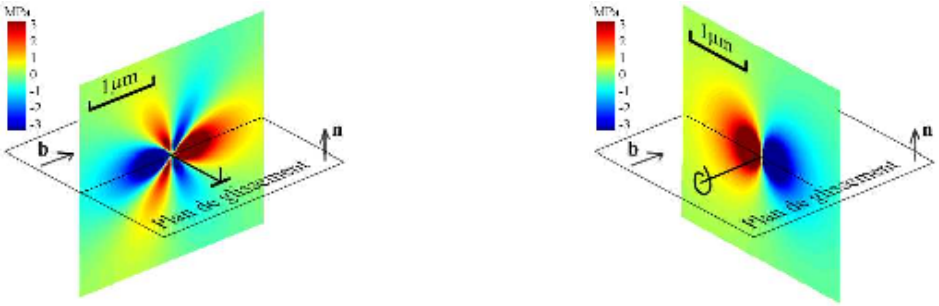
Les dislocations se déplacent en réponse à la contrainte qui s'applique localement sur chacune d'elles à travers la force de PEACH-KOEHLER [93] décrite par la formule I.3. Cette force par unité de longueur est orthogonale à la ligne de dislocation, mais n'est pas forcément contenue dans le plan de glissement. Elle dépend du tenseur de contrainte locale $\boldsymbol{\sigma}$.

$$\boldsymbol{f}_{PK} = (\boldsymbol{\sigma}\cdot\boldsymbol{b})\times\boldsymbol{\xi} \tag{I.3}$$

La contrainte locale sur chaque dislocation résulte des forces externes appliquées au solide, mais aussi de forces internes. Les contraintes externes sont par exemple une compression effectuée par une presse sur le matériau. Les contraintes internes, quant à elles, résultent de la déformation élastique du réseau induite par tous les défauts qu'il contient, notamment les dislocations.

$$\boldsymbol{\sigma}_{tot} = \boldsymbol{\sigma}_{int} + \boldsymbol{\sigma}_{ext} \tag{I.4}$$

Le champ de contrainte interne généré par le réseau de dislocations (fig. I.13) peut être calculé dans le cadre de l'élasticité linéaire [6, 33]. Il est donc possible de calculer les forces d'interactions entre les dislocations. Les forces d'interaction peuvent être de nature attractive ou répulsive selon la géométrie et le système de glissement des dislocations.



(a) Pour une dislocation coin.

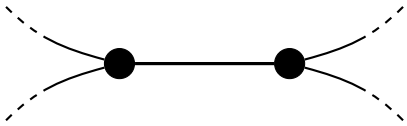
(b) Pour une dislocation vis.

FIGURE I.13 – Champ de contrainte généré par une dislocation [33].

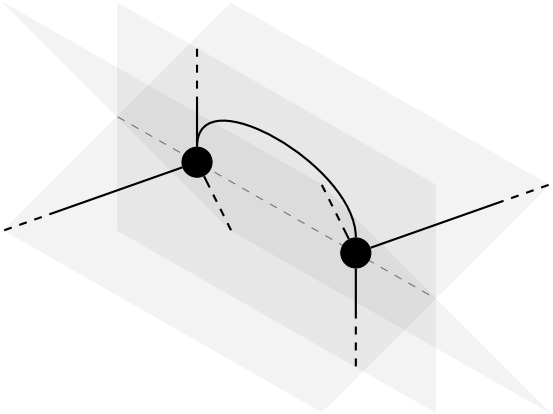
I.1.3.3 Formation de jonctions

Des phénomènes locaux peuvent se produire lorsque deux lignes de dislocation entrent en contact. Les lignes qui entrent en contact peuvent fusionner et former une *jonction* si cette dernière est plus favorable énergétiquement. Le vecteur de Burgers de la jonction $\boldsymbol{b}_j$ est alors la somme des Burgers des dislocations $\boldsymbol{b}_1$ et $\boldsymbol{b}_2$: $\boldsymbol{b}_j = \boldsymbol{b}_1 + \boldsymbol{b}_2$.

Les jonctions peuvent se former à partir de la collision entre deux dislocations ou plus. Par exemple on observe sur la figure I.14a deux dislocations dans le même plan formant une jonction. Les cas où plusieurs dislocations entrent en contact peuvent mener à des situations plus complexes comme dans la figure I.14b.



(a) Une jonction planaire entre deux dislocations.



(b) Une multi-jonction formant une source de Frank-Read [19].

FIGURE I.14 – Formation de jonctions.

La sommation des Burgers lors de la fusion des dislocations peut générer des topologies spéciales qui jouent un rôle important dans le comportement mécanique des matériaux. Des dislocations avec des vecteurs de Burgers opposés s'annihilent en fusionnant, par exemple. Certaines jonctions peuvent habiter des systèmes de glissement non favorables, générant alors des jonctions sessiles qui ne peuvent pas glisser. De nombreuses études s'intéressent aux phénomènes de formation de jonctions et à leur impact, notamment sur les phénomènes d'écrouissage [19, 80]. Nous verrons plus tard que la gestion des phénomènes locaux comme la formation de jonctions est un enjeu majeur afin de simuler de manière fidèle la dynamique des dislocations.

I.2 Simulation des dislocations dans un contexte de modélisation multi échelle

La modélisation multi échelle vise à comprendre le comportement des matériaux en analysant les mécanismes physiques élémentaires à plusieurs échelles [106]. Elle consiste à remonter aux principes les plus élémentaires des interactions au sein de la matière, pour ensuite modéliser son comportement à des échelles de temps et d'espace de plus en plus grandes.

La simulation des dislocations est utilisée pour rendre compte de la plasticité à l'échelle du grain [18, 72]. Les dislocations peuvent être simulées à l'échelle atomique par la dynamique moléculaire, et à l'échelle mésoscopique par la dynamique des dislocations.

I.2.1 Différentes échelles de temps et d'espace

Différents phénomènes sont modélisés de l'échelle atomique jusqu'à la structure complexe d'une cuve de centrale, par exemple. La modélisation à chaque étape se base sur les résultats obtenus aux échelles inférieures, et sert à son tour de base pour la modélisation des échelles supérieures.

La dynamique des dislocations fait le lien entre l'échelle atomique utilisée pour les simulations de dynamique moléculaire (MD - *Molecular Dynamics*) et l'échelle macroscopique utilisée dans des simulations de mécanique des milieux continus qui utilisent par exemple la méthode des éléments finis (FEM - *Finite Element Method*).

La dynamique moléculaire simule le comportement de chaque atome du matériau considéré [5]. Elle modélise de manière très précise le comportement des matériaux, mais ne peut simuler que de petits objets pendant des temps très courts. Il est possible d'en extraire des informations sur le comportement des dislocations à petites échelles de temps et d'espace pour calibrer les modèles de dynamique des dislocations. La figure I.15 donne l'exemple de la simulation d'une dislocation en dynamique moléculaire.

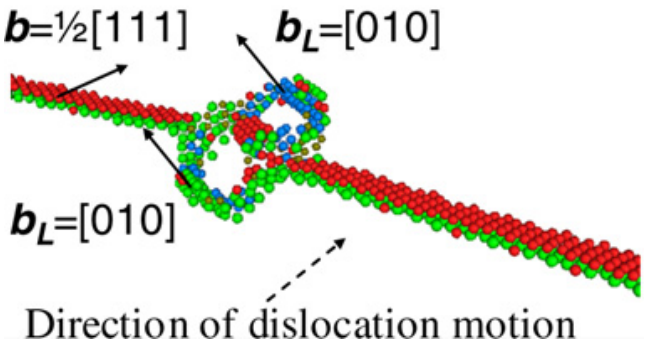


FIGURE I.15 – Atomes autour d'une dislocation et d'une boucle d'irradiation dans le fer en dynamique moléculaire [109].

À l'échelle macroscopique, la plasticité cristalline est simulée dans une approche continue en utilisant la méthode des éléments finis ou des méthodes de résolution spectrales (FFT - *Fast Fourier Transform*). Les échelles considérées permettent leur utilisation directe dans l'industrie, mais nécessitent une calibration des lois de comportement, notamment par la dynamique des dislocations.

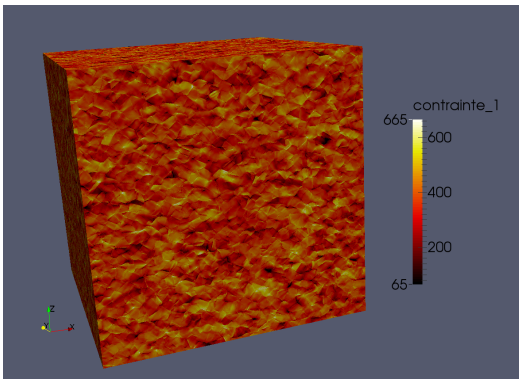
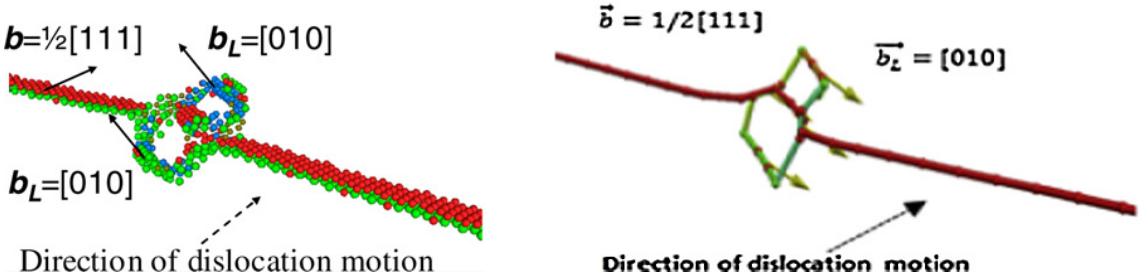


FIGURE I.16 – Distribution de contraintes dans un polycristal CFC durant une simulation FFT [58].

I.2.2 De l'échelle atomique à la dynamique des dislocations

La dynamique moléculaire apporte une base de paramètres physiques, mais sert aussi de validation aux simulations de dynamique des dislocations. À titre d'exemple, la figure I.17 illustre les comparaisons entre la dynamique moléculaire et la dynamique des dislocations menées par X. SHI [103]. Ces résultats montrent une adéquation entre les deux méthodes de simulation pour les interactions entre les dislocations et les boucles d'irradiation dans le fer. Des travaux récents permettent par exemple d'extraire les lignes de dislocations des simulations de dynamique moléculaire [107] afin de comparer plus finement les deux méthodes.



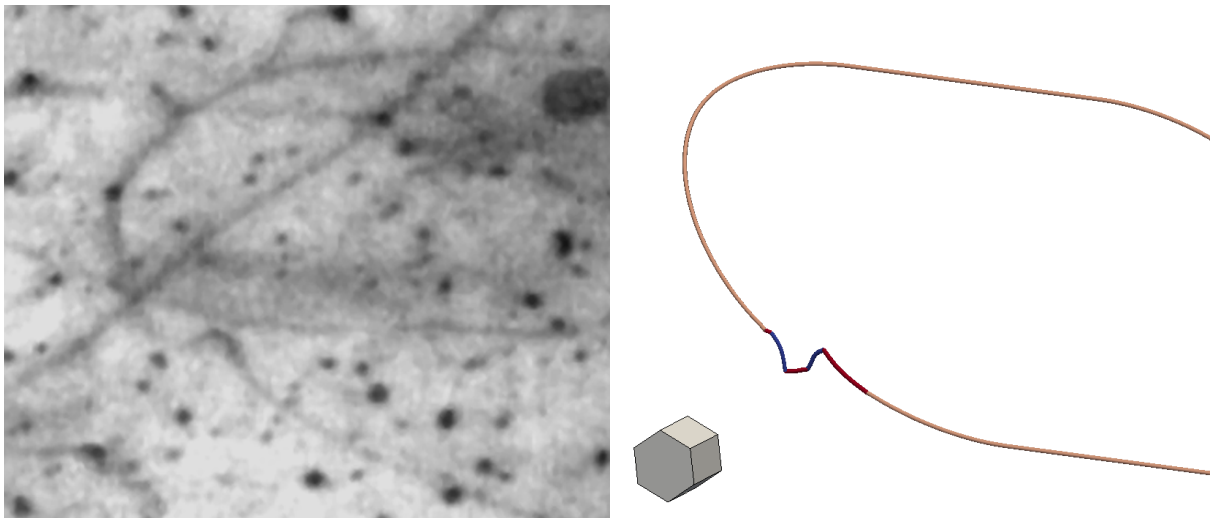
(a) Dynamique moléculaire [109].

(b) Dynamique des dislocations.

FIGURE I.17 – Comparaison entre dynamique moléculaire et dynamique des dislocations pour simuler les interactions entre dislocations et boucles d'irradiation dans le fer [103].

La dynamique moléculaire permet également de calibrer les modèles utilisés pour la dynamique des dislocations, par exemple en déterminant la mobilité des dislocations dans les différents systèmes de glissement [103].

D'autres méthodes sont aussi utilisées pour valider les simulations de dynamique des dislocations, comme par exemple la comparaison avec l'expérience. Des comparaisons à des observations MET in-situ ont été menées par J. DROUET [35] (figure I.18) et montrent une adéquation entre les observations in-situ et les résultats de dynamique des dislocations pour les interactions des dislocations avec les boucles d'irradiation.



(a) Microscope Électronique en Transmission. (b) Simulation de dynamique des dislocations.

FIGURE I.18 – Validation par des observations MET in-situ (Zr) [35]. L'interaction de la ligne de dislocations avec la boucle d'irradiation forme la même géométrie en dynamique des dislocations (b) que celle observée expérimentalement (a).

I.2.3 De la dynamique des dislocations aux échelles supérieures

La simulation des dislocations permet la transition d'échelle vers la plasticité cristalline et les milieux continus, notamment en permettant la mise au point de lois analytiques décrivant le comportement statistique d'une population de dislocations. Il en découle par exemple des lois de comportement basées sur la *densité* de dislocations qui peuvent être utilisées à plus grande échelle [42, 96, 84].

Par exemple, le modèle de Taylor pour les phénomènes d'écrouissage peut être étendu en considérant les différences entre l'activité des différents plans de glissement. La formule de Taylor (sec. I.1.2) peut être étendue sous une forme plus sophistiquée qui donne la contrainte d'activation τ_c^p d'un système de glissement p [45] en présence de densités de dislocations dans les autres systèmes :

$$\tau_c^p = \mu b \sqrt{\sum_s a^{ps} \rho^s} \tag{I.5}$$

Avec μ le module de cisaillement du matériau, b la norme du vecteur de Burgers, et ρ^s la densité des dislocations dans le système de glissement s. Les coefficients a^{ps} représentent l'intensité des interactions entre les systèmes de glissement p et s. La simulation de dynamique des dislocations est utilisée pour déterminer les coefficients a^{ps} [84] (fig. I.19) en permettant des expériences "numériques" qu'il serait impossible de réaliser autrement.

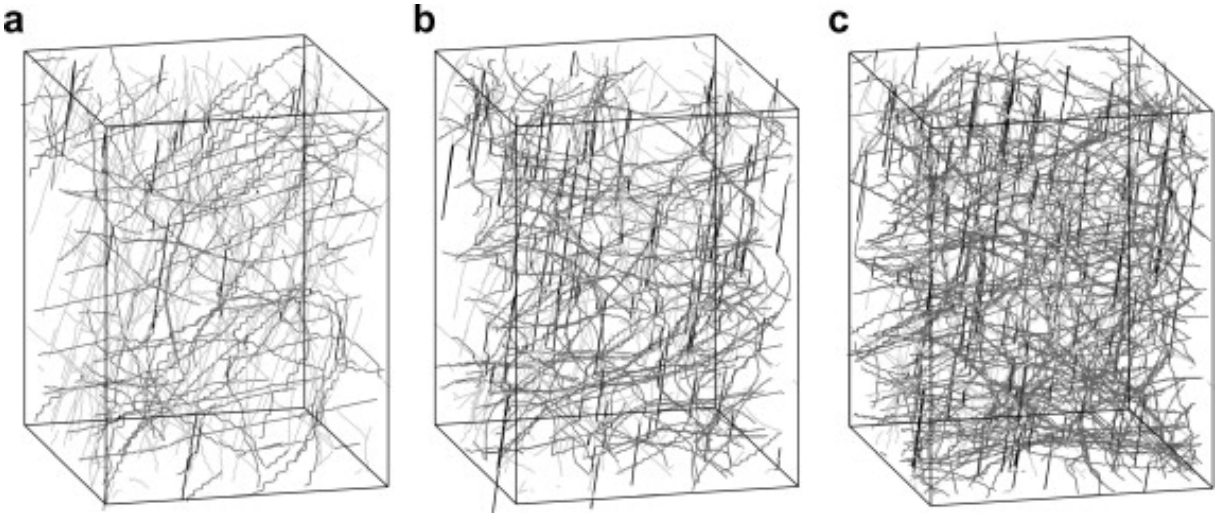


FIGURE I.19 – Des simulations de dynamique des dislocations pour déterminer les coefficients a^{ps} avec le code microMegas [84].

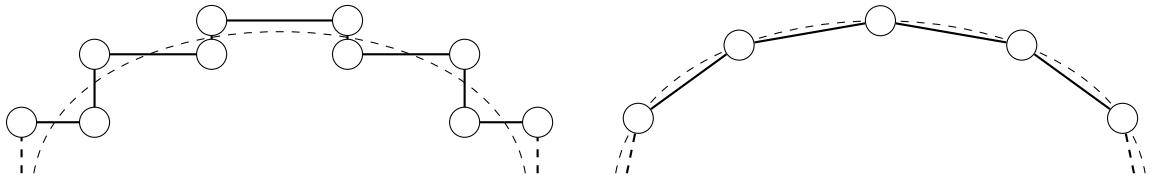
Ces simulations sont effectuées dans un grain infini en utilisant des conditions périodiques. Elles permettent de simuler le comportement de grands cristaux à moindre coût, mais ne sont cependant pas aussi précises que la simulation d'un grain complet non périodique. Par exemple, des problèmes d'auto-annihilation des dislocations [81] ont été identifiés comme un phénomène gênant dans les simulations périodiques. Pour simuler de manière plus précise les phénomènes liés à l'écrouissage, il faudrait simuler des domaines plus grands de manière à modéliser correctement un grain complet. Le volume de calcul devient alors trop important, et c'est ici qu'interviennent les simulations massives de dynamique des dislocations.

Les effets de l'irradiation sur la plasticité peuvent aussi être quantifiés par les simulations de la dynamique des dislocations [9]. L'étude des interactions entre les dislocations et les boucles d'irradiations permettent de mieux comprendre les phénomènes de durcissement par irradiation [8, 102]. L'étude de la formation de bandes claires [87] permet de mieux comprendre la fragilisation sous irradiation.

Les boucles d'irradiation sont souvent des objets de petite taille, et leur discrétisation doit être faite en conséquence. Une discrétisation plus fine implique un plus grand nombre de segments à simuler pour la dynamique des dislocations, ce qui nous mène encore une fois aux simulations massives.

I.2.4 Les codes de dynamique des dislocations

Les premières simulations 2D voient le jour dans les années 1960 pour observer les interactions entre dislocations dans le plan [43]. Les simulations 3D naissent dans les années 1990 avec le code Micromegas/Tridis (CNRS-ONERA, CEA) [23] qui repose sur une description spatiale des lignes sur une grille discrète : les dislocations sont alors représentées par une succession de segments vis et coin (fig. I.20a), voire mixtes [80].



(a) Modèle Vis/Coin. (b) Modèle nodal.

FIGURE I.20 – Différentes modélisations des dislocations.

Le modèle nodal de discrétisation (fig. I.20b) utilise des nœuds se déplaçant de manière continue, sans grille [101]. Il constitue une meilleure approximation des courbures des dislocations, ce qui permet de mieux modéliser les phénomènes comme la formation des jonctions. Model (UCLA) [52], le simulateur de Karlsruhe [119], ParaDis (Stanford) [18] et Numodis/Optidis (CEA) [36] utilisent tous le modèle nodal de discrétisation des dislocations.

I.3 Déroulement d'une simulation de dynamique des dislocations

Afin de comprendre les algorithmes mis en jeux et les difficultés calculatoires rencontrées dans la simulation de la dynamique des dislocations, je présente les différentes phases du calcul. Cette section présente l'approche nodale de la dynamique des dislocations discrète utilisée dans les codes tels que Numodis, Optidis et ParaDis.

I.3.1 Modèle nodal de discrétisation des dislocations

Dans le modèle nodal de la dynamique des dislocations, les lignes sont discrétisées afin de résoudre numériquement par un formalisme *éléments finis* leur déplacement. Le réseau de dislocations est représenté par un ensemble de nœuds connectés par des segments qui portent des propriétés telles que le système de glissement auxquels ils appartiennent. Les segments peuvent être orientés dans toutes les directions, permettant de modéliser des dislocations mixtes. La figure I.21 représente un réseau de dislocation discrétisé avec une représentation nodale. Contrairement aux nœuds physiques (en noir), les nœuds topologiques (en blanc), aussi appelés nœuds de discrétisation, n'ont pas de signification physique et ne servent qu'à discrétiser les lignes entre deux nœuds physiques. Chaque segment porte des propriétés physiques, comme par exemple son vecteur de Burgers (flèches).

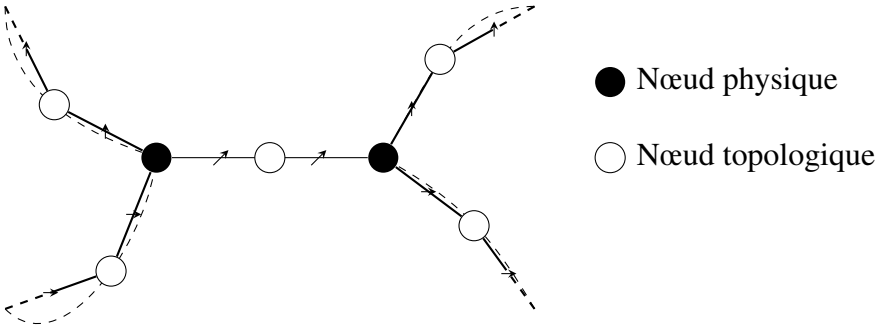


FIGURE I.21 – Représentation nodale d'un réseau de dislocations. Les dislocations sont discrétisées par un ensemble de nœuds et de segments.

I.3.2 Déroulement global de la simulation

La figure I.22 décrit les différentes étapes d'une itération en temps de la simulation et leurs dépendances en données.

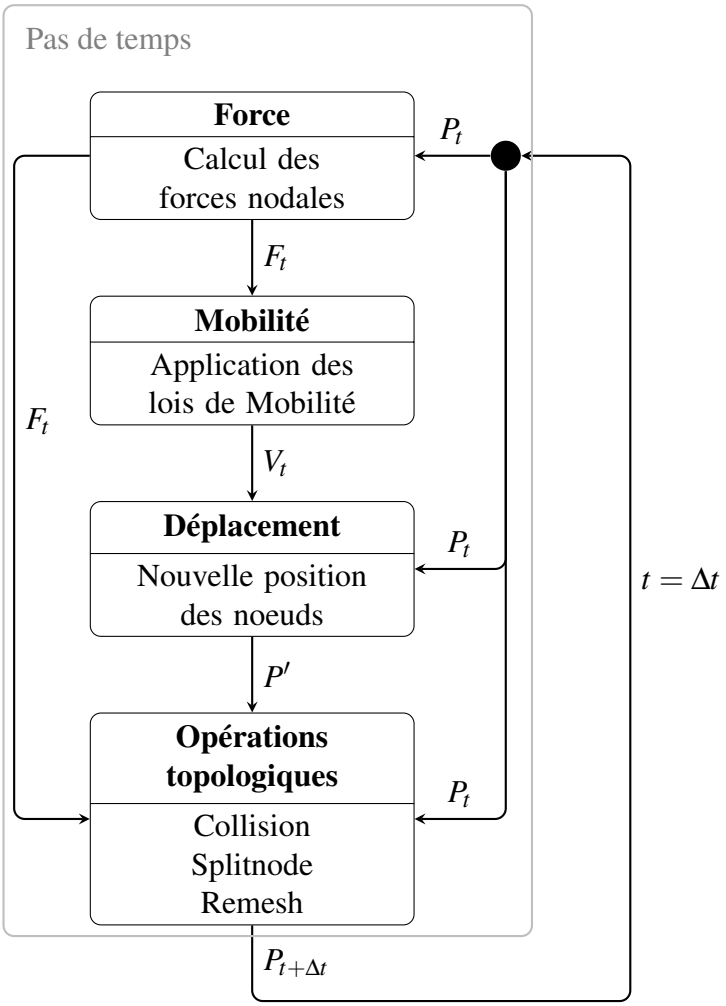


FIGURE I.22 – Les différentes étapes et leurs dépendances en données.

Les **Forces** nodales F_t sont calculées à partir de la formule de PEACH-KOEHLER en utilisant

la position P_t des nœuds et les propriétés physiques des segments. Les lois de **Mobilité** sont ensuite appliquées pour calculer les vitesses nodales V_t à partir des forces. Le **Déplacement** des nœuds est effectué en intégrant la vitesse pour déterminer une position prédictive P'. Les **Opérations topologiques** liées au remaillage et à la formation des jonctions sont effectuées : la position prédictive est corrigée en fonction des modifications topologiques qui ont lieu. On passe ensuite au pas de temps suivant.

Les sections suivantes détaillent certaines parties importantes, comme le calcul de forces ou la formation des jonctions.

I.3.3 Calcul des forces nodales

Dans notre approche éléments finis, la force $\boldsymbol{f_i}$ appliquée sur chaque nœud i est calculée comme l'intégrale de la force de PEACH-KOEHLER $\boldsymbol{f}^{PK}$ [93] (section I.1.3.2) sur l'ensemble D_i des segments de dislocations S_k qui lui sont connectés ($D_i = \bigcup S_k$) multipliée par la fonction de forme N_k :

$$\boldsymbol{f_i} = \sum_{S_k \in D_i} \int_{S_k} N_k(\boldsymbol{x_k}) \boldsymbol{f}_k^{PK}(\boldsymbol{x_k}) d\boldsymbol{x_k} \tag{I.6}$$

En élasticité linéaire, la contrainte utilisée pour le calcul de la force se décompose en une contrainte externe $\boldsymbol{\sigma}_{ext}$ et en des contributions $\boldsymbol{\sigma}_j$ de tous les segments S_j contenus dans la boite de simulation :

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{ext} + \sum_j \boldsymbol{\sigma}_j \tag{I.7}$$

On peut alors décomposer f_i en plusieurs contributions :

$$\boldsymbol{f}_i = \boldsymbol{f}_{i,ext} + \sum_j \boldsymbol{f}_{i,j} \tag{I.8}$$

avec $\boldsymbol{f}_{i,ext}$ la contribution de la contrainte externe à la force et $\boldsymbol{f}_{i,j}$ la contribution du segment S_j :

$$\boldsymbol{f}_{i,ext} = \sum_{S_k \in D_i} \int_{S_k} N_k(\boldsymbol{x_k}) \boldsymbol{f}_{k,ext}^{PK}(\boldsymbol{x_k}) d\boldsymbol{x_k} \tag{I.9}$$

$$\boldsymbol{f}_{i,j} = \sum_{S_k \in D_i} \int_{S_k} N_k(\boldsymbol{x_k}) \boldsymbol{f}_{k,j}^{PK}(\boldsymbol{x_k}) d\boldsymbol{x_k} \tag{I.10}$$

Le calcul de $\boldsymbol{f}_{i,ext}$ est une application directe de la formule de PEACH-KOEHLER car $\boldsymbol{\sigma}_{ext}$ est la contrainte imposée par les paramètres de la simulation. Les $\boldsymbol{f}_{i,j}$ peuvent se calculer par une quadrature numérique en utilisant la formule de MURA [85], mais les formules analytiques d'ARSENLIS [7], plus récentes, sont moins coûteuses et permettent de calculer directement les forces d'interaction entre deux segments sans passer par le calcul de la contrainte en s'appuyant sur la théorie élastique non-singulière de CAI *et. al.* [21]. L'étape du calcul des forces d'interaction est la plus coûteuse de la simulation, car elle nécessite de calculer chacune des interactions $\boldsymbol{f}_{k,j}^{PK}$ entre tous les segments du réseau de dislocations.

I.3.4 Déplacement des dislocations

Une fois le calcul des forces terminé, on évalue la vitesse des nœuds par la méthode des éléments finis. La nouvelle position des nœuds est ensuite déterminée en intégrant la vitesse au cours du

temps.

Le calcul de la vitesse des nœuds utilise, pour le moment dans Optidis, uniquement un modèle de glissement visqueux linéaire basé sur l'hypothèse d'un régime dissipatif amorti [18]. Ce modèle est une approximation qui ne prend pas en compte les effets de l'inertie des dislocations qui pourraient apparaître à de très fortes contraintes et vitesses de déformation. Dans cette approximation, on considère que la force appliquée en tout point est à l'équilibre avec une force de frottement visqueuse linéaire :

$$\boldsymbol{B}\cdot\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) \tag{I.11}$$

où $\boldsymbol{f}$ est la force, $\boldsymbol{v}$ la vitesse et $\boldsymbol{B}$ le coefficient de viscosité qui dépend du type de matériau et de la dislocation considérée. Dans le cas du glissement seul, le mouvement doit être contraint dans le plan de glissement de la dislocation. Le coefficient de viscosité peut être décomposé en une composante normale B_{glide} selon $\boldsymbol{m} = \xi \wedge \boldsymbol{n}$ qui représente le déplacement physique du segment et en une composante tangentielle à la dislocation B_{tan} qui, bien que non physique, est introduite pour limiter la vitesse tangentielle des nœuds et maintenir la stabilité numérique :

$$\boldsymbol{B} = B_{tan}(\xi \otimes \xi) + B_{glide}(\boldsymbol{m} \otimes \boldsymbol{m}) \tag{I.12}$$

On utilise la formulation variationnelle de l'équation (I.12) sur le réseau de dislocations avec les fonctions de base N_i de la section précédente :

$$\forall i, \int_D \boldsymbol{B}\boldsymbol{v}(\boldsymbol{x})N_i(\boldsymbol{x})d\boldsymbol{x} = \int_D \boldsymbol{f}(\boldsymbol{x})N_i(\boldsymbol{x})d\boldsymbol{x} \tag{I.13}$$

La vitesse en tout point $\boldsymbol{v}(x)$ s'exprime en fonction des vitesses nodales $\boldsymbol{v}_i$:

$$\boldsymbol{v}(\boldsymbol{x}) = \sum_{i=1}^{N} N_i(\boldsymbol{x})\boldsymbol{v}_i \tag{I.14}$$

En reportant l'expression de la vitesse (I.14) dans l'équation (I.13), les vitesses nodales sont solutions du système linéaire :

$$\boldsymbol{A}\boldsymbol{v} = \boldsymbol{f} \tag{I.15}$$

avec $\boldsymbol{f}_i$ les forces nodales, $\boldsymbol{v}_i$ les vitesses nodales et $\boldsymbol{A}$ la matrice $N \times N$ dont les coefficients à valeurs tensorielles sont :

$$A_{ij} = \int_{\mathscr{D}} N_i(\boldsymbol{x})N_j(\boldsymbol{x})\boldsymbol{B}d\boldsymbol{x} \tag{I.16}$$

De par la définition des fonctions de base N_i, A_{ij} est nul lorsque les nœuds i et j ne sont pas connectés par un segment. La matrice $\boldsymbol{A}$ est donc une matrice creuse. Il est possible de résoudre le système explicitement, mais cela est coûteux. Une simplification qui consiste à supposer que les vitesses des segments connectés entre eux sont similaires $\boldsymbol{v}_i \approx \boldsymbol{v}_j$ [18] permet d'obtenir une vitesse nodale de manière locale. Le système linéaire à résoudre se simplifie :

$$\sum_j A_{ij}\boldsymbol{v}_j \approx (\sum_j A_{ij})\boldsymbol{v}_i \tag{I.17}$$

$$\forall i \; A_{i,tot}\boldsymbol{v}_i = \boldsymbol{f}_i \tag{I.18}$$

avec

$$A_{i,tot} = \sum_{j \text{ connecté à } i} A_{ij} \tag{I.19}$$

Pour chaque nœud i il suffit alors de résoudre le système 3×3 de l'équation I.18 pour obtenir la vitesse nodale.

Une fois la vitesse calculée, il faut l'intégrer pour déterminer la nouvelle position du nœud :

$$\boldsymbol{p}(t+\Delta t) = \boldsymbol{p}(t) + \int_{\tau=t}^{d+\Delta t} \boldsymbol{v}(\tau) d\tau \tag{I.20}$$

La méthode la plus simple est le schéma d'Euler défini par :

$$\boldsymbol{p}(t+\Delta t) = \boldsymbol{p}(t) + \boldsymbol{v}(t)\Delta t \tag{I.21}$$

Cette méthode d'ordre 1 nécessite un pas de temps très faible pour être stable. Il dépend de la vitesse de déplacement et de la longueur caractéristique des objets. Une condition empirique consiste à limiter la distance que parcourent les objets à une fraction de la longueur des segments. Par exemple, avec C une constante déterminée empiriquement, L_{min} la longueur du plus petit segment et v_{max} la vitesse du nœud le plus rapide :

$$dt_{max} = C\frac{L_{min}}{v_{max}} \tag{I.22}$$

Des intégrateurs prédicteur-correcteur peuvent aussi être utilisés [105]. L'ordre d'un tel intégrateur est plus élevé, et reste stable pour des pas de temps plus grands. Le nombre de pas de temps à effectuer pour la même simulation serait moindre, mais chacun serait plus coûteux. Il y a ici un compromis à trouver entre précision et rapidité pour l'intégrateur.

I.3.5 Opérations topologiques

Les opérations topologiques se chargent de modifier la connectivité du réseau de dislocation en fonction des évènements de la simulation. Les segments qui entrent en collision peuvent fusionner et former des jonctions, et le maillage doit parfois être modifié pour maintenir l'homogénéité de la taille des segments.

Les opérations topologiques à effectuer lorsque deux lignes de dislocations entrent en collision sont gérées par l'algorithme de collision. Son objectif est de construire une topologie qui respecte la physique des dislocations en fusionnant les segments lorsqu'une collision se produit. Il s'agit de l'une des étapes relativement coûteuses de la simulation, car il faut effectuer une détection de collision pour toutes les paires de segments de la simulation.

La formation de jonctions est un phénomène complexe, et la topologie qui résulte du contact entre deux dislocations dépend beaucoup de la nature de celles-ci [19, 109, 102]. Après l'exécution de l'algorithme de collision, un algorithme de scission des nœuds physiques (*splitnode*) se charge de sélectionner la configuration qui maximise la dissipation du réseau de dislocations. Lorsqu'un nœud physique est connecté à plus de 2 segments, il peut être plus favorable énergétiquement que ce nœud soit scindé en deux pour former une jonction. La figure I.23 énumère quelques configurations possibles pour la formation d'une jonction lorsque deux lignes entrent en contact.

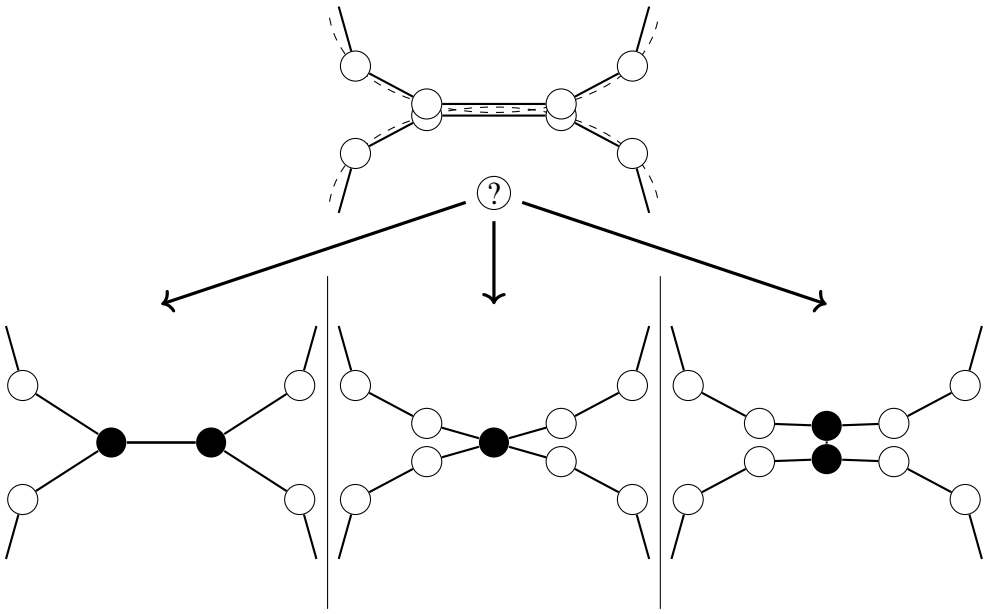


FIGURE I.23 – Configurations possibles lors de la formation de jonctions. Lorsque les dislocations entrent en contact, l'algorithme de collision effectue une des modifications topologiques possibles. L'algorithme de scission choisit ensuite la configuration la plus favorable énergiquement pour les nœuds physiques créés.

Afin de maintenir une discrétisation homogène, il peut être nécessaire de rediscrétiser le réseau de dislocations. L'algorithme de remaillage se charge de fusionner les segments trop courts et de scinder les segments trop longs.

Nous verrons que les opérations topologiques posent de nombreux problèmes d'optimisation des performances. Les modifications de la connectivité du réseau de dislocations nécessitent une synchronisation dans un environnement parallèle. L'ajout et la suppression d'objets dans le réseau de dislocations conduisent à des variations potentiellement importantes dans le nombre d'objets simulés et nuisent à la régularité du calcul.

I.4 Vers les simulations massives de dynamique des dislocations

Si l'on est capable de faire la transition entre l'échelle mésocopique et l'échelle des milieux continus pour certains phénomènes (sec. I.2.3), d'autres mécanismes sont encore hors de portée des simulations de dynamique des dislocations et demandent d'effectuer des simulations à plus grande échelle. Pour permettre les simulations massives qui mettent en jeu de nombreux segments, un travail d'optimisation des algorithmes et des codes est nécessaire. L'utilisation des ordinateurs parallèles et distribués permet d'augmenter la puissance de calcul disponible.

I.4.1 La méthode des multipôles rapide

L'optimisation des algorithmes, et notamment des modèles numériques, permet de réduire la complexité dans les phases coûteuses comme le calcul des forces. Certains codes utilisent des approximations en champ lointain. Par exemple MicroMegas/Tridis [104] où les dislocations

sont placées dans des cellules, et les contributions des particules au champ lointain sont regroupées au centre de chaque cellule. D'autres codes comme ParaDis et Optidis utilisent une méthode hiérarchique pour l'approximation du champ lointain.

Le calcul des forces d'interactions entre dislocations décrit en section I.3.3 est un exemple de calcul N-corps. Dans le cas de la dynamique des dislocations discrète, N segments de dislocations interagissent via la force de PEACH-KOEHLER et les champs de contraintes qu'ils génèrent. Les simulations N-corps sont coûteuses en temps de calcul, car elles nécessitent de calculer les interactions entre chaque paire d'objets (les f_{ij} dans la section I.3.3). La complexité d'un tel calcul est quadratique.

La Méthode des Multipôles Rapide (FMM - *Fast Multipole Method*) permet d'accélérer les calculs N-corps grâce à des approximations de champ lointain [56]. Elle est utilisée dans Optidis [16] et dans ParaDis [7] pour accélérer le calcul des forces d'interactions entre dislocations. ScalFMM est une librairie qui permet d'utiliser la FMM sur des architectures parallèles et distribuées [4] et est utilisée dans Optidis.

Pour réduire la complexité du calcul des forces qui pose problème lorsque l'on souhaite augmenter le nombre de segments simulés on utilise la FMM qui effectue un calcul exact pour les segments qui sont proches, mais qui approxime la contribution des objets lointains.

L'espace est partitionné hiérarchiquement en boites. Chaque boite est successivement découpée en 8 sous-boites dans le cas de la dimension 3. Les boites des différents niveaux de découpage sont organisées selon un arbre, comme l'illustre la figure I.24. Les boites du niveau le plus bas appelées *feuilles* contiennent les particules (i.e. les objets à simuler, dans notre cas les segments), celles des autres niveaux sont appelées *cellules*. Les cellules contiennent les développements multipolaires et les développements locaux qui sont utilisés pour le calcul du champ lointain.

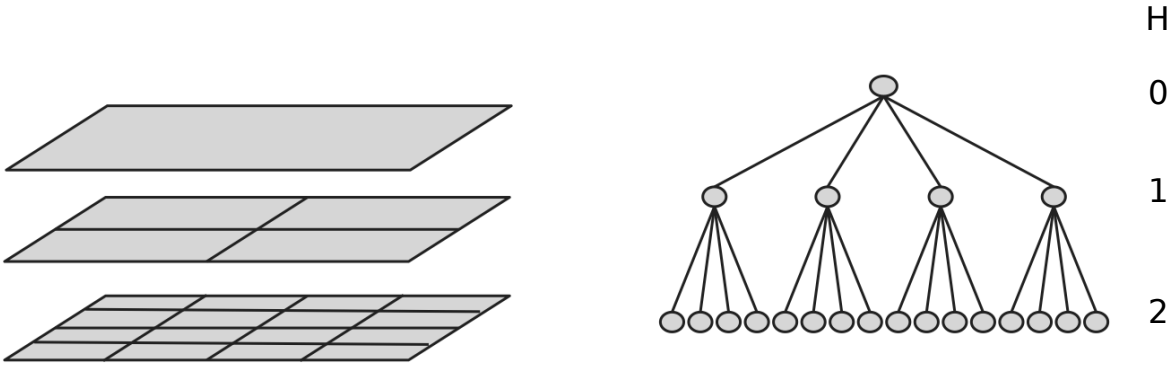


FIGURE I.24 – Découpage du plan en boites FMM. L'arbre représente les découpages successifs du plan. Le plan entier est découpé en 4 cellules, puis chaque cellule est découpée en 4 feuilles.

Le champ proche est l'interaction entre les particules voisines dans les feuilles, comme l'illustre la figure I.25. Les interactions entre les particules de boites voisines sont calculées de manière exacte (P2P - *Particle to Particle*).

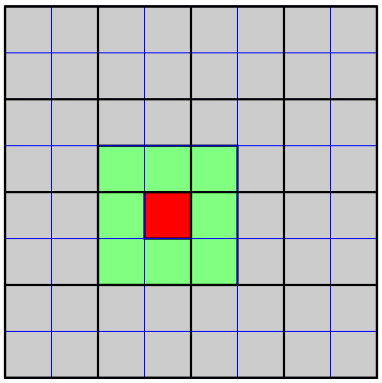


FIGURE I.25 – Voisins directs d'une boite. Les particules des boites vertes voisines de la boite rouge contribuent au champ proche de cette dernière. Celles des boites grises contribuent au champ lointain qui est approximé.

Le champ lointain est calculé en regroupant hiérarchiquement les contributions des particules. Les contributions au champ lointain des particules lointaines sont regroupées dans des multipôles pour ensuite être transmises aux particules locales via les développements locaux.

Les développements multipolaires sont calculés en remontant l'arbre depuis les feuilles (P2M - *Particles to Multipole*) vers les cellules des niveaux successifs (M2M - *Multipole to Multipole*). Les multipôles sont alors convertis (M2L - *Multipole to Local*) en développements locaux qui représentent le champ lointain à appliquer à toutes les particules des sous-boites d'une cellule. Ces développements locaux sont ensuite transmis vers le bas aux niveaux hiérarchiques successifs (L2L - *Local to Local*) puis sont appliqués aux particules (L2P - *Local to Particle*).

La FMM a été adaptée à la dynamique des dislocations dans Optidis [16]. L'interpolation de Chebyshev ou l'interpolation sur une grille uniforme est utilisée dans les multipôles pour approcher le champ lointain. Afin d'appliquer la FMM aux dislocations, certaines différences avec les particules classiques doivent être prises en compte. Le caractère non ponctuel des segments de dislocations nécessite par exemple d'utiliser des boites étendues pour l'interpolation.

I.4.2 Utilisation des architectures parallèles et distribuées

Les simulations de dynamique des dislocations se complexifient et nécessitent de plus en plus de puissance de calcul. Les codes de dynamique des dislocations font appel au calcul haute performance (HPC). Si les architectures parallèles et distribuées permettent d'augmenter la puissance de calcul disponible, leur utilisation nécessite un investissement significatif d'adaptation des codes.

I.4.2.1 Parallélisme

Depuis les années 2000, la fréquence des processeurs n'évolue plus de manière significative, pourtant la puissance des ordinateurs continue d'augmenter. Pour maintenir l'augmentation de la puissance de calcul, des architectures parallèles ont vu le jour. Les supercalculateurs possèdent de plus en plus de cœurs de calcul pour permettre aujourd'hui d'effectuer plusieurs millions de milliards d'opérations par seconde : le supercalculateur le plus rapide permet de calculer à une vitesse de 122 PFlops/s en juin 2018 [1].

Les architectures actuelles sont complexes avec l'utilisation de nombreux niveaux de parallélisme. Les supercalculateurs sont constitués de plusieurs machines interconnectées et exécutent parallèlement le même programme sur des données différentes (SPMD - *Single Program Multiple Data*) en communiquant pour se synchroniser, par exemple en utilisant la bibliothèque d'échange de messages MPI (*Message Passing Interface*). Chacune de ces machines utilise le parallélisme : elles peuvent être composées de plusieurs processeurs possédant chacun plusieurs cœurs de calcul. Les programmes peuvent utiliser parallèlement plusieurs cœurs de calcul au sein d'une même machine avec l'utilisation du *multithreading*, par exemple avec des outils comme OpenMP. Enfin, chacun des cœurs de calculs permet le parallélisme en effectuant plusieurs opérations simultanément, avec par exemple des unités de calcul *vectorielles* qui permettent d'effectuer des opérations sur plusieurs valeurs en une seule instruction (SIMD - *Single Instruction Multiple Data*). L'exploitation de ces différents niveaux de parallélisme est une tâche complexe qui nécessite un travail d'adaptation des codes de calcul. Plus l'architecture utilisée exploite le parallélisme, plus le travail d'adaptation est important.

Les accélérateurs de calcul permettent d'aller encore plus loin dans le parallélisme en proposant des architectures spécialisées pour certains types de calculs. Par exemple les accélérateurs graphiques (GPU) sont très efficaces pour effectuer des traitements simples sur un grand nombre de données. Des accélérateurs plus génériques, comme par exemple le Xeon Phi qui peuvent interpréter les mêmes programmes que les processeurs classiques existent. D'autres accélérateurs sont plus spécialisés, mais plus efficaces, comme par exemple les ASICs (*Application-specific integrated circuit*) dans lesquels le programme est intégré directement dans le circuit électronique.

I.4.2.2 Dans les codes de dynamique des dislocations

Les codes de dynamique des dislocations utilisent principalement le parallélisme à gros grain (SPMD) pour accélérer les calculs. Le calcul distribué avec MPI est par exemple utilisé dans Tridis [104], Model [115], ParaDis [18, 7] et le support du multithreading est utilisé dans le code de Karlsruhe et Optidis afin de permettre des simulations à plus grande échelle. Certains simulateurs, comme le code de Karlsruhe, se parallélisent sur un nombre réduit de cœurs de calcul, alors que ParaDis et Optidis visent des simulations à plus grande échelle. D'autres codes comme DDLab commencent à utiliser les GPUs pour accélérer le calcul [41].

Les problèmes de performance de ces codes de dynamique des dislocations en mémoire distribuée sont principalement rencontrés dans le calcul des forces. Les méthodes non hiérarchiques d'approximation du champ lointain utilisées par exemple dans Tridis génèrent un volume important de communications MPI, ce qui limite la capacité du code à être efficace sur un nombre important de cœurs de calcul.

Les problématiques de parallélisme à grain fin ne sont que peu abordées dans le domaine de la simulation de dynamique des dislocations. Les codes utilisent souvent seulement une parallélisation MPI (Tridis, ParaDis, etc.), et ceux qui utilisent le parallélisme de threads (Karlsruhe) n'utilisent pas la mémoire distribuée. La gestion du parallélisme intra-nœud, avec l'utilisation de OpenMP par exemple, permet de mieux exploiter plusieurs cœurs d'un même processeur qu'en utilisant seulement une parallélisation MPI. Les structures de données utilisées sont souvent un obstacle au parallélisme d'instructions (vectorisation), avec l'utilisation, par exemple dans NUMODIS, de structures comme les listes chainées. Le caractère dynamique des données freine la parallélisation à grain fin (intra-nœud).

Optidis est un projet qui consiste à porter le simulateur Numodis sur les architectures parallèles et distribuées, et est le code utilisé tout au long de ce manuscrit. La mise en place de la méthode des multipôles rapides [16] dans Optidis a permis de lever certaines des contraintes liées au volume des données échangées via MPI. Le calcul des forces est notamment parallélisé à travers la bibliothèque ScalFMM [4]. Optidis fait aussi évoluer certains algorithmes de Numodis pour répondre à d'autres problématiques du parallélisme [40]. Un changement majeur réside dans la représentation des lignes de dislocations. Numodis considère les lignes de dislocations comme des objets à part entière : une ligne est composée de plusieurs nœuds et segments stockés dans une liste chainée, et le réseau de dislocations est composé d'un ensemble de lignes interconnectées par les nœuds physiques. Optidis introduit un maillage où chaque segment est indépendant, ce qui permet d'être en accord avec la vision particulaire de la FMM, et de distribuer plus facilement les dislocations sur plusieurs machines. Les différents algorithmes de la simulation ont été adaptés en utilisant un parallélisme hybride MPI et OpenMP.

I.4.3 Positionnement

L'objectif de cette thèse est de répondre aux nombreux défis informatiques que posent encore les simulations massives de dynamique des dislocations sur architectures parallèles et distribuées.

Le nombre de segments dans la dynamique des dislocations varie beaucoup au cours du temps. Les opérations topologiques de formation de jonctions et de remaillage ajoutent et suppriment des objets à la simulation. Le caractère dynamique des données pose un défi majeur pour la performance et la parallélisation à grain fin des simulations massives. Comme première contribution, je propose une structure de données adaptée à la dynamique des dislocations qui prend en compte le caractère dynamique des données, tout en permettant une bonne performance dans les différents niveaux de parallélisme. Elle permet d'accéder simplement et rapidement aux données, même dans un environnement distribué.

Ma seconde contribution concerne plus particulièrement l'algorithme de collision (sec. I.3.5) qui pose des problématiques multiples. La fiabilité de l'algorithme de collision est cruciale pour le bon déroulement de la simulation, notamment pour la formation des jonctions. Il est important que les collisions soient traitées de manière fiable pour éviter des instabilités qui forcent, par exemple, à limiter le pas de temps utilisé. Ces instabilités sont encore plus présentes lorsque le nombre de segments est grand. C'est pourquoi je propose un nouvel algorithme de détection et de gestion des collisions plus fiable. L'algorithme de détection des collisions est un algorithme N-corps, une attention est donc aussi portée à sa performance.

L'impact sur la performance de mes contributions sera mesuré dans les différentes parties. La performance globale du code sera étudiée dans la partie finale sur des simulations à grande échelle.

Bilan

Nous avons vu dans cette première partie ce que sont les dislocations et quel est leur impact sur le comportement mécanique des matériaux. L'étude et la simulation du comportement des dislocations s'inscrit à l'échelle mésoscopique dans la modélisation multi échelle des matériaux, et permet de modéliser les phénomènes liés à la plasticité. La complexité croissante des simulations de dynamique des dislocations augmente les besoins en calcul et nécessite de recourir au calcul haute performance. L'objectif de cette thèse est d'améliorer la fiabilité et la performance du simulateur Optidis dans un environnement parallèle et distribué.

Partie II

Structure de données

Introduction

La manière de stocker les informations liées aux objets simulés est d'une importance cruciale pour la flexibilité et la performance des algorithmes. L'annihilation et la multiplication des dislocations font varier fortement le nombre de segments au cours du temps. C'est pourquoi il est nécessaire de mettre en place une structure spécifique qui prenne en compte le caractère dynamique des données. Après avoir posé le problème et fait un état des lieux des solutions de stockage des données dynamiques, je propose une structure de données qui permet d'accéder aux informations liées au réseau de dislocations de manière transparente et performante afin d'écrire des algorithmes plus fiables et plus maintenables sans compromis sur la vitesse d'exécution, même dans un contexte distribué.

II.1 Une structure de données adaptée à la dynamique des dislocations

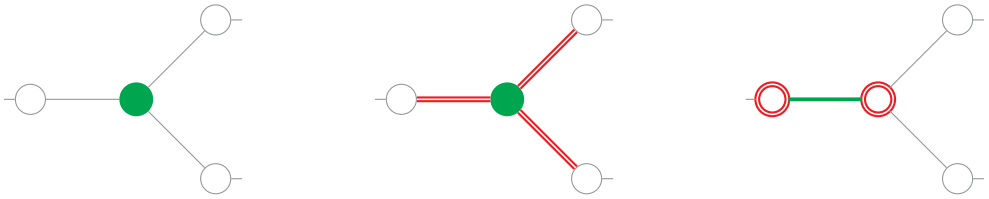
Les données stockées et utilisées sont en majorité contenues dans le réseau de dislocations. Les lignes de dislocations sont discrétisées en une multitude de segments qui connectent des nœuds. Les algorithmes utilisent un certain nombre d'informations, qu'elles soient portées par la topologie du réseau, ou par les objets contenus dans ce réseau.

Le réseau est considéré comme un maillage éléments finis dans les phases calculatoires où l'accès aux données doit être rapide. Dans les autres cas, c'est sa topologie qui est au centre des algorithmes, comme lors de la phase de collision. Le caractère unidimensionnel de ce maillage ainsi que la dynamicité de sa topologie poussent à réfléchir hors du cadre des maillages classiques éléments finis afin d'avoir une solution de stockage plus adaptée.

II.1.1 Les phases de calcul intensif

Comme dans tout maillage éléments finis, chaque nœud et segment porte des propriétés physiques et des inconnues qui doivent être stockées dans une structure de données afin de transiter entre les différentes étapes de la simulation. À chaque nœud, on associe une position x_i, une vitesse v_i, une force nodale f_i ainsi que d'autres données spécifiques à certains algorithmes de la simulation. Chaque segment contient des informations physiques comme le système de glissement g_j composé d'un vecteur de Burgers b_j et d'un plan de glissement n_j.

Les algorithmes calculatoires parcourent les objets du maillage pour les modifier. Par exemple, l'algorithme de calcul du déplacement (section I.3.4) accède à la position et la vitesse de chaque nœud pour en déterminer sa nouvelle position. On distingue seulement trois types de parcours des données dans les algorithmes, illustrés en figure II.1.



(a) Nœuds seuls. (b) Nœuds avec segments connectés. (c) Segments avec nœuds connectés.

FIGURE II.1 – Accès aux données lors des parcours du réseau de dislocations. Les objets verts peuvent être accédés en lecture/écriture alors que les objets rouges sont accédés en lecture seule.

Le parcours simple des nœuds (fig. II.1(a)) accède en lecture et en écriture à des données portées par les nœuds. Ce parcours est par exemple utilisé dans l'algorithme de déplacement des nœuds.

Le parcours des nœuds avec segments connectés (fig. II.1(b)) accède en lecture et écriture aux données des nœuds comme le parcours simple, mais peut aussi accéder en lecture seule aux données portées par les segments connectés. Il est par exemple utilisé dans le calcul des vitesses nodales qui applique les lois de mobilités sur chaque segment connecté.

Enfin, le parcours des segments avec nœuds connectés (fig. II.1(c)) permet d'accéder en lecture et écriture aux données des segments ainsi qu'en lecture seule aux données des nœuds connectés. Ce type de parcours permet par exemple de calculer le centre ou la longueur des segments en accédant aux positions des nœuds.

Ces parcours sont au cœur de nombreux algorithmes de la dynamique des dislocations et se doivent d'être réalisables efficacement. Une interface d'accès aux données doit donc pouvoir effectuer ces parcours simplement et avec une bonne performance. Les problématiques de parcours des données sont typiques pour les maillages de type éléments finis [12, 25], mais la spécificité du réseau de dislocations est son caractère dynamique.

II.1.2 Une topologie dynamique

Les informations topologiques décrivent la manière dont les nœuds et les segments sont connectés au sein du réseau de dislocations. Chaque segment est connecté aux deux nœuds à son extrémité, et chaque nœud peut être connecté à plusieurs segments. La topologie est modifiée par certains algorithmes au travers d'*opérations topologiques*, comme lors de la formation de jonctions ou le remaillage. Les algorithmes qui modifient la topologie doivent avoir la possibilité d'ajouter ou de supprimer des nœuds et des segments. À la différence des simulations éléments finis classiques [13], le réseau de dislocations voit son nombre d'objets varier fortement, et sa topologie changer au cours du temps.

Les données dynamiques sont difficiles à stocker efficacement. Les ajouts et suppressions fréquents dans une structure de données nuisent à la cohérence spatiale. La cohérence spatiale et temporelle est une base des principes d'accès aux données [32]. Les architectures matérielles des ordinateurs sont optimisées pour accéder à des données stockées proches les unes des autres, et permettent un accès plus rapide aux données contiguës et récemment utilisées.

Dans ScalFMM par exemple, les particules sont triées afin que celles qui sont proches dans l'espace le soient aussi en mémoire [4]. On utilise une *courbe de remplissage* (*space-filling curve*) [98] comme la *Z-curve*, aussi appelée *courbe de Lebesgue* ou *ordre de Morton*, illustrée figure II.2. Elle permet de linéariser l'espace pour obtenir une relation d'ordre totale qui garantit une bonne cohérence spatiale des données [4, 112]. L'espace est découpé en boites numérotées selon leur ordre sur la Z-curve, appelé *indice de Morton*. Les particules sont rangées par boites et ordonnées selon leur indice.

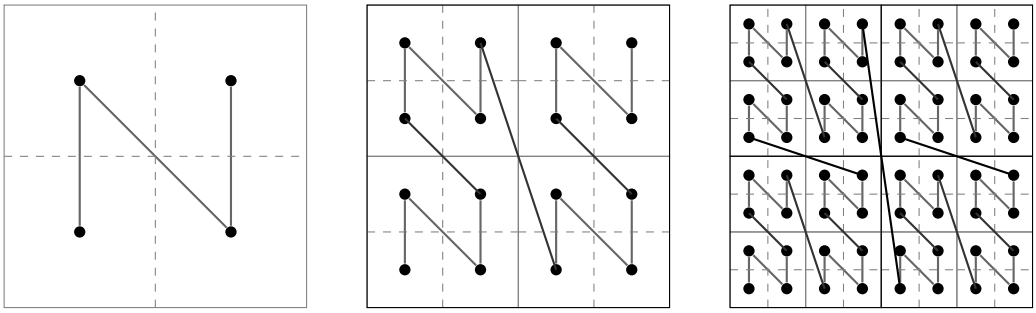


FIGURE II.2 – Z-curve : le plan est subdivisé récursivement et le motif en Z (ou N ici) est répété dans les sous-boites.

Le maintien de la localité spatiale s'obtient le plus souvent au prix d'une structure de données plus complexe ou d'un surcoût lié au tri des données. Pour les données dynamiques comme dans la dynamique des dislocations, il faut trouver un compromis entre performance d'insertion des données et maintien de la localité spatiale.

II.1.3 Interface de haut niveau

Idéalement, celui qui écrit les algorithmes physiques ne devrait pas avoir à se préoccuper des problématiques de performances. Il est possible d'abstraire les problématiques de stockage des données de l'implémentation des algorithmes en mettant en place une interface pour les parcours de la structure ainsi que pour les opérations topologiques. Des interfaces génériques pour les données géométriques existent [14], mais résolvent des problèmes bien plus larges que le nôtre. Il est plus adapté de mettre en place une interface *ad hoc* pour les dislocations.

En plus d'abstraire les problématiques de performances, une interface permet de cacher la complexité du maintien de la cohérence de la structure. Les algorithmes peuvent alors être écrits plus aisément. C'est encore plus crucial lorsque les données sont distribuées. L'objectif est donc de proposer une interface qui permette l'accès transparent aux données distribuées sans s'encombrer de maintenir la cohérence de la topologie ou d'optimiser les parcours de données. Il est ensuite possible d'implémenter cette interface de manière efficace en adressant les problématiques de stockage des maillages dynamiques.

II.2 État de l'art et positionnement

II.2.1 Représentation des maillages

Les maillages éléments finis se décomposent en deux catégories, les maillages *structurés* et *non-structurés* [51]. Les maillages dits *structurés* sont réguliers et leur topologie peut le plus souvent être représentée de manière implicite par leur position dans la structure de données. Cela permet un stockage plus efficace en termes de consommation mémoire et de vitesse d'exécution. Le réseau de dislocations est à l'inverse un maillage *non-structuré*. Il est moins régulier et sa topologie doit être représentée explicitement.

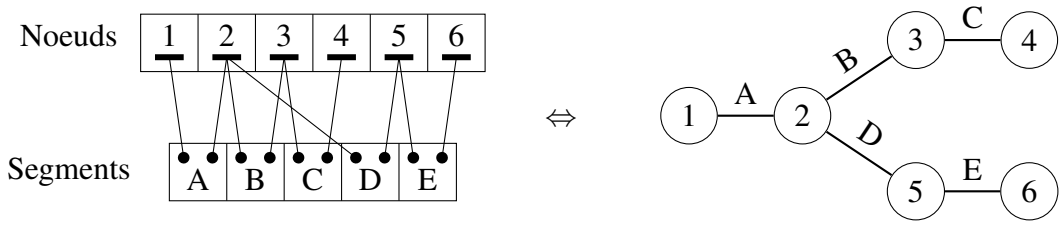
Des travaux [12, 25] présentent des structures de données compactes pour les maillages éléments finis, mais ne discutent pas des problématiques liées aux modifications des maillages. Des techniques adaptatives (AMR - Adaptive Mesh Refinement) existent pour raffiner les maillages localement [70]. Elles utilisent des maillages qui peuvent changer au cours du temps, mais les topologies reposent sur des structures hiérarchiques et ont un caractère structuré [79] non adapté aux dislocations.

Le réseau de dislocations est un maillage unidimensionnel extrêmement dynamique dans un espace à trois dimensions. Les problématiques de représentation informatique de ce type de maillage se rapprochent plus des problématiques de la représentation des graphes que de la représentation classique des maillages éléments finis. Un graphe est un objet mathématique constitué d'un ensemble de nœuds V reliés par un ensemble de segments E [118]. Cette vision s'adapte bien aux dislocations où V est l'ensemble des nœuds du réseau et E l'ensemble des segments qui les relient. À la différence des maillages classiques, aucun élément volumique ou surfacique n'apparait. Cette vision de graphe s'adapte bien aux opérations topologiques qui peuvent avoir lieu. L'ajout et la suppression de nœuds ou de segments se traduit directement en termes d'ajout et de suppression des objets du graphe.

Il existe plusieurs manières de représenter informatiquement un graphe, décrites dans de nombreux ouvrages d'algorithmique, comme par exemple GOODRICH *et. al.* [55] ou CORMEN *et. al.* [28]. Il est possible d'utiliser des représentations implicites des graphes. C'est par exemple ce qui est utilisé pour les maillages structurés où la topologie n'a pas besoin d'être explicitée. Bien que les représentations implicites des graphes permettent de réduire l'empreinte mémoire, elles ne s'adaptent ni aux maillages non structurés ni aux données dynamiques. Une deuxième représentation, la *matrice d'adjacence*, utilise une matrice carrée où les lignes représentent le nœud de départ d'un segment et les colonnes le nœud d'arrivée. Chaque case de la matrice représente alors un segment. Ce type de représentation est adaptée aux graphes denses dont une partie importante de la matrice d'incidence est remplie. De plus, l'ajout de nœuds au graphe nécessite d'augmenter la taille de la matrice, ce qui en fait un mauvais candidat pour les données dynamiques. Pour finir, la représentation par *liste d'adjacence* représente le graphe comme un ensemble de nœuds qui possèdent chacun une liste de connectivité avec d'autres nœuds. Cette représentation est plus adaptée aux graphes dont le degré des nœuds est faible comme c'est le cas du réseau de dislocations. De plus, les modifications topologiques n'impactent que les listes des nœuds concernés, ce qui fait de la représentation par *liste d'adjacence* la bonne méthode pour les topologies dynamiques.

La représentation utilisée dans Optidis [40] découle d'une implémentation orientée objet de graphe [55] qui se rapproche d'une représentation par *liste d'adjacence*. Les nœuds et segments

sont représentés par des objets contenant les données présentées en section II.1.1 et des informations sur la connectivité du graphe. Chaque nœud possède une liste des indices des segments qui lui sont incidents, et chaque segment contient les indices des deux nœuds connectés, comme l'illustre la figure II.3.



(a) Représentation mémoire.

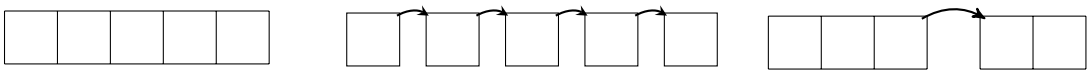
(b) Maillage des dislocations.

FIGURE II.3 – Représentation du maillage. Les nœuds numérotés de 1 à 6 et les segments de A à E sont stockés dans deux conteneurs différents. Le nœud 2 est par exemple connecté aux segments A, B et D et le segment C est connecté aux nœuds 3 et 4.

On prendra soin de stocker les données des nœuds et des segments dans des conteneurs adaptés aux données dynamiques.

II.2.2 Conteneurs adaptés aux données dynamiques

Pour que le parcours des données soit performant, il faut que leur organisation respecte certaines règles. Différentes manières de stocker les données dynamiques existent et possèdent des avantages et des contraintes différents. Trouver la bonne structure pour son utilisation, c'est trouver le bon compromis entre la performance du parcours et la capacité à insérer et supprimer des objets. On identifie deux grandes classes de conteneurs, les tableaux et les structures avec indirection [55, 28].



(a) Tableaux contigus.

(b) Liste chainée.

(c) *Blocklist*.

FIGURE II.4 – 3 types de conteneurs pour données dynamiques.

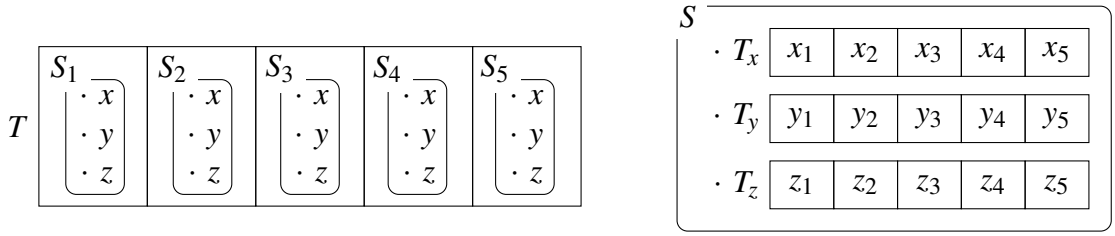
Les *tableaux contigus* (fig. II.4(a)) sont souvent utilisés pour représenter les maillages fixes, car ils permettent un parcours efficace des données grâce au stockage contigu des éléments. La contiguïté des données est importante pour la performance, car elle permet aux compilateurs d'effectuer certaines optimisations, comme la *vectorisation* [15]. Toutefois, ce type de conteneurs pose des problèmes pour stocker des données dynamiques. La suppression d'un élément nécessite soit de déplacer d'autres éléments pour maintenir la contiguïté, soit de laisser un "trou" qui fragmente la mémoire. L'ajout d'un élément lorsque le tableau est plein peut nécessiter une copie du tableau dans un espace mémoire plus grand. Il faut alors réallouer le tableau en trouvant un bon compromis entre la consommation mémoire et la fréquence de réallocation.

Des structures avec indirections, comme par exemple la *liste chainée* (fig. II.4(b)) permettent d'ajouter ou de supprimer facilement des maillons à la chaine, mais ne permettent pas de parcourir les données de manière efficace. Des travaux existent pour augmenter la localité spatiale de ce type de conteneurs [48] et ont inspiré la mise en place d'une structure de données hybride appelée *Blocklist* (fig. II.4(c)) dans Optidis [40]. Elle consiste en une liste chainée de tableaux et permet un parcours contigu des blocs tout en permettant d'insérer et de supprimer des éléments facilement.

Ces conteneurs permettent de stocker une donnée par case, alors que les objets de la simulation possèdent plusieurs champs. L'organisation en mémoire de ces champs est importante pour la performance des parcours dans les phases calculatoires.

II.2.3 Organisation des champs au sein des conteneurs

Il est possible d'organiser les champs des objets selon deux dispositions. Toutes les informations relatives à un objet peuvent être regroupées en mémoire, comme l'illustre la Figure II.5a. On parle de tableau de structures (AoS - *Array of Structures*). À l'inverse, il est possible de regrouper les champs des objets dans une structure de tableaux (SoA - *Structure of Arrays*), comme dans la figure II.5b.



(a) Array of Structures :
Chaque objet occupe une case du tableau.

(b) Structure of Arrays :
Chaque champ possède son tableau dédié.

FIGURE II.5 – Représentation mémoire d'un tableau d'objets à plusieurs champs.

La représentation AoS est la plus intuitive, mais l'utilisation d'une représentation SoA permet souvent de meilleures performances [59]. Par exemple, les champs non utilisés n'ont pas à être chargés dans le cache et la contiguïté des valeurs permet la vectorisation [15, 63].

Bien que la figure II.3 illustrant la représentation de graphe utilise une organisation *Array of Structures*, la disposition réelle des données en mémoire reste libre. Il est donc possible d'utiliser une organisation SoA ou AoS pour stocker les objets du maillage.

II.2.4 Positionnement

Au-delà du choix d'une représentation des données efficace, ma contribution vise à formaliser une interface d'accès aux données qui permette d'abstraire leur représentation de leur utilisation. Cette abstraction prend la forme d'un *type abstrait de données*.

Un Type Abstrait de Données (TAD) est une définition formelle d'un ensemble d'opérations sur les données qui constitue une interface entre le domaine de l'applicatif et de l'implémentation [31]. En d'autres termes, il s'agit d'une séparation entre les considérations qui relèvent de l'implémentation des algorithmes et les problématiques d'optimisation de la performance.

Dans notre cas, un type abstrait de données qui représente le réseau de dislocations doit permettre d'écrire des algorithmes de haut niveau sans avoir à se préoccuper de la cohérence des données tout en permettant une bonne performance. Le but est de donner au physicien écrivant les algorithmes un outil pour accéder aux données sans se préoccuper des problématiques liées à la performance, et à l'informaticien le moyen d'essayer différents schémas de stockage et d'optimiser la performance sans interférer avec les algorithmes.

L'absence d'une telle abstraction dans Optidis posait des problèmes de maintenabilité et de fiabilité du code, mais aussi de performances. Maintenir manuellement la cohérence des données entre les processus MPI lors des opérations topologiques nuisait à la lisibilité des algorithmes et l'absence d'une interface unifiée pour le parcours des données rendait l'optimisation des parcours plus difficile.

J'ai implémenté les différentes méthodes du type abstrait décrit en section II.3.1. Les données sont stockées selon le modèle de graphe décrit en section II.2.1. Les informations liées aux nœuds et aux segments sont stockées en utilisant des tableaux dynamiques `std::vector`. La disposition mémoire choisie est l'organisation *Structure of Array* pour ses performances, cependant les champs x, y et z des vecteurs à trois dimensions sont regroupés.

La représentation des données utilise des conteneurs de la STL afin de permettre une meilleure fiabilité et une meilleure maintenabilité du code. L'objectif est d'avoir une organisation des données simple qui permette d'aborder plus sereinement les problématiques comme le parallélisme et la distribution des données. L'interface proposée a d'ailleurs été pensée dans ce contexte et offre des moyens d'accéder à des données distantes en mémoire distribuée.

Nous verrons en mesurant les performances de l'implémentation à base de tableaux que la simplicité d'une représentation n'en fait pas forcément un mauvais choix en termes de performances.

II.3 Structure de données à base de tableaux simples

La nouvelle interface d'accès aux données au sein d'Optidis permet de lire, de modifier et de parcourir les informations liées au réseau de dislocations de manière simple et efficace. Je commence par présenter le type abstrait de données qui constitue l'interface d'accès aux informations du réseau de dislocations, puis je propose une implémentation de la structure de données à base de tableaux simples et je détaille les solutions mises en place pour répondre aux différents challenges que posent les données dynamiques.

II.3.1 L'interface

La structure de données est composée de deux conteneurs associatifs `Nodes` et `Segments` pour stocker respectivement les nœuds et les segments. Ces deux sous-classes s'inspirent de `std::map` de la librairie standard C++ [66, 64]. Par exemple, `Nodes` possède une interface similaire à `std::map<NodeIndex_local,NodeInfo>` qui est détaillée en figure II.6.

```
struct Nodes{
    //Modifications topologiques
    NodeIndex_global insert(const NodeInfo&  n, int insert_rank = -1);
    void             erase (const NodeIndex_global& n);
    //Accès à la topologie
    SegmentIndex_list getConnectedSegments (const NodeIndex_local& n) const;
    //Accès aux données des objets
    size_t size() const;
    NodeInfo get(const NodeIndex_global& n) const;
    void     set(const NodeIndex_global& n, const NodeInfo& n_info);
    NodeInfo_ref at         (const NodeIndex_local& n);
    NodeInfo_ref operator[] (const NodeIndex_local& n);
    //Accès aux itérateurs
    NodeIterator begin      () ;
    NodeIterator end        () ;
};
```

FIGURE II.6 – Interface d'accès aux nœuds dans la structure de données. Elle utilise des noms de méthodes proches de celles de `std::map`, comme par exemple `insert()`, `erase()` et l'opérateur `[]`.

L'accès aux données portées par les objets se fait via l'opérateur `[]` ou les méthodes `get()` et `set()`. L'opérateur `[]` permet à la fois la lecture et l'écriture des données alors que `get()` permet seulement la lecture et `set()` seulement l'écriture. Pour les nœuds, ces fonctions utilisent des indices `NodeIndex_local` et `NodeIndex_global` pour repérer respectivement les objets locaux et ceux potentiellement distants en mémoire distribuée. L'accès aux données se fait via les objets `NodeInfo` et `NodeInfo_ref`. `NodeInfo` est une structure classique qui contient les champs des objets, alors que `NodeInfo_ref` est une interface d'abstraction qui permet d'accéder aux différents champs en lecture et écriture via des méthodes qui retournent des références.

Il est possible de modifier la topologie en ajoutant ou en supprimant des objets avec les méthodes `insert()` et `erase()`. Les informations topologiques sont accessibles via la méthode `find()` qui permet de déterminer si deux nœuds sont connectés par un segment et la méthode `getConnectedSegments()` qui donne la liste des segments connectés à un nœud.

Pour parcourir les données, des itérateurs sont mis à disposition avec les méthodes `begin()` et `end()` classiques en C++. Des fonctions utilisant des *lambda-fonctions* permettent d'effectuer plus efficacement les parcours décrits en section II.1.

L'interface complète avec la description des différents types, des méthodes d'accès ainsi que des fonctions de parcours est disponible en l'annexe A.1. Des exemples d'utilisation des fonctions parcours y sont aussi présentés.

Cette interface a été implémentée en utilisant des tableaux `std::vector`. Je décris dans les sections suivantes comment les problématiques liées aux données dynamiques ont été traitées pour implémenter efficacement cette interface.

II.3.2 Ajouts, suppression et fragmentation mémoire

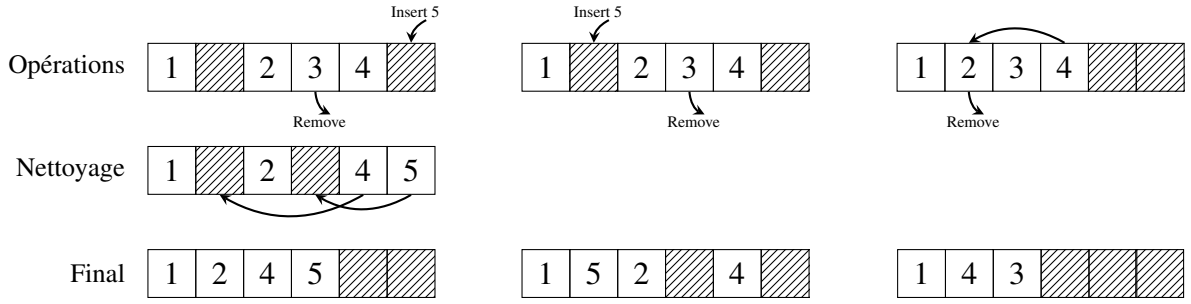
Les ajouts/suppressions d'objets dans les tableaux posent des problèmes de fragmentation mémoire en laissant des "trous" dans la structure de données. Plusieurs solutions d'implémentation de `insert()` et `erase()` ont été explorées pour limiter la fragmentation.

II.3.2.1 Méthodes envisagées pour réduire la fragmentation mémoire

La première solution réduit la fragmentation mémoire en utilisant un *nettoyage périodique* (fig. II.7a), aussi appelé *garbage collection*. Rien n'est fait à l'insertion et à la suppression pour éviter la fragmentation. Les objets supprimés sont invalidés en laissant un trou dans le tableau, et les insertions se font à la fin de l'espace mémoire. Les données sont réarrangées périodiquement afin de rétablir la contiguïté des données. Cette solution ne nécessite pas d'effectuer des opérations complexes à chaque insertion et suppression, mais ne respecte pas la contiguïté des données : lors des parcours il faudra tester que l'élément est valide, ce qui peut limiter la performance.

La deuxième solution effectue un *remplissage à l'insertion* (fig. II.7b). Les éléments supprimés sont invalidés de la même manière, mais les éléments sont insérés de façon à remplir les "trous" dans le tableau. Cette solution complexifie l'implémentation de la méthode d'insertion mais permet une meilleure efficacité mémoire sans avoir recours à un nettoyage. Des éléments invalidés peuvent tout de même subsister donc cette solution pose les mêmes problèmes de contiguïté que la solution précédente.

La dernière solution force la contiguïté des données en utilisant un *remplissage à la suppression* (fig. II.7c). Lorsqu'un objet est supprimé, le dernier objet du tableau est déplacé à la position libérée. Cette solution permet de maintenir la contiguïté des données, ce qui permet de meilleures performances pour les parcours. Elle complexifie cependant la méthode de suppression. Le déplacement d'un objet nécessite de mettre à jour la topologie ce qui peut être très coûteux.



(a) Nettoyage périodique : 3 laisse un vide lorsqu'il est supprimé et 5 s'insère à la fin du tableau. Une opération de nettoyage ultérieure trie et rassemble les données.

(b) Remplissage à l'insertion : 5 est inséré au premier emplacement vide.

(c) Remplissage à la suppression : lorsque 2 est supprimé, 4 est déplacé dans l'emplacement vide.

FIGURE II.7 – Solutions à la fragmentation mémoire.

J'ai choisi d'implémenter première solution présentée (*Nettoyage périodique*) pour des raisons de simplicité, car l'objectif premier de cette implémentation est d'améliorer la fiabilité du simulateur. Cette implémentation possède des propriétés intéressantes du point de vue de l'invalidation des itérateurs dont il est important de se préoccuper pour la validité du programme [92].

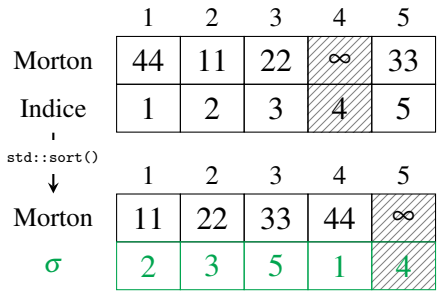
Cette solution ne déplace pas les éléments à l'ajout ou à la suppression, ce qui assure que les indices pointent toujours sur le même objet après l'opération. Ce n'est par exemple pas le cas du *Remplissage à la suppression*. De plus, les cases vides ne sont pas réécrites à l'insertion ou

à la suppression, ce qui permet de savoir si un indice correspond à un objet supprimé. Prenons pour exemple la situation de la figure II.7c. On possède un indice pointant vers la 2^{eme} case du tableau. Cet indice permet avant les modifications l'accès à l'élément 2. Après la suppression de l'élément 2, cet indice pointe vers 4. Il n'est pas possible de savoir si notre indice pointe vers un élément supprimé car l'élément a été remplacé.

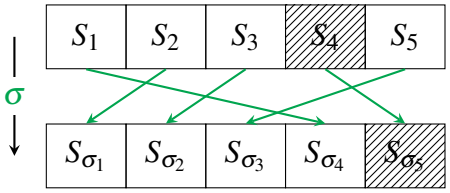
Les algorithmes d'Optidis utilisent ces propriétés induites par l'utilisation du *Nettoyage périodique*. Un indice doit pointer vers le même objet avant et après les opérations d'ajout / suppression, et il doit être possible de déterminer si un indice pointe vers un objet supprimé. Ces propriétés sont vraies pour les opérations d'ajout et de suppression, mais l'opération nettoyage de la structure de données peut invalider tous les indices. Il est possible d'utiliser les autres méthodes pour éviter la fragmentation, mais l'implémentation devra respecter ces propriétés.

II.3.2.2 Nettoyage périodique

Pour éviter la fragmentation mémoire, un nettoyage périodique est effectué. La solution mise en place consiste à trier les objets selon leur indice de Morton déterminé en section II.4.2.1, puis à supprimer les objets invalidés. Cette solution permet de maintenir la cohérence spatiale des objets en plus de remplir les trous dans les tableaux. Le tri s'effectue en deux étapes comme décrit dans la figure II.8. Une première étape de tri de complexité $O(nlog(n))$ génère une permutation. Dans un second temps, la permutation est appliquée aux éléments des conteneurs en les copiant dans l'ordre dans un nouveau tableau. Ce tri en deux temps permet de réduire le nombre de déplacements des données.



(a) La permutation σ est générée en triant une liste de paires {Morton, indice} avec `std::sort()`.



(b) Les éléments du conteneur sont copiés dans un nouveau tableau selon la permutation σ.

FIGURE II.8 – Tri des objets lors de la défragmentation.

La permutation est générée en utilisant `std::sort()` sur un tableau σ contenant des paires $\{morton, indice\}$. σ est construit en utilisant l'indice de Morton du i^{eme} objet, et la position courante de l'objet :

$$\sigma_i \leftarrow \{m_i, i\} \tag{II.1}$$

Pour les objets invalidés, on utilise $+\infty$ (`INT_MAX` informatiquement) comme indice de Morton, de manière à ce que ces objets soient déplacés en fin de tableau. Une fois le tableau σ trié selon les indices de Morton, l'indice contenu dans chaque paire donne la permutation $\sigma(i)$ qu'il faut

appliquer pour que les objets soient triés. Il ne reste plus qu'à les copier dans une nouvelle liste en respectant cette permutation :

$$new\_objects[i] \leftarrow old\_objects[\sigma(i)] \tag{II.2}$$

La fonction de nettoyage fait partie de l'interface de la structure de données afin de contourner l'interface d'ajout / suppression des objets. Cela permet un accès direct aux données pour plus de performances, mais nécessite de maintenir à jour la topologie. Après leur déplacement, la connectivité des objets n'est plus à jour. Ceux qui étaient connectés à un objet qui a été déplacé pointent maintenant vers un autre objet. Pour garder une trace du déplacement des objets, on maintient une table de correspondance qui donne la nouvelle position en fonction de l'ancienne. Il est ainsi possible de parcourir les objets et de mettre à jour leur connectivité.

Dans Optidis, ce nettoyage est effectué à chaque pas de temps. Il serait possible de l'effectuer moins souvent en testant par exemple le taux d'éléments invalidés, mais nous verrons en section II.4.2.3 qu'il peut être fait au moment de la migration avec un surcoût faible.

II.3.2.3 Allocation mémoire

Le caractère dynamique du réseau de dislocations conduit à ajouter et supprimer fréquemment des objets des conteneurs. Lorsque l'espace alloué n'est plus suffisant pour ajouter un élément à la fin du tableau, il faut réallouer un espace mémoire et copier les données existantes. Plusieurs solutions existent pour diminuer l'impact de ce problème.

Le coût d'une seule insertion peut être linéaire en nombre d'objets présents dans le conteneur si une copie des données dans un espace plus grand est nécessaire. Dans la librairie standard C++, `std::vector::push_back()` qui insère un objet à la fin du tableau possède une complexité constante amortie [66]. Sa complexité est constante en moyenne lorsque plusieurs appels sont effectués. Une manière d'obtenir cette complexité est d'utiliser une suite géométrique pour les tailles des tableaux, par exemple en utilisant un espace mémoire deux fois plus grand que le précédent à chaque réallocation. L'utilisation de `std::vector` dans Optidis garantit donc que la réallocation ne nuit pas aux performances.

Si l'on connait la taille maximale des données, il est possible d'allouer un espace mémoire suffisamment grand pour éviter de nombreuses réallocations lors de l'insertion d'un nombre important d'objets. Dans Optidis, j'utilise `std::vector::reserve()` qui réserve un espace mémoire d'une taille donnée lorsque le nombre d'objets varie beaucoup, comme lors de l'initialisation.

Il est aussi important de faire remarquer que par défaut `std::vector` ne fait qu'augmenter sa capacité sans jamais la réduire. Cela ne semble pas poser de problème dans le cas des dislocations qui ont plutôt tendance à se multiplier qu'à s'annihiler. Il est toujours possible de résoudre les cas pathologiques en utilisant la méthode `std::vector::shrink_to_fit()`. Elle n'est pour l'instant pas utilisée dans Optidis, mais pourrait être utilisée dans le nettoyage des données (section II.3.2).

II.4 Adaptation aux données distribuées

De nouvelles problématiques apparaissent lorsque l'on introduit du parallélisme en mémoire distribuée (MPI). Il faut discerner les données locales au processus et les données distantes (sur un autre processus) et adapter les algorithmes à cette contrainte. La cohérence des données sur les différents processus doit être maintenue.

II.4.1 Interface distribuée

L'interface mise en place a été pensée pour répondre à ces problématiques en proposant des méthodes d'accès aux éléments locaux, mais aussi aux éléments distants dans le cadre des parcours. Elle propose aussi un moyen de modifier la topologie de manière cohérente sur tous les processus. Son implémentation a été optimisée pour répondre aux problèmes d'équilibrage de charge et d'accès aux données distantes.

On distingue deux types d'opérations, les *opérations locales* peuvent être effectuées indépendamment par chaque processus, alors que les *opérations collectives* (en référence aux *communications collectives* de la bibliothèque MPI) doivent être exécutées simultanément sur tous les processus qui participent à la simulation.

Parmi les opérations locales, on compte par exemple l'opérateur `[]` ou les méthodes `begin()` et `end()` qui permettent d'accéder aux données locales dans le sens décrit en section II.4.2.1. On remarque que ces méthodes utilisent des indices d'objets locaux (c.f. annexe A.1.1) pour repérer les nœuds et les segments.

Les opérations qui accèdent à des données distantes comme `insert()` et `erase()`, `get()` et `set()` ou encore les fonctions de parcours, sont collectives. Ces opérations utilisent des indices globaux (c.f. annexe A.1.1) pour accéder à objets qui peuvent donc être locaux ou distants.

La fonction `migrate()` est une opération seulement nécessaire en mémoire distribuée qui vise à redistribuer les objets sur les processus selon la décomposition de domaine. Comme les objets sont mobiles, ils peuvent changer de domaine et doivent être communiqués au processus associé via MPI pour être relocalisés.

II.4.2 Stockage des données distribuées

Les données sont distribuées sur les différents processus. L'implémentation interne de la structure de données permet de traiter les problématiques de distribution et d'équilibrage de charge.

II.4.2.1 Décomposition de domaine

Dans Optidis, les données sont distribuées selon une décomposition de domaine suivant la Z-curve pour une compatibilité avec ScalFMM [40, 4]. L'espace est découpé en boites selon une grille uniforme qui sont distribuées sur les différents processus MPI. Les boites sont ordonnées selon leur indice de Morton, et des intervalles d'indices de Morton sont associés aux différents processus MPI, comme l'illustre la figure II.9.

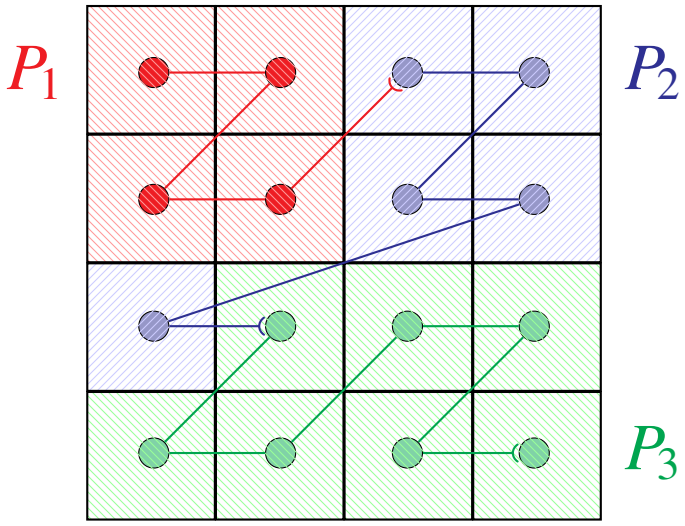


FIGURE II.9 – Décomposition de domaine par intervalles de Morton.

Pour distribuer les objets sur les processus MPI, on utilise les règles d'Optidis [40] :

— L'indice de Morton associé à un nœud $\mathscr{M}(n)$ est l'indice de Morton de la boite où il se trouve

— L'indice de Morton associé à un segment $\mathscr{M}(s)$ est l'indice de Morton de la boite où se trouve son *centre*

— Chaque processus P_i se voit attribué un intervalle I_{P_i} d'indices de Morton. Ces intervalles forment une partition du domaine

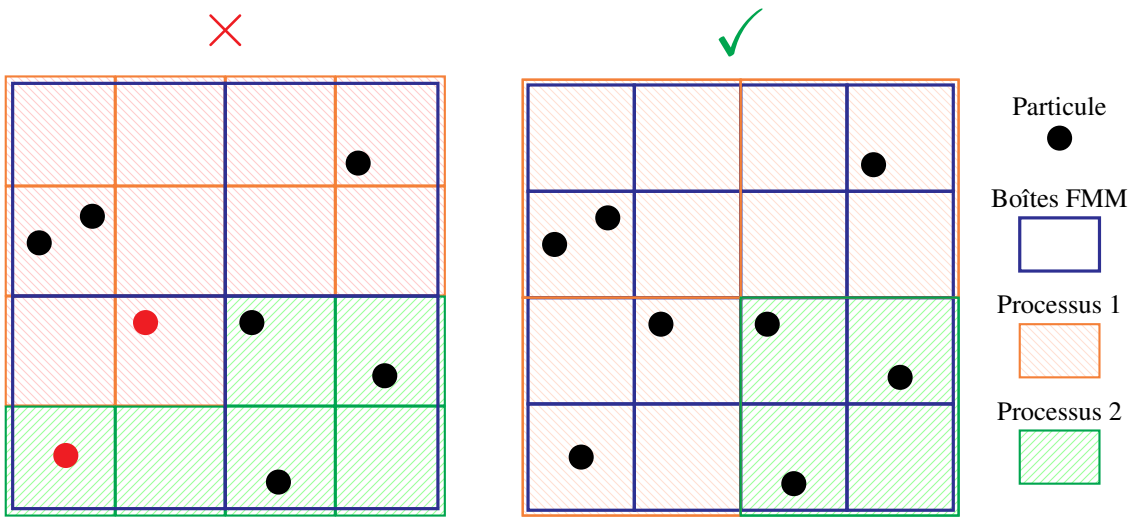
— Chaque objet o (segment ou nœud) est associé au processus P_i unique dont l'intervalle contient son indice de Morton :

$$o \in P_i \Leftrightarrow \mathscr{M}(o) \in I_{P_i} \tag{II.3}$$

La charge de calcul de chaque processus peut être équilibrée en ajustant les intervalles I_{P_i} de manière à homogénéiser la distribution des objets.

La grille utilisée pour la décomposition de domaine n'est pas forcément identique à la grille FMM : les hauteurs d'arbres utilisées peuvent être différentes. En effet, une grille trop fine pourrait augmenter la complexité géométrique des domaines et par conséquent le volume de communications MPI, comme par exemple dans la figure II.9 où le domaine bleu possède une cellule orpheline dans la partie gauche du schéma. Pour que la décomposition de domaine soit compatible avec ScalFMM, il faut s'assurer que toutes les particules d'une feuille de l'octree FMM se trouvent sur le même processus MPI. La décomposition de domaine doit se faire selon les boites d'un octree d'une hauteur H_{DD} inférieure à l'arbre utilisé pour le calcul FMM (eq II.4), comme l'illustre la figure II.10b, sinon certaines boites FMM seraient éclatées sur plusieurs processus (ce que ScalFMM ne permet pas), comme dans la figure II.10a.

$$H_{DD} < H_{FMM} \tag{II.4}$$



(a) Les particules en rouge sont dans la même boite FMM mais pas sur le même processus.

(b) Le découpage FMM est plus fin que la décomposition de domaine.

FIGURE II.10 – Compatibilité de la décomposition de domaine avec ScalFMM.

II.4.2.2 Équilibrage de charge

L'équilibrage de charge effectué dans Optidis utilise initialement une approche dynamique basée sur le transfert de morceaux d'intervalles avec les processus voisins [40]. Dans une démarche de simplification du code, j'ai mis en place une nouvelle méthode pour équilibrer le nombre de nœuds locaux à chaque processus. La charge est toujours équilibrée en ajustant les intervalles de distribution des données, mais la différence réside dans le choix des nouveaux intervalles qui utilise maintenant un algorithme inspiré d'un tri distribué [26].

Une première étape détermine les intervalles optimaux. Le k_i^{eme} plus petit indice de Morton de tous les objets de la simulation marquera le début de l'intervalle I_{P_i} avec :

$$k_i = \frac{i \cdot size}{nb_{procs}} \tag{II.5}$$

Cela assure que chacun des processus possède le même nombre de nœuds locaux $N = \frac{size}{nb_{procs}}$. La seconde étape est le déplacement des objets sur les bons processus, détaillé en section II.4.2.3.

L'algorithme qui sélectionne le k-ieme plus petit élément d'une liste est appelé algorithme de sélection. J'utilise l'algorithme de sélection distribué décrit par Cheng *et. al.* [26]. Dans cet algorithme chaque processus trie sa liste d'indice de Morton localement, puis les processus communiquent pour déterminer tous les k_i en une unique exécution de l'algorithme.

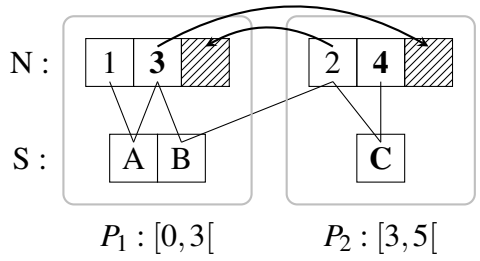
Cet algorithme permet de calculer de manière exacte la répartition optimale des intervalles de Morton : chaque processus possède $N = \frac{size}{nb_{procs}}$ éléments.

II.4.2.3 Migration

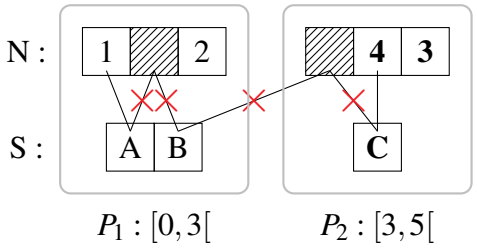
Après avoir déplacé les objets dans la phase d'intégration ou après avoir modifié les intervalles de décomposition de domaine dans la phase d'équilibrage de charge (II.4.2.2), certains objets peuvent se trouver dans des boites qui appartiennent à un autre processus MPI. La phase de migration déplace les objets vers les bons processus MPI.

Tout comme l'algorithme de nettoyage, la migration contourne l'interface d'insertion et de suppression évoquée en section II.3.1 pour des raisons de performances. L'implémentation de cet algorithme doit donc mettre à jour les informations internes de la structure de données qui sont normalement maintenues en cohérence par les opérations collectives d'ajout/suppression.

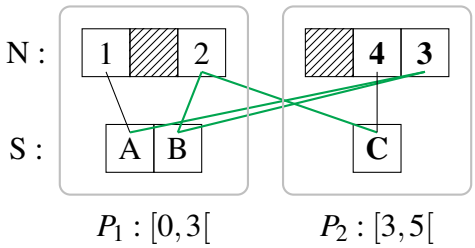
La migration se déroule en plusieurs étapes illustrées en figure II.11. Les objets qui ne sont pas locaux sont déplacés sur le bon processus, puis la topologie est mise à jour pour reconnecter les objets déplacés.



(a) Les objets qui ne sont pas locaux sont déplacés sur le bon processus.



(b) Les connexions aux objets déplacés sont supprimées.



(c) Les objets déplacés sont reconnectés.

FIGURE II.11 – Étapes de la migration.

L'étape de migration réorganise les données dans les conteneurs et invalide donc les indices (c.f. II.3.2). De plus, on remarque en figure II.11c qu'il peut y avoir une dégradation de la localité spatiale : les connexions avec des objets distants sont plus fréquentes. Dans Optidis, la migration est effectuée conjointement avec le nettoyage des données de manière à ne mettre qu'une seule fois à jour la topologie.

II.4.3 Opérations topologiques

Les fonctions `insert()` et `erase()` sont des *opérations collectives* (section II.4.1). Pour insérer ou supprimer un nœud, il faut appeler ces méthodes sur tous les processus à la fois. J'ai fait ce choix pour que l'interface garantisse la cohérence des données topologiques entre tous les processus. Même si l'ajout et la suppression des objets sont des opérations collectives, l'implémentation ne nécessite pas forcément l'échange de messages MPI.

À l'insertion, tous les processus doivent connaître les données portées par l'objet à insérer. Ce dernier est ajouté au conteneur local d'un des processus, choisi selon la décomposition de domaine. Pour les segments, les informations topologiques portées par les nœuds doivent être

mises à jour, même lorsqu'ils sont situés sur un processus distant. `insert()` retourne sur tous les processus l'indice global de l'objet.

Une implémentation naïve de `insert()` consiste à diffuser l'indice depuis le processus qui possède l'objet en utilisant `MPI_Bcast()`. Dans Optidis, l'indice est déterminé par chaque processus sans utiliser de communications MPI. Chacun possède une représentation de l'état de tous les autres processus afin de déterminer localement l'indice d'insertion. L'opération reste tout de même collective, car les informations topologiques ainsi que de l'état des autres processus doivent être mis à jour.

Par exemple, lors de la génération initiale du réseau de dislocations tous les processus lisent les données des nœuds à insérer depuis un fichier. Tous les processus connaissent la position du nœud et appellent la méthode `node.insert()`, mais un processus seulement insère réellement le nœud dans son conteneur local. Chaque processus connait l'état interne de tous les autres, il est donc possible de savoir exactement où a été inséré le nœud : sur quel processus, mais aussi à quelle position dans liste locale du processus distant. On construit alors un indice global `NodeIndex_global` qui pourra être utilisé, par exemple, pour connecter un segment à ce nœud.

Pour supprimer un objet, tous les processus doivent connaître l'indice global de l'objet à supprimer. L'objet est supprimé localement du processus qui le possède. Dans le cas du segment, les informations topologiques des nœuds à son extrémité doivent être mises à jour, même si le nœud est distant.

Il est possible d'éviter totalement les communications MPI lors de la suppression, mais l'implémentation de `segments.erase()` dans Optidis utilise des communications pour déterminer si le segment est connecté à un nœud local. L'optimisation n'a pas été effectuée car la suppression n'a pas été identifiée comme un frein à la performance [62].

II.4.4 Parcours

Il est possible de parcourir les données via deux interfaces différentes. L'utilisation d'itérateurs permet de parcourir les données locales, et des fonctions externes présentées en figure A.4 de l'annexe A.1 permettent d'effectuer les parcours décrits en section II.1.

L'interface du réseau de dislocations permet d'accéder à des itérateurs sur les données locales avec les méthodes `begin()` et `end()`, qui permettent l'utilisation du *Range-based for* du C++11 [66]. Les itérateurs supportent OpenMP et peuvent être utilisés dans une région parallèle, comme l'illustre la figure II.12. `begin()` et `end()` retournent des intervalles différents pour chaque thread qui forment une partition des objets locaux.

```
1  #pragma omp parallel
2  for(Mesh::NodeInfo_ref n : mesh.nodes)
3  {
4      f( n );
5  }
```

FIGURE II.12 – Utilisation des itérateurs pour parcourir les données locales.

Les parcours décrits en section II.1 accèdent aux données locales, mais peuvent avoir besoin de lire des données distantes. Des fonctions indépendantes de l'interface de la structure de données permettent d'abstraire ces parcours et s'occupent des communications MPI de manière transparente pour l'utilisateur. L'algorithme 1 détaille l'implémentation du parcours des segments avec nœuds connectés. Les communications MPI se déroulent dans une première phase afin de récupérer les informations sur les nœuds distants connectés aux segments locaux. Les segments locaux sont ensuite parcourus, les nœuds connectés sont récupérés depuis les données locales ou depuis le cache de nœuds distants. Ces fonctions permettent l'utilisation d'OpenMP avec un parallélisme de boucles.

Algorithme 1 : Implémentation du parcours des segments avec nœud connectés

```
   Données : Réseau de dislocations
   Données : f : Fonction à exécuter
1  cache ← echange_noeuds (m)          // Echange MPI des nœuds distants
2  #pragma omp for                     // Multithreading avec OpenMP
3  pour S parmi les segments locaux faire
4  |  n1 ← cache.get_noeud(S.n1)       // Les informations des nœuds sont
5  |  n2 ← cache.get_noeud(S.n2)       // récupérées depuis les données
   |                                     locales ou le cache
6  |  f (S,n1,n2)                      // La fonction est appelée pour
   |                                     chaque segment local
```

TABLE II.1

Pour qu'il soit possible d'effectuer la communication MPI avant le parcours des données, la liste des nœuds connectés distants - aussi appelés *nœuds fantômes* - doit être connue. Dans Optidis, une liste des indices des nœuds fantômes est maintenue sur chaque processus tout au long de la simulation lors de l'ajout ou la suppression d'objets. L'accès à ces données stockées dans la structure n'est pas possible via l'interface décrite en section II.1. Les fonctions de parcours contournent donc l'encapsulation de la structure de données.

La mise en place de fonctions pour parcourir les objets en plus des itérateurs se justifie car elles offrent de meilleures performances. Il est possible d'utiliser les itérateurs pour écrire les fonctions de parcours de manière générique, mais les contourner permet un accès direct et plus rapide aux données de la structure. La performance a été choisie ici au détriment de la généricité. Une version générique utilisant les itérateurs a tout de même été implémentée pour faciliter la mise en place d'une nouvelle implémentation de la structure mais n'est pas utilisée dans Optidis.

II.5 Validation et performances

La structure de données permet d'effectuer toutes les opérations nécessaires à la simulation au travers d'une interface abstraite. On vérifie dans ce qui suit la performance de l'implémentation à base de tableaux. Les études de performances menées ici peuvent aussi s'appliquer à toute implémentation ultérieure.

II.5.1 Conditions de test des parcours

La performance des parcours a été mesurée dans un programme de test hors d'Optidis. Les temps d'exécution des trois parcours décrits en section II.1 ont été mesurés dans différentes situations, et comparés au parcours témoin d'un tableau statique simple.

II.5.1.1 Paramètres

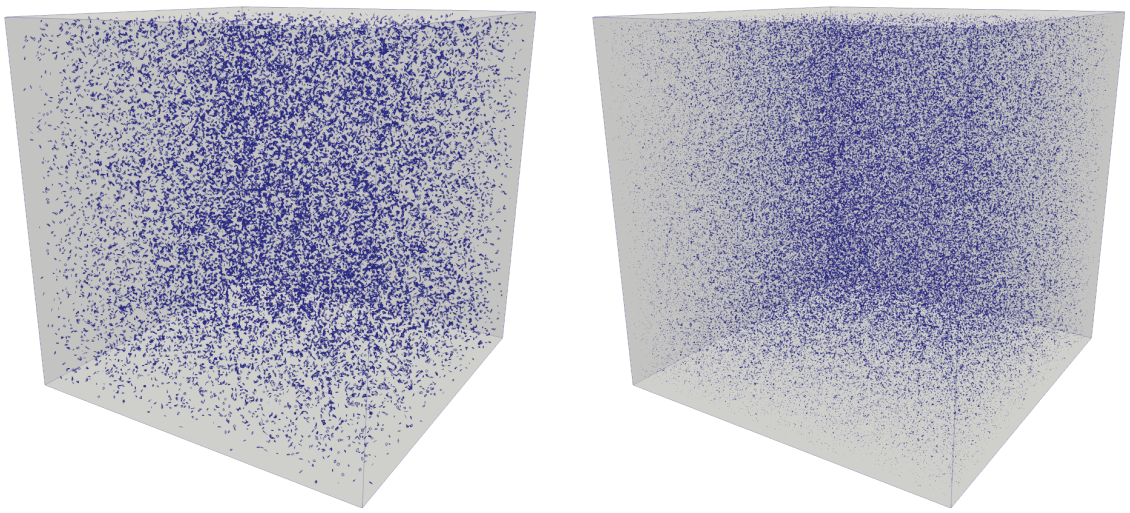
Les mesures ont été effectuées en faisant varier plusieurs paramètres dont les notations sont détaillées dans le tableau II.2. La taille des données, c'est-à-dire le nombre de segments, prend deux valeurs afin de tester une situation où le volume de données tient en cache L3 (10^5 éléments), et une situation qui dépasse le cache (10^6 éléments). Le nombre de threads OpenMP varie pour couvrir trois situations : 1 seul cœur de calcul pour obtenir la performance séquentielle, 1 nœud NUMA complet (8 threads) pour obtenir la performance multithreadée, et un nœud de calcul complet (2 NUMA, 16 threads) pour observer d'éventuels effets NUMA [17].

S	Nombre de segments
I	Intensité arithmétique
N_{mpi}	nombre de processus MPI
$N_{threads}$	Le nombre de threads OpenMP

TABLE II.2 – Paramètres du benchmark de la structure de données.

II.5.1.2 Données

Les données à parcourir sont générées en fonction de la taille demandée. Le réseau de dislocations généré se compose de $\frac{S}{4}$ boucles carrées uniformément réparties et composées de 4 segments chacune, comme illustré dans la figure II.13. La taille des boucles est modifiée en fonction du nombre de segments pour maintenir une densité de dislocations constante. Maintenir une densité constante est important pour que les parcours génèrent un volume de communication constant : plus le réseau est dense, plus les segments ont de chance d'être à cheval sur deux domaines.



(a) 10^5 Segments de taille 50 Å. (b) 10^6 Segments de taille 5 Å.

FIGURE II.13 – Données générées pour tester la performance des parcours de la structure de données. Taille de domaine :1 µm ; densité : $5 \cdot 10^{14}$ m/m^3.

La taille des segments est adaptée de manière à ce que la densité de dislocations ρ soit constante. La densité choisie est de $5 \cdot 10^{14}$ m/m^3. Tous les segments ont la même longueur L qui peut être déterminée comme suit selon la densité et le nombre de segments S souhaité :

$$\rho = \frac{\sum_s L_s}{\mathrm{V}} = Cst \tag{II.6}$$

$$L = \frac{\rho \times V}{N} \tag{II.7}$$

Pour le parcours témoin, des tableaux de flottants double précision de taille $3 \times S$ sont alloués et initialisés avec des valeurs quelconques différentes les unes des autres. Chaque thread alloue le tableau qu'il parcourt. Les éléments du tableau sont regroupés par trois flottants double précision (*AoS*), comme les tableaux internes à la structure de données (sec. II.2.4). Une variante avec trois tableaux de taille S (*SoA*) a aussi été essayée pour laquelle les résultats ne sont pas exposés ici car la performance est très légèrement inférieure.

II.5.1.3 Opérations effectuées pour chaque objet

Pour chaque élément contenu dans la structure de données, on effectue un certain nombre d'opérations en fonction de l'intensité arithmétique I demandée. La fonction exécutée pour chaque objet est décrite dans l'algorithme 2.

Parcours	v_1	v_2	v_3
Référence	Tableau 1	Tableau 2	Tableau 3
Nœuds seuls	Nouvelle position	Ancienne position	Vitesse
Segments avec Nœuds	Force sur le segment	Force nœud 1	Force nœud 2
Nœuds avec Segments	Force nodale	Force sur le 1^{er} segment connecté	Force sur le 2^{eme} segment connecté

TABLE II.4 – Champs choisis pour le benchmark des parcours.

Algorithme 2 : Fonction exécutée pour chaque objet

Paramètres : I : L'intensité arithmétique
Données : v_1, v_2 et v_3 : des champs vectoriels de la structure de données.

```
1 v1 ← 0
2 pour i dans [1,I] faire
3 |   v1 ← v1 + (0.1 * i) * v2 + (0.15 * i) * v3
```

TABLE II.3

La boucle `for` est en réalité implémentée en utilisant les *templates* afin que le compilateur puisse appliquer toutes les optimisations possibles. Les constantes flottantes $(0.1 \cdot i)$ et $(0.15 \cdot i)$ sont connues à la compilation et différentes pour chaque multiplication. Cela permet de s'assurer que le compilateur ne factorise pas les multiplications, et effectue bien le nombre d'opérations flottantes attendu. L'option de compilation `ffast-math` est désactivée, et le nombre d'opérations dans le code assembleur généré a été vérifié.

v_1, v_2 et v_3 sont des vecteurs en 3 dimensions en double précision. Pour chaque objet et pour chaque tour de boucle, 9 flottants doubles précisions (8 octets) sont chargés et 3 sont écrits. Le nombre d'octets écrits et chargés au cours du parcours est :

$$M(I,S) = 8 \cdot (9+3) \cdot S = 96 \cdot \text{S Octets} \tag{II.8}$$

Pour chaque objet, et pour chaque tour de boucle, on effectue 2 additions et 2 multiplications vectorielles. Le nombre d'opérations flottantes au cours d'un parcours est le suivant :

$$Flops(I,S) = 12 \cdot \text{I} \cdot \text{S Flops} \tag{II.9}$$

L'intensité en Flops/Octet en fonction de I est :

$$Intensite(I) = \frac{\text{I}}{8} \text{ Flops/Octet} \tag{II.10}$$

Selon les parcours considérés, les champs choisis pour v_1, v_2 et v_3 varient.

II.5.1.4 Méthode de mesure du temps d'exécution

La mesure du temps d'exécution des parcours de la structure de données se fait en utilisant la fonction de la librairie standard `std::chrono::steady_clock::now()`. La mesure retenue est le meilleur temps d'exécution de la fonction de parcours sur 100 appels successifs en excluant la première mesure [82].

II.5.2 Performance des parcours en mémoire partagée

Cette section s'intéresse à la performance de la structure de données en mémoire partagée avec OpenMP. Nous proposons une méthode pour vérifier que la structure de données utilise correctement le matériel dans un contexte multithreadé sur un seul nœud de calcul.

II.5.2.1 Roofline model et machine de test

Le *roofline model* est une représentation graphique de la performance d'une application. Il permet de comparer la puissance théorique du matériel à la vitesse d'exécution de l'application [120]. Il s'agit d'un graphe dont l'axe des abscisses représente l'*intensité arithmétique* (aussi appelée *Intensité opérationnelle*), et l'axe des ordonnées la performance de l'application. L'intensité arithmétique est le nombre d'opérations arithmétiques effectuées par le processeur pour chaque Octet chargé depuis la mémoire et s'exprime en Flops/Octet. La performance est la vitesse d'exécution des opérations arithmétiques et se mesure en Flops/s. Sur ce graphique, on superpose la performance maximale de la machine avec la performance de l'application à étudier. La figure II.14 présente un exemple de roofline.

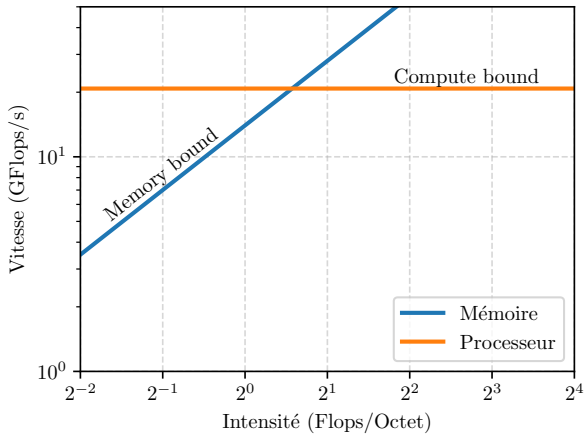


FIGURE II.14 – La performance maximale théorique sur un roofline.

La performance de la machine peut être limitée par la vitesse de la mémoire, on est alors dans la partie *Memory bound* du roofline. Elle peut aussi être limitée par la vitesse du processeur, c'est la partie dite *Compute bound*. La performance maximale P_{max} selon l'intensité I se calcule en utilisant la vitesse du processeur P_{CPU} et la bande passante mémoire B_{RAM} [120] :

$$P_{max} = min(P_{CPU}, I \times B_{RAM}) \tag{II.11}$$

La machine utilisée pour effectuer les mesures est le cluster Poincaré [2] de la Maison de la Simulation. La topologie d'un nœud de la machine est donnée par la figure II.15. Chaque nœud de calcul est composé de deux processeurs Sandy Bridge Intel E5-2670 à 2.6Ghz.

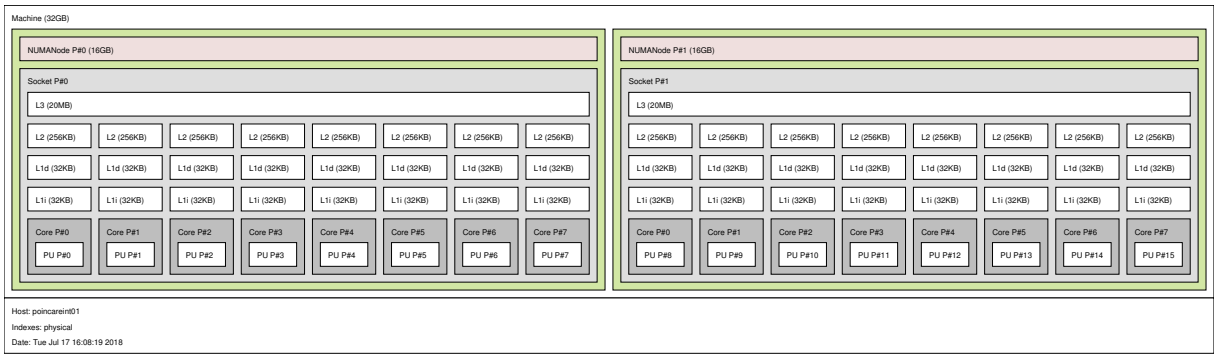


FIGURE II.15 – Topologie d'un nœud de la machine Poincaré.

La performance maximale d'un nœud de la machine Poincaré est résumée dans les roofline models de la figure II.16. Les trois rooflines représentent les performances maximales de la machine avec différents nombres de threads.

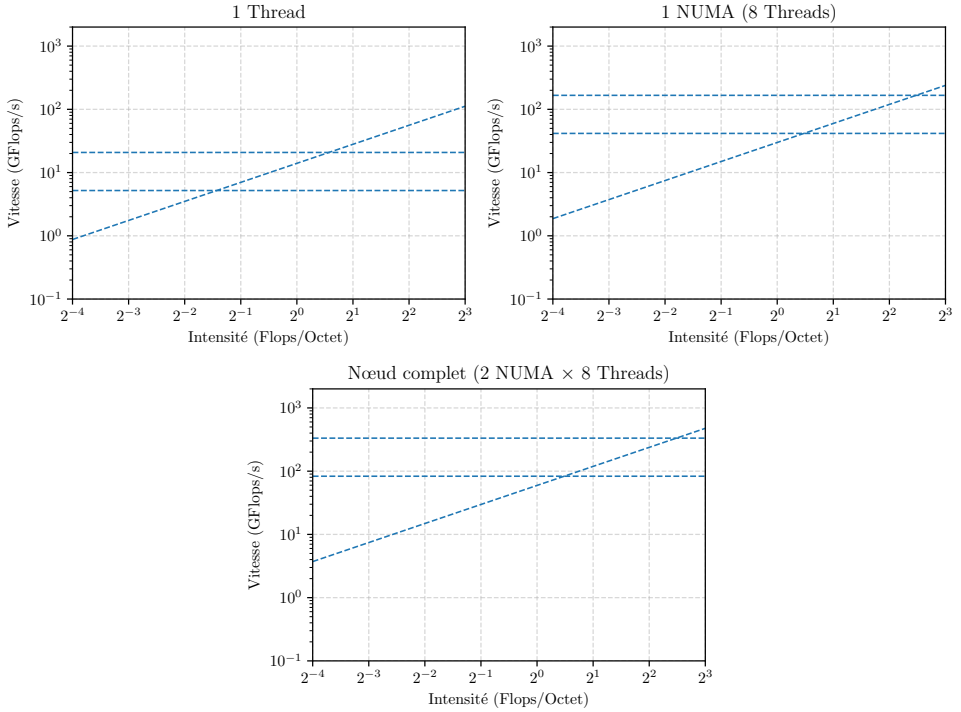


FIGURE II.16 – Performances de référence de la machine Poincaré (Roofline).
- La ligne oblique représente la performance maximale liée à la bande-passante mémoire alors que les lignes horizontales représentent la vitesse maximale du processeur lorsque la vectorisation est activée (haut), et lorsqu'elle est déactivée (bas).

La bande passante maximale (partie linéaire) a été mesurée via le benchmark STREAM [82] et la performance crête (plateau) a été déduite des spécifications constructeur du processeur [67]. On distingue la puissance crête avec vectorisation P_{AVX2} de celle sans vectorisation $P_{scalaire}$:

$$P_{AVX2} = 2.6\,\text{GHz} \times 2\,\text{ALUs} \times 4\,\text{double/inst} = 20.8\,\text{GFlops/s/coeur} \quad \text{(II.12)}$$

$$P_{scalaire} = 2.6\,\text{GHz} \times 2\,\text{ALUs} \times 1\,\text{double/inst} = 5.2\,\text{GFlops/s/coeur} \quad \text{(II.13)}$$

Le tableau II.5 résume les performances maximales de la machine.

	1 coeur	1 NUMA	Noeud complet
Bande Passante (GO/s)	14.0	29.9	59.5
Performance crête scalaire (GFlops/s)	5.2	41.6	83.2
Performance crête AVX2 (GFlops/s)	20.8	166.4	332.8

TABLE II.5 – Performances maximales de la machine.

Une autre manière de présenter ces résultats pour les faibles intensités arithmétiques est de tracer la courbe de la bande passante en fonction de l'intensité arithmétique. La figure II.17 affiche par exemple bande passante maximale du matériel sur un nœud NUMA en fonction du nombre de threads utilisés.

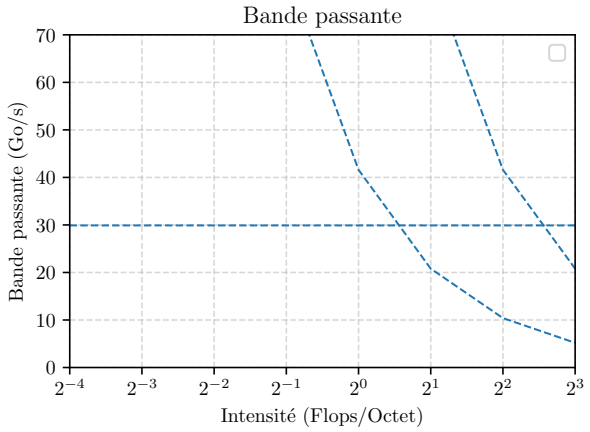


FIGURE II.17 – Performances de référence de la machine Poincaré sur un nœud NUMA (Bande passante).

Placement des algorithmes sur le roofline

Il est possible de calculer l'intensité arithmétique pour les algorithmes implémentés au sein d'Optidis afin de les placer sur le Roofline model. J'ai choisi d'étudier plusieurs algorithmes qui ont des intensités arithmétiques différentes.

Le premier algorithme est l'algorithme de déplacement des nœuds du réseau de dislocations. Il fait partie de l'étape d'intégration décrite en section I.3.4, et utilise le parcours simple des nœuds décrit en section II.1. Il possède une faible intensité arithmétique et est consultable en figure II.18.

Le second algorithme est celui chargé de calculer l'incrément de déformation. Il n'a pas encore été évoqué, mais il permet de déterminer la déformation plastique en calculant l'aire balayée par les dislocations. Il est consultable en figure A.6(b), en annexe A.1 et a été choisi comme exemple pour sa concision.

La figure II.18 est un extrait de code de la fonction chargée de déplacer les nœuds à leur nouvelle position en fonction de la vitesse calculée au préalable. En plus de calculer la nouvelle position, la fonction enregistre la position précédente. Cette fonction est prise comme exemple afin d'expliquer comment est calculée l'intensité arithmétique théorique d'une boucle.

```
double dt;
[...]
MeshHelper::fastiterate_nodes( mesh,
[dt](Mesh::NodeInfo_ref& n)
{
    n.x_old() = n.x();
    n.y_old() = n.y();
    n.z_old() = n.z();
    n.x() += dt*n.vx();
    n.y() += dt*n.vy();
    n.z() += dt*n.vz();
});
[...]
```

FIGURE II.18 – Déplacement des nœuds.

A chaque itération, des champs de la structure sont chargés ou écrits en mémoire :

— Position `{n.x(),n.y(),n.z()}` : lecture + écriture

— Vitesse `{n.vx(),n.vy(),n.vz()}` : lecture seule

— Ancienne vitesse `{n.x_old(),n.y_old(),n.z_old()}` : écriture seule

Pour un total de 12 flottants double précision lus ou écrits. Le volume d'opérations mémoire par itération est donc :

$$M = 12 \cdot \text{sizeof}(\texttt{double}) = 96 \text{ Octets} \tag{II.14}$$

On compte 3 additions et 3 multiplications :

$$F = F_{add} + F_{mul} = 6 \text{ Flops} \tag{II.15}$$

On en déduit l'intensité arithmétique : On compte 3 additions et 3 multiplications :

$$I = F/M = \frac{1}{16} \text{ Flops/Octet} = 0.0625 \text{ Flops/Octet} \tag{II.16}$$

De la même manière, on calcule l'intensité arithmétique pour l'incrément de déformation, et on obtient le tableau II.6.

Algorithme	F (Flops)	M (Octets)	I (Flops/Octet)
Déplacement des nœuds	6	96	0.0625
Incrément de déformation	72	96	0.75

TABLE II.6 – Calcul des intensités arithmétiques de différents algorithmes au sein d'Optidis.

Il est alors possible de les placer sur un Roofline, comme le montre la figure II.19.

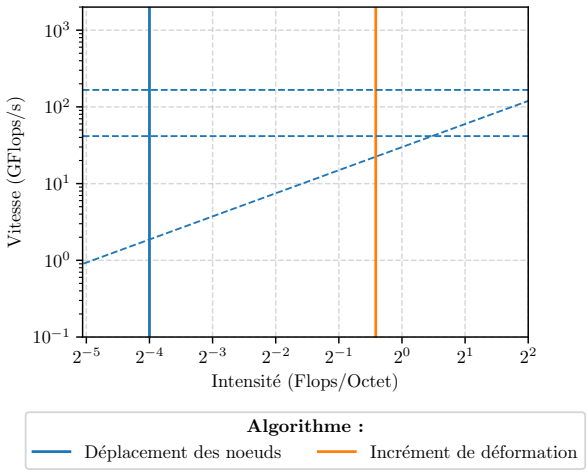


FIGURE II.19 – Placement des intensités arithmétiques des algorithmes d'Optidis sur un roofline model (1 Nœud NUMA).

Pour placer la vitesse d'exécution d'un algorithme sur la courbe il faut mesurer son temps d'exécution. La vitesse V peut être calculé à partir du décompte des opérations à chaque tour de boucle F_{Iter} (ex. tableau II.6), du nombre d'itérations N_{Iter} et du temps d'exécution total de la boucle T_{Tot} (eq. II.17). C'est cette vitesse d'exécution qui est analysée tout au long des sections suivantes.

$$V = \frac{F_{Iter} \times N_{Iter}}{T_{Tot}} \tag{II.17}$$

II.5.2.2 Comportement général

Les sections suivantes mettent en avant certains phénomènes que l'on peut observer dans les mesures effectuées. Toutes les mesures sont données dans l'annexe A.2.

Les graphiques de la figure II.20 donnent un aperçu de la performance des parcours de la structure de données dans le cas qui représente le mieux son utilisation dans Optidis, à savoir avec des données qui ne tiennent pas en cache et sur un nœud NUMA.

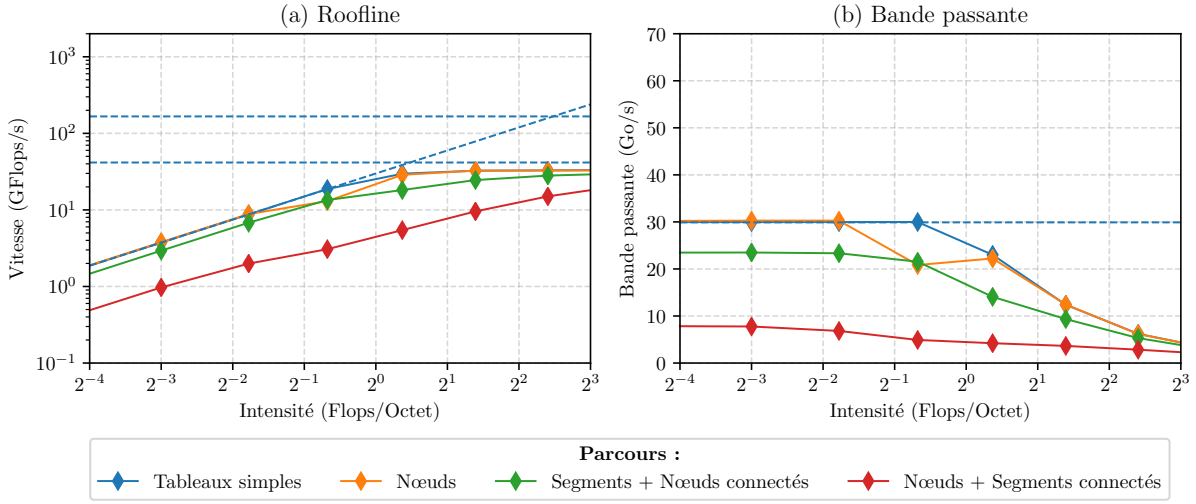


FIGURE II.20 – Performance des parcours de la structure de données pour un volume de données dépassant la capacité du cache L3 (10^6 éléments) sur un nœud NUMA (8 threads).

On remarque que dans ces conditions les parcours de la structure de données utilisent efficacement les capacités de la machine. Les différents parcours ont des performances variables en accord avec la complexité des schémas d'accès aux données. Les performances du parcours de référence et du parcours simple des nœuds sont comparables et proches de la performance maximale de la machine sans vectorisation ni effet de cache. On déduit que la couche d'abstraction que constitue l'interface d'accès aux données n'affecte pas significativement la performance du parcours simple des nœuds.

Les deux autres parcours sont moins performants à cause d'un accès aux données plus irrégulier. Le parcours des segments avec nœuds connectés génère une bande passante utile 20% inférieure à la bande passante maximale car l'accès aux nœuds nécessite une indirection supplémentaire. Il faut charger depuis la mémoire les indices des segments, ce qui consomme de la bande passante. Le parcours des nœuds avec segments connectés est encore plus irrégulier car le nombre de segments connectés à chaque nœud est inconnu à la compilation. Le compilateur ne peut pas effectuer autant d'optimisations, et il en résulte une performance moindre.

II.5.2.3 Effets de la taille des données

Le nombre d'objets contenus dans la structure de données joue un rôle important dans la performance des parcours, surtout pour les faibles intensités arithmétiques. La figure II.21 compare les performances des parcours sous forme de Roofline et de graphe de bande passante pour 10^5 et 10^6 éléments sur un nœud NUMA.

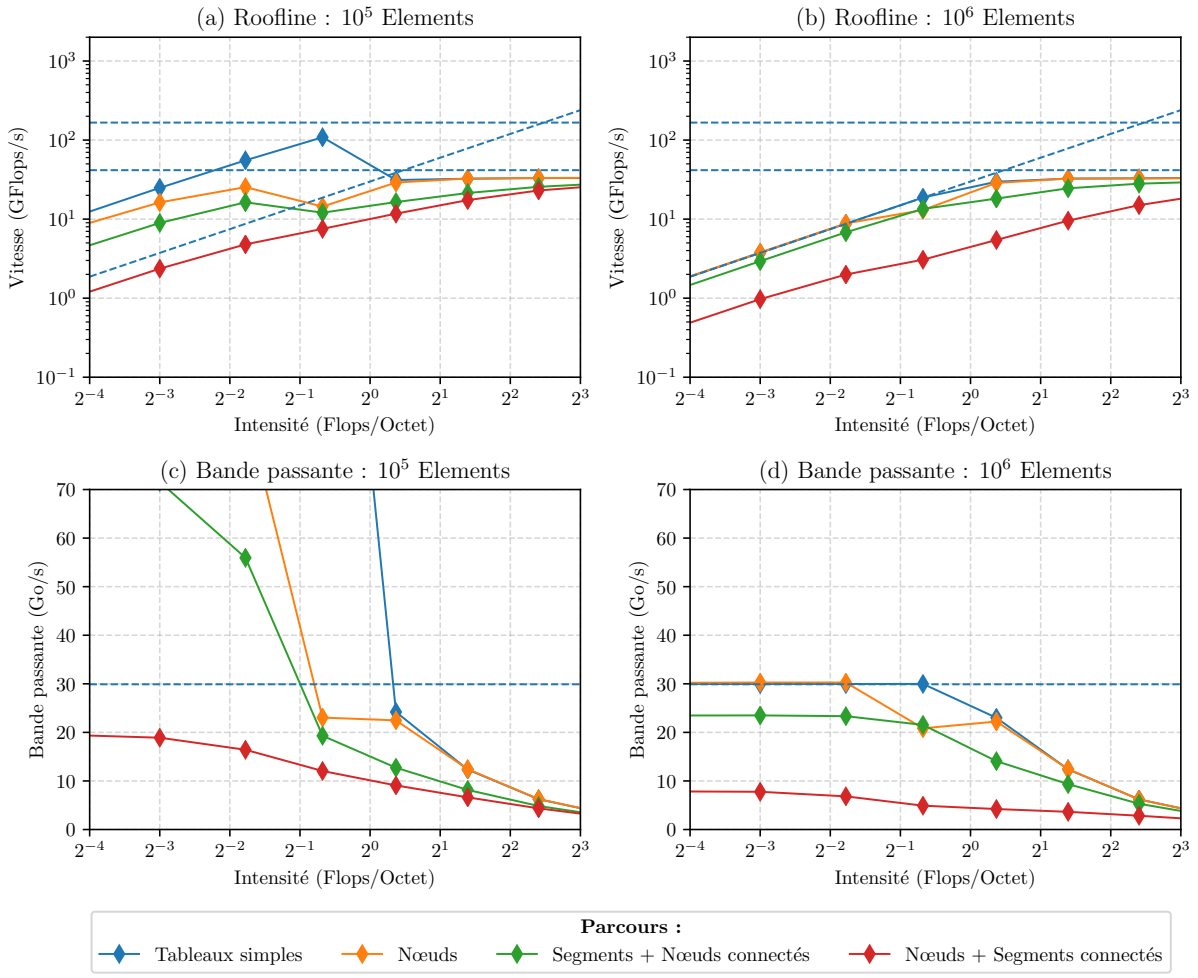


FIGURE II.21 – Impact du nombre d'objets contenus dans la structure sur un nœud NUMA (8 threads). À gauche (a, c) pour 10^5 éléments, et à droite (b, d) pour 10^6 éléments.

On observe des effets de cache lorsque la taille des données est suffisamment petite. La bande passante pour $S = 10^5$ (fig. II.21c) est supérieure à la bande passante maximale entre le processeur et la RAM pour presque tous les parcours, ce qui indique que les données sont stockées dans le cache qui possède une bande passante supérieure. Pour des données plus grandes, ces effets n'apparaissent pas, et la performance est limitée par la vitesse de la RAM.

Cependant, les algorithmes d'Optidis ne bénéficient pas de ces effets de cache. Dans Optidis, les différents algorithmes utilisent le plus souvent des champs différents de la structure, ce qui limite la réutilisation des données. De plus, les données du cache sont remplacées à chaque itération par le calcul FMM coûteux en mémoire. Le cas le plus représentatif de la performance dans Optidis est donc celui où les effets de cache n'apparaissent pas.

Bien que les algorithmes tels qu'ils sont mis en place ne bénéficient pas des effets de cache, les résultats nous montrent que la structure de données, elle, est capable d'en bénéficier. Une meilleure utilisation du cache pourrait améliorer la performance, par exemple en fusionnant des boucles pour augmenter l'intensité arithmétique [69], ou en mettant en place une forme quelconque de *Cache blocking* [117]. Ce type d'optimisation n'est pas utilisé dans Optidis car les modifications de la structure du programme seraient lourdes pour des gains hypothétiques

faibles.

II.5.2.4 Effets du nombre de threads

La figure II.22 compare les performances des parcours d'une structure de données contenant 10^6 objets en fonction du nombre de threads utilisés. Cette figure vise à montrer l'impact des effets NUMA [17] sur la structure de données. Je compare une situation multithreadée sur un unique nœud NUMA (fig. II.22a et c) à une situation où les threads sont distribués sur deux nœuds NUMA (fig. II.22b et d).

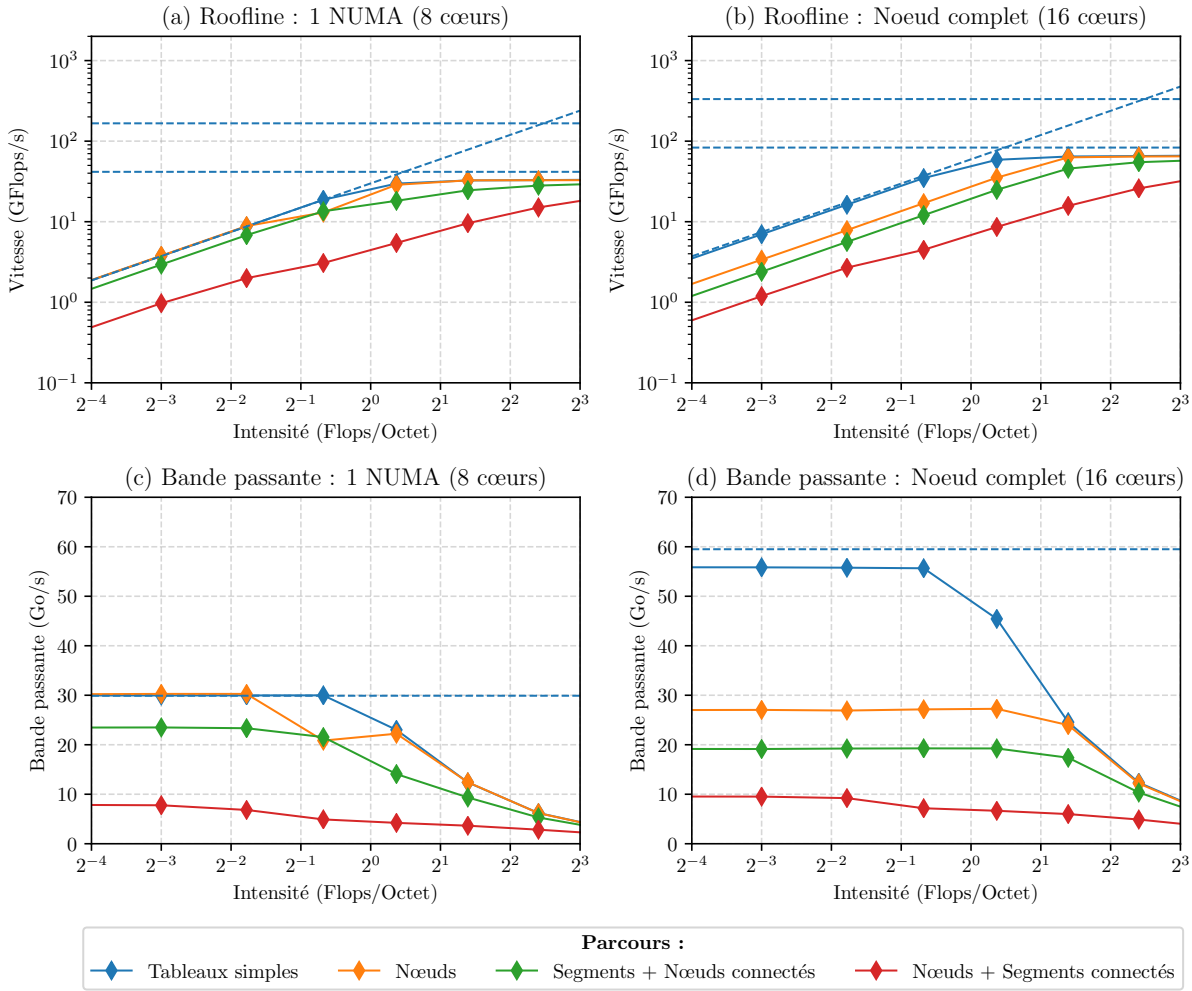


FIGURE II.22 – Impact du nombre de threads sur la performance de la structure de données pour 10^6 objets. À gauche (a, c) pour un nœud NUMA (8 threads), et à droite (b, d) pour un nœud de calcul complet (2 NUMA $\times$ 8 threads).

Comme l'indique la section II.5.2.2, la performance sur un unique nœud NUMA est proche de la performance maximale. Cependant, dans un contexte NUMA (fig. II.22b et d), on observe une différence notable. L'ajout d'un second nœud NUMA ne permet pas d'accélérer les parcours de la structure de données lorsque la performance est limitée par la mémoire. Le parcours de référence, quant à lui voit sa bande passante utile augmenter.

Ce résultat nous montre que la structure de données n'est pas adaptée à un environnement NUMA. Les performances observées sont symptomatiques d'une mauvaise distribution des

données sur les nœuds NUMA, et il faudrait mieux les distribuer pour bénéficier de la bande passante supplémentaire qu'offre l'architecture NUMA. Ce résultat était à prévoir au vu de l'initialisation des données dans la structure : tous les tableaux sont initialisés depuis le même thread. Il serait possible de résoudre ces problèmes en distribuant mieux les données sur les nœuds NUMA [24]. La solution la plus simple est tout de même de n'utiliser qu'un nœud NUMA par processus MPI. C'est ce qui est fait dans les mesures effectuées en mémoire partagée.

II.5.3 Performance des parcours en mémoire distribuée

Dans cette section, je mesure la performance de la structure de données lorsque les objets sont distribués sur plusieurs processus MPI. Les graphiques proposés représentent le temps ou l'efficacité en fonction du nombre de processus. Pour chacun des graphiques présentés, on utilise toujours un nœud NUMA complet, comme le conseille la section II.5.2.4. Le nombre d'éléments pourra être de 10^5 ou 10^6 par processus MPI et sera indiqué sur chaque graphe. L'intensité arithmétique est de 0.04 Flops/Octet. Elle a été choisie faible pour mieux observer le surcoût MPI.

La performance MPI des parcours de la structure est mesurée en scalabilité faible : la taille des données augmente avec le nombre de processus. Ce choix a été fait car la scalabilité forte nous oblige à utiliser des tailles soit trop petites pour mesurer correctement les temps avec beaucoup de processus, soit trop grandes pour être représentatives du nombre d'objets simulés. Afin que la proportion de segments fantômes ne varie pas, il est important de maintenir une densité linéique de dislocations constante comme décrit en section II.5.1.2. Toutes les mesures effectuées sont données en annexe A.2.

II.5.3.1 Comportement général

Les graphiques de la figure II.23 donnent un aperçu de la performance des parcours de la structure de données dans le cas qui représente le mieux son utilisation dans Optidis : l'intensité choisie de 0.04 Flops/Octet est représentative de l'algorithme de déplacement des nœuds par exemple, et le volume de données de 10^6 éléments par processus MPI est représentatif de la non-réutilisation des données dans les caches au sein d'Optidis (sec. II.5.2.3).

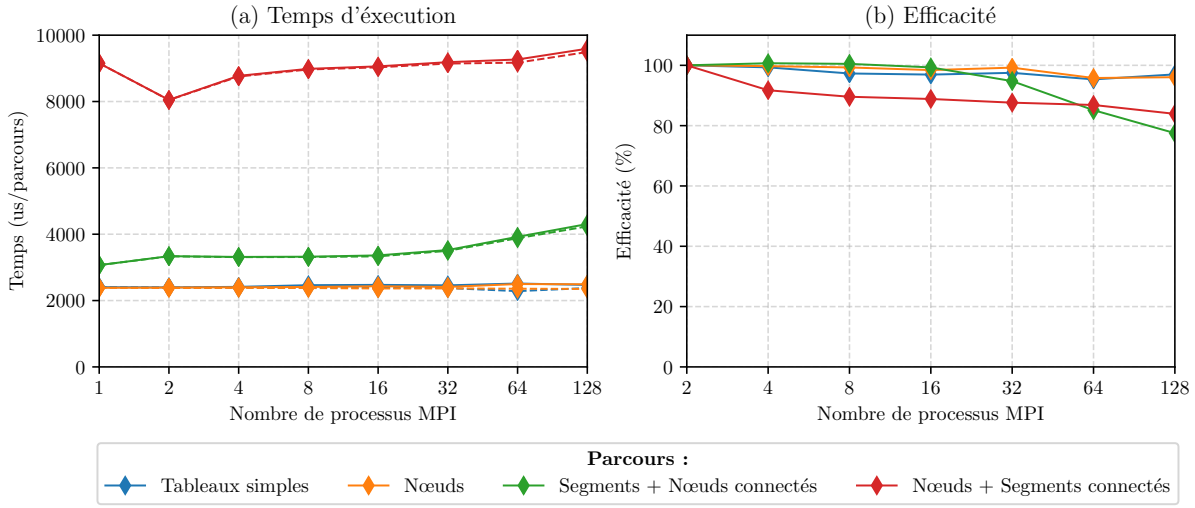


FIGURE II.23 – Mesures de scalabilité des parcours pour 10^6 éléments par processus MPI, une intensité arithmétique de 0.04 Flops/Octet et un processus MPI par nœud NUMA (8 threads).

Le temps d'exécution varie peu, ce qui nous indique que les parcours de la structure de données sont efficaces dans ces conditions. L'efficacité de la plupart des parcours est proche de 100 %, à l'exception du parcours des nœuds avec segments connectés dont l'efficacité s'approche des 90%.

On remarque que le temps d'exécution du parcours des nœuds avec segments connectés est minimal lorsque deux processus MPI sont exécutés sur le même nœud de calcul. Nous avons donc choisi le temps d'exécution sur un nœud complet avec un processus par nœud NUMA (2 MPI $\times$ 8 Threads) comme valeur de référence pour l'efficacité afin d'avoir des valeurs d'efficacité cohérentes. Le temps choisi pour calculer l'efficacité est le plus grand parmi tous les processus, qui est représenté par une ligne pleine sur les courbes de temps d'exécution. Cela permet de prendre en compte les problèmes de répartition de charge dans le calcul de l'efficacité.

II.5.3.2 Impact de la taille des données

La figure II.24 donne la performance des parcours qui ont recours aux communications MPI lorsqu'il y a un nombre faible d'éléments (10^5).

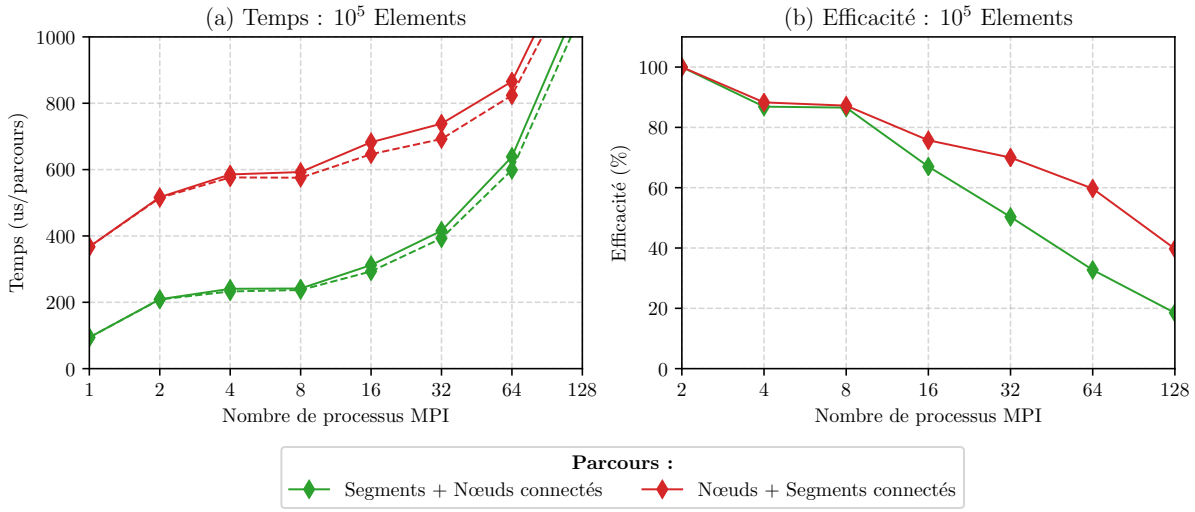


FIGURE II.24 – Mesures de scalabilité des parcours pour 10^5 éléments par processus MPI, une intensité arithmétique de 0.04 Flops/Octet et un processus MPI par nœud NUMA (8 threads).

Les temps de parcours mesurés pour une taille de 10^5 sont très courts, seules les mesures pour les parcours avec communications MPI sont présentées car le temps d'exécution des autres parcours est trop faible pour être mesuré précisément. Il s'agit ici d'un cas extrême où le temps d'exécution est très faible avec à la fois une intensité arithmétique faible et un faible nombre d'éléments. Ces mesures mettent en avant le coût des communications MPI.

Lorsque le temps d'exécution est court, l'impact des communications MPI est plus important : on remarque que l'efficacité est bien inférieure à celle de la figure II.23. Le temps d'exécution reste relativement constant lorsque l'on augmente le nombre de processus pour une taille de 10^6 (fig. II.23a) alors qu'il augmente rapidement lorsque la taille est de 10^5 (fig. II.24b).

Ces résultats peuvent être extrapolés pour l'intensité arithmétique : plus le temps de calcul est long, moins les communications jouent un rôle important dans la performance.

II.5.3.3 Effet des échanges MPI

Dans les figures II.25 et II.26, les parcours qui nécessitent des échanges MPI ont été modifiés afin d'éviter les envois de messages lorsque les valeurs liées aux objets fantômes (*Ghosts*) ne changent pas. Lorsque les données ne sont pas modifiées entre les itérations, il est possible d'éviter les communications MPI en sauvegardant les valeurs liées aux objets fantômes dans un *cache* [1] pour s'en rappeler lors de l'appel suivant. Ces figures comparent la performance lorsque les processus effectuent des communications MPI, et lorsque les données sont gardées en cache.

1. à ne pas confondre avec les caches des processeurs

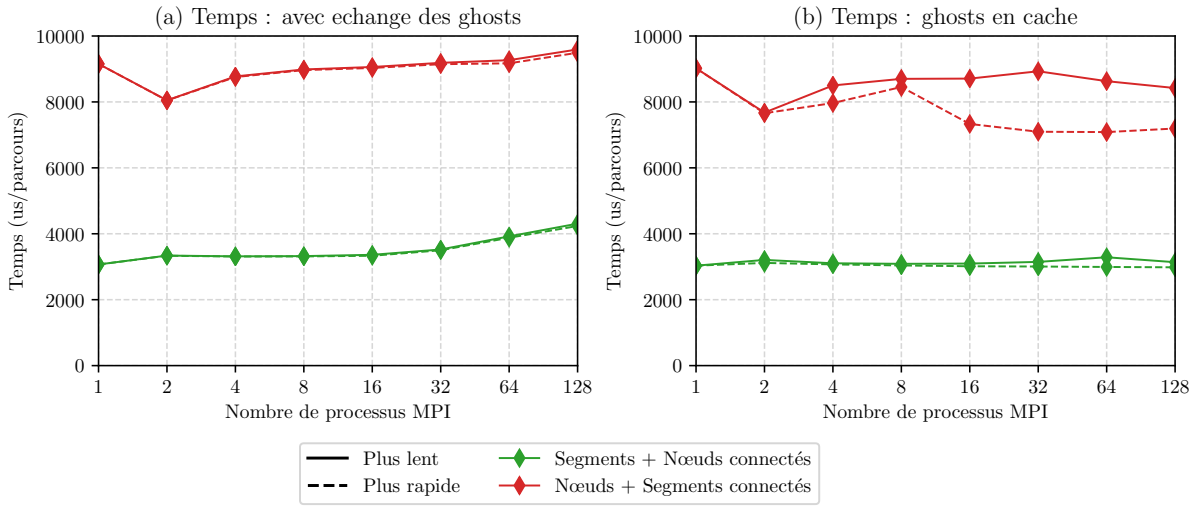


FIGURE II.25 – Effet des échanges MPI sur la performance des parcours pour une taille de 10^6 éléments par processus, une intensité de 0.04 Flops/Octet avec un processus MPI par nœud NUMA (8 threads).

Lorsque les données sont suffisamment grandes (figure II.25), les communications MPI ne changent pas la performance de manière significative. La courbe en pointillés indique le temps du processus le plus rapide, et la courbe pleine du plus lent. Les deux courbes sont disjointes dans la figure II.25b car une différence de charge existe entre les différents processus. Cette différence s'explique par l'inhomogénéité du nombre d'objets fantômes à traiter : les processus du bord ont moins de fantômes. On remarque que pour une taille de 10^6 éléments par processus les échanges MPI ne font que synchroniser tous les processus sur le plus lent.

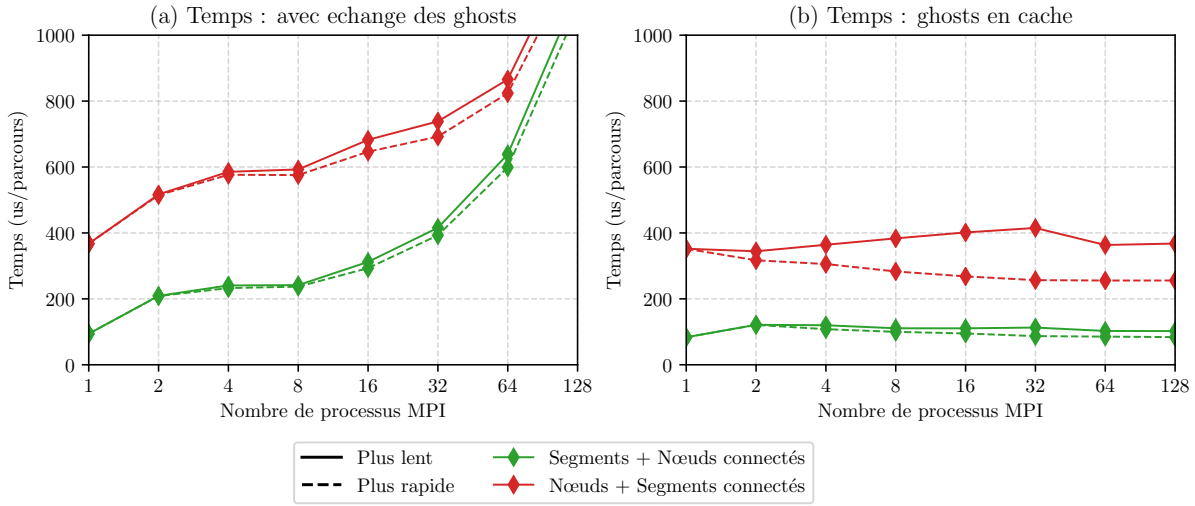


FIGURE II.26 – Effet des échanges MPI sur la performance des parcours pour une taille de 10^5 éléments par processus, une intensité de 0.04 Flops/Octet avec un processus MPI par nœud NUMA (8 threads).

Lorsque la taille des données est faible II.26), le temps de parcours est faible et l'impact des communications est plus fort. Pour les petits volumes de données et les faibles intensités arithmétiques, l'utilisation d'un cache pour les données fantômes permettrait d'améliorer les per-

formances. Malheureusement, comme pour les effets de caches (sec. II.5.2.3), les algorithmes réutilisent rarement les mêmes champs de la structure de données. Pour cette raison, bien que le mécanisme de cache ait été mis en place dans la structure de données, il n'est pas utilisé dans Optidis.

II.5.4 Performance des opérations topologiques

L'objectif principal de la mise en place d'un type de données abstrait pour la structure de données est d'augmenter la fiabilité des algorithmes au sein d'Optidis. Certains des aspects comme les parcours ont été optimisés pour la performance, mais d'autres opérations comme les opérations de modification de la topologie nécessitent encore des optimisations afin que leur performance soit optimale dans un environnement parallèle et distribué. Cette section a pour but de faire un état des lieux de la performance de ces opérations.

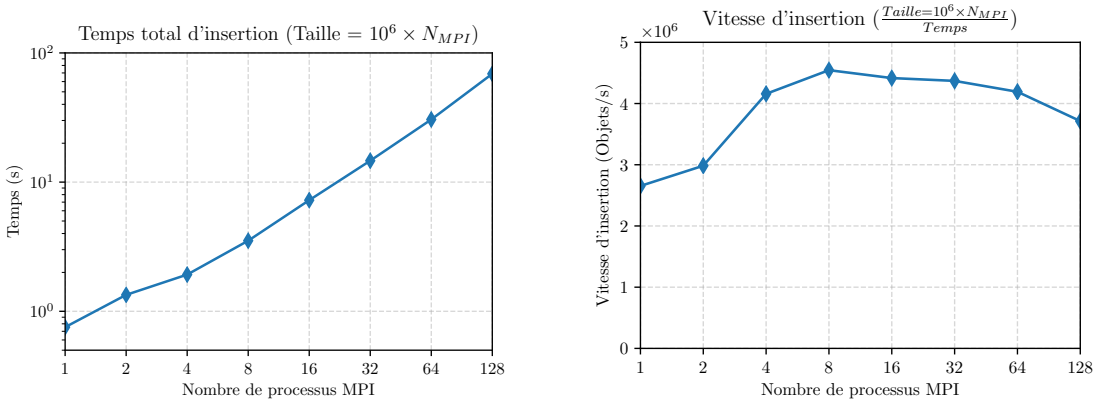
II.5.4.1 Performances théoriques

Les opérations topologiques d'ajout et de suppression de nœuds ou de segments se font de manière collective. Tous les processus doivent participer à l'insertion de chaque objet. Les opérations topologiques collectives, à l'image des opérations MPI telles que `MPI_Bcast` sont inévitablement un obstacle à la scalabilité. Malgré les efforts mis en place pour éviter au maximum les communications pour ces opérations (sec. II.4.3), le temps nécessaire à l'ajout ou à la suppression est linéaire en fonction du nombre d'objets quel que soit le nombre de processus MPI utilisés : ces opérations sont *séquentialisées*. Avec N le nombre d'objets à insérer/supprimer et quel que soit P le nombre de processus MPI, le temps nécessaire à effectuer les opérations topologiques est linéaire :

$$T_{insertion}(N,P) = T_{suppression}(N,P) = O(N) \tag{II.18}$$

II.5.4.2 Mesure des performances d'insertion

Afin d'illustrer la performance des opérations topologiques dans la structure de données, j'ai mesuré le temps nécessaire à l'insertion des objets lors de la construction du réseau de dislocation pour le benchmark des parcours. Les boucles du cas-test présenté en section II.5.1.2 sont insérées les unes après les autres, sans aucune communication MPI. La figure II.27 donne la vitesse d'insertion des objets en fonction du nombre de processus MPI. Chaque point du graphe est déduit de la mesure du temps nécessaire pour construire le réseau de dislocations utilisé pour les graphiques de scalabilité présentés en section II.5.3, et dont la structure est décrite en section II.5.1.2.



(a) Temps total de l'initialisation.

(b) Vitesse moyenne d'insertion.

FIGURE II.27 – Insertion de 10^6 segments par processus MPI dans la structure de données.

Après une légère augmentation, la vitesse d'insertion stagne à $4 \cdot 10^6$ Objets/s malgré le nombre croissant de processus MPI. Le temps d'insertion est donc bien linéaire en nombre d'objets.

Bilan

Dans cette partie, j'ai développé une interface d'accès aux données dans Optidis. Cette interface permet de modifier de manière fiable les données de la simulation sans se préoccuper des problématiques de performances et de distribution des données. L'implémentation proposée à base de tableaux simples stocke les données topologiques du réseau de dislocations en utilisant une représentation orientée objet de graphe. Les propriétés physiques des objets du réseau sont placées dans des conteneurs dynamiques qui utilisent `std::vector` pour plus de fiabilité et une représentation *Structure of Array* pour des performances optimales. Les performances de cette implémentation ont été mesurées dans un contexte OpenMP et MPI.

L'interface proposée permet bien d'abstraire les choix d'implémentation. Par exemple, une autre implémentation qui utilise la *BlockList* (section II.2.2) développée précédemment dans Optidis [40] a aussi été testée. Cette implémentation n'a pas été détaillée ici, car elle n'a pas fait l'objet du même travail d'optimisation que celle à base de tableaux simples. Les résultats des mesures de performances ont montré que la structure de données utilise correctement les ressources disponibles lors des parcours, et que l'abstraction mise en place n'est pas un frein à la performance.

La performance des parcours pourrait être encore améliorée en exploitant la réutilisation des données. Une réorganisation du code permettrait d'utiliser plus efficacement les caches. Une optimisation des communications MPI permettrait de réduire le volume de communications en utilisant le cache de données fantômes décrit en section II.5.3.3.
Les performances des opérations topologiques peuvent aussi être améliorées en développant une interface d'insertion qui ne nécessite pas d'opérations collectives. Un bon point de départ est probablement d'étudier les mécanismes des bases de données transactionnelles [116], notamment le concept de *transaction distribuée*.

Partie III

Détection et traitement des collisions

Introduction

L'algorithme chargé de détecter et traiter les collisions est une étape importante des simulations de dynamique des dislocations qui permet notamment de former les jonctions (sec. I.1.3.3). Il fait partie des opérations topologiques effectuées à la fin de chaque pas de temps. Des nœuds et des segments du réseau de dislocation peuvent par exemple fusionner lorsqu'ils entrent en contact.

Les modifications topologiques effectuées au cours de l'algorithme de collision sont très sensibles et peuvent rapidement être source d'instabilités. La gestion des collisions a d'ailleurs été identifiée comme un frein à l'augmentation du pas de temps [105]. La prise en compte des collisions doit être effectuée de manière fiable et précise en prenant en compte les contraintes cristallographiques afin de respecter la physique des dislocations. La difficulté principale réside dans la gestion de collisions multiples lorsque le pas de temps est grand.

Le caractère massif des simulations menées nous contraint aussi à considérer l'aspect performance des algorithmes. Un double enjeu est donc de permettre une gestion à la fois fiable et rapide de ces collisions.

Dans cette partie, je propose un algorithme de détection et de traitement pour la dynamique des dislocations fiable et performant. Dans un premier temps, le problème à résoudre est présenté, et les différentes solutions proposées dans la littérature aux problèmes de détection et de gestion des collisions sont exposées. Différents algorithmes de collision sont ensuite détaillés et comparés. Je décris ensuite comment l'algorithme retenu a été implémenté dans Optidis. Enfin, les performances de l'algorithme dans un contexte parallèle et distribué sont présentées et discutées.

III.1 État de l'art

L'algorithme de gestion de collision a pour objectif de modifier la topologie du réseau afin de permettre la prise en compte de la physique inhérente aux interactions de contact [61]. Il est par exemple à l'origine de la formation des jonctions (sec. I.1.3.3).

La gestion fiable de ces interactions de contact est cruciale pour mettre en avant certains comportements des matériaux comme l'écrouissage qui émane de la formation des jonctions [80] ou le durcissement par irradiation qui provient des interactions entre dislocations et défauts cristallins produits par l'irradiation [103].

III.1.1 Les collisions pour la DD

La méthode utilisée dans certains codes de dynamique des dislocations est une approche statique de la gestion des collisions [73, 18]. Le traitement statique des collisions - aussi appelé détection de proximité [105] ou détection d'intersection [39] - revient à détecter et gérer les collisions à un instant donné. On calcule la distance entre chaque objet, et les objets dont la distance est inférieure à un rayon de capture ($dist < r_{col}$) sont fusionnés. Le mouvement des objets n'est pas pris en compte donc si une collision a lieu plus tard au cours du pas de temps, elle ne sera pas détectée.

Un tel algorithme a pour avantage d'être facile à mettre en œuvre et est peu coûteux en temps de calcul. Les formules impliquées dans calcul des distances entre des objets immobiles sont

simples et peu complexes. Le défaut de la gestion statique des collisions est son imprécision lorsque le déplacement des objets devient important. Des études récentes ont montré que ce type d'algorithme limite fortement le pas de temps utilisable [105]. Si le pas de temps est trop grand par rapport au rayon de capture les collisions ne sont pas détectées et les objets se traversent sans interagir. Pour que toutes les collisions soient détectées, les objets ne doivent pas se déplacer hors du rayon de capture. Le pas de temps est alors limité par le rayon de capture r_{col} et la vitesse de déplacement des objets v : $dt < r_{col}/2v$.

Pour contourner cette limite, il est possible de découper le pas de temps dt en sous-pas dt_{col} tel que $dt_{col} < 2r_{col}/v_{max}$. L'algorithme effectue des sous-pas de temps successifs en alternant une détection de collisions statique, et une fusion de proximité et déplaçant à chaque sous-pas les dislocations. Ce type d'approche est par exemple utilisée dans microMegas/Tridis [73]. Le diagramme III.1 illustre cet algorithme.

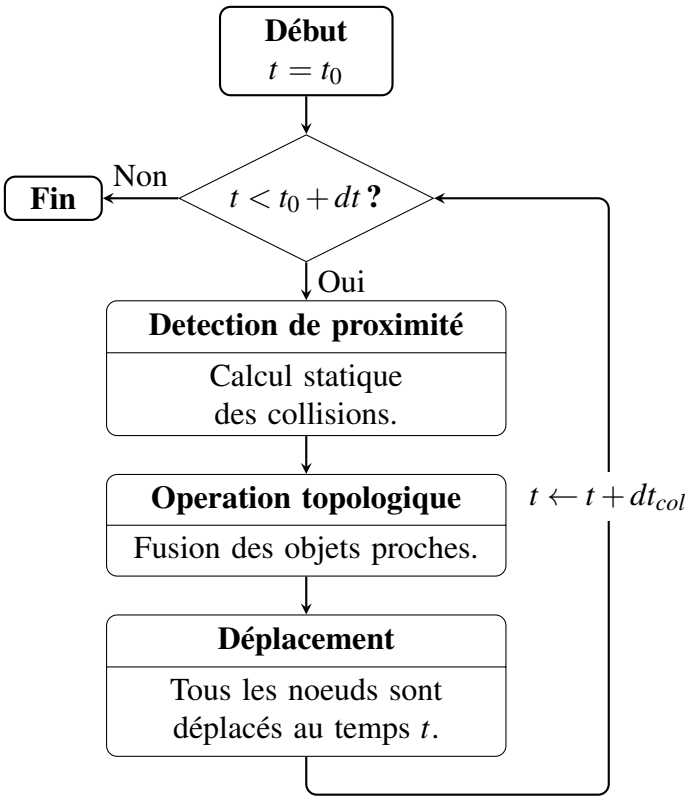


FIGURE III.1 – Algorithme de collision : Fusion de proximité. Plusieurs sous-pas de temps sont effectués en alternant détection statique et fusion des objets.

Cet algorithme est coûteux car il nécessite d'effectuer plusieurs détections de collisions. La détection des collisions, même statique, est un problème à n-corps d'une complexité élevée.

Plus récemment, des méthodes dynamiques de traitement des collisions ont vu le jour, par exemple dans NUMODIS [36] et dans ParaDis [105]. Un traitement dynamique des collisions permet de prendre en compte le déplacement des objets lors de la détection des collisions. Pour chaque paire d'objets en mouvement, on détermine s'ils s'intersectent au cours du pas de temps, mais aussi à quel moment a lieu cette collision.

Une gestion dynamique permet de détecter toutes les collisions qui ont lieu quelque soit l'intervalle de temps considéré et ne limitent pas le pas de temps utilisable. La mise œuvre du

traitement dynamique des collisions est toutefois plus délicate que la détection de proximité. Il pose des problèmes de chronologie des intersections. Les collisions doivent être gérées dans l'ordre sous peine de générer des conflits. De plus, la détection des collisions entre objets dynamiques est bien plus complexe qu'un simple calcul de distance comme pour la gestion statique des collisions.

III.1.2 Détection des collisions dynamiques

Le sujet de la détection de collisions entre objets en mouvement, aussi appelée détection dynamique ou détection continue des collisions (CCD - *Continuous Collision Detection*) intéresse depuis longtemps les chercheurs en robotique [60, 30, 53]. Elle est utilisée principalement pour détecter les collisions avant qu'elles ne se produisent pour les éviter. Le domaine de l'animation 3D utilise aussi la détection dynamique des collisions entre solides depuis les années 90 [10, 113] pour animer de manière fidèle les objets physiques, comme par exemple les vêtements ou les cheveux. Elle est aujourd'hui utilisée dans un contexte temps réel [39], notamment dans l'industrie du jeu vidéo où la performance joue un rôle crucial.

Les algorithmes pour la dynamique des dislocations sont des algorithmes de détection de collision n-corps car tous les objets du maillage peuvent entrer en collision. Le réseau de dislocations est modélisé par un ensemble de cylindres de rayon r_{col} autour des segments sur le modèle de la géométrie de construction de solides (CSG - *Constructive Solid Geometry*) [97].

L'algorithme naïf où chaque paire d'objets est testée possède une complexité quadratique $O(N^2)$. Il peut être utilisé comme algorithme de référence, mais sa complexité est trop importante pour qu'il soit utilisé sur des données massives. Des solutions basées sur la localité spatiale permettent de réduire la complexité de l'algorithme, notamment en triant spatialement les objets et en hiérarchisant l'espace. En effet, certains objets sont souvent trop éloignés les uns des autres pour qu'une collision soit possible.

Dans la littérature, plusieurs solutions existent pour réduire la complexité du calcul en réduisant le nombre de tests à effectuer. L'espace peut être découpé spatialement afin de séparer les objets qui sont trop éloignés pour entrer en collision. On peut citer la méthode *sweep and prune* [27], aussi appelée *sort and sweep* [11] qui trie les objets spatialement. Des méthodes qui consistent en un découpage de l'espace en une grille uniforme [54], voire hiérarchique [83], sont aussi utilisées. Enfin, des techniques comme l'utilisation des BSP-Trees [50, 99] permettent de hiérarchiser les géométries plus complexes. Une seconde technique consiste à utiliser des volumes englobants (BV - *Bounding Volumes*) [27]. Le principe consiste à inscrire les objets à tester pour la collision dans un volume, puis à tester l'intersection de ces volumes. Cette technique qui vise à rejeter certaines paires d'objets qui ne peuvent pas entrer en collision est appelée *fast-reject*. Un *fast-reject* efficace doit permettre de détecter toutes les collisions tout en filtrant le mieux possible les faux positifs. L'utilisation de volumes englobants simples, comme des sphères par exemple, permet d'effectuer ce filtrage rapidement. Les volumes englobants peuvent aussi être utilisés au sein de hiérarchies pour effectuer ce filtrage par étapes successives [71]. Ces solutions peuvent être utilisées conjointement pour accélérer la détection des collisions.

III.1.3 Positionnement

Mon approche consiste à utiliser un algorithme dynamique afin de rentre plus fiable la détection et le traitement des collisions. L'objectif est de lever les limitations liées au pas de temps induites

par le traitement des collisions tel qu'il a été mis en place dans Optidis au cours des travaux de thèse de A. ETCHEVERRY [40].

Pour améliorer la fiabilité de l'algorithme de collision, il faut tout d'abord détecter de manière fiable les collisions. De nouvelles primitives dynamiques de détection de collisions entre objets plus précises et robustes son proposées. Les objets considérés sont les segments mais aussi les nœuds afin de situer plus précisément le lieu et le moment de la collision.

Pour accélérer la détection des collisions dynamiques, je propose d'utiliser des techniques de partitionnement de l'espace et de volumes englobants.

A. ETCHEVERRY [40] a mis en place une technique de partitionnement basée sur le découpage utilisé pour la FMM dans le calcul de forces. Il utilise le calcul de champ proche (P2P) pour ne tester les collisions qu'entre objets de boites voisines (*cutoff*). Je propose de décorréler le partitionnement utilisé dans les collisions de celui utilisé dans la FMM en utilisant un découpage de l'espace selon une grille uniforme indépendante. Cela permet de régler la finesse de la grille en fonction de critères basés sur les collisions et non sur le calcul des forces.

J'utilise également la technique des volumes englobants pour accélérer la détection qui s'effectue alors en deux phases [83]. Dans une première phase (*Broad Phase* ou *fast-reject*) des collisions entre les volumes englobants simples sont détectées. Le choix des volumes utilisés s'est porté sur les sphères car leur intersection est facile à tester [89]. Dans un second temps (*Narrow Phase*) les collisions exactes entre les nœuds et les segments présélectionnés sont évaluées.

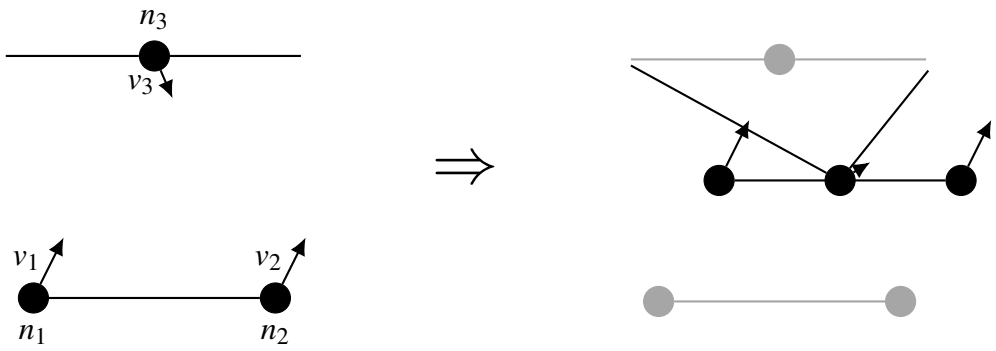
Enfin, je propose un nouvel algorithme de gestion des collisions. J'ai comparé plusieurs algorithmes en étudiant leur fiabilité, mais aussi leur performance. Parmi ces derniers, le meilleur compromis entre la précision du résultat obtenu et le temps d'exécution a été sélectionné et mis en place dans Optidis. À l'inverse de l'algorithme précédent, il est capable de gérer les collisions entre géométries complexes, même lorsque le pas de temps est grand et que plusieurs collisions ont lieu simultanément.

III.2 Définition du problème

Le problème de détection et de gestion des collisions est présenté ici dans sa version dynamique. Il faut donc détecter et traiter les collisions entre les différents objets (nœuds et segments) en mouvement au cours d'un pas de temps.

III.2.1 Définition générale

Les collisions interviennent lorsque des dislocations s'intersectent. Par exemple, dans la figure III.2, deux lignes de dislocations se dirigent l'une vers l'autre et entrent en contact. La dislocation de la partie haute fusionne avec la partie basse. Le rôle de l'algorithme de collision est de détecter les intersections qui ont lieu au cours d'un pas de temps, puis d'effectuer les modifications topologiques qui en découlent.



(a) Avant collision. (b) Après collision.

FIGURE III.2 – Un nœud entre en collision avec un segment et fusionne avec celui-ci.

Lorsque plusieurs collisions ont lieu au cours d'un pas de temps, le rôle de l'algorithme de collision est d'effectuer toutes les opérations topologiques nécessaires dans le bon ordre. La figure III.3 illustre par exemple la fusion entre deux lignes de dislocations qui occupent le même plan de glissement. On remarque que lors de la fusion des deux lignes une multitude de modifications topologiques successives ont lieu.

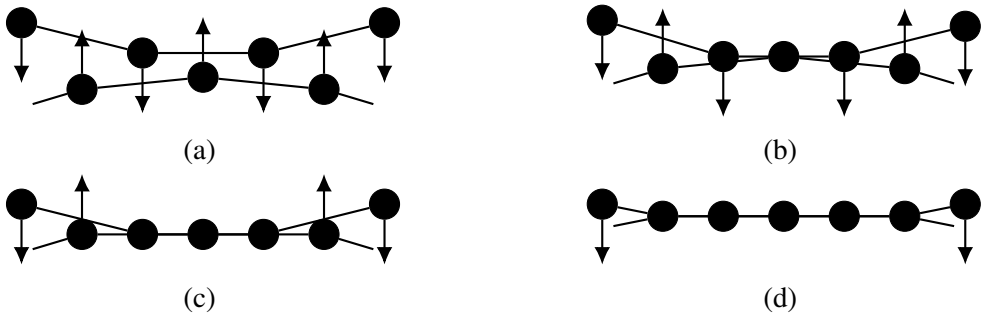


FIGURE III.3 – Étapes successives de la fusion de deux lignes de dislocation dans le plan : les deux lignes se dirigent l'une vers l'autre. Les nœuds fusionnent successivement avec les segments.

Ce problème est décomposé en deux sous-problèmes. Dans un premier temps, il faut détecter les collisions, puis les traiter en modifiant la topologie du réseau. La détection des collisions vérifie pour chaque paire d'objets s'il y a contact, et la gestion des collisions modifie le réseau de dislocations en fonction des intersections qui ont lieu. Les sections suivantes définissent ces deux sous-problèmes.

III.2.2 Détection des collisions

La détection s'effectue à la suite de la phase d'intégration (**Déplacement** sur la figure I.22 en section I.3.2). On connait donc la position initiale $\boldsymbol{x}_i(t)$ et la position finale $\boldsymbol{x}_i(t+dt)$ de chacun des nœuds du réseau de dislocations. On suppose que les nœuds se déplacent à vitesse $\boldsymbol{v}_i$ constante dans un espace à 3 dimensions pendant un pas de temps dt. La figure III.4 illustre la situation initiale du problème.

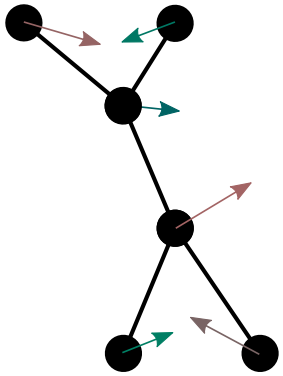


FIGURE III.4 – Le réseau de dislocations portant les vitesses nodales.

Dans le cas du schéma d'intégration explicite d'Euler, v_i est la vitesse calculée lors de l'application des lois de mobilité (sec. I.3.4). Afin de s'abstraire du type d'intégration utilisé, la vitesse peut être calculée à partir des positions initiales et finales des nœuds :

$$\boldsymbol{v}_i = \frac{\boldsymbol{x}_i(t+dt) - \boldsymbol{x}_i(t)}{dt} \tag{III.1}$$

Le problème à résoudre consiste à détecter quels objets (nœuds ou segments) entrent en contact au cours d'un pas de temps, et à déterminer le temps où le contact à lieu. L'algorithme de détection exécute des primitives de collisions pour chaque paire d'objets, et construit une liste des collisions détectées. Deux objets entrent en collision lorsque leur distance de séparation passe sous un certain seuil r_{col} appelé *rayon de capture*. La formule III.2 formalise la condition de collision entre deux objets o_1 et o_2 au cours d'un pas de temps dt :

$$\text{collision}(o_1, o_2) \equiv \exists t \in [0, dt], \text{dist}(o_1, o_2) < r_{col} \tag{III.2}$$

La difficulté de ce problème réside dans sa complexité quadratique. En effet, une manière naïve de détecter les collisions serait de tester pour chaque paire d'objets s'ils entrent en contact.

III.2.3 Gestion des collisions

La seconde étape gère les collisions détectées dans l'étape précédente. Elle consiste à déplacer et fusionner les lignes de dislocation en fonction des contacts qui ont lieu. Il faut effectuer dans l'ordre toutes les collisions qui ont été détectées au cours du pas de temps. De nouvelles collisions non détectées précédemment pourront être induites par la modification du réseau, il pourra donc être nécessaire d'effectuer une détection partielle à chaque modification pour ne pas courir le risque de rater des collisions.

Les modifications qui sont liées à la topologie peuvent être génératrices de fortes instabilités. La non-détection d'une seule collision peut changer radicalement les résultats. Par exemple, ne pas détecter la collision entre deux dislocations peut mener à l'absence totale d'une jonction. La gestion des collisions se doit donc d'être fiable afin de fusionner correctement tous les objets qui entrent en contact.

Pour modifier la topologie du réseau, les propriétés qui nous intéressent en plus des propriétés géométriques sont celles liées au système de glissement. Chaque segment se déplace dans un plan de glissement et est caractérisé par un vecteur de Burgers. Il est important que chaque modification géométrique respecte les contraintes cristallographiques : les déplacements doivent se faire dans les plans de glissement des segments et les données portées par les nouveaux segments insérés, comme le vecteur de Burgers, doivent être cohérentes.

Les propriétés géométriques du maillage généré ont aussi leur importance. Il est crucial que la géométrie du réseau ne soit pas génératrice d'instabilités. Les angles forts et les fortes différences entre les longueurs des segments sont à proscrire.

III.3 Étude comparative de différents algorithmes

Plusieurs stratégies sont envisageables pour résoudre ce problème. Par exemple, nous avons vu en section III.1.1 qu'il est possible d'effectuer une gestion statique ou dynamique des collisions. J'ai comparé plusieurs algorithmes pour déterminer le plus adapté.

Les algorithmes proposés vont d'un algorithme de gestion statique des collisions à des algorithmes dynamiques plus complexes. J'ai étudié ces algorithmes d'un point de vue théorique pour mieux comprendre les limitations et les complexités de chacun. Ils ont été implémentés dans un démonstrateur hors d'Optidis en s'abstrayant des contraintes cristallographiques pour observer leur fiabilité et leur performance.

L'étude de ces différents algorithmes a permis d'en choisir un en particulier à intégrer dans Optidis.

III.3.1 Algorithmes comparés

Quatre algorithmes sont présentés. Le premier est un algorithme de fusion de proximité avec sous-pas de temps (sec. III.1.1). Les autres algorithmes sont dynamiques et sont des améliorations incrémentales de l'algorithme implémenté dans Numodis [36] et Optidis [40].

III.3.1.1 Fusion de proximité

Le premier algorithme étudié est un algorithme de fusion de proximité (section III.1.1). La détection de collision est effectuée en calculant la distance entre les différents objets, puis les modifications topologiques sont effectuées en utilisant les opérations topologiques détaillées en annexe B.2.

Cet algorithme a pour avantage d'être facile à implémenter et robuste, ce qui en fait un bon algorithme de référence pour vérifier la validité des autres algorithmes. Cependant, même en utilisant une méthode d'intégration explicite qui ne permet pas d'utiliser des pas de temps très grands, un grand nombre de sous-pas de temps est nécessaire. Selon la section I.3.4, les pas de temps dt utilisés pour un intégrateur *forward euler* sont de l'ordre de $dt \approx L/v_{max}$ avec L de l'ordre de grandeur de la longueur d'un segment. L est grand devant r_{col}, donc le nombre de sous-pas de temps à effectuer est important, de l'ordre de $\frac{L}{r_{col}}$.

III.3.1.2 Algorithme dynamique séquentiel

L'algorithme dynamique séquentiel est une adaptation de l'algorithme utilisé dans Numodis [36] et Optidis [40]. La différence principale est l'ajout d'une phase de re-détection pour ne rater aucune collision. Les primitives de détection et les opérations topologiques ont aussi été réécrites pour être plus précises.

Cet algorithme commence après une phase de détection dynamique complète des collisions effectuées avec les primitives détaillées en annexe B.1. Les collisions détectées sont stockées dans une liste globale de collisions. L'algorithme, présenté en figure III.5, possède une liste de collisions détectées au cours du pas de temps qu'il traite dans l'ordre chronologique. Pour chaque collision détectée au temps t_{col} qui implique les objets o_1 et o_2, l'ensemble du réseau de dislocations est déplacé au temps t_{col} puis o_1 et o_2 sont fusionnés. Les mêmes opérations topologiques que pour la fusion de proximité sont utilisées (Annexe B.2). Il faut ensuite tester

si les objets modifiés entrent en collision avec des objets voisins dans une phase de *re-détection* locale. On passe ensuite à la collision suivante, et l'algorithme s'arrête lorsqu'il n'y a plus de collisions à traiter.

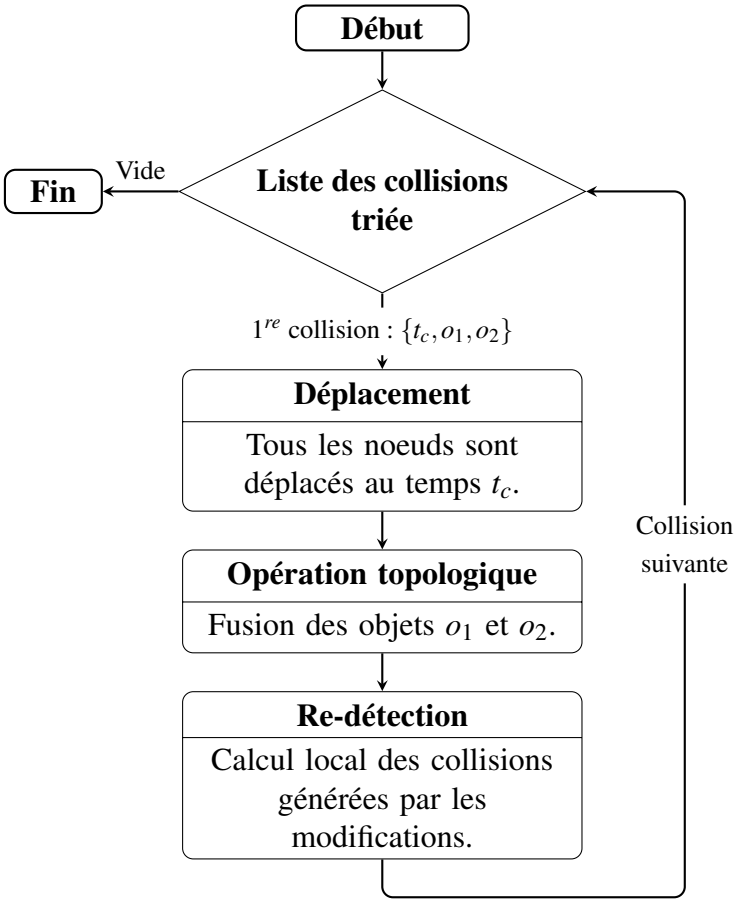


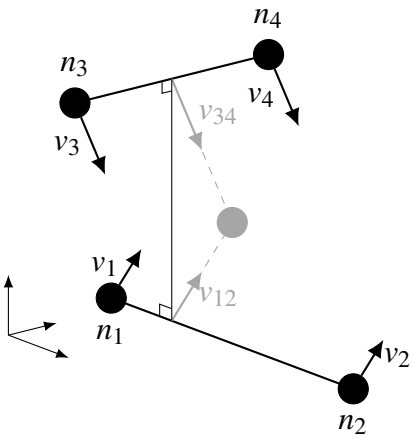
FIGURE III.5 – Algorithme de collision : dynamique séquentiel. Les collisions sont traitées chronologiquement en déplaçant tout le réseau de dislocation à chaque opération topologique.

Cet algorithme possède une complexité moins importante que l'algorithme à fusion de proximité, et reste relativement facile à mettre en œuvre. Le déplacement des dislocations et la re-détection des collisions sont les opérations les plus coûteuses de l'algorithme et peuvent être optimisés. La re-détection des collisions pour les objets qui ont été modifiés au cours des opérations topologiques est une détection locale et peut utiliser les mêmes techniques d'accélération que pour la détection globale des collisions. Le découpage en grille uniforme permet de ne tester que les dislocations des boites voisines des objets modifiés.

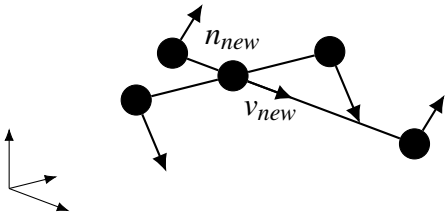
III.3.1.3 Gestion séquentielle sans déplacement

Il est possible de réduire la complexité de l'algorithme précédent (fig. III.5) en supprimant l'étape de déplacement des objets. En effet, cette étape possède une complexité linéaire en fonction du nombre d'objets présents dans la simulation. Je propose donc un algorithme amélioré où le déplacement n'est effectué qu'une seule fois.

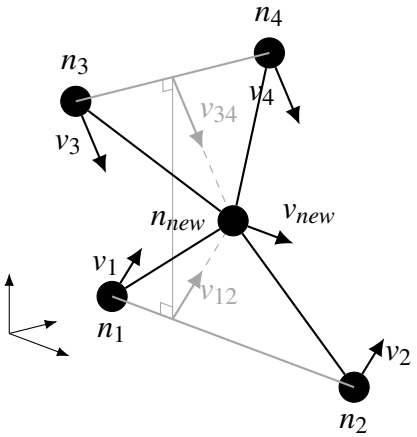
Dans cet algorithme, le réseau de dislocations n'est déplacé qu'une seule fois à la fin de l'algorithme plutôt qu'au traitement de chaque collision. Pour ce faire, les opérations topologiques doivent être adaptées afin de supporter la fusion d'objets éloignés. L'algorithme est modifié pour devenir l'algorithme décrit en figure III.7. La figure III.6 explique la différence entre les deux algorithmes.



(a) État initial du réseau.



(b) Opération **avec** déplacement :
Les nœuds n_1, n_2, n_3 et n_4 sont déplacés **puis** les segments sont fusionnés en n_{new}.



(c) Opération **sans** déplacement :
n_{new} est crée sans déplacer les nœuds n_1, n_2, n_3 et n_4. Les nœuds ne sont déplacés qu'une fois toutes les opérations topologiques effectuées.

FIGURE III.6 – Comparaison des opérations topologiques avec et sans déplacement.

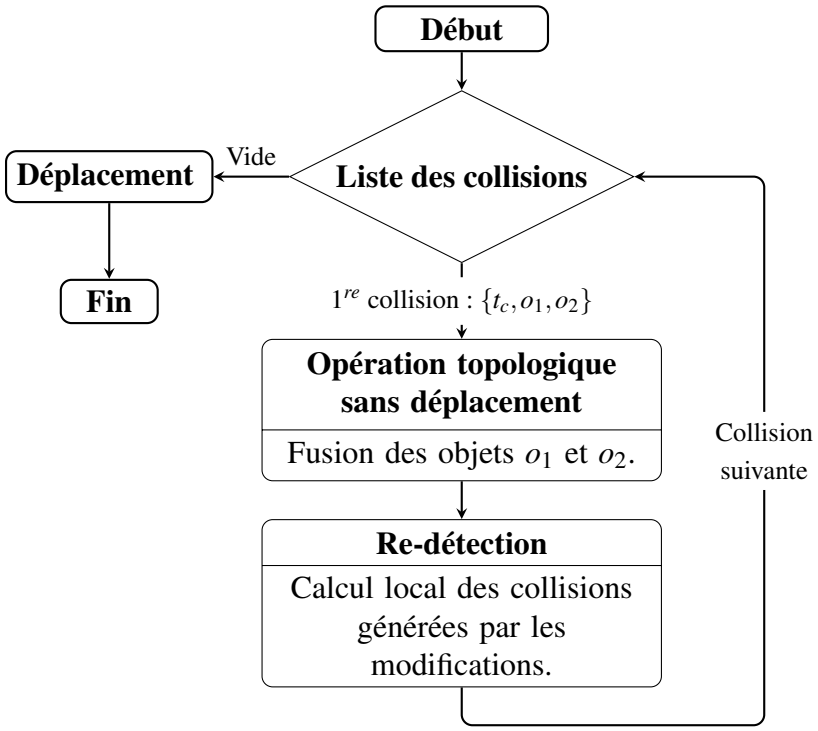


FIGURE III.7 – Algorithme de collision : séquentiel sans déplacement. L'algorithme est similaire à celui de la figure III.5, mais l'opération de **Déplacement** effectuée une seule fois à la fin. Les **opérations topologiques** doivent être adaptées a cette modification.

Les opérations topologiques utilisées pour cet algorithme sont détaillées dans l'annexe B.3. En plus de définir la nouvelle topologie comme les opérations topologiques précédentes, ces nouvelles opérations doivent définir la position des objets fusionnés de manière à ne pas interférer avec les autres collisions. La complexité de cet algorithme est moins grande, mais les opérations topologiques sont plus compliquées à mettre en place.

III.3.1.4 Gestion de collision parallèle

Un autre moyen d'accélérer la gestion des collisions est d'utiliser le parallélisme. Pour cela, il faut définir un ordre de traitement des collisions. Intuitivement, lorsque deux collisions sont *suffisamment lointaines* le traitement de l'une n'influe pas sur l'autre et il est possible de les traiter parallèlement. La notion de collisions *suffisamment lointaines* s'appuie en fait sur un ordre partiel dans lequel les collisions doivent être traitées. En contraste avec l'algorithme séquentiel qui utilise un ordre total, un algorithme parallèle doit pouvoir gérer plusieurs collisions simultanément. Définir un ordre partiel sur les collisions revient à définir la relation $<$ telle que pour une collision c, l'ensemble $\{c_i | c_i < c\}$ soit l'ensemble des collisions qui doivent être traitées avant la collision c. Par exemple, la relation d'ordre utilisée pour les algorithmes séquentiels est l'ordre chronologique (ordre total) :

$$c_1 <_{seq} c_2 \Leftrightarrow c_1.t_{col} < c_2.t_{col} \tag{III.3}$$

Je propose un ordre partiel pour les collisions qui garantit que toutes les collisions sont gérées dans l'ordre :

Définition .1. Une collision c_1 doit être traitée avant c_2 si c_1 modifie des objets impliqués dans c_2 avant que la collision c_2 ait lieu. Soient M_c les objets modifiés par c et O_c les objets qui entrent en collision pendant la collision c :

$$c_1 <_{parallel} c_2 \Leftarrow c_1.t_{col} < c_2.t_{col} \wedge M_{c_1} \cap O_{c_2} = \emptyset \tag{III.4}$$

De cet ordre partiel on peut déduire un **test de précédence**. Ce test décide si une collision peut être traitée ou si d'autres collisions doivent être traitées auparavant. Une collision peut être traitée si elle est minimale selon $<_{parallel}$. C'est-à-dire qu'il n'existe pas de collision devant être gérée antérieurement. Grâce à la définition de $<_{parallel}$, ce test de précédence peut être effectué en ayant seulement une connaissance locale du réseau de dislocation. En effet, les objets impliqués sont les quelques objets contenus dans M_c et dans O_c.

L'algorithme de traitement parallèle proposé utilise un modèle à agents [91, 90] : chaque objet de la simulation exécute indépendamment l'algorithme de la figure III.8 de manière concurrente.

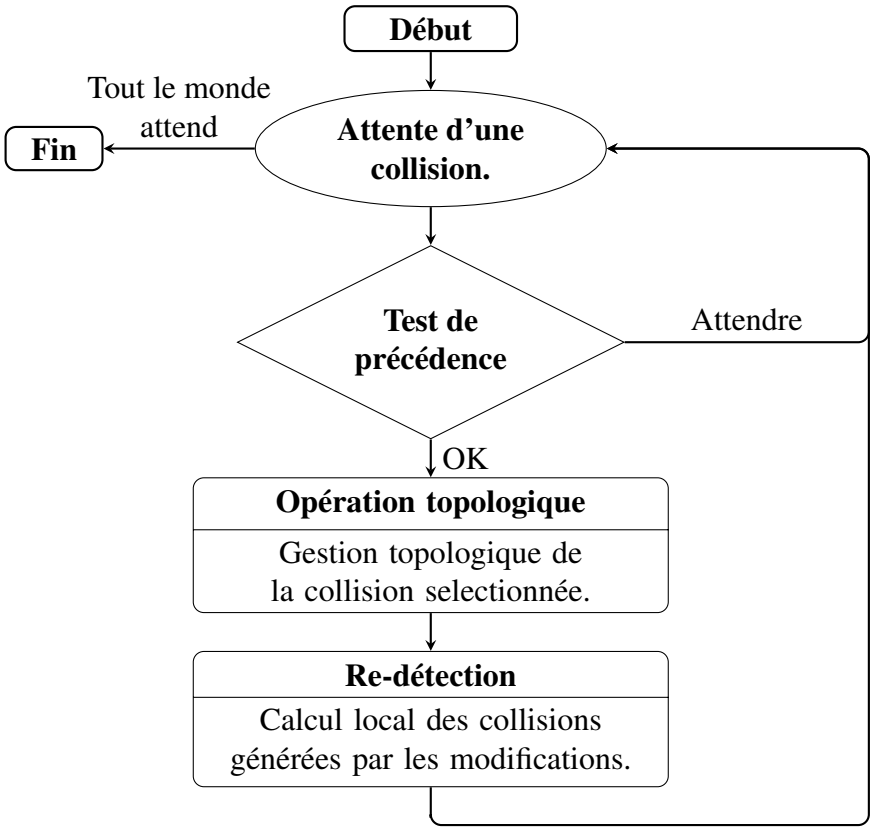


FIGURE III.8 – L'algorithme parallèle de gestion des collisions.
Chaque objet exécute cet algorithme et possède une liste des collisions où il intervient. Si tous les événements antérieurs ont été gérés (**Test de précédence**), l'objet traite sa première collision (**Opération topologique**). L'algorithme se termine quand tous les objets ont fini de traiter leurs collisions.

Chaque objet possède une liste C_i des collisions dans lesquelles il intervient et traite dans l'ordre chronologique ses collisions. Avant de traiter une collision, le segment doit s'assurer que la collision vérifie le test de précédence décrit plus haut : l'objet est mis en attente tant que des collisions antérieures selon $<_{parallel}$ existent. Une fois que toutes les collisions antérieures ont été

traitées par les autres objets, le segment courant peut effectuer les modifications topologiques liées à la collision. Lorsque tous les segments ont géré toutes leurs collisions, l'algorithme termine. Comme la re-détection peut ajouter une collision à un objet à tout moment ils doivent tous attendre de nouvelles collisions, même si leur liste est vide.

Dans cette implémentation, le test de précédence parcourt les listes de collisions de tous les éléments $o \in M_c$ et vérifie si leur liste de collisions C_o contient une collision chronologiquement antérieure. Tant qu'une telle collision existe, le traitement est mis en attente. Les opérations topologiques sans déplacement évoquées en section III.3.1.3 et détaillées dans l'annexe B.3.

La gestion parallèle des collisions pose plusieurs problèmes. Premièrement, l'accès à la structure de données contenant les dislocations pour modifier la topologie doit se faire de manière concurrente. Le code de démonstration ne proposant pas d'accès parallèle à cette structure, les modifications topologiques sont *séquentialisées*. Ensuite, le grain de parallélisme proposé dans cet algorithme est trop fin : traiter chaque segment dans un thread différent n'est pas possible. Pour grossir le grain de parallélisme, il est possible de grouper les segments. Une gestion réellement parallèle des collisions est difficile à mettre en place, et nous verrons qu'elle n'est pas forcément nécessaire.

III.3.2 Comparaison de la précision des algorithmes

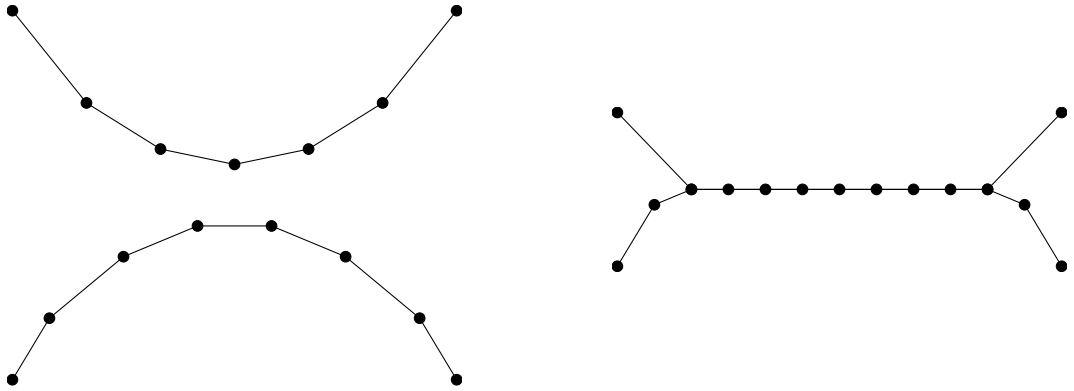
Les algorithmes précédents ont été implémentés dans un démonstrateur en faisant abstraction des contraintes cristallographiques. Cela nous permet de comparer leur fiabilité, ainsi que la vitesse d'exécution de ces derniers. En plus des expériences menées dans le démonstrateur, les complexités des différents algorithmes sont comparées. Ces résultats nous permettent de choisir l'algorithme à mettre en place dans Optidis.

III.3.2.1 Méthode : banc de test

Le banc de test est un programme qui permet de tester différentes implémentations de la gestion des collisions. Une géométrie initiale est générée, puis les différents algorithmes de gestion des collisions sont exécutés afin de comparer leurs résultats.

Les algorithmes implémentés dans le banc de test effectuent les opérations géométriques nécessaires à la formation de jonctions, mais ne respectent pas forcément les contraintes cristallographiques des dislocations. Les plans de glissement, par exemple, ne sont pas pris en compte.

Les algorithmes présentés ont été validés par des tests unitaires qui les exécutent sur des géométries simples avec peu d'objets. Parmi les situations testées on compte notamment des géométries planaires et dans l'espace, avec une ou plusieurs collisions ayant lieu au cours d'un pas de temps qui peut être plus ou moins grand. L'exemple présenté pour illustrer la précision des algorithmes est une jonction coplanaire (fig. III.9). Les deux dislocations se déplacent l'une vers l'autre dans le même plan et s'intersectent pour former une jonction droite.



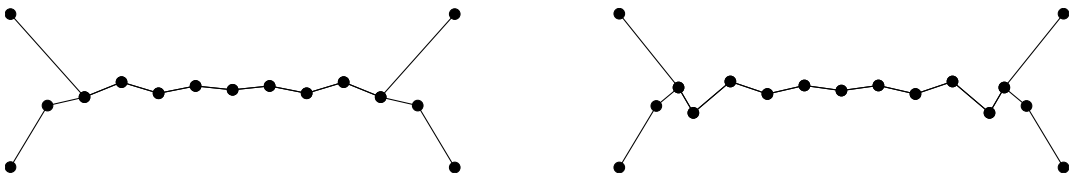
(a) Initial $t = 0$ (b) Final $t = dt$

FIGURE III.9 – Validation des collisions : jonction coplanaire. La figure (b) représente la situation attendue après l'exécution de l'algorithme de collision sur les dislocations de la figure (a) qui se déplacent l'une vers l'autre.

La situation initiale, illustrée dans la figure III.9a, met en scène deux lignes de dislocations. Le déplacement $v \cdot dt$ des nœuds au cours du pas de temps est supérieur à la longueur L d'un segment : $v \cdot dt = 2L$. Le pas de temps dépasse les conditions de stabilité de l'intégrateur explicite (i.e $dt < \frac{L}{v}$) afin d'amplifier les résultats obtenus : il est 4 fois plus grand que le pas de temps maximal utilisé dans des simulations réelles qui utilisent l'intégrateur *forward euler*. Le rayon de capture est petit par rapport à L : $r_{col} = 0.025L$, soit 1.5% du déplacement des nœuds. Cela correspond par exemple à des segments de taille 40 Å pour un rayon de capture de 1 Å. Dans la situation finale, les deux parties fusionnent pour former la jonction visible dans la figure III.9b.

III.3.2.2 Résultats

La figure III.10 donne les résultats renvoyés par les différents algorithmes dans le banc de test.



(a) Opérations topologiques avec déplacement : fusion de proximité et algorithme dynamique séquentiel.

(b) Opérations topologiques sans déplacement : algorithmes dynamiques sans déplacement et algorithme parallèle.

FIGURE III.10 – Résultats de la collision pour les différents algorithmes.

La topologie est celle attendue dans tous les cas, mais la géométrie est un peu plus irrégulière que le résultat de référence. Les dents-de-scie observées ne posent pas de problème car elles sont atténuées par les calculs de tension de ligne au pas de temps suivant. On remarque tout de même que le résultat de l'image III.10b est un peu différent de l'image III.10a car les opérations topologiques utilisées ne sont pas les mêmes.

III.3.2.3 Influence de la re-détection des collisions.

Dans les trois algorithmes présentés précédemment, après chaque itération de l'algorithme les collisions sont re-détectées soit totalement (Proximité) soit en partie (Dynamique séquentiel et parallèle). Lorsque la re-détection des collisions est désactivée, tous les algorithmes perdent en précision et peuvent ne pas détecter un certain nombre de collisions. Voyons comment ces algorithmes réagissent à la désactivation de la re-détection.

Fusion de proximité

Lorsque l'on n'effectue une seule itération de l'algorithme de fusion de proximité, on observe le phénomène remarqué dans [105]. Toutes les collisions qui n'ont pas lieu à $t = 0$ sont ignorées. Le résultat en fin de pas de temps est représenté dans la figure III.11. Le calcul est bien plus rapide que lorsque des sous-pas de temps sont exécutés, au prix d'une détection de collision approximative. La topologie obtenue est fausse d'un point de vue physique.

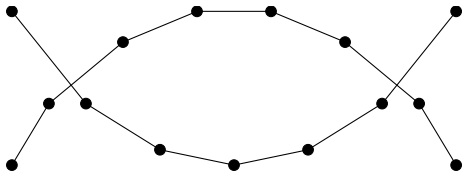


FIGURE III.11 – Effet de la re-détection : Fusion de proximité.

Cependant, pour des pas de temps suffisamment petits et un rayon de capture suffisamment grand, la fusion de proximité sans détection peut être suffisante. De plus, les dislocations capables de fusionner s'attirent généralement et la détection des collisions ratées pourra avoir lieu aux itérations suivantes. Pour certaines simulations où la formation de jonctions n'a pas besoin d'être précise, l'algorithme de fusion de proximité pourrait être une solution rapide.

Une solution pour que l'algorithme sans re-détection soit juste est de limiter le pas de temps à $dt_{col} = 2r_{col}/v_{max}$ afin que les objets ne se déplacent pas en dehors de la sphère de rayon r_{col}.

Algorithmes dynamiques.

Les algorithmes dynamiques sans re-détection, qu'il s'agisse de l'algorithme séquentiel ou parallèle, détectent seulement une partie des collisions. Comme le montre la figure III.12, le résultat obtenu n'est pas juste physiquement.

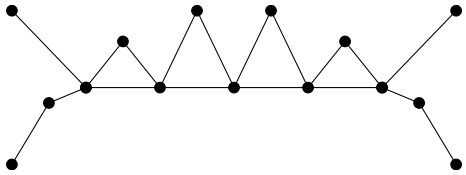


FIGURE III.12 – Effet de la re-détection : Algorithmes dynamiques.

Lorsqu'un nombre réduit de collisions a lieu au cours d'un pas de temps, les effets de la désactivation de la re-détection peuvent être négligeables. Cependant, la détection partielle des collisions peut entraîner des géométries génératrices d'instabilités dans les autres phases du calcul.

L'utilisation de l'algorithme séquentiel est toutefois possible sans re-détection en arrêtant le pas de temps à la première collision. Le pas de temps est alors diminué seulement lorsque nécessaire. Dans ce cas, un algorithme hybride dynamique/proximité peut être utilisé pour ne pas avoir à trop limiter le pas de temps.

III.3.3 Calculs de complexité

L'algorithme de détection des collisions naïf possède une complexité quadratique $O(N^2)$, mais le découpage en grille uniforme permet de rendre le calcul linéaire dans certaines conditions. La complexité calculée pour un nombre N de dislocations uniformément réparties dans B blocs contenant chacun en moyenne D_B dislocations est décrite par la formule suivante :

$$C_{\text{Detection}} = 27B \cdot (D_B)^2 = O(D_B \cdot N) \tag{III.5}$$

Pour chacun des B blocs, on exécute les tests de collision entre les D_B dislocations du bloc et les D_B dislocations de chacun des 27 blocs voisins. À $D_B = \frac{N}{B}$ constant (i.e. si le nombre de blocs augmente avec le nombre d'objets), le calcul de détection avec grille uniforme est linéaire en nombre d'objets à tester.

L'utilisation des sphères comme volume englobant, quant à elle, ne modifie pas la complexité, mais réduit le coût unitaire de chaque test de collision. L'intersection entre deux sphères est plus facile à tester qu'entre deux segments.

Chaque itération de l'algorithme de fusion de proximité nécessite une détection de collision. La complexité dépend de l'algorithme de détection utilisé, ainsi que du nombre de sous-pas de temps à effectuer. Avec dt le pas de temps considéré, dt_{col} le sous-pas de temps utilisé, et une détection par blocs :

$$C_{\text{Proximite}} = C_{\text{Detection}} \cdot \frac{dt}{dt_{col}} = O(D_B \cdot N \cdot \frac{dt}{dt_{col}}) \tag{III.6}$$

En section III.1.1, on évalue l'ordre de grandeur du nombre de pas de temps à effectuer : $\frac{dt}{dt_{col}} \approx \frac{L}{r_{col}}$. Dans ce cas, la complexité du calcul est :

$$C_{\text{Proximite}} = O(D_B \frac{L}{r_{col}} \cdot N) \tag{III.7}$$

La complexité de l'algorithme dynamique séquentiel dépend essentiellement de l'algorithme de détection utilisé et du nombre de collisions N_{col} à gérer. Si l'algorithme déplace les nœuds à chaque itération, la complexité est :

$$C_{\text{Dynamique}} = C_{\text{Detection globale}} + N_{col} \cdot C_{\text{Detection locale}} \cdot N \tag{III.8}$$

En utilisant les opérations sans déplacement, la complexité se réduit à :

$$C_{\text{Dynamique}} = C_{\text{Detection globale}} + N_{col} \cdot C_{\text{Detection locale}} \tag{III.9}$$

Lorsque l'on utilise une grille uniforme, la complexité de la détection locale $C_{\text{Detection locale}}$ vaut $27{D_B}^2$. On obtient la complexité suivante :

$$C_{\text{Dynamique}} = O(D_B \cdot N + {D_B}^2 \cdot N_{col}) \tag{III.10}$$

L'algorithme parallèle doit effectuer le même nombre d'opérations mais le calcul peut être distribué. Le temps d'exécution dépendra aussi de l'indépendance des collisions.

Le tableau III.1 résume les complexités des algorithmes. Les plus avantageux semblent être les algorithmes dynamiques sans déplacement. Au contraire de l'algorithme de fusion de proximité, ils n'effectuent qu'une seule détection globale. Le deuxième terme de leur complexité dépend du nombre de collisions ayant lieu au cours d'un pas de temps. Nous verrons (sec. III.5.3.4, IV.3.2) qu'en dynamique des dislocations le nombre de collisions par pas de temps est faible, et la complexité de ces algorithmes est dominée par le coût de la détection. Cette observation rend l'intérêt de l'algorithme de gestion parallèle limité.

Algorithme	**Complexité**
Fusion de proximité	$O(D_B \frac{L}{r_{col}} \cdot N)$
Dynamique (avec déplacement)	$O(D_B \cdot N + {D_B}^2 \cdot N_{col} \cdot N)$
Dynamique (sans déplacement)	$O(D_B \cdot N + {D_B}^2 \cdot N_{col})$

TABLE III.1 – Complexité des algorithmes de collision où N est le nombre d'objets à simuler, D_B le nombre d'objets par case de la grille uniforme et N_{col} le nombre de collisions à traiter.

Bilan

L'algorithme de gestion de collision dynamique séquentiel sans déplacement (sec. III.3.1.3) a été choisi pour être intégré dans Optidis. Il permet une bonne performance comparé à l'algorithme de fusion de proximité et génère des résultats plus précis que les algorithmes sans re-détection. Il est plus facile à maintenir que l'algorithme parallèle et fournit des résultats plus précis que l'algorithme précédemment implémenté dans Optidis.

III.4 Intégration dans Optidis

L'objectif des travaux menés sur les algorithmes de collision est d'améliorer la fiabilité de la simulation tout en permettant une bonne performance dans un environnement parallèle et distribué. La comparaison entre différents algorithmes en partie III.3 nous a permis de choisir le plus adapté. L'algorithme de gestion dynamique et séquentiel des collisions avec re-détection et sans déplacement décrit en section III.3.1.3 a donc été choisi pour être intégré dans Optidis.

III.4.1 Implémentation de la détection

La détection des collisions implémentée dans Optidis est une détection dynamique. J'ai mis en place de nouvelles primitives de détection ainsi que différentes optimisations pour détecter de manière fiable et rapide les collisions.

III.4.1.1 Primitives de détection

Les primitives détaillées en annexe B.1 décrivent la manière de calculer la distance entre les objets, et de déterminer le temps de collision. Elles permettent de déterminer si deux objets entrent en collision au cours du pas de temps. On appelle ces primitives pour chaque couple d'objets. Afin de gérer tous les cas qui se présentent à nous, plusieurs primitives ont été implémentées.

La primitive de détection des collisions entre segments représente la situation la plus courante. Lorsque deux segments s'intersectent à proximité de l'une de leurs extrémités, de très petits segments peuvent être générés. Une primitive de collision entre points et segments permet de traiter différemment ce cas pour obtenir une géométrie plus régulière. De même, une primitive de collision entre points a été mise en place. Séparer les primitives permet de gérer les cas particuliers plus facilement. Les primitives de détection utilisent une approche analytique plus rapide que des approches numériques comme la dichotomie par exemple utilisée dans ParaDis [105].

Lorsque deux points se déplacent dans l'espace, il est très peu probable qu'une collision exacte se produise entre ces 2 points. Pourtant, si ces points s'approchent suffisamment l'un de l'autre, nous voudrions qu'une collision soit détectée. Les primitives utilisées ici effectuent des calculs de collision à r_{col} près afin d'éviter les problèmes d'instabilités numériques qui pourraient intervenir, ainsi que pour s'approcher mieux de la réalité physique des dislocations.

III.4.1.2 Optimisations : découpage de l'espace et volumes englobants

Les techniques d'accélération de la détection décrites en section III.1.2 ont été mises en place dans Optidis.

Afin de séparer les objets qui n'entrent pas en collision, l'espace est découpé selon une *grille uniforme*. Ce type de partitionnement de l'espace est adapté aux segments de dislocations car leurs longueurs sont homogènes. Les objets de chaque boite ne doivent pas parcourir plus de la moitié d'une boite au cours d'un pas de temps. Les nœuds peuvent s'éloigner au maximum de $v_{max}\ dt + r_{col}$ de leur boite initiale, avec v_{max} la vitesse maximale des nœuds sur le domaine, dt le pas de temps et r_{col} le rayon de capture. Les extrémités des segments peuvent s'éloigner à une distance maximale $\frac{l_{max}}{2} + v_{max}\ dt + r_{col}$. Ce qui nous donne l'inéquation III.11 à respecter pour que deux segments qui ne sont pas dans des boites voisines de taille W ne puissent pas entrer en collision.

$$\frac{l_{max}}{2} + v_{max}\ dt + r_{col} < \frac{W}{2} \tag{III.11}$$

La figure III.13 illustre la zone atteignable par un segment contenu dans une boite. Lorsque la largeur de la boite W est supérieure au double de l'éloignement maximal, deux objets qui sont dans des boites non voisines ne peuvent pas entrer en collision.

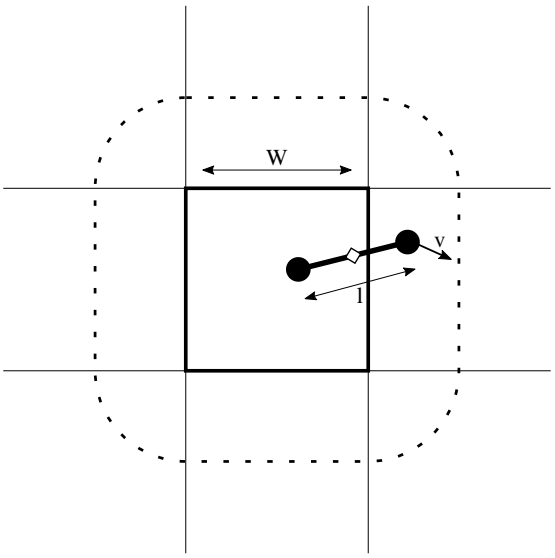


FIGURE III.13 – Éloignement maximal d'un segment. Le segment ne peut pas sortir de la zone en pointillé qui dépend de sa vitesse de déplacement v et de sa longueur L.

La largeur minimale de la boite pour n'avoir à détecter les collisions qu'avec les boites voisines est donc :

$$W_{min} = l_{max} + 2v_{max} + 2r_{col} \tag{III.12}$$

La taille de boite est mise à jour à chaque itération et calculée avant la détection globale de collisions. Elle est choisie légèrement supérieure à W_{min} pour que le domaine comporte un nombre entier de boites dans chaque dimension.

La technique des volumes englobants permet de simplifier les tests de collisions entre les objets. Les volumes utilisés ici sont des **sphères englobantes**, à l'image de techniques utilisant des hiérarchies de sphères [89]. Chaque segment est simplifié par deux sphères centrées sur les quarts de segments de rayon $\frac{l}{4} + r_{col}$, comme l'illustre la figure III.14.

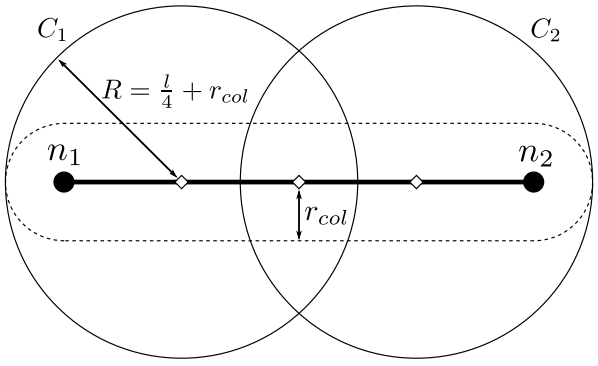


FIGURE III.14 – Simplification d'un segment par deux sphères englobantes.

Lors de la détection de collisions, une première étape détecte la collision entre les sphères englobantes. Ensuite, les points et les segments contenus dans les sphères qui entrent en collision

sont testés. Les sphères C_1 et C_2 qui entrent en collision contiennent respectivement les points n_1 et n_2 ainsi que la moitié des segments s_1 et s_2. Les collisions exactes sont testées en utilisant les primitives de collision dans cet ordre : les collisions entre points sont testées en premier lieu, puis entre points et segments, et enfin entre segments. La première collision exacte détectée est ajoutée à la liste des collisions. Les deux collisions point / segment peuvent être ajoutées en même temps si elles se produisent. Il est important de gérer ces collisions dans cet ordre car les opérations topologiques ont été pensées dans cette logique. Si par exemple une collision entre deux points est détectée, les segments qui leur sont connectés entreront aussi en collision. Cependant, effectuer l'opération topologique de fusion des segments générera une topologie instable avec des nœuds très proches.

Le découpage en grille uniforme peut être utilisé pour la détection de collision entre sphères, et la taille de boite peut être raffinée. Le diamètre des sphères étant de l'ordre de la moitié de la longueur d'un segment, il est possible d'utiliser des boites plus petites, et donc de réduire la complexité du calcul. La largeur minimale d'une boite devient alors :

$$\begin{aligned} W_{min} &= 2r_{max} + 2v_{max} \\ &= l_{max}/2 + 2v_{max} + 2\varepsilon \end{aligned} \tag{III.13}$$

III.4.1.3 Détails d'implémentation

Pour chaque segment, les sphères englobantes sont construites. Le centre $\boldsymbol{c_i}$, la vitesse $\boldsymbol{v_i}$ et le rayon r_i de chaque sphère sont calculés et stockés dans une structure de données triée selon le découpage en grille uniforme. À chaque sphère sont aussi associés les indices du nœud n_i et du segment s_i qu'elle contient. Cette structure de données est comparable à un octree FMM ne contenant que des feuilles. La structure permet d'accéder à la liste des sphères contenues dans une boite de la grille.

La solution choisie pour implémenter ce conteneur consiste en un tableau T trié par indice de Morton indexé par une table de hachage H. Les sphères sont placées dans le tableau T trié selon la z-curve (figure II.2). La table de hachage est utilisée pour associer chaque intervalle I_{ijk} du tableau T à une case de la grille indexée par sa position i, j, k ou son indice de Morton. Pour parcourir les sphères d'une boite i, j, k, il faut interroger la table de hachage pour obtenir l'intervalle I_{ijk} du tableau à parcourir. Le tableau T est en fait implémenté en Structure of Arrays au lieu de Array of Structures comme représenté en figure III.15 afin d'améliorer la performance.

```
struct Sphere_Blocks{
    //Types
    struct Sphere{
        double center_x, center_y, center_z;//c_i : Centre de la sphère
        double v_x, v_y, v_z;               //v_i : Vitesse de déplacement
        double r;                           //r_i : Rayon de la sphère
        NodeIndex node;                     //n_i : Noeud contenu dans la sphère
        SegmentIndex segment;               //s_i : Segment contenu dans la sphère
    };
    struct Indice_Block{                   //Position du bloc dans la grille
        int i,j,k;
    };
    struct Intervalle{                     //Intervalle dans le tableau T
        int min, max;
    }

    //Données
    std::vector< Sphere > T;               //Tableau contenant les sphères
    //Table de hachage contenant les I_ijk
    std::unordered_map< Indice_Block, Intervalle > H;
};
```

FIGURE III.15 – Structure de données contenant les sphères dans la grille uniforme.

L'empreinte mémoire de la liste des sphères est linéaire en nombre de segments et celle de la table de hachage est linéaire en nombre de blocs remplis. La construction de cette structure peut être coûteuse, car les sphères doivent être triées selon leur indice de Morton, ce qui induit une complexité en $O(Nlog(N))$. En pratique, la consommation mémoire de cette structure est largement inférieure à celle de l'arbre FMM utilisé pour le calcul des forces, et son coût de construction, pour le nombre de segments considéré, est surtout dominé par la construction (linéaire) de la table de hachage.

L'algorithme se décompose en deux phases [83]. La première phase (*Broad Phase*) filtre les paires d'objets trop éloignées pour avoir une chance d'entrer en collision. La seconde phase (*Narrow Phase*) s'occupe de détecter exactement les collisions, en extrayant des informations importantes pour la suite, à savoir la position et le temps où a lieu la collision. Les collisions probables détectées lors de la première phase sont enregistrées dans une liste temporaire pour être vérifiées lors de la seconde phase.

Algorithme 3 : Détection de collisions

Données : Grille uniforme $\{H,T\}$ contenant les sphères. (Figure III.15)
Résultat : Liste des collisions C

```
1  C_preselected ⟵ {}
                                                        // Broad phase
2  pour b1 bloc de la grille faire
3  |  pour b2 voisin de b1 faire
4  |  |  pour i1 indice dans la boite b1 faire
5  |  |  |  pour i2 indice dans la boite b2 faire
6  |  |  |  |  si test_collision_spheres(T[i1], T[i2]) alors
7  |  |  |  |  |  C_preselected.Insert({i1, i2})
8  C ⟵ {}                                               // Narrow phase
9  pour {i1, i2} dans C_preselected faire
10 |  C_spheres ⟵ detect_exact_collisions(T[i1], T[i2])
11 |  C.Insert(C_spheres)
```

TABLE III.2

Dans l'algorithme 3, `test_collision_spheres` utilise la primitive de détection de la section B.1.1 pour déterminer si les sphères entrent en collision. `detect_exact_collisions` détecte les collisions exactes entre les objets contenus dans les sphères, comme décrit en section III.4.1.2.

III.4.1.4 Parallélisation

Qu'il s'agisse d'une parallélisation en mémoire partagée ou distribuée, l'algorithme de détection de collision se parallélise facilement. Chaque test de collision étant indépendant, il est possible de distribuer le travail sur plusieurs processeurs de calcul.

La méthode choisie consiste à distribuer les blocs de sphères lors de la *Broad Phase*. À chaque processeur, on assigne un ensemble de blocs à traiter que l'on appellera **blocs locaux**. L'algorithme se déroule de la même manière que précédemment à la différence que chaque processeur ne parcourt que ses blocs locaux. Chaque processeur enregistre une liste locale des collisions présélectionnées lors de la phase de collision entre sphères, pour ensuite effectuer la détection exacte de collision. Les collisions exactes sont stockées dans les listes locales aux threads qui doivent être fusionnées pour obtenir la liste globale des collisions.

Lorsque les données sont distribuées (MPI), les sphères sont distribuées de manière à ce que chaque processus en possède un nombre équivalent à traiter. La répartition des sphères est effectuée de la même manière que pour les objets de la structure de données (section II.4.2.2). Les blocs d'un processus sont contigus selon la z-curve, ce qui permet de garder une cohérence spatiale. Les blocs à la frontière d'un domaine forment une zone de *fantômes* qui doit être dupliquée sur plusieurs processus.

La distribution des segments sur les différents processus peut ne pas correspondre à la distribution des sphères. Certains nœuds ou segments peuvent ne pas être locaux au moment du calcul des collisions exactes. Il faut donc communiquer les données des objets distants référencés dans la liste des collisions présélectionnées.

III.4.2 Implémentation de la gestion des collisions

Les modifications topologiques sont effectuées en utilisant les opérations topologiques sans déplacement (Annexe B.3). Certains détails d'implémentation, notamment sur les plans de glissement, sont décrits dans l'annexe B.2. On utilise l'interface de modification de la topologie de la structure de données décrite en section II.3.1. Les opérations topologiques sont donc séquentialisées entre les processus. Le choix de favoriser la fiabilité au détriment de la performance est par exemple décrit en section III.3.3. Il se justifie par la rareté des collisions au cours d'un pas de temps (voir sec. III.5.3.4). De manière générale, les modifications topologiques sont effectuées en supprimant tous les objets impliqués et en insérant de nouveaux objets. Par exemple lorsque deux points sont fusionnés on supprime les deux points pour en insérer un nouveau au lieu de réutiliser un des deux points. Ne pas réutiliser les objets permet de s'assurer que les objets modifiés ne sont pas référencés par d'autres collisions.

Lors d'une collision entre nœuds, les opérations topologiques sont effectuées selon l'algorithme 4. Les nœuds peuvent fusionner lorsqu'une collision entre segments se produit proche des extrémités des deux segments, ou si un segment rétrécit suffisamment.

Algorithme 4 : Gestion d'une collision point/point

Données : n_1 et n_2 Des nœuds du réseau de dislocations m qui entrent en collision
Données : c des informations sur la collision

```
1 n_new ← m.nodes.insert(new_point_position(n_1,n_2,c))   // Crée un nouveau
    nœud
2 transfert_segments(m,n_1,n_new)      // Transfère segments de n_1 vers n_new
3 m.nodes.erase(n_1)                   // Supprime n_1
4 transfert_segments(m,n_2,n_new)      // Transfère segments de n_2 vers n_new
5 m.nodes.erase(n_2)                   // Supprime n_2
6
7 redetect()                           // Re-détection des collisions
8 projection_plans(n_new)              // Projection de la vitesse sur les
                                          plans de glissement
```

TABLE III.3

Dans l'algorithme 4, le nouveau nœud est créé (ligne 1) avec les propriétés indiquées dans l'annexe B.3, les segments connectés aux nœuds qui entrent en collision sont transférés au nouveau nœud (lignes 2 et 4) et enfin les anciens nœuds sont supprimés (lignes 3 et 5). Le transfert des segments se fait en supprimant les segments connectés à l'ancien nœud pour les réinsérer connectés au nouveau nœud.

Lors d'une collision entre un point et un segment, les opérations topologiques sont effectuées selon l'algorithme 5. Un nœud peut fusionner à un segment si une collision entre segments se produit proche d'une extrémité. Cette situation permet de gérer précisément le cas de la collision planaire. En effet, si les segments se déplacent dans le même plan, la collision a toujours lieu à une extrémité de l'un d'eux.

Algorithme 5 : Gestion d'une collision point/segment

Données : n et s le nœud et le segment du réseau de dislocations m qui entrent en collision

Données : c des informations sur la collision

```
1  n_new ← m.nodes.insert(new_point_position(n,s,c))      // Crée un
                                                             nouveau nœud
2  s_info ← m.segments.get(s)          // Copie les propriétés de s
3  m.segments.insert(n_1,n_new,s_info) // Crée le segment n_1 ↔ n_new
4  m.segments.insert(n_new,n_2,s_info) // Crée le segment n_new ↔ n_2
5  m.segments.erase(s)                 // Supprime s
6
7  transfert_segments(m,n,n_new)       // Transfère les segments de n vers
                                          n_new
8  m.nodes.erase(n)                    // Supprime n
9
10 redetect()                          // Re-détection des collisions
11 projection_plans(n_new)             // Projection de la vitesse sur les
                                          plans de glissement
```

TABLE III.4

Dans l'algorithme 5, le segment est supprimé (ligne 5) et remplacé par deux segments (lignes 3 et 4) en insérant un nouveau nœud (ligne 1) avec les propriétés indiquées dans l'annexe B.3. Les segments connectés au nœud qui entre en collision sont transférés au nouveau nœud (ligne 7), et l'ancien nœud est supprimé (ligne 8).

Lors d'une collision entre segments, les opérations topologiques sont effectuées selon l'algorithme 6. La fusion des segments se produit dans le cas général lorsque deux segments s'intersectent loin de leurs extrémités et qu'ils ne sont pas dans le même plan.

Algorithme 6 : Gestion d'une collision segment/segment

Données : s_1 et s_2 le nœud et le segment du réseau de dislocations m qui entrent en collision

Données : c des informations sur la collision

```
1  n_new ← m.nodes.insert(new_point_position(n,s,c))     // Créé un nouveau nœud
2
3  s1_info ← m.segments.get(s_1)                         // Copie les propriétés de s_1
4  m.segments.insert(s_1.n_1,n_new,s_info)               // Crée le segment s_1.n_1 ↔ n_new
5  m.segments.insert(n_new,s_1.n_2,s_info)               // Crée le segment n_new ↔ s1.n_2
6  m.segments.erase(s_1)                                 // Supprime s_1
7
8  s2_info ← m.segments.get(s_2)                         // Copie les propriétés de s_2
9  m.segments.insert(s_2.n_1,n_new,s_info)               // Crée le segment s_2.n_1 ↔ n_new
10 m.segments.insert(n_new,s_2.n_2,s_info)               // Crée le segment n_new ↔ s2.n_2
11 m.segments.erase(s_2)                                 // Supprime s_2
12
13 redetect()                                            // Re-détection des collisions
14 projection_plans(n_new)                               // Projection de la vitesse sur les
                                                            plans de glissement
```

TABLE III.5

Dans l'algorithme 6, deux segments sont coupés en deux par l'insertion d'un nouveau nœud avec les propriétés indiquées dans l'annexe B.3. Un nouveau nœud est inséré (ligne 1), puis des segments sont ajoutés entre ce nouveau nœud et les quatre extrémités des deux segments (lignes 4, 5, 8 et 9). Les anciens segments sont supprimés (lignes 6 et 11)

Tous les algorithmes présentés terminent par une re-détection des collisions (`redetect()`) et une projection des vitesses dans les plans de glissement (`projection_plans()`). Après chaque modification topologique les objets modifiées doivent être testés pour savoir s'ils entrent en collision avec d'autres objets. Pour cela on effectue une détection locale de collisions. Il est possible d'utiliser le découpage en grille uniforme pour ne tester que les collisions entre objets proches. Pour chaque objet modifié, on détermine la boite à laquelle il appartient selon sa position, puis on utilise le découpage en grille uniforme pour tester la collision avec les objets des boites voisines.

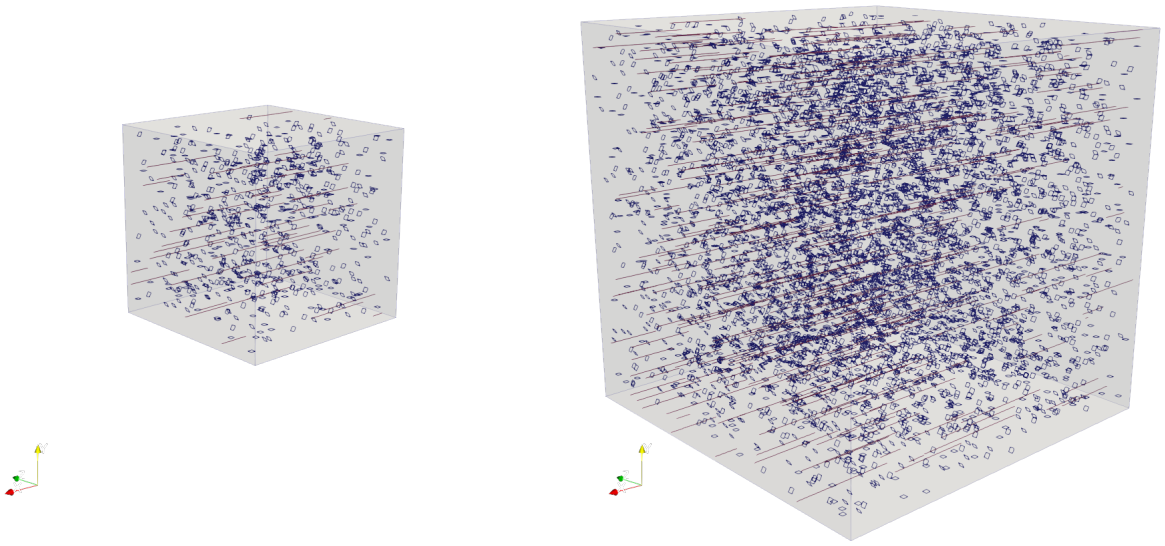
III.5 Performances

Des mesures de performances ont été effectuées sur l'algorithme de collision mis en place en section III.4 afin de vérifier qu'il est utilisable avec un nombre important d'objets, même dans un cadre distribué.

III.5.1 Conditions expérimentales

Les performances de l'algorithme de collision implémenté dans Optidis sont mesurées en exécutant l'algorithme de collision sur un maillage avec un nombre important de dislocations. Les mesures sont effectuées en faisant varier le nombre de segments, ainsi que le nombre de threads et de processus MPI utilisés afin d'observer l'impact de la taille des données et du parallélisme sur la performance. J'ai mesuré les temps d'exécution des différentes parties de l'algorithme, mais aussi le nombre de primitives testées afin de mesurer l'efficacité des différentes méthodes de réduction de la complexité. La machine utilisée est Poincaré [2], déjà décrite en section II.5.2.1.

Le réseau de dislocation est composé de boucles d'irradiation et de sources de Frank-Read dans un grain de Zirconium (HCP). Les boucles peuplent à la fois les plans prismatiques et basal alors que les sources de Frank-Read sont des dislocations contraintes dans le plan basal. Pour faire varier le nombre de segments de dislocations, on augmente la taille du domaine afin de maintenir une densité de dislocations ainsi qu'un nombre d'objets par blocs de la grille uniforme constants. Les dislocations sont en mouvement et la valeur des vitesses provient d'une simulation complète sur un pas de temps. La figure III.16 représente deux exemples de réseaux de dislocations utilisés. Le volume pour $T = 8$ (fig. III.16b) est 8 fois plus gros que pour $T = 1$ (fig. III.16a).



(a) Taille $T = 1$: boite de largeur 20 000.

(b) Taille $T = 8$: boite de largeur 40 000.

FIGURE III.16 – Situation initiale pour la mesure de performances des collisions. Le volume pour $T = 8$ est 8 fois plus gros que pour $T = 1$.

Les paramètres de la simulation en fonction d'une taille souhaitée T sont détaillés dans le ta-

bleau III.6. La taille du réseau varie avec le nombre de processus en *scalabilité faible* ($T = N_{MPI}$). En *scalabilité forte* le réseau a une taille constante $T = 16$.

Largeur boite de Simulation	$20000 \times \sqrt[3]{T}$ Å
Densité de boucles	10^{20} boucles/m^2
Taille des boucles	500 Å
Nombre de lignes	$50 \times T$
Longueur des lignes	de 1500 à 15000 Å
Rayon de capture r_{col}	0.5 Å
dt	1 ns

TABLE III.6 – Paramètres de la simulation pour la mesure de performances des collisions.

III.5.2 Mesures de la complexité et de la distribution des objets

Cette section s'intéresse aux mesures effectuées concernant la quantité de calculs effectués au cours de la détection et la gestion des collisions. Cela permet par exemple de mesurer l'impact des stratégies mises en place pour réduire la complexité de la détection décrites en section III.4.2. Cette section s'intéressera aussi à la répartition de charge.

III.5.2.1 Nombre de sphères et équilibrage de charge

Le nombre de sphères présentes dans la simulation dépend directement du nombre de segments simulés. Cette section vise à comprendre comment les sphères sont reparties entre les différents processus MPI. Les sphères à la frontière des domaines doivent être dupliquées afin d'effectuer le calcul : on les appelle sphères fantômes (*ghosts*). La figure III.17 comptabilise le nombre de sphères locales et fantômes sur chaque processus MPI en fonction du nombre de processus utilisés. La première ligne présente (a) le nombre de sphères locales par processus MPI et (b) la différence relative entre ce nombre et la valeur moyenne sur tous les processus. La seconde ligne présente (c) le nombre de sphères fantômes par processus MPI et (d) leur proportion par rapport aux sphères locales.

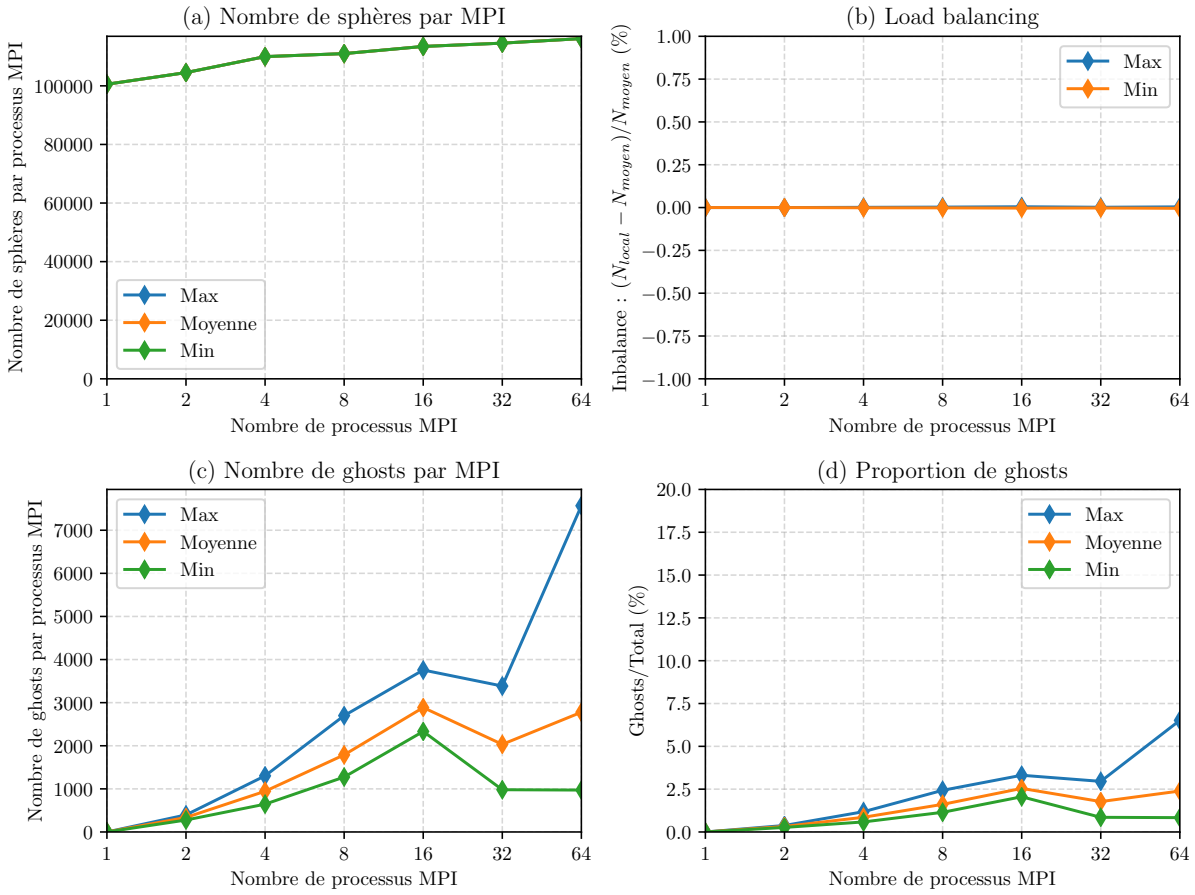


FIGURE III.17 – Décompte du nombre de sphères en fonction du nombre de processus MPI (*scalabilité faible* : $T = N_{MPI}$). La première ligne présente (a) le nombre de sphères locales et (b) la différence relative de ce nombre entre les processus. La seconde ligne présente (c) le nombre de sphères fantômes et (d) leur proportion par rapport aux sphères locales.

La figure III.17a nous montre que le nombre de sphères locales par processus MPI reste relativement constant. Les sphères sont uniformément réparties entre les processus comme le montre la figure III.17b. Cela montre que l'équilibrage de charge fonctionne comme prévu en distribuant les sphères uniformément. Il faut cependant relativiser ce résultat car le réseau est réparti de manière homogène dans tout le domaine.

Sur la deuxième ligne de la figure III.17 (c et d), on remarque que le nombre de sphères fantômes augmente avec le nombre de processus, et que leur répartition est de moins en moins homogène. Ces deux phénomènes s'expliquent par l'augmentation de la surface des interfaces entre les domaines. Lorsque le nombre de domaines grandit, les domaines au centre de la boite de simulation ont plus de surfaces partagées que les domaines au bord de la boite de simulation. L'augmentation du nombre de fantômes augmente le volume de communications MPI. L'homogénéité de leur répartition influe sur l'équilibrage de charge.

III.5.2.2 Décompte des appels aux primitives de test

La complexité de l'algorithme de détection dépend directement du nombre de primitives de test exécutées. La figure III.18 résume le nombre d'appels aux différentes primitives de collision.

Ces figures sont générées à partir de mesures effectuées en *scalabilité faible* avec pour valeur de T (section III.5.1) le nombre de processus. La première ligne présente (a) le nombre de primitives de collisions entre sphères dans la *broad phase*, et (b) le nombre de tests de collision exacts dans la *narrow phase*. Sur la seconde ligne, on affiche le taux de filtration de chaque étape de la détection. Le ratio entre le nombre de paires testées et le nombre de paires total (N^2) (c) nous donne une idée de l'impact du découpage en grille uniforme sur le nombre de paires à tester. Le ratio entre le nombre de paires d'objets testés de manière exacte et le nombre de paires de sphères testées (d) nous donne une idée de l'efficacité du *fast-reject*.

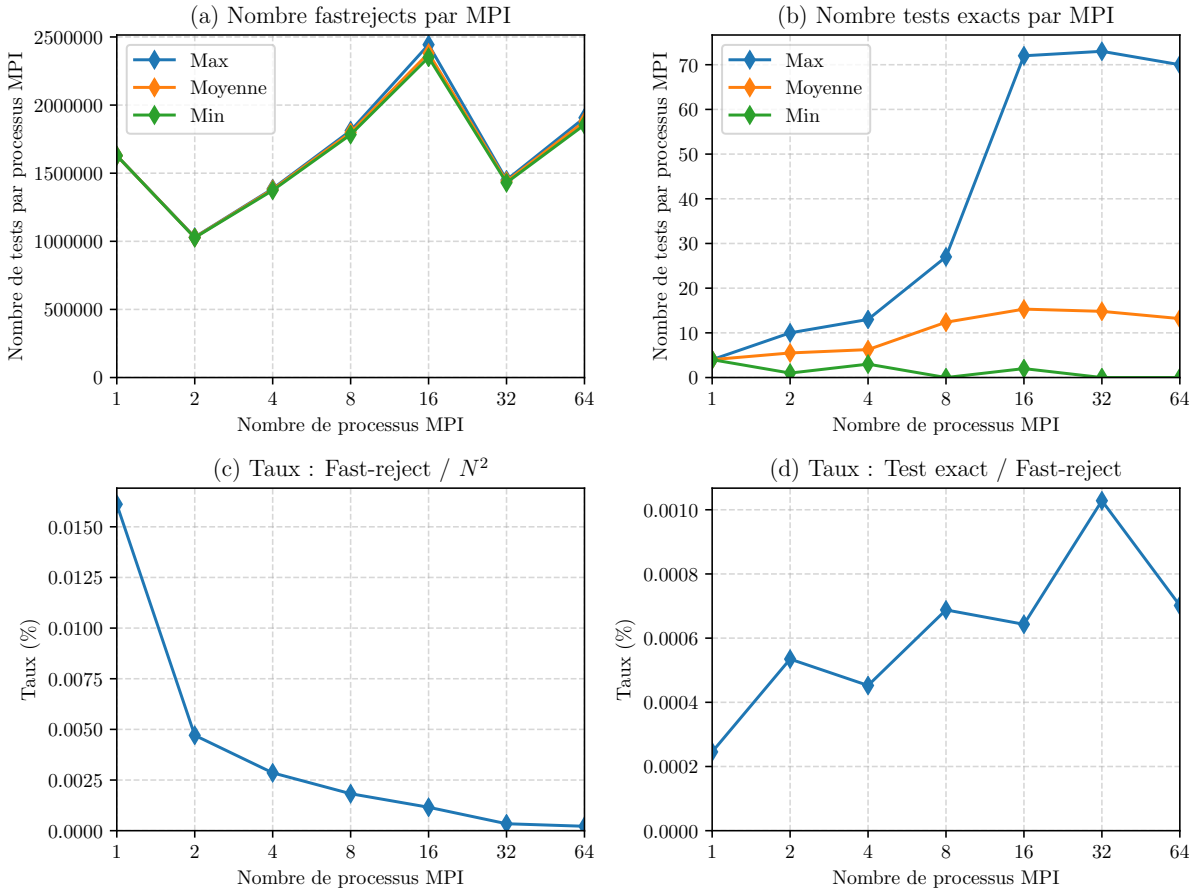


FIGURE III.18 – Décompte du nombre d'appels aux primitives de test et taux de collision (*scalabilité faible* : $T = N_{MPI}$). La première ligne donne le décompte des primitives exécutées, alors que la seconde nous donne le taux de filtration des différentes étapes.

Sur ces figures, on remarque que les deux techniques mises en place pour réduire la complexité de la détection sont efficaces.

Le découpage en grille est efficace car le nombre de tests par processus MPI reste borné lorsque l'on augmente le nombre d'objets. On remarque tout de même une variation périodique due au fait que la hauteur de l'arbre utilisée pour le découpage en grille évolue de manière discrète. Par exemple, dans la figure III.18a la hauteur de l'arbre augmente entre 16 et 32 processus car la condition sur la largeur de boite le permet. La taille des boites est plus petite et la complexité diminue. De plus, la figure III.18c nous montre que l'on effectue beaucoup moins de tests qu'avec l'algorithme naïf en N^2.

L'utilisation des collisions entre sphères comme *fast-reject* permet de réduire significativement le nombre de tests exacts à effectuer, comme nous le montre la figure III.18d. On remarque tout de même dans la figure III.18b que le nombre de tests exacts augmente avec le nombre d'objets, et qu'il varie significativement entre les processus.

III.5.3 Mesure du temps d'exécution

On étudie la performance en mémoire partagée en *scalabilité forte* ($T = 16$). La performance MPI de l'algorithme de collision est étudiée en *scalabilité forte* ($T = 16$) et en *scalabilité faible* ($T = N_{MPI}$).

Les temps d'exécution de plusieurs parties du code ont été mesurés. La phase de *Préparation* est la construction et la distribution des sphères. La *Détection* exécute les différentes primitives de détection de collision. La *Gestion* est l'application des différentes opérations topologiques. Notons que la mesure des performances de partie *Gestion* n'est pas l'objectif de ces mesures (voir section III.5.3.4)

III.5.3.1 Temps d'exécution en mémoire partagée (OpenMP)

On étudie la performance en mémoire partagée en *scalabilité forte*. En effet, le multi-threading vise à apporter une accélération avec une quantité de mémoire constante. On utilise un réseau de dislocations de taille $T = 16$ (section III.5.1). Le nombre total de segments est d'environ 900 000. La figure III.19 présente les temps d'exécution des différentes parties du calcul des collisions. La figure III.19a donne les temps d'exécution bruts de chaque partie de l'algorithme, et la figure III.19b présente le temps total cumulé et la contribution des différentes parties de l'algorithme.

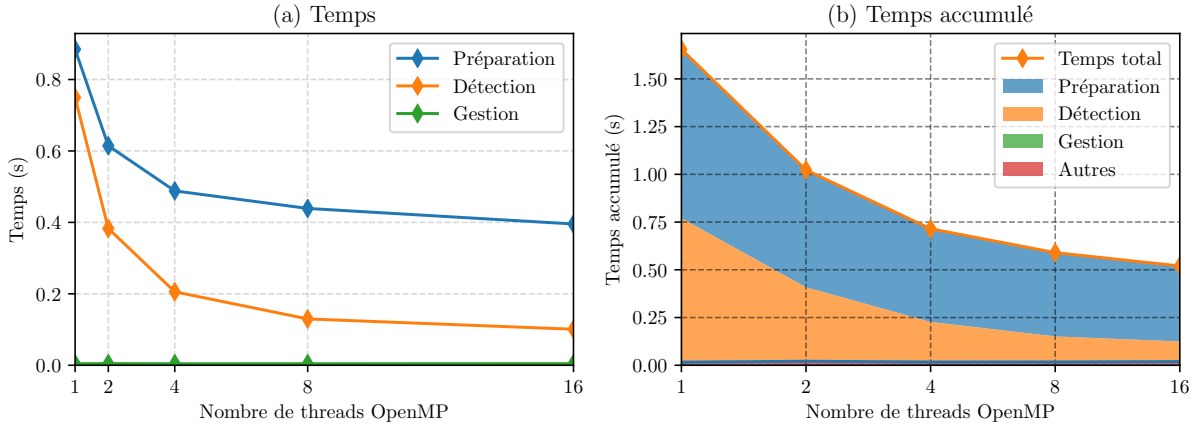


FIGURE III.19 – Temps d'exécution de l'algorithme de collision en fonction du nombre de threads OpenMP (*scalabilité forte* : $T = 16$). Les temps d'exécution sont présentés sous forme de temps bruts (a) et de temps cumulés (b).

Sur la figure III.19a, on remarque que les temps d'exécution des étapes de préparation et de détection forment un plateau lorsque le nombre de processus augmente. Cela s'explique par des parties séquentielles dans ces deux étapes. Certaines parties ne sont pas totalement parallélisables, comme par exemple le tri des sphères par indice de Morton effectué au cours de la préparation. Certaines opérations peuvent être séquentialisées à cause de la contention mémoire,

comme par exemple des copies de tableaux. La figure III.19b nous montre que la préparation est la partie la plus coûteuse, surtout lorsque l'on augmente le nombre de threads. La figure III.20 présente l'efficacité des différentes étapes du calcul des collisions.

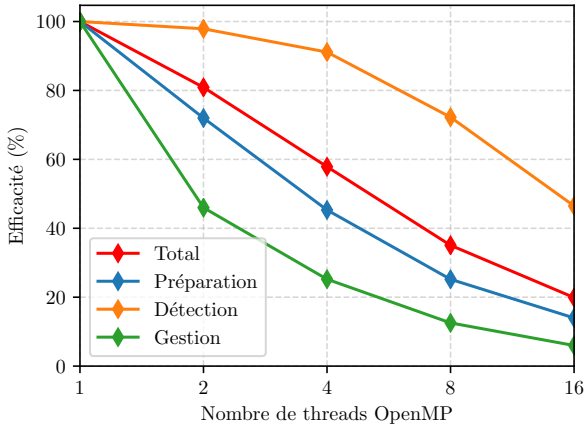


FIGURE III.20 – Efficacité de l'algorithme de collision en fonction du nombre de threads OpenMP (*scalabilité forte*).

L'efficacité globale en mémoire partagée de l'algorithme de collision pour 8 threads (1 nœud NUMA) est d'environ 40% et chute à 20% sur le nœud complet (16 Threads).

III.5.3.2 Temps d'exécution en mémoire distribuée (MPI) en *scalabilité faible*

Pour mesurer la performance MPI en *scalabilité faible* ($T = N_{MPI}$), chaque processus s'exécute sur 8 threads et gère environ 50 000 segments. On remarquera tout de même que le nombre d'objets par processus MPI varie en fonction du nombre de processus MPI. Ce phénomène s'observe sur la figure III.17a, et apparait à cause de la manière de générer les boucles. Les temps d'exécution de l'algorithme et son efficacité en *scalabilité faible* en mémoire distribuée sont présentés respectivement en figures III.21 et III.22.

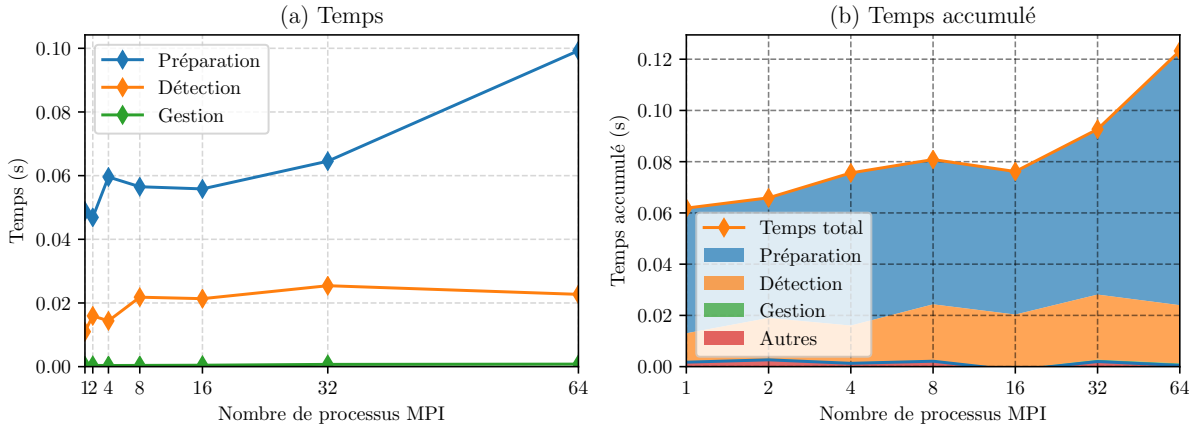


FIGURE III.21 – Temps d'exécution de l'algorithme de collision en fonction du nombre de processus MPI (*scalabilité faible* : $T = N_{MPI}$).

Comme la charge par processeur ne varie pas, les temps d'exécution devraient rester constants

lorsque l'on augmente le nombre de processus MPI. On remarque que le temps de préparation augmente alors que le temps de détection reste relativement constant. Ce phénomène s'explique par le fait que les communications MPI sont principalement effectuées au cours de la préparation. La figure III.22 donne l'efficacité des différentes étapes du calcul des collisions.

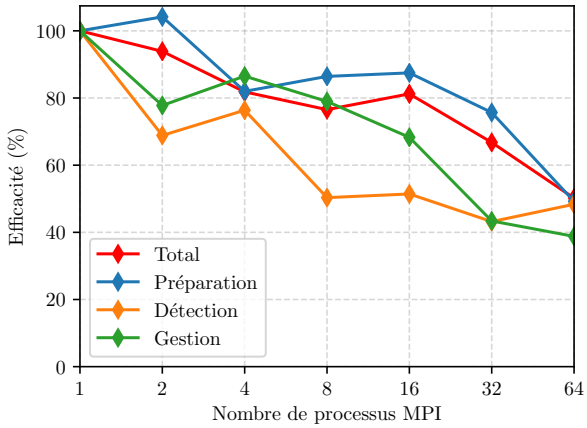


FIGURE III.22 – Efficacité de l'algorithme de collision en fonction du nombre de processus MPI (*scalabilité faible* : $T = N_{MPI}$).

L'efficacité de l'algorithme de collision en *scalabilité faible* atteint environ 70% pour 32 processus MPI, mais chute à 50% pour 64 processus. Toutes les étapes du calcul sont plus efficaces qu'en *scalabilité forte* OpenMP car l'augmentation du nombre de processus augmente aussi la bande passante disponible pour effectuer les opérations à fortes contraintes mémoires, comme les copies de tableaux par exemple. L'efficacité chute tout de même, par exemple à cause des communications MPI utilisées pour échanger les sphères fantômes.

III.5.3.3 Temps d'exécution en mémoire distribuée (MPI) en *scalabilité forte*

Pour mesurer la performance MPI en *scalabilité forte*, chaque processus s'exécute sur 8 threads et le réseau de dislocations correspond à celui utilisé sur 16 nœuds en *scalabilité faible* ($T = 16$). Le nombre total de segments est d'environ 900 000. Les figures III.23 et III.24 illustrent la scalabilité de l'algorithme de collision en *scalabilité forte*.

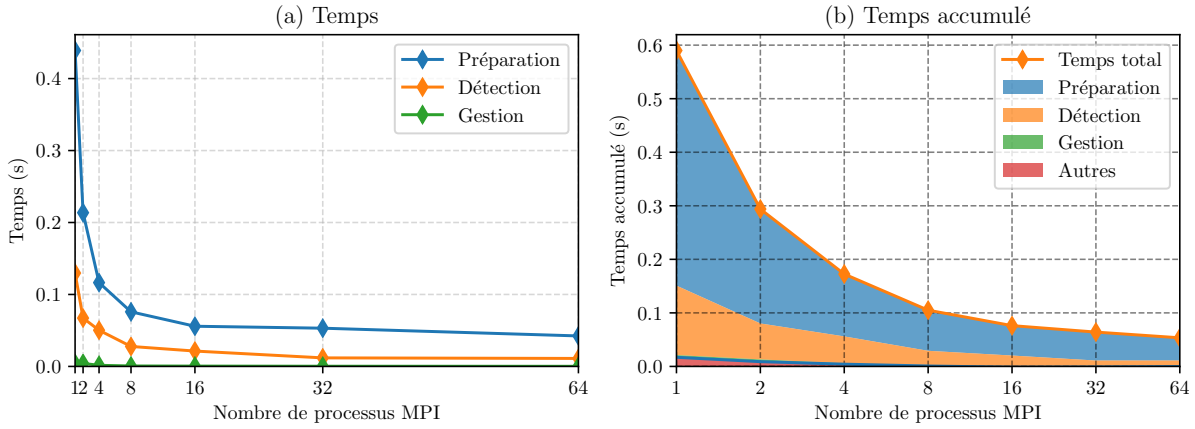


FIGURE III.23 – Temps d'exécution de l'algorithme de collision en fonction du nombre de processus MPI (*scalabilité forte*).

Sur la figure III.23 on observe que le temps d'exécution de la préparation forme un plateau. C'est la conséquence des échanges MPI effectués pour échanger les sphères fantômes durant cette phase. Le temps de préparation prédomine largement dans le temps d'exécution.

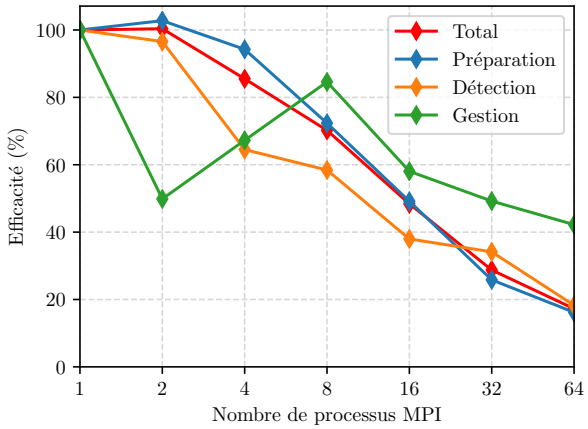


FIGURE III.24 – Efficacité de l'algorithme de collision en fonction du nombre de processus MPI (*scalabilité forte*).

La figure III.24 nous montre que l'efficacité chute rapidement. Ceci laisse présager que l'utilisation de MPI doit plutôt servir à augmenter le nombre de segments simulés plutôt qu'à réduire le temps de calcul. Ce phénomène s'explique par l'augmentation du nombre de sphères fantômes qui accroit le volume de communications MPI. En effet, réduire la taille de domaine de chaque processus sans changer la finesse de la grille augmente la proportion de sphères fantômes.

III.5.3.4 Remarques sur la performance des opérations topologiques

Ces mesures ne permettent pas de mesurer la performance des modifications topologiques lorsque des collisions sont détectées. La collision entre deux dislocations est un événement rare : cela n'arrive pas dans le cas test présenté en section III.5.1. Lorsque l'on augmente la densité de boucle, les collisions arrivent plus souvent. Cependant leur apparition n'est pas consistante, et ne permet pas de donner de tendance en fonction du nombre d'objets présents. Par exemple, lorsque l'on augmente la densité de boucles à 10^{21} boucles/m^3, avec des boucles de taille 100 Å, le nombre de collisions n'augmente pas forcément avec la taille du domaine. On remarque en effet en figure III.25 que le temps de traitement des collisions ne croit pas toujours avec la taille du réseau de dislocations. On ne peut donc pas en déduire une tendance quant à la performance de ces opérations. L'impact de la gestion des collisions sur la performance doit se faire sur plusieurs pas de temps et sera discuté davantage en partie IV sur des simulations complètes.

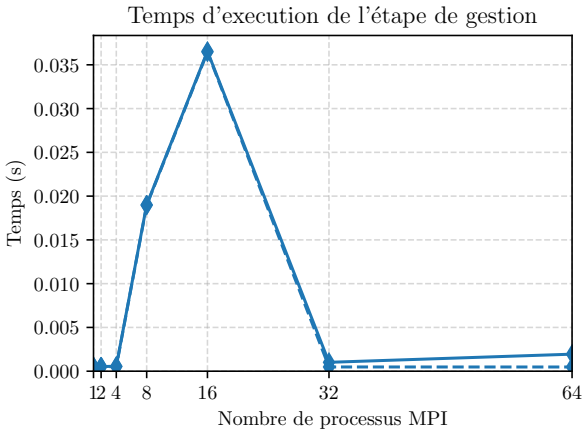


FIGURE III.25 – Temps d'exécution des opérations topologiques de collision en fonction du nombre de processus MPI (*scalabilité faible* : $T = N_{MPI}$).

La performance des opérations topologiques pour cette implémentation qui utilise la nouvelle structure de données peut cependant être extrapolée depuis les résultats présentés en section II.5.4. Ces opérations étant séquentialisées entre tous les processus MPI, on peut s'attendre à ce que le temps d'exécution de la gestion des collisions soit proportionnel au nombre total de collisions à gérer au cours d'un pas de temps, quel que soit le nombre de processus.

III.5.4 Bilan des mesures

Les résultats sur le décompte des sphères et des primitives exécutées nous montrent que les sphères sont bien distribuées et que les différentes méthodes d'optimisation sont efficaces. La méthode de partitionnement en grille uniforme mise en place permet bien de réduire la complexité initialement quadratique à une complexité linéaire lorsque la densité de dislocations reste constante. Le faible nombre de tests exacts nous montre bien que l'utilisation des sphères englobantes est une optimisation efficace.

Les mesures de temps d'exécution nous permettent d'identifier des freins à la scalabilité. Que ce soit en mémoire distribuée ou partagée, la scalabilité forte, est un problème à cause du coût initial de fabrication et de distribution des sphères. Le coût initial de fabrication de la grille pourrait par exemple être réduit en réutilisant la grille uniforme entre les itérations en mettant à jour seulement les objets ayant changé de boite. En mémoire distribuée, l'augmentation du nombre de sphères fantômes pose aussi un problème de performances. Comme pour la distribution des segments, il est possible de réduire le volume de données fantômes en optimisant la décomposition de domaine afin de trouver le bon compromis entre le volume des données partagées entre les processus et tous les autres phénomènes impactés par la décomposition de domaine, comme la répartition de charge et la localité spatiale. Les opérations topologiques séquentielles sont aussi un frein potentiel à la scalabilité de cet algorithme. Mettre en place des opérations topologiques non-collectives dans la structure de données ainsi que l'algorithme dynamique parallèle présenté en section III.3.1.4 permettrait de paralléliser le traitement des collisions.

La scalabilité faible est moins problématique : on atteint environ 50% d'efficacité sur 64 processus MPI, soit 512 processeurs de calcul. Nous verrons en partie IV que le calcul des collisions ne représente pas une partie majeure de la simulation et que ces valeurs sont acceptables.

Bilan

Un nouvel algorithme de collision a été mis en place pour répondre aux problématiques de fiabilité que pose la gestion des collisions dans la dynamique des dislocations. Après une étude de différentes stratégies possibles, un algorithme dynamique a été retenu puis a été intégré dans Optidis. Il permet de détecter et traiter de manière fiable les collisions, même lorsque le pas de temps est grand. Il met en œuvre des techniques d'optimisation de la détection des collisions, s'appuyant sur un partitionnement de l'espace en grille uniforme et sur l'utilisation de sphères englobantes. Sa fiabilité repose sur une re-détection des collisions après chaque modification topologique. La performance a été optimisée en supprimant le besoin de mettre à jour tout le réseau de dislocations à chaque opération topologique avec l'utilisation d'opérations topologiques sans déplacement.

Des mesures ont permis d'observer la performance de cet algorithme et de montrer que les optimisations mises en place permettent de réduire le nombre de primitives calculées. Des mesures de scalabilité nous montrent que la performance parallèle n'est pas parfaite, mais que l'algorithme est capable de fonctionner dans un contexte parallèle et distribué avec des performances acceptables. Pour améliorer la performance et la scalabilité de l'algorithme il reste un certain nombre d'optimisations à mettre en place, comme par exemple l'utilisation d'une grille uniforme qui persiste entre les itérations et la mise en place de l'algorithme de traitement parallèle des collisions.

Partie IV

Validation, performances et simulations massives

Introduction

Cette partie s'intéresse à l'étude de la fiabilité et de la performance d'Optidis pour mettre en avant l'apport des différentes contributions présentées dans les parties précédentes. Dans un premier temps, des comparaisons ont été menées entre Optidis et Numodis pour mesurer l'impact des changements entre les deux codes sur la précision du calcul. Dans un second temps, des mesures de performances ont été effectuées avec des boucles d'irradiation et des sources de Frank-Read sur un faible nombre de pas de temps afin d'observer l'impact des différents paramètres de la simulation sur la performance. Enfin, une simulation complète avec de multiples sources de Frank-Read est présentée pour observer le comportement d'Optidis sur un plus grand nombre de pas de temps.

IV.1 Validation sur des simulations simples

La validité du calcul a été vérifiée en comparant des résultats de simulations simples avec ceux de Numodis. Une comparaison entre Optidis et Numodis permet de vérifier la validité de nos développements. Certaines étapes du calcul sont très différentes entre les deux codes, comme par exemple le calcul de forces qui utilise la FMM ou le calcul de la vitesse qui est déterminée localement, plutôt qu'en résolvant explicitement le système linéaire. La comparaison avec Numodis permet de mettre en avant les différences de précision entre les deux codes. L'impact de différents paramètres a aussi été mesuré afin de vérifier que l'utilisation de la FMM ou de MPI ne nuit pas à la validité du calcul.

IV.1.1 Présentation des cas test

Les simulations choisies pour valider la précision d'Optidis contiennent peu de segments. Cela permet de les exécuter aussi avec Numodis et de comparer les résultats. La source de Frank-Read et la jonction de Lomer-Cottrell sont des cas d'école déjà vérifiés sur Numodis : ces résultats peuvent donc être utilisés comme base de comparaison pour valider ceux retournés par Optidis.

IV.1.1.1 Source de Frank-Read

La source de Frank-Read est un mécanisme à l'origine de la multiplication des dislocations décrit pour la première fois par Frank et Read en 1950 [46]. Une source de Frank-Read est formée d'une ligne de dislocation dont les extrémités sont épinglées. L'évolution d'une source de Frank-Read lorsqu'une contrainte en cisaillement est appliquée est illustrée en figure IV.1. La source est mise en mouvement par la contrainte en cisaillement τ et se courbe peu à peu en s'enroulant autour de ses points d'ancrage. Lorsque les deux parties de la source entrent en contact, les dislocations s'annihilent pour former deux lignes de dislocations distinctes. Une boucle continue de se propager à travers le matériau alors qu'une autre ligne forme une nouvelle source de Frank-Read. La source recommence alors son cycle, formant des boucles successives à l'origine de la multiplication des dislocations.

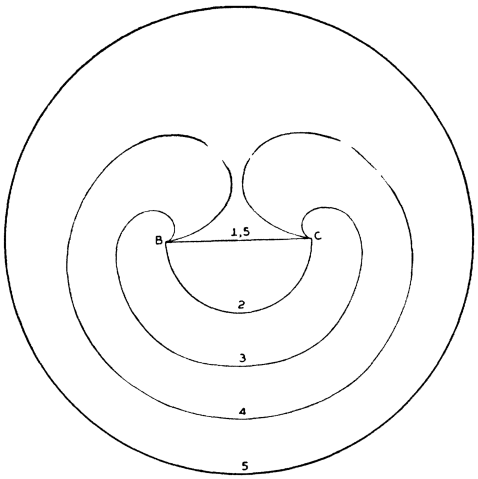


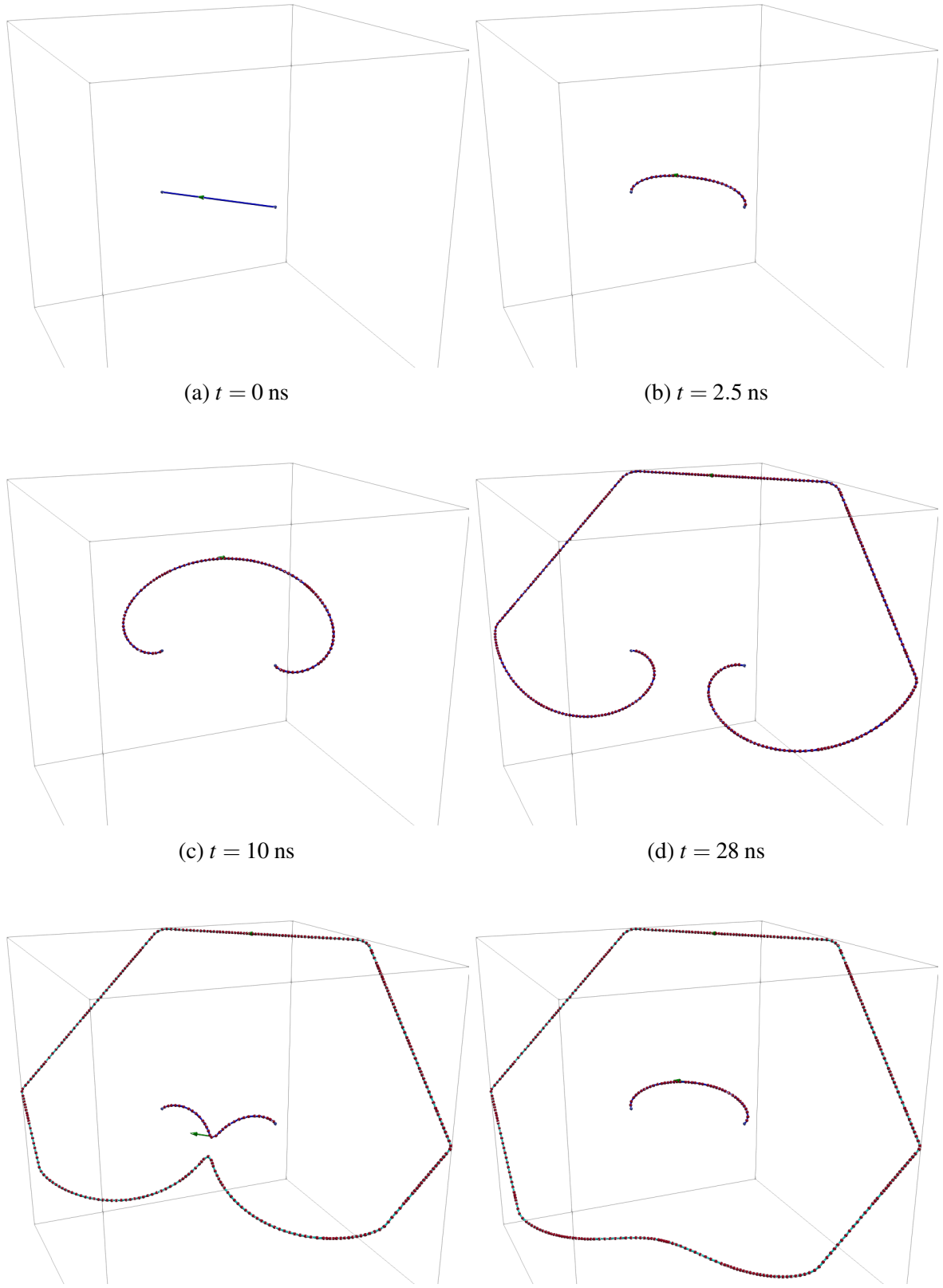
FIGURE IV.1 – Évolution d'une source de Frank-Read [47].

La simulation utilisée consiste en une source de Frank-Read placée au centre d'un grain cubique d'aluminium. Dans sa situation initiale, la source est composée d'un ensemble de segments formant une ligne de dislocation se terminant par deux nœuds immobiles. Une contrainte constante en cisaillement est appliquée au matériau pour activer la source. Les paramètres exacts de la simulation sont décrits dans le tableau IV.1.

Longueur de la source	1 500 Å
Direction	$[\bar{1}\,0\,1]$
Vecteur de Burgers	$[\bar{3}\,0\,3]$
Plan de glissement	$(1\,1\,1)$
Chargement	Contrainte constante en cisaillement selon le plan $(1\,1\,1)$ dans la direction $[\bar{1}\,0\,1]$
Contrainte	180 MPa
Matériau	Aluminium
Taille du grain	4 000 Å
Discrétisation	30 Å $> L_{seg} >$ 75 Å
dt	0.01 ns

TABLE IV.1 – Paramètres de la simulation pour la source de Frank-Read.

Le déroulement de la simulation est illustré par les images provenant du simulateur Numodis en figure IV.2. On peut y observer la source rectiligne initiale (a) qui se déforme sous la contrainte (b et c). La source entre ensuite en contact avec le joint de grain (d), puis les deux parties de la source entrent en contact et s'annihilent (e). Enfin, la boucle formée ainsi qu'une nouvelle source continuent de se propager dans le matériau (f).



(a) $t = 0$ ns (b) $t = 2.5$ ns

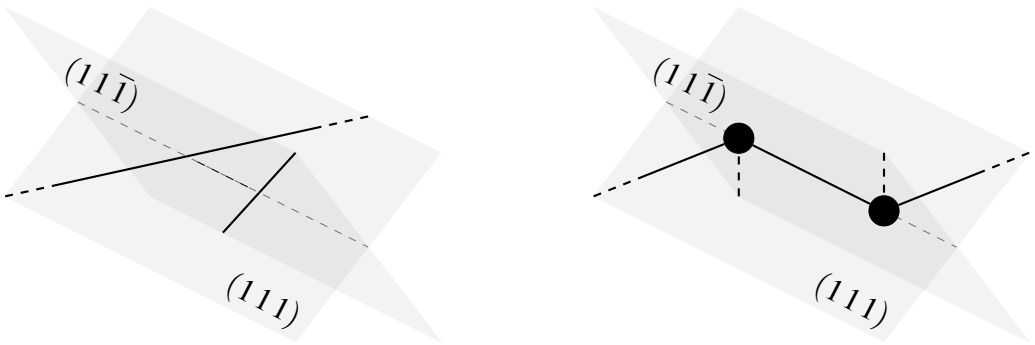
(c) $t = 10$ ns (d) $t = 28$ ns

(e) $t = 31$ ns (f) $t = 35$ ns

FIGURE IV.2 – Source de Frank-Read : déroulement de la simulation dans Numodis.

IV.1.1.2 Lomer-Cottrell

La jonction de Lomer-Cottrell est une jonction sessile dans les matériaux cubiques à faces centrées décrite par W.M. Lomer [77] et A.H. Cottrell [29] dans les années 1950. Ses propriétés sont bien connues, comme par exemple la longueur de la jonction sous contrainte [37]. Ce type de jonctions aussi appelé verrou de Lomer-Cottrell joue un rôle important dans certains phénomènes de durcissement, comme par exemple l'écrouissage. Elle se forme lorsque deux dislocations qui se déplacent dans des plans de glissement différents entrent en contact. Elles fusionnent alors et forment une jonction dont le système de glissement n'est pas actif. La figure IV.3 illustre la topologie d'une jonction de Lomer-Cottrell.



(a) Situation initiale.

(b) Situation à l'équilibre.

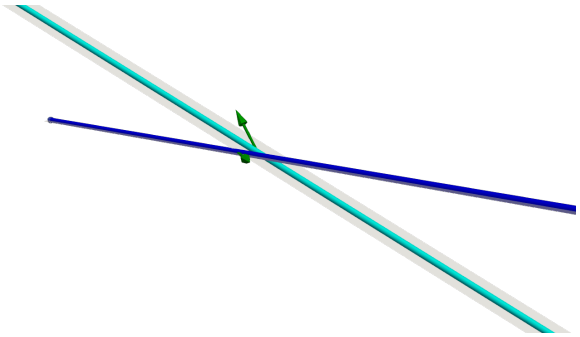
FIGURE IV.3 – Topologie d'une jonction de Lomer-Cottrell. Les deux lignes initiales s'attirent pour former une jonction sessile contenue dans les plans de glissement des deux lignes.

La simulation utilisée contient deux sources de Frank-Read placées respectivement dans les plans $(1\,1\,1)$ et $(1\,1\,\bar{1})$ de manière à s'attirer et former une jonction. Une contrainte de cisaillement constante par morceaux est appliquée : la simulation démarre avec une contrainte imposée nulle jusqu'à obtenir un premier équilibre à $t = 30$ ns, on applique ensuite une contrainte de 150 MPa jusqu'à atteindre un second équilibre à $t = 120$ ns, on applique enfin une contrainte de 200 MPa suffisante pour défaire la jonction. Ce chargement permet de comparer la longueur de la jonction entre Optidis et Numodis à l'équilibre pour 0 et 150 MPa. Les paramètres exacts de la simulation sont décrits dans le tableau IV.2.

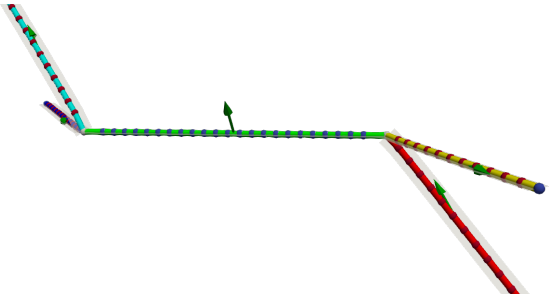
Ligne 1	L=2000 Å, ξ=$[\bar{1}\,2\,1]$, b=$[0\,\bar{3}\,\bar{3}]$, Plan=$(1\,1\,\bar{1})$
Ligne 2	L=2000 Å, ξ=$[\bar{2}\,1\,1]$, b=$[\bar{3}\,0\,3]$, Plan=$(1\,1\,1)$
Chargement	Contrainte constante en cisaillement
	selon le plan $(0\,1\,0)$ dans la direction $[0\,0\,\bar{1}]$
Contrainte	$t = 0$ ns $\rightarrow \tau = 0$ MPa
	$t = 30$ ns $\rightarrow \tau = 150$ MPa
	$t = 120$ ns $\rightarrow \tau = 200$ MPa
Matériau	Aluminium
Discrétisation	30 Å $> L_{seg} >$ 75 Å
dt	0.04 ns

TABLE IV.2 – Paramètres de la simulation pour la jonction de Lomer-Cottrell.

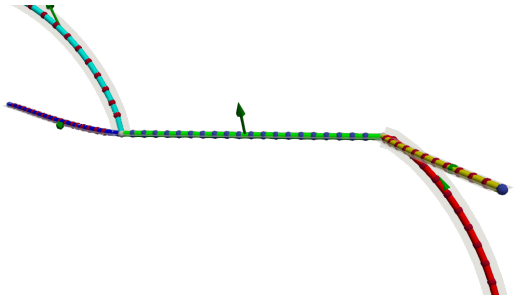
La figure IV.4 présente le déroulement de la simulation sous la forme d'images obtenues avec le simulateur Numodis. On y observe les deux sources de Frank-Read dans leur état initial (a). Les sources se déplacent et fusionnent pour former une jonction qui s'agrandit pour atteindre un équilibre (b). Une fois le premier équilibre atteint, on applique une contrainte à $\tau = 150$ MPa qui rétrécit la jonction jusqu'à atteindre un second équilibre (c). Une contrainte supérieure à la contrainte critique de séparation de la jonction est appliquée et la jonction rétrécit (d) jusqu'à se séparer en deux sources de Frank-Read.



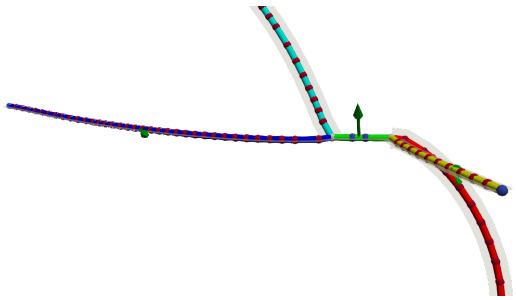
(a) Initial
($t = 0$ ns, $\tau = 0$ MPa).



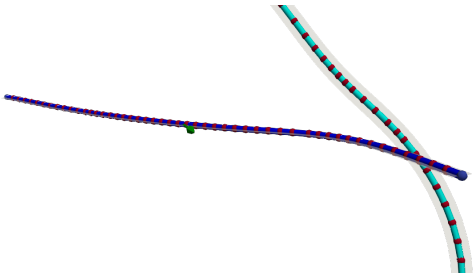
(b) Équilibre 1 : $L = 964$ Å
($t = 30$ ns, $\tau = 0$ MPa).



(c) Équilibre 2 : $L = 802$ Å
($t = 120$ ns, $\tau = 150$ MPa).



(d)
($t = 168$ ns, $\tau = 200$ MPa).



(e)
($t = 175$ ns, $\tau = 200$ MPa).

FIGURE IV.4 – Déroulement de la simulation de la jonction de Lomer-Cottrell. Chaque ligne est colorée d'une couleur différente, les plans de glissement sont représentés par le ruban gris clair autour des dislocations.

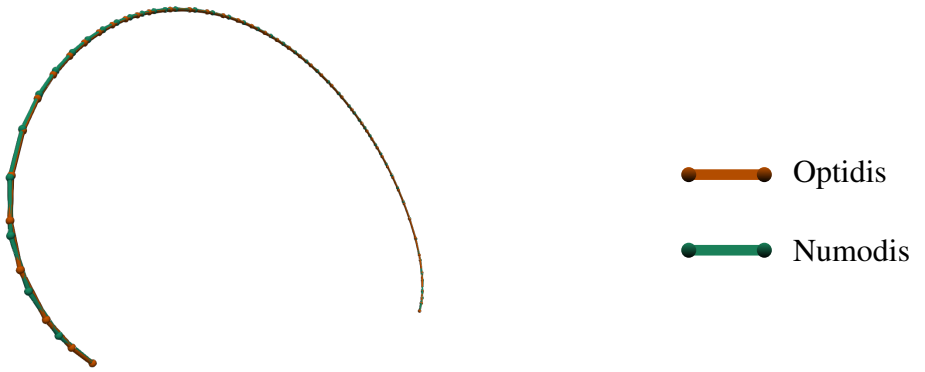
IV.1.2 Analyse des simulations

Des simulations identiques ont été exécutées avec Optidis et Numodis afin de comparer les résultats obtenus. Une comparaison visuelle des géométries obtenues permet d'observer les diffé-

rences entre les deux simulateurs. Les valeurs de déformation plastique ont aussi été comparées afin de quantifier ces différences pour comprendre l'impact des méthodes d'accélération utilisées dans Optidis sur la précision du calcul. L'impact de l'utilisation de la FMM par rapport à un calcul direct des forces a été quantifié, ainsi que celui d'autres paramètres comme l'utilisation de MPI ou du nouvel algorithme de calcul des collisions.

IV.1.2.1 Comparaison avec Numodis

Les simulations présentées en section IV.1.1 ont été exécutées de la même manière sur Optidis et Numodis. Les topologies obtenues avec Numodis et Optidis ont été comparées en superposant les réseaux de dislocation. La figure IV.5 montre la superposition des résultats pour la source de Frank-Read, et la figure IV.6 pour la jonction de Lomer-Cottrell. Le calcul direct est le seul calcul de forces disponible dans Numodis, et a été aussi utilisé dans Optidis afin que les deux simulateurs soient configurés de la même manière.



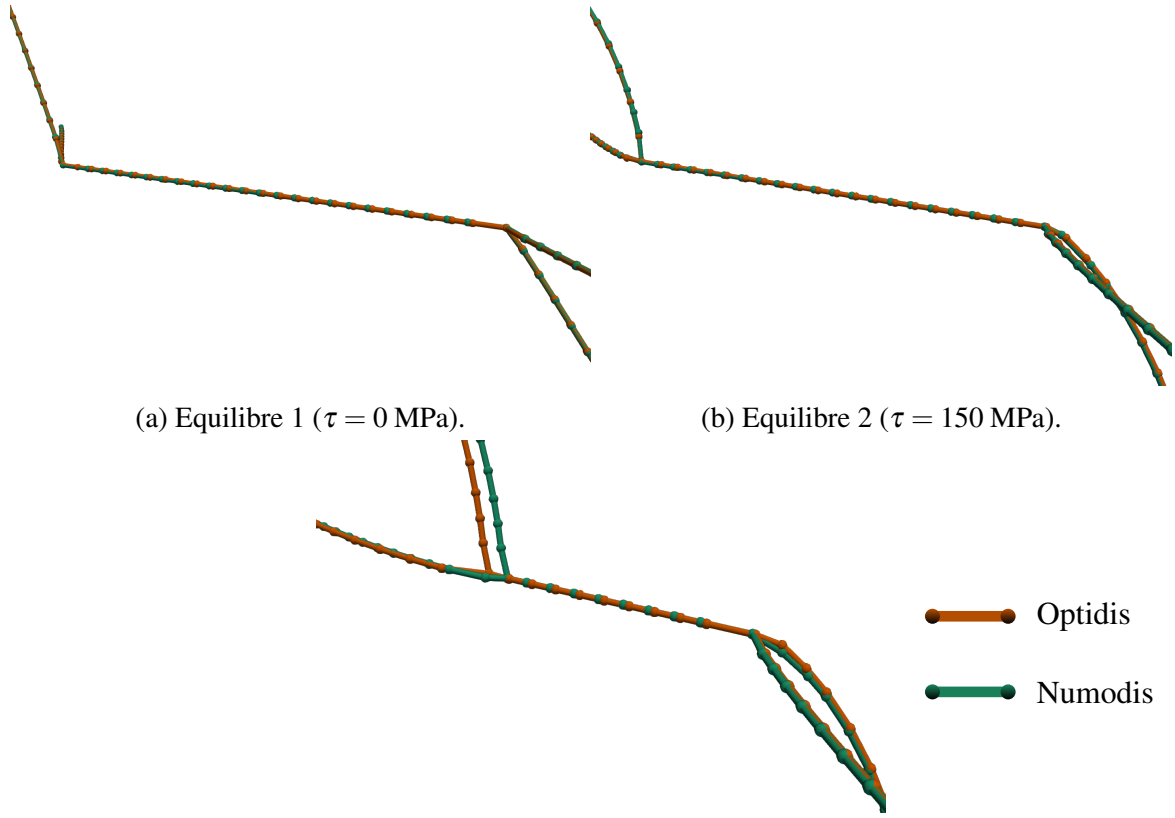
(a) Formation de la boucle ($t = 10$ ns).

(b) Pendant la collision ($t = 31$ ns).

(c) Après la collision ($t = 35$ ns).

FIGURE IV.5 – Superposition des résultats d'Optidis et de Numodis pour la source de Frank-Read à plusieurs étapes de la simulation.

La figure IV.5 nous montre que les géométries générées par Optidis et Numodis sont très similaires. Lors de la croissance de la boucle (a), les lignes de dislocations sont confondues. On remarque tout de même que la discrétisation est différente : les nœuds du réseau de dislocation ne se superposent pas. Une autre différence apparaît lors de la collision entre les deux parties de la boucle (b). L'annihilation des segments est plus rapide dans Numodis car le *splitnode* n'a pas encore été implémenté dans Optidis : lors de la collision (b) la partie haute reste connectée à la basse dans Optidis (orange) alors qu'elles se sont séparées dans Numodis (vert).



(a) Equilibre 1 ($\tau = 0$ MPa).

(b) Equilibre 2 ($\tau = 150$ MPa).

(c) *Unzipping* ($\tau = 200$ MPa).

FIGURE IV.6 – Superposition des résultats d'Optidis et de Numodis pour la jonction de Lomer-Cottrell à plusieurs étapes de la simulation.

Dans la figure IV.6, on observe que les positions d'équilibre (a et b) sont proches. La différence de position des nœuds physiques à l'équilibre entre Numodis et Optidis est inférieure à 0.1 Å, ce qui donne une erreur relative de l'ordre de 10^{-4} sur la longueur de la jonction. Cependant, la vitesse de déplacement des nœuds physiques est plus lente sur Optidis que sur Numodis. En effet, on observe sur la figure IV.6c que les deux branches en haut de la figure ne sont pas superposées. Dans cette figure, ces branches se déplacent vers la droite : Optidis est en retard par rapport à Numodis.

La principale différence entre Numodis et Optidis qui peut expliquer ce retard est le calcul local des vitesses. L'approximation utilisée suppose que les vitesses des nœuds voisins sont similaires, ce qui est une approximation forte pour les nœuds physiques.

Les résultats ont aussi été comparés numériquement en utilisant la déformation plastique totale. La déformation totale du matériau se décompose en une composante plastique et élastique [22]. La déformation élastique est calculée directement à partir de la contrainte appliquée en utilisant la loi de Hooke. La déformation plastique quant à elle découle du déplacement des dislocations dans le matériau. L'incrément de déformation plastique $\boldsymbol{\Delta\epsilon}_{plastique}$ est déterminé à partir de l'aire balayée par les dislocations [18] :

$$\boldsymbol{\Delta\epsilon}_{plastique} = \sum_{s \in segments} \frac{\boldsymbol{b}_s \otimes \boldsymbol{n}_s + \boldsymbol{n}_s \otimes \boldsymbol{b}_s}{2V} \Delta A_s \tag{IV.1}$$

avec V le volume de la boite de simulation, $\boldsymbol{b}_s$ le vecteur de Burgers, $\boldsymbol{n}_s$ la normale au plan de glissement et ΔA_s l'aire balayée par le segment. La déformation plastique $\varepsilon_{plastique}$ est la somme des incréments de déformation sur tous les pas de temps et quantifie le déplacement des dislocations. Cette grandeur macroscopique peut donc être utilisée pour comparer les simulations.

Les déformations plastiques calculées par les deux simulateurs sont présentées sous forme de courbes qui représentent la déformation plastique [1] du matériau en fonction du temps. Ces courbes sont présentées en figure IV.7a pour la source de Frank-Read, et en figure IV.8a pour la jonction de Lomer-Cottrell. Les différences relatives entre les courbes au cours du temps sont présentées en figures IV.7b et IV.8b pour les deux cas-test.

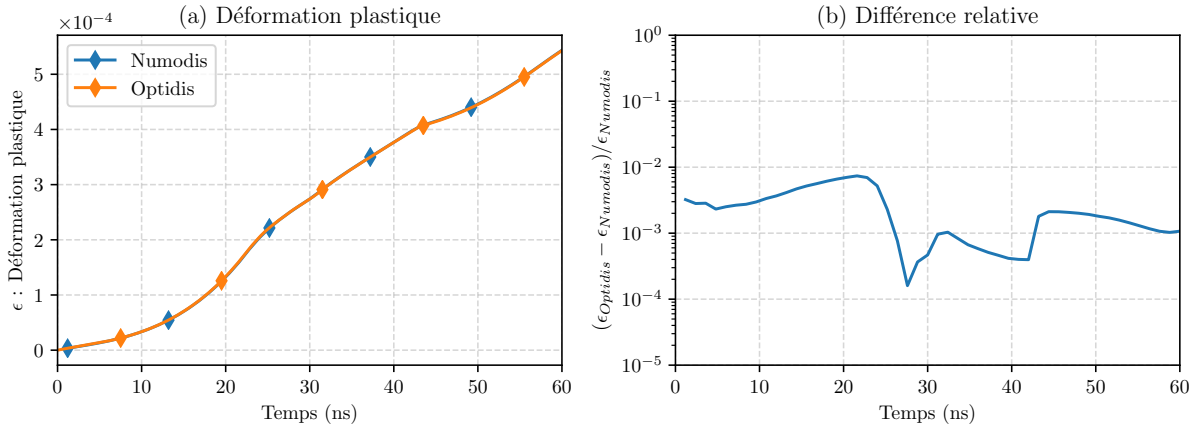


FIGURE IV.7 – Comparaison des déformations plastiques entre Numodis et Optidis pour la source de Frank-Read.

La figure IV.7 confirme que les résultats de simulation sont très proches pour les deux simulateurs. Les courbes sont pratiquement confondues, et l'erreur relative est inférieure à 1%.

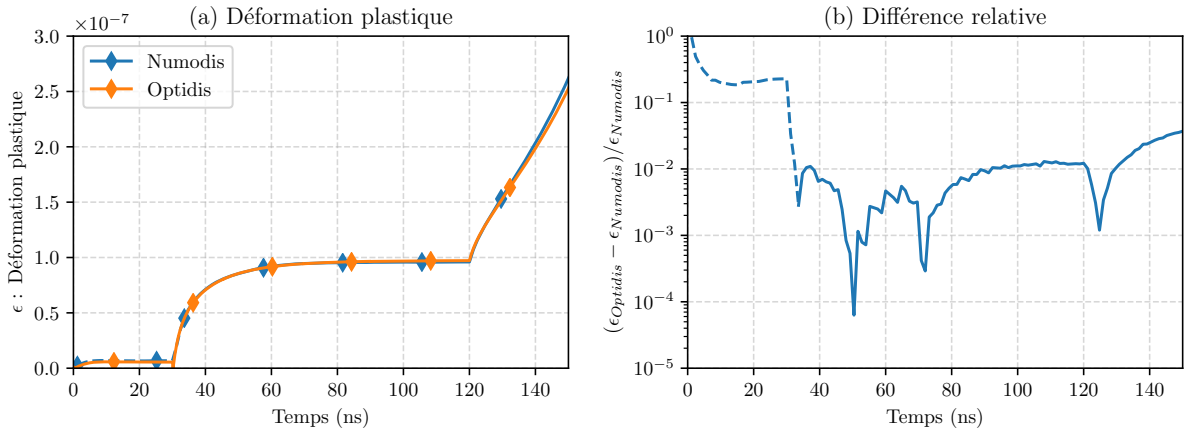


FIGURE IV.8 – Comparaison des déformations plastiques entre Numodis et Optidis pour la jonction de Lomer-Cottrell.

Sur la figure IV.8 on remarque que les différences relatives sont plus importantes pour la jonction de Lomer-Cottrell que pour la source de Frank-Read. Cela s'explique par le retard observé

1. La valeur tracée est le coefficient maximum du tenseur de déformation

visuellement précédemment, mais aussi par les déformations plus faibles et les modifications topologiques qui ont lieu au cours de la simulation. La première partie de la courbe IV.8b (entre 0 et 30ns) n'est pas représentative car la déformation plastique est trop faible. La figure IV.9 nous montre en effet qu'en zoomant sur la courbe de déformation entre 60 et 120ns, on observe des variations périodiques de la déformation sous forme de crans sur la ligne orange. Ces crans correspondent à des temps ou des remaillages ont lieu et sont d'une amplitude d'environ 10^{-10}. En effet, lors d'un remaillage l'aire du triangle remaillé est soustraite, formant alors un cran sur la courbe de déformation plastique. On peut en déduire que la valeur de la déformation plastique dans ce cas n'est précise qu'à 10^{-10} près en erreur absolue à cause des remaillages. Si les valeurs de déformation plastique de l'ordre de 10^{-7} au deuxième équilibre sont suffisamment grandes pour que l'erreur relative avec Numodis soit significative, la déformation inférieure à 10^{-9} avant 30ns rend la première partie de la courbe non-représentative. D'après ces chiffres, l'erreur relative générée par le remaillage est d'environ 10^{-1} au premier équilibre (30ns), et d'environ 10^{-3} au deuxième équilibre (120ns).

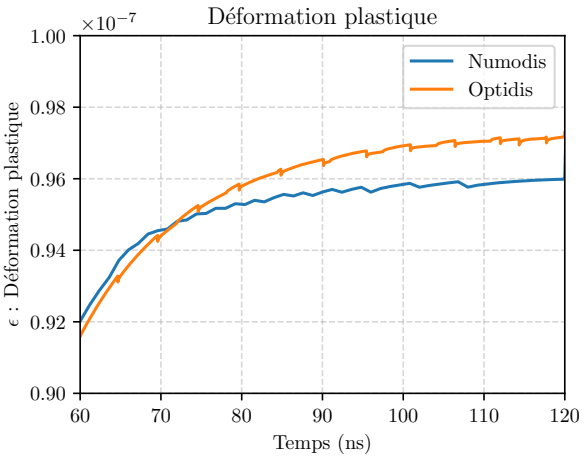


FIGURE IV.9 – Précision du calcul de la déformation plastique pour la jonction de Lomer-Cottrell.

On remarque des crans plus prononcés sur Optidis que NUMODIS car les remaillages sont plus fréquents, surtout autour des positions d'équilibre. En regardant les vitesses des nœuds proches des nœuds physique, on observe que leur vitesse est importante par rapport aux autres nœuds, comme le montre la figure IV.10.

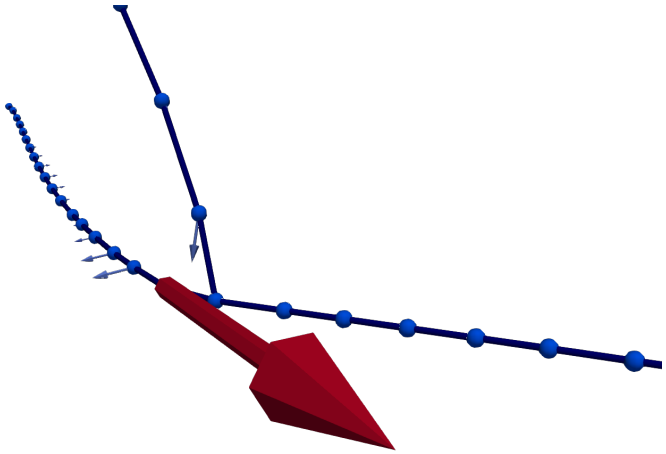


FIGURE IV.10 – Vitesses des nœuds autour d'un nœud physique de la jonction de Lomer-Cottrell ($t = 119$ ns). Le nœud avec la vitesse en rouge est beaucoup plus rapide que les autres, ce qui induit de nombreux remaillages.

Ces comparaisons nous montrent que les résultats obtenus avec Optidis sont proches des résultats de Numodis. Cependant les valeurs de déformation plastique entre les simulateurs peuvent varier en fonction du maillage. Pour la source de Frank-Read, le maillage au moment de la collision diffère à cause du *splitnode*, et pour la jonction de Lomer-Cottrell on observe du bruit dans les valeurs de la déformation plastique à cause du remaillage. Les résultats pour la jonction de Lomer-Cottrell nous indiquent aussi que les vitesses des nœuds physiques ne sont pas identiques entre Numodis et Optidis. Certains nœuds proches des nœuds physiques aussi se comportent différemment : ce sont les fortes vitesses de ces nœuds qui génèrent plus de remaillages dans Optidis. Le calcul local des vitesses ne s'applique donc pas bien aux nœuds physiques et mérite d'être amélioré.

IV.1.2.2 Impact de la FMM

L'utilisation de la méthode des multipôles rapides pour le calcul de forces dans Optidis peut avoir un impact sur la précision du résultat. Afin de quantifier cet impact, des simulations en calcul direct et en calcul FMM utilisant les cas-tests présentés en section IV.1.1 ont été exécutées dans Optidis. Quatre calculs de forces différents sont utilisés. La première simulation calcule les forces de manière directe. Les autres utilisent un calcul FMM uniforme avec un ordre d'interpolation $O = 5$ et une hauteur d'arbre $H = 5$. Pour ces trois simulations FMM, seule la fréquence de calcul du champ lointain $f_{farfield}$ varie. Dans le premier cas, la *FMM complète* est exécutée et le champ lointain est calculé à chaque itération ($f_{farfield} = 1$). Pour le calcul *Cutoff* le champ lointain n'est jamais calculé ($f_{farfield} = \infty$). Enfin, le calcul appelé *FMM partielle* recalcule le champ lointain toutes les 20 itérations.

La figure IV.11 compare les différents types de calcul des forces pour la source de Frank-Read en comparant les courbes de déformation plastique. La même démarche est appliquée pour la jonction de Lomer-Cottrell en figure IV.13. Les figures IV.11b et IV.13b donnent la différence relative entre chaque calcul de force et le calcul direct au cours du temps.

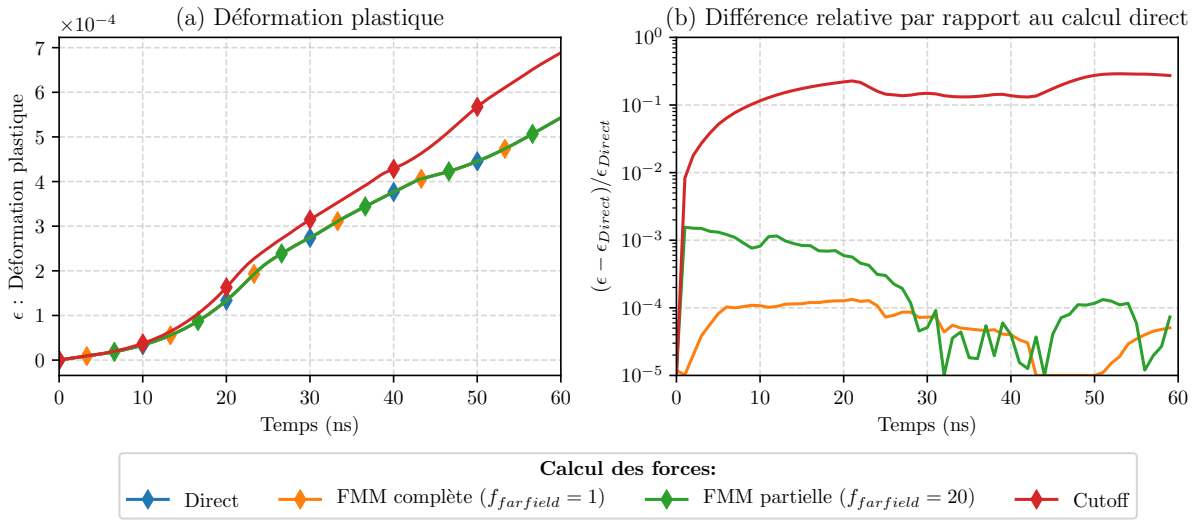


FIGURE IV.11 – Comparaison des déformations plastiques entre les différents types de calcul de forces dans Optidis pour la source de Frank-Read.

La figure IV.11 nous montre que les résultats en FMM et en calcul direct sont proches, à l'exception du calcul en *Cutoff*. Lorsque le champ lointain est calculé, même toutes les 20 itérations, la différence entre les courbes de traction reste inférieure à celle observée entre Numodis et Optidis. Lorsque seul le champ proche est calculé, l'erreur augmente significativement pour atteindre environ 10%. Cette erreur est observable sur la géométrie et se manifeste par un déplacement plus rapide de la boucle comme on peut l'observer en figure IV.12. Dans les autres cas, les lignes sont presque indiscernables.

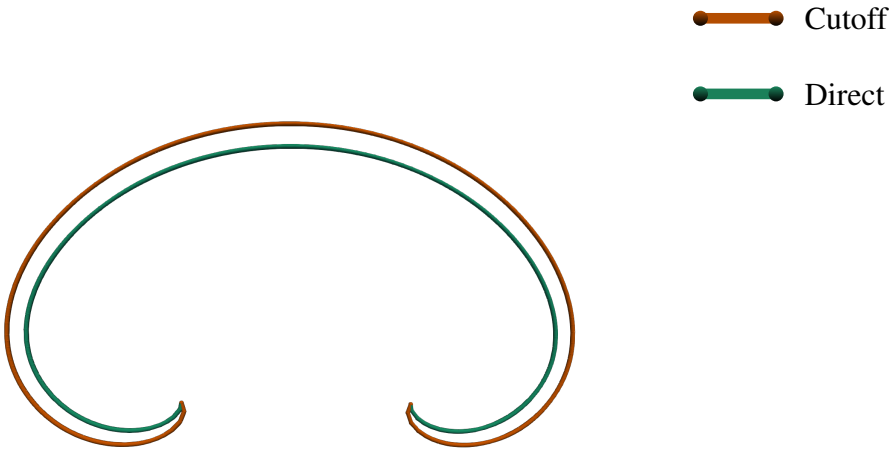


FIGURE IV.12 – Superposition des résultats en calcul direct et cutoff pour la source de Frank-Read.

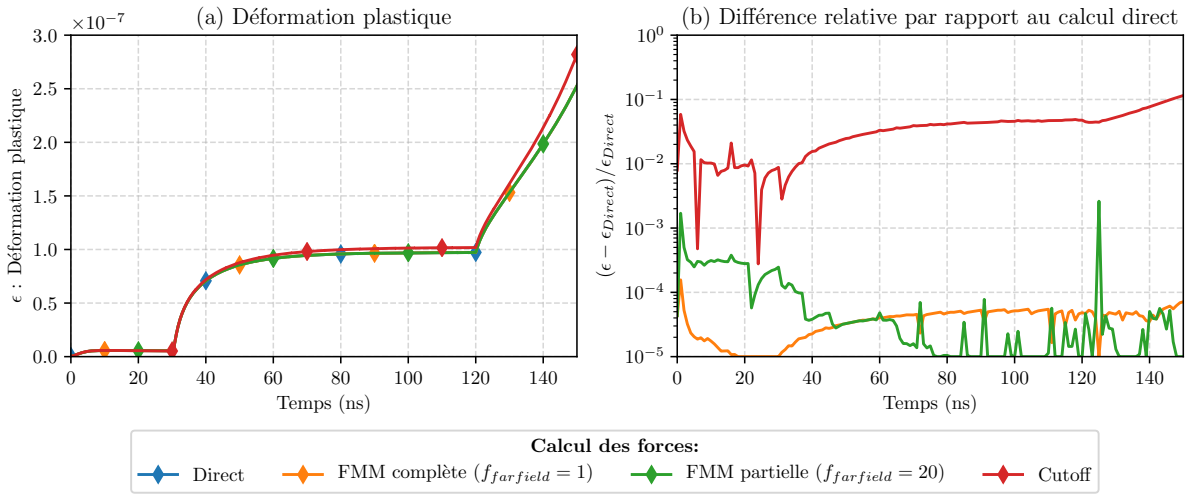


FIGURE IV.13 – Comparaison des déformations plastiques entre les différents types de calcul de forces dans Optidis pour la jonction de Lomer-Cottrell.

La comparaison des résultats pour la jonction de Lomer-Cottrell en figure IV.13 donne des résultats comparables aux résultats précédents. L'utilisation du champ lointain permet d'obtenir une précision supérieure à la différence entre Optidis et Numodis, alors que le calcul en Cutoff nous donne des erreurs plus importantes.

Les résultats nous montrent que dans les cas-tests présentés l'utilisation de la FMM induit une erreur relativement faible par rapport à la différence entre Numodis et Optidis. Lorsque le champ lointain est désactivé totalement, l'erreur est plus importante et induit des différences visibles sur la géométrie. Le calcul du champ lointain augmente substantiellement la précision du calcul, même lorsqu'il n'est pas actualisé à tous les pas de temps.

Les simulations présentées contiennent peu de segments afin d'être capable de les exécuter dans Numodis, l'importance du champ lointain reste donc faible. Dans certaines simulations massives, le champ lointain peut impacter la précision de manière plus importante. Des études ont été menées sur la précision du calcul des forces en FMM pour la dynamique des dislocations dans les travaux de P. Blanchard [16] qui montre que la précision du calcul du champ de force croit selon l'ordre d'interpolation en $O(10^{-Ordre})$. Des éléments de réponse sur l'impact de la fréquence du calcul du champ lointain sont présents dans les travaux de A. Etcheverry [40]. Il utilise une approche adaptative qui calcule par exemple le champ lointain toutes les 15 itérations environ pour des simulations de sources de Frank-Read, malheureusement le pas de temps qu'il utilise n'est pas précisé.

IV.1.2.3 Validité du calcul MPI

L'utilisation de la mémoire distribuée avec MPI peut entrainer des différences sur le résultat final du calcul : des différences dans l'ordre des calculs effectués peuvent changer le résultat obtenu. Afin de vérifier que l'utilisation de MPI ne nuit pas à la précision du calcul les deux simulations présentées en section IV.1.1 ont été exécutées avec et sans l'utilisation de MPI. Cette comparaison a été effectuée pour les deux cas tests et pour deux types de calcul de forces : le calcul direct et le calcul FMM complet. Le calcul séquentiel est exécuté sur un seul processus de calcul, et le calcul MPI sur 8 processus.

La figure IV.14 présente les différences relatives des déformations entre le calcul séquentiel et le calcul MPI en fonction du temps pour les deux cas-tests et pour différents types de calcul de forces.

(a) Source de Frank-Read

(b) Jonction de Lomer-Cottrell

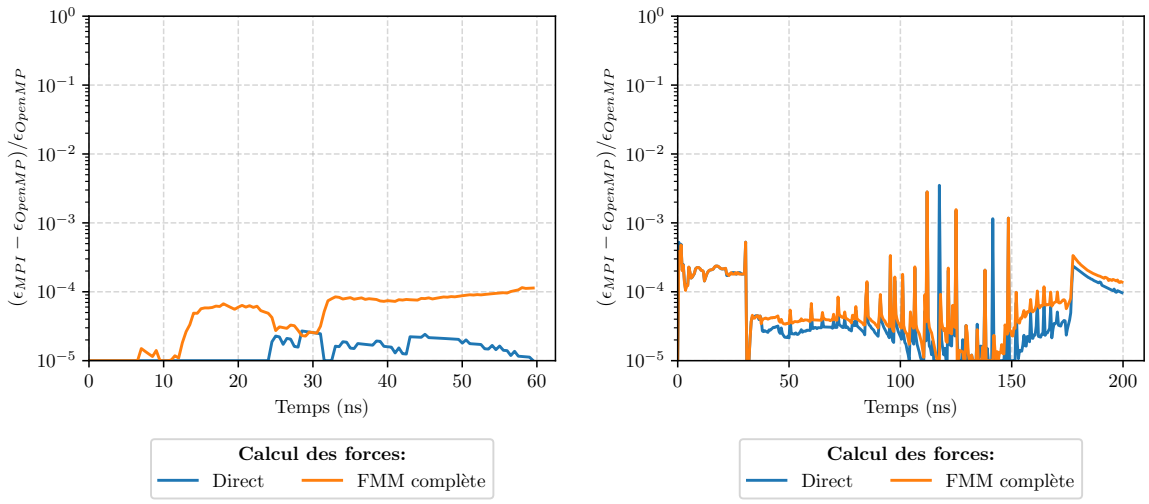


FIGURE IV.14 – Différences relatives des déformations entre les calculs sur 1 et 8 processus MPI.

Dans la figure IV.14a pour la source de Frank-Read, on observe des différences relatives entre 10^{-4} et 10^{-5} qui sont inférieures de plus d'un ordre de grandeur aux erreurs observées précédemment. Pour la jonction de Lomer-Cottrell en figure IV.14b, l'erreur observée est le plus souvent en dessous de 10^{-4} avec des pics lors des remaillages qui peuvent s'effectuer à des pas de temps différents en MPI.

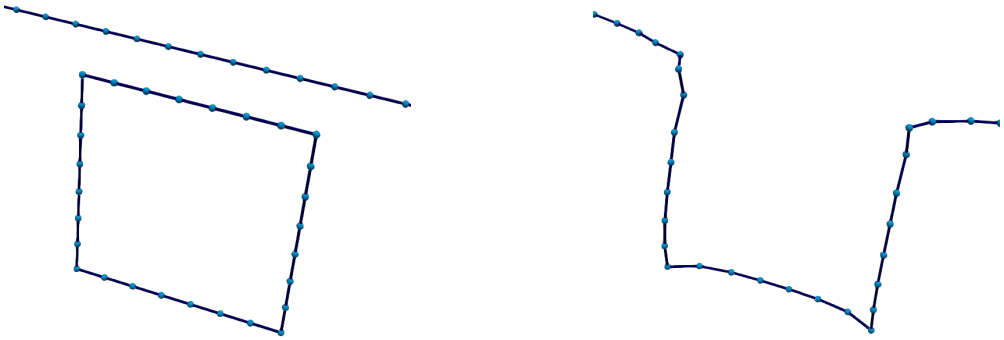
Ces résultats nous permettent de conclure que les erreurs observables en MPI sont inférieures aux erreurs que peut générer la discrétisation. Il faut cependant noter que ces cas-tests sont relativement petits et ne posent pas les problèmes que pourraient poser des sommations sur un grand nombre d'objets.

IV.1.2.4 Vérification de l'algorithme de collision

Nous avons vu en section IV.1.2.3 que les algorithmes qui modifient la topologie peuvent amplifier les erreurs numériques au cours de la simulation. Il est donc important que l'algorithme de collisions ne génère pas de géométrie instable contenant des segments de taille trop hétérogène, ou avec des angles entre segments trop importants.

Pour mettre en avant l'apport du nouvel algorithme de collision (Partie III), nous l'avons comparé avec l'ancienne implémentation dans une simulation complète avec Optidis. Le cas test utilisé est issu d'un article de Serra et Bacon [100] (fig. 3a) et contient une boucle d'irradiation ainsi qu'une ligne de dislocation de même vecteur de Burgers. La ligne se déplace dans le plan contenant un des bords de la boucle d'irradiation et est attirée par cette dernière. Ce cas-test a été choisi pour montrer les améliorations apportées à l'algorithme de collision car l'attraction entre la boucle et la ligne génère des collisions avec des vitesses relatives importantes. La figure IV.15 représente la situation initiale (a) et la situation finale attendue (b), après que la collision

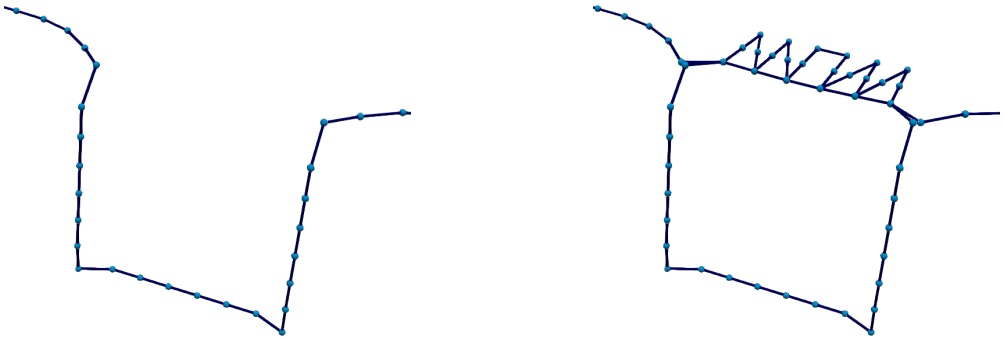
a eu lieu. La ligne de dislocation et le bord de la boucle s'annihilent pour former un cran. La figure IV.15b est le résultat de l'exécution de 10 pas de temps de $dt = 10^{-5}$ ns, et donne le résultat attendu.



(a) $t = 0$ ns (b) $t = 10^{-4}$ ns

FIGURE IV.15 – Absorption d'une boucle par une ligne ($dt = 10^{-5}$ ns, nouvel algorithme). La ligne et la boucle s'attirent et la boucle est absorbée pour former un cran.

Lorsque l'on augmente le pas de temps à $dt = 10^{-4}$ ns, le nouvel algorithme (fig. IV.16a) donne un résultat similaire et le cran se forme correctement. L'ancien algorithme (fig IV.16b) forme des dents de scie et les dislocations ne s'annihilent pas : la géométrie est incorrecte.



(a) Nouvel algorithme. (b) Ancien algorithme.

FIGURE IV.16 – Comparaison du résultat entre l'ancien et le nouvel algorithme ($dt = 10^{-4}$ ns, $t = 10^{-4}$ ns). La dislocation absorbe correctement la boucle avec le nouvel algorithme, alors que l'ancien forme des dents-de-scie.

En plus d'avoir une connectivité incorrecte, ce type de géométrie est instable. On observe par exemple dans la figure IV.17 ce qu'il se passe si l'on continue la simulation après la situation de la figure IV.16b avec l'ancien algorithme. La géométrie est génératrice d'instabilités numériques et le mouvement des nœuds est erratique au niveau de la jonction.

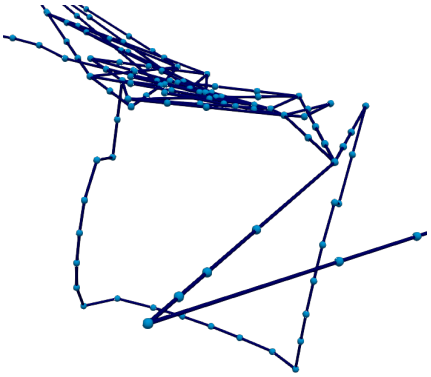


FIGURE IV.17 – Géométrie instable ($dt = 10^{-4}$ ns, $t = 2 \cdot 10^{-4}$ ns). Si l'on continue l'exécution avec l'ancien algorithme après la situation de la figue IV.16b, la géométrie devient de plus en plus instable.

Cette expérience permet de visualiser l'impact de l'amélioration de l'algorithme de collision. Avec ce nouvel algorithme, le pas de temps utilisé au cours de la simulation n'est plus limité par le traitement des collisions, mais bien par la méthode d'intégration. Cela permettra dans le futur d'améliorer la méthode d'intégration pour augmenter le pas de temps utilisé afin de réduire le nombre d'itérations à effectuer pour une simulation donnée.

IV.2 Grandes simulations et mesures de performances

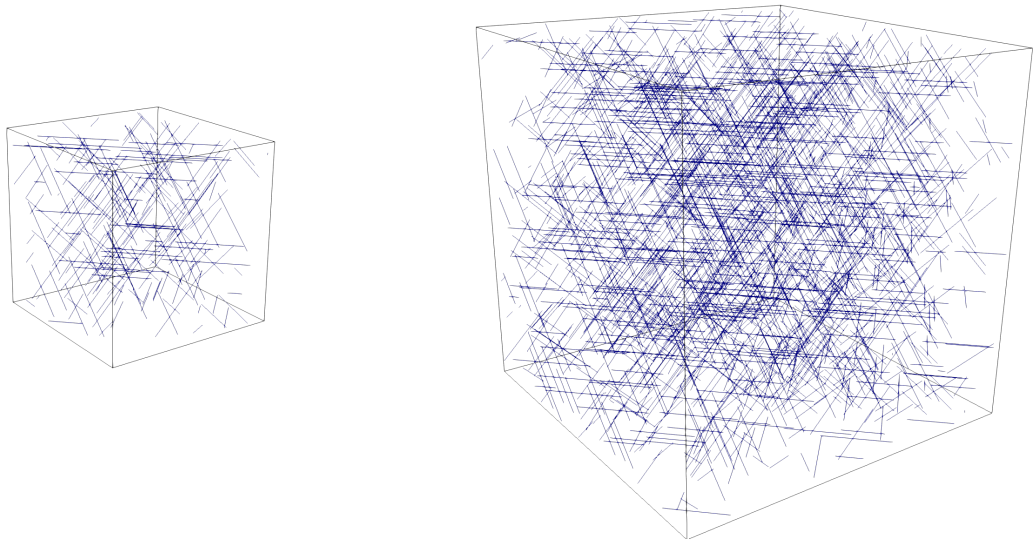
Des mesures de performances ont été effectuées sur des simulations plus imposantes avec un faible nombre de pas de temps mais en faisant varier plusieurs paramètres. Ces résultats permettent de comprendre l'impact des différents paramètres sur la performance, mais pourront aussi servir de base de comparaison pour des améliorations ultérieures.

IV.2.1 Présentation des cas-test

Deux familles de cas-test ont été utilisées pour ces mesures de performance. La première contient uniquement des sources de Frank-Read très mobiles alors que la seconde contient des boucles d'irradiation relativement statiques. Les simulations présentées ont comme point commun d'être des simulations massives avec de nombreux segments. Ces cas-test ont été pensés pour pouvoir faire varier le nombre de segments sans changer les propriétés de la microstructure.

IV.2.1.1 Cas 1 : sources de Frank-Read multiples dans l'aluminium

La première famille de cas-tests implique un grand nombre de sources de Frank-Read dans un grain infini (périodique) d'aluminium. Des sources de Frank-Read sont dispersées dans le grain et habitent tous les 24 systèmes de glissement $\{1\,1\,1\}[\bar{1}\,1\,0]$ de l'aluminium. Le cas test est paramétrisé par la taille des données T : plus T est grand, plus la taille du domaine est grande et le nombre de sources important. Un chargement en contrainte imposée est appliqué afin d'activer les sources de Frank-Read en glissement multiple. La figure IV.18 représente deux exemples de réseaux de dislocations utilisés.



(a) Taille $T = 1$: Domaine de largeur 20 000 Å. (b) Taille $T = 8$: Domaine de largeur 40 000 Å.

FIGURE IV.18 – Situation initiale pour les sources de Frank-Read dans l'aluminium.

Les paramètres de la simulation en fonction d'une taille souhaitée T sont détaillés dans le tableau IV.3.

Largeur boite de Simulation	$20\,000 \times \sqrt[3]{T}$ Å
Nombre de lignes	$240 \times T$
Longueur des lignes	de 5 000 à 10 000 Å
Chargement	Contrainte constante en allongement selon la direction $[0\,0\,1]$
Contrainte	$\sigma_{33} = 100$ MPa
Rayon de capture r_{col}	5 Å
Matériau	Aluminium
Discrétisation	30 Å $> L_{seg} >$ 75 Å
dt	$< 5 \cdot 10^{-4}$ ns (adaptatif)
Periodicité	Périodique

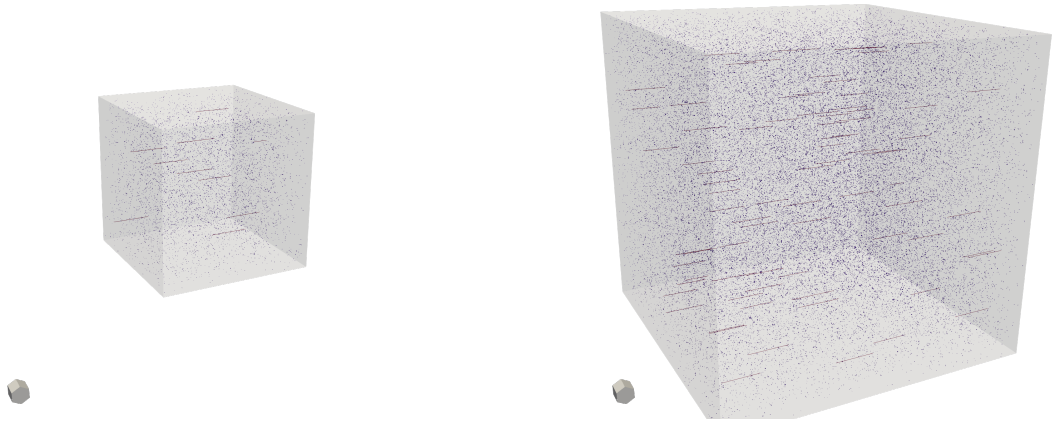
TABLE IV.3 – Paramètres de la simulation pour les sources de Frank-Read dans l'aluminium.

La nature des sources de Frank-Read rend ce cas-test très dynamique. La multiplication des dislocations fait rapidement augmenter le nombre de segments de dislocations au cours du temps, et le mouvement des dislocations génère beaucoup de modifications topologiques, comme les collisions ou le remaillage. Le nombre de segments initial est d'environ $30\,000 \times T$ pour une densité de dislocations d'environ $2 \cdot 10^{13}$ m/m^3.

IV.2.1.2 Cas 2 : Boucles d'irradiation dans le Zirconium

La seconde famille de cas-test contient principalement des boucles d'irradiation dans les plans prismatiques d'un grain de Zirconium. Des sources de Frank-Read se déplacent aussi dans le plan basal. Il s'apparente au cas test utilisé pour les mesures de performances de l'algorithme de

collisions (section III.5.1) mais certains paramètres diffèrent, notamment la taille des boucles et le nombre de sources de Frank-Read. Le cas test est paramétrisé par la taille des données T. Pour faire varier la taille des données, on modifie la taille du domaine de simulation et on change le nombre de sources de Frank-Read tout en maintenant la même densité de boucles d'irradiation. La figure IV.19 représente deux exemples de réseaux de dislocations utilisés. Une coupe dans le plan basal permet de mieux comprendre le cas test en figure IV.20



(a) Taille $T = 1$: domaine de largeur 40 000. (b) Taille $T = 8$: domaine de largeur 80 000.

FIGURE IV.19 – Situation initiale pour les boucles dans le Zirconium.

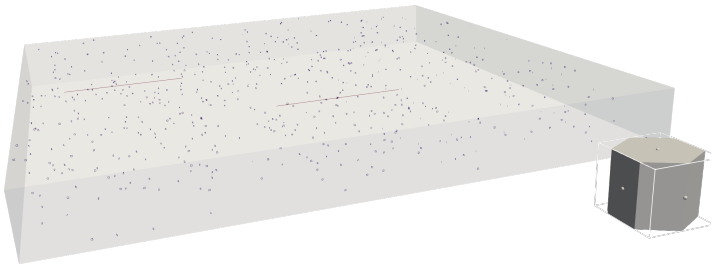


FIGURE IV.20 – Coupe d'épaisseur 4000 Å dans le plan basal pour les boucles dans le Zirconium ($T = 1$).

Les paramètres de la simulation en fonction d'une taille souhaitée T sont détaillés dans le tableau IV.4.

Largeur boite de Simulation	$40\,000 \times \sqrt[3]{T}$ Å
Densité de boucles	10^{20} boucles$/\mathrm{m}^3$
Taille des boucles	100 Å
Nombre de lignes	$10 \times T$
Longueur des lignes	10 000 Å
Chargement	Contrainte constante en cisaillement
Contrainte	$\sigma_{12} = 100$ MPa
Rayon de capture r_{col}	0.5 Å
Matériau	Zirconium
Discrétisation	20 Å $> L_{seg} >$ 50 Å
dt	$< 10^{-4}$ ns (adaptatif)
Périodicité	Non-périodique

TABLE IV.4 – Paramètres de la simulation pour les boucles dans le Zirconium.

Les boucles d'irradiation sont des objets relativement immobiles par rapport aux sources de Frank-Read. Le nombre de segments varie donc très peu et le nombre collisions et de remaillages est faible. Le nombre de segments par processus est d'environ $70\,000 \times T$.

IV.2.2 Étude de performances

Les mesures de performances ont été effectuées sur la machine *Poincaré* [2] présentée en section II.5.2.1. Les cas-tests présentés en section IV.2.1 sont exécutés en faisant varier différents paramètres : la configuration du calcul de force, la taille des données T et le nombre de processus MPI. Les calculs de forces utilisés sont le calcul FMM complet et le calcul en Cutoff décrits en section IV.1.2.2. Ces deux cas représentent deux extrêmes qui permettent de borner les performances si l'on ne souhaite pas calculer le champ lointain à toutes les itérations. Les mesures ont été effectuées avec différentes hauteurs de l'arbre FMM afin de déterminer la valeur la plus efficace. Le nombre de processus et la taille des données varient afin d'effectuer des mesures de scalabilité en scalabilité faible et forte. En scalabilité faible, la taille des données est égale au nombre de processus : $T = N_{proc}$ afin de garder un nombre de segments par processus constant. Les mesures de la scalabilité forte sont effectuées avec une taille de données intermédiaire $T = 8$. Le nombre de processus MPI varie dans l'intervalle [1,64] avec 1 processus par nœud NUMA, soit 8 threads OpenMP par processus, pour une scalabilité jusqu'à 512 cœurs de calcul.

Comme le laissent supposer les résultats sur les s simples, le calcul en *Cutoff* est en réalité trop imprécis pour effectuer des simulations significatives d'un point de vue physique. Mesurer les performances du calcul *Cutoff* est tout de même utile pour deux raisons. Premièrement, réduire le temps du calcul de forces en ne calculant pas le champ lointain permet de mettre en avant la performance des autres étapes de la simulation. Ensuite, les performances mesurées en *Cutoff* sont représentatives des itérations ou le champ lointain n'est pas calculé lorsque l'on choisit de ne pas le recalculer à toutes les itérations.

La principale mesure effectuée est le temps d'exécution. Les temps d'exécution sont collectés pour chaque étape de la simulation, ce qui permet de comprendre l'impact de chaque partie du code sur la performance. Les temps présentés dans les sections suivantes sont les temps cumulés sur 100 itérations.

IV.2.2.1 Résultats généraux

Cette section présente les différents types de graphiques utilisés pour analyser la performance d'Optidis. Je présente une sélection de mesures dans cette partie, les résultats complets sont disponibles en annexe C.1 et dans les sections suivantes. L'exemple utilisé ici est celui des boucles d'irradiation (cas 2), mais les mêmes mesures ont été effectuées sur les sources de Frank-Read.

Les temps d'exécution de la simulation peuvent être présentés sous forme de graphes synthétiques comme celui de la figure IV.21. Ce graphe présente le temps d'exécution de la simulation en fonction du nombre de processus utilisé. Chaque aire sous la courbe représente la contribution d'une étape du calcul au temps d'exécution total. Pour chaque valeur, la hauteur d'arbre optimale est choisie en prenant le meilleur temps d'exécution total.

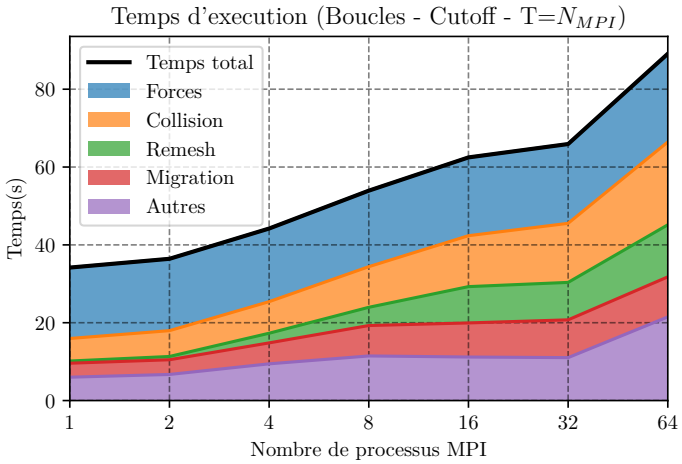


FIGURE IV.21 – Temps d'exécution des différentes étapes du calcul en fonction du nombre de processus pour les boucles d'irradiation (cas 2) en scalabilité faible avec un calcul de forces Cutoff (sur 100 itérations).

En scalabilité faible parfaite, le temps d'exécution devrait rester contant à mesure que le nombre de processus augmente car la charge par processus ne varie pas. Dans nos mesures (fig. IV.21), on remarque que le coût des étapes de collision ou de remaillage augmente en fonction du nombre de processus, ce qui indique une mauvaise scalabilité de ces algorithmes.

En scalabilité forte, le temps de calcul décroit avec le nombre de processus car la taille des données par processus diminue. Il est difficile de représenter de manière lisible les temps d'exécution car ils décroissent rapidement, et qu'une échelle logarithmique ne représente pas bien les contributions de chaque étape de la simulation. Les résultats en scalabilité forte sont donc présentés sous la forme de courbes de temps MPI cumulé en fonction du nombre de processus. Le graphique de la figure IV.22 présente la somme des temps d'exécution de chaque processus MPI : cette grandeur est proportionnelle au nombre d'heures de calcul consommées. Comme les temps de calcul sont similaires sur tous les processus, le temps cumulé est en fait calculé à partir du temps d'un des processus :

$$t_{cumule} = \sum_{i \in MPI} t_i \approx N_{MPI} \times t_0 \tag{IV.2}$$

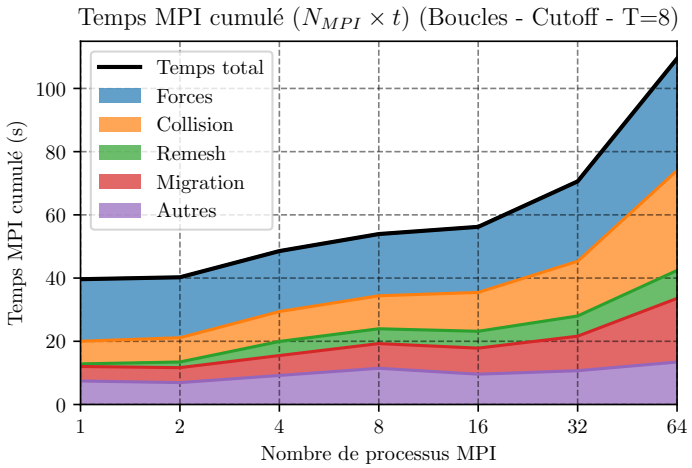


FIGURE IV.22 – Temps MPI cumulé (somme sur les processus) des différentes étapes du calcul pour les boucles d'irradiation (cas 2) en scalabilité faible avec un calcul de forces Cutoff (sur 100 itérations).

En scalabilité forte parfaite, le temps MPI cumulé devrait rester constant à mesure que le nombre de processus MPI augmente, car la charge totale ne varie pas. Dans la figure IV.22 on remarque que certaines étapes du calcul comme la gestion des collisions ou la migration n'ont pas une bonne scalabilité forte.

Les mesures de performances MPI peuvent aussi être présentées en utilisant l'efficacité, comme le montre la figure IV.23. L'efficacité est le ratio entre le temps d'exécution en scalabilité parfaite et le temps d'exécution réel. La hauteur d'arbre qui minimise le temps d'exécution est choisie pour chaque point du graphe.

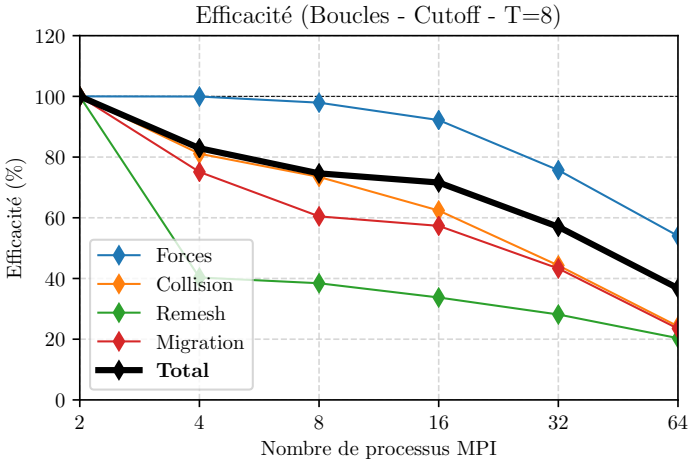


FIGURE IV.23 – Efficacité des différentes étapes du calcul en fonction du nombre de processus pour les boucles d'irradiation (cas 2) en scalabilité forte avec un calcul de forces Cutoff (sur 100 itérations).

Les courbes d'efficacité sont générées en prenant comme temps de référence le temps pour 2 processus MPI : c'est le temps d'exécution sur un nœud de calcul entier. Cette valeur a été

choisie pour que les effets NUMA n'entrent pas en jeu et pour ne pas voir apparaitre des efficacités supérieures à 1, comme pour les mesures de performances de la structure de données. Cette courbe nous permet de voir plus spécifiquement quelles étapes de la simulation ont une mauvaise efficacité.

De ces résultats généraux et de ceux présents en annexe C.1 on peut conclure que certaines étapes de la simulation perdent rapidement en efficacité lorsque l'on augmente le nombre de processus. Les sections suivantes étudient plus en détail les effets de certains paramètres comme le cas test utilisé, les paramètres du calcul des forces ou la taille des données.

IV.2.2.2 Différences entre les cas-tests

Les deux familles de simulations utilisées possèdent des différences. Les sources de Frank-Read sont plus dynamiques et génèrent plus d'opérations topologiques que les boucles d'irradiation, mais c'est surtout sur le calcul de force que la différence est la plus grande. Premièrement, le cas-test contenant les sources de Frank-Read est périodique : le calcul des forces, et notamment celui du champ proche, est plus coûteux. La densité de dislocations est aussi une différence notable entre les deux cas-tests : le cas-test contenant des sources de Frank-Read possède une densité 4 fois plus élevée que celui avec les boucles d'irradiation avec respectivement 3750 Seg $/\mu m^3$ et 1000 Seg $/\mu m^3$.

Ces différences impactent principalement le calcul des forces, surtout le champ proche. La figure IV.24 montre les différences sur les temps d'exécution entre les deux cas-tests en scalabilité faible.

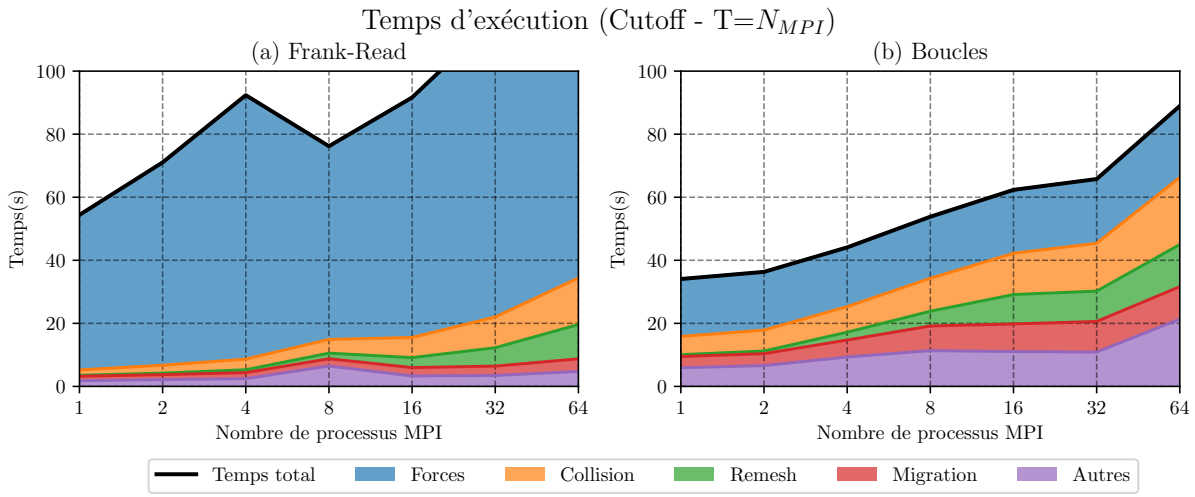


FIGURE IV.24 – Comparaison des temps d'exécution entre les cas-tests en Cutoff et en scalabilité faible (sur 100 itérations).

On remarque en effet que le calcul de forces est bien plus coûteux pour le cas test Frank-Read à cause de la périodicité et de la densité plus forte.

Les autres étapes de la simulation sont moins impactées par le changement de densité entre les cas-tests, comme le montre la figure IV.25 où les performances sont normalisées selon le nombre de segments : les temps d'exécution par segment des autres étapes sont du même ordre de grandeur pour les sources de Frank-Read et les boucles.

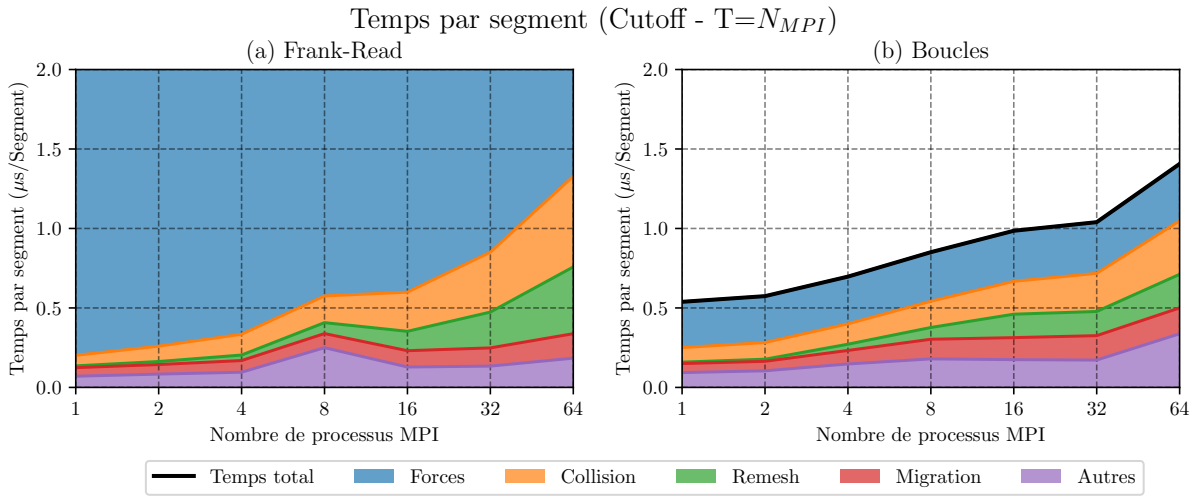


FIGURE IV.25 – Comparaison des temps d'exécution **par segment** entre les cas-tests en Cutoff et en scalabilité faible.

Les étapes hors calcul de forces sont capables de traiter chaque segment en environ 1 µs pour les 2 cas. On remarque tout de même une différence de l'ordre de 25% du temps d'exécution de ces étapes due à la densité plus forte et au caractère plus dynamique. La dynamicité des sources de Frank-Read génère plus de remaillage et de collisions, comme on peut le voir dans la figure IV.26 qui compare le nombre de collisions dans les deux cas tests.

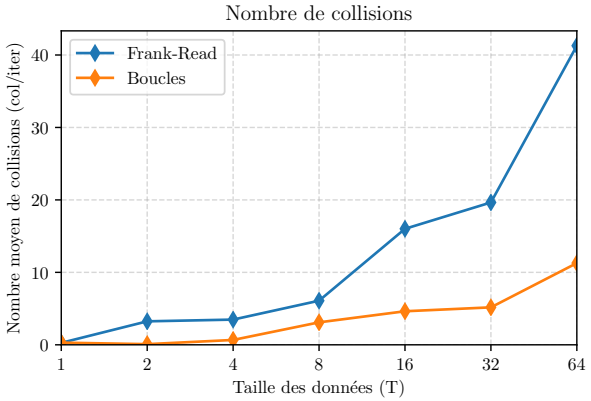


FIGURE IV.26 – Comparaison du nombre moyen de collisions par itération en fonction de la taille des données.

Pour le calcul des forces, une densité de segments plus forte nécessite d'utiliser une hauteur d'arbre plus petite, ce qui augmente le temps de calcul. La figure IV.27 illustre ce phénomène et représente les temps d'exécution en fonction de la taille des données pour différentes hauteurs d'arbres.

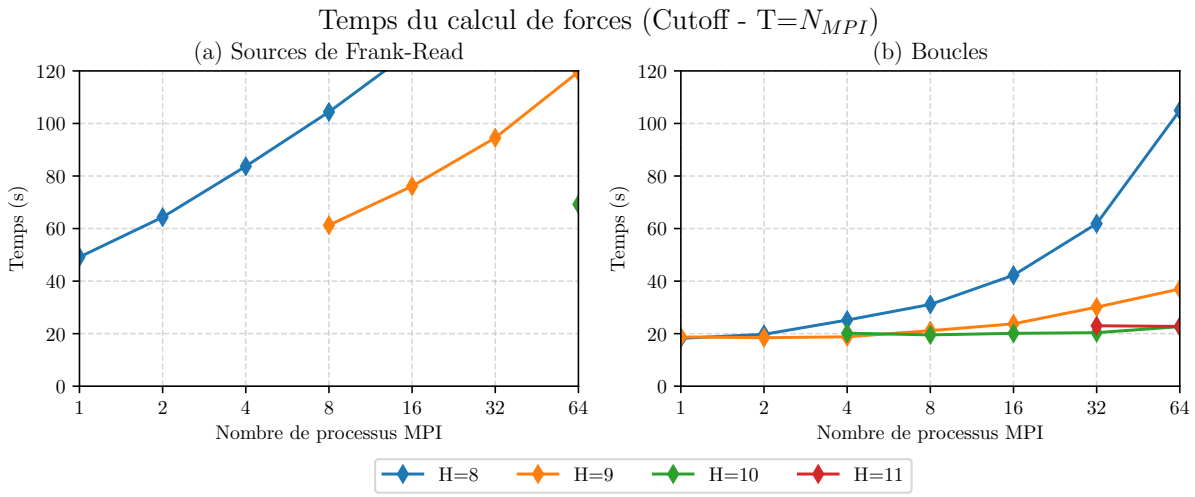


FIGURE IV.27 – Comparaison des temps d'exécution du calcul des forces entre les cas-tests en Cutoff et en scalabilité faible pour différentes hauteurs d'arbre.

Dans Optidis la hauteur d'arbre est limitée par la taille des segments : chaque boite FMM doit être deux fois plus large qu'un segment. Les mesures ont été effectuées pour toutes les hauteurs d'arbres compatibles avec la taille des données dans l'intervalle $H \in [8, 11]$. On remarque sur la figure IV.27 que les hauteurs d'arbre compatibles sont plus petites de manière générale dans le cas Frank-Read. Les hauteurs d'arbres faibles font que le temps de calcul du champ proche augmente rapidement dans le cas Frank-Read. Pour les boucles d'irradiation, les grandes hauteurs d'arbres permettent de réduire le temps de calcul.

Ces résultats nous montrent l'impact de la densité de segments sur la hauteur d'arbre, et par conséquent sur le temps de calcul du champ proche. Pour minimiser le temps de calcul du champ proche, il faut de manière générale choisir la hauteur la plus grande compatible avec le cas-test.

De ces mesures on déduit aussi la hauteur d'arbre optimale à utiliser dans ces cas-tests précis. Pour nos études en scalabilité forte à $T = 8$, la hauteur d'arbre optimale pour le cas-test contenant les sources de Frank-Read est $H = 9$ et $H = 10$ pour les boucles d'irradiation.

IV.2.2.3 Équilibre entre champ proche et champ lointain

En calcul FMM complet, le champ lointain permet une meilleure précision du calcul comme nous avons pu l'observer en section IV.1.2.2, mais augmente le temps d'exécution. Alors que le coût du champ proche diminue en fonction de la hauteur d'arbre, le champ lointain voit son temps de calcul augmenter avec la hauteur d'arbre. Afin d'obtenir la meilleure performance, il faut donc trouver un équilibre entre champ proche et champ lointain. La figure IV.28 compare les temps d'exécution lorsque l'on utilise un calcul de forces Cutoff et FMM uniforme complet. L'ordre d'interpolation pour le champ lointain vaut 5 et a été choisi selon les résultats de P. Blanchard [16] afin d'obtenir un calcul de forces précis à 10^{-4} près.

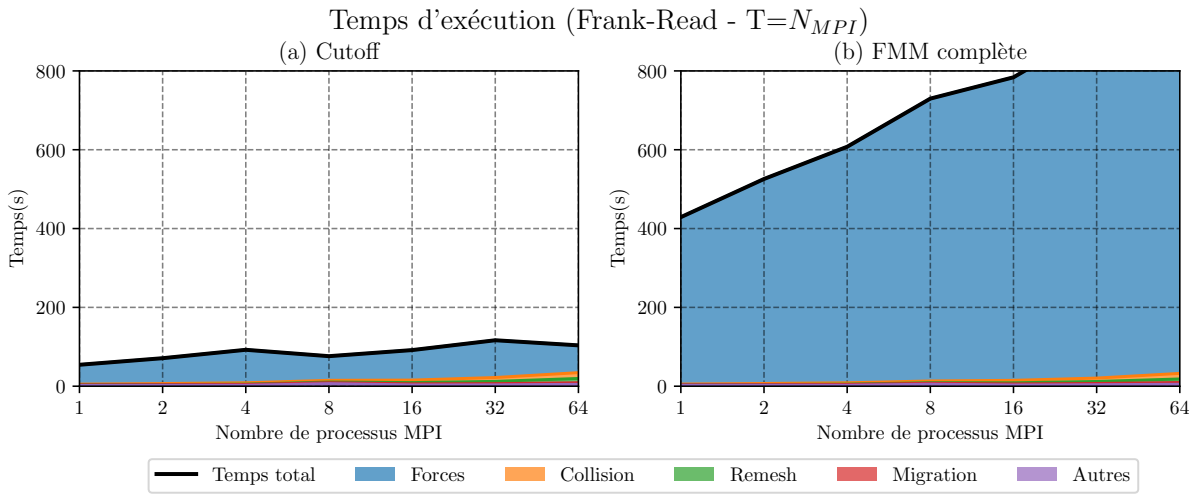


FIGURE IV.28 – Comparaison des temps d'exécution selon le type de calcul des forces pour le cas test Frank-Read (cas 1) en scalabilité faible.

On remarque sur cette figure que le temps de calcul des forces est beaucoup plus important en FMM complète qu'en Cutoff. Le coût des autres étapes de la simulation reste constant, et le calcul des forces domine fortement le temps d'exécution.

Afin d'obtenir les meilleures performances, il faut trouver un équilibre entre champ proche et champ lointain en choisissant une hauteur d'arbre adaptée. La figure IV.29 compare les temps d'exécution du champ proche et du champ lointain selon la hauteur d'arbre utilisée pour les boucles d'irradiation.

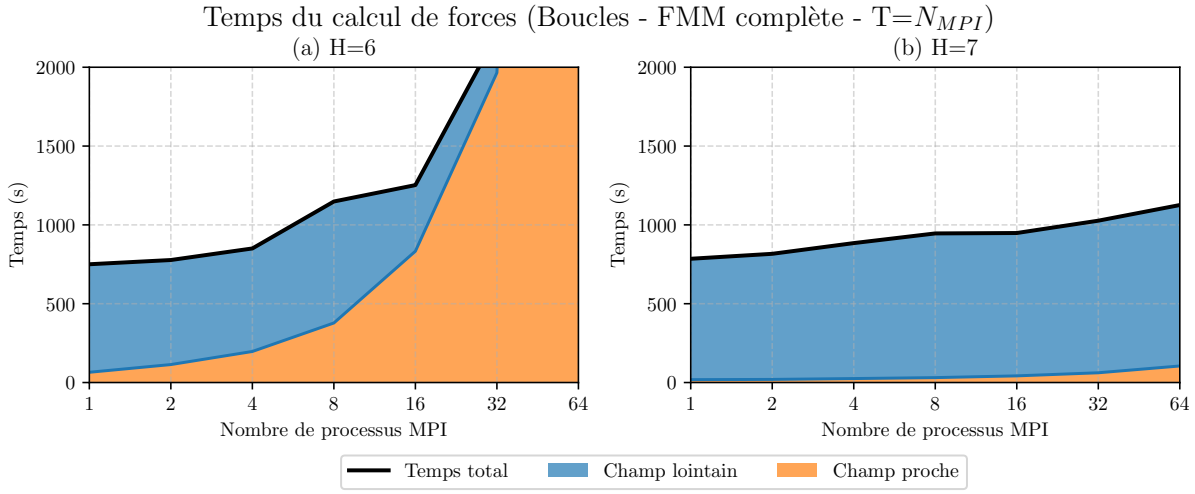


FIGURE IV.29 – Proportion du temps d'exécution entre champ proche et champ lointain pour le cas test des boucles (cas 2) en scalabilité faible.

Pour une faible hauteur d'arbre, le temps de calcul du champ proche augmente rapidement pour dominer le temps de calcul lorsque la taille des données augmente. Lorsque l'on augmente la hauteur d'arbre, le champ proche reste négligeable par rapport au champ lointain.

Afin de choisir la hauteur d'arbre optimale, la figure IV.30 trace les temps d'exécution du calcul de forces en fonction de la taille des données pour les deux cas-tests et pour différentes hauteurs d'arbres.

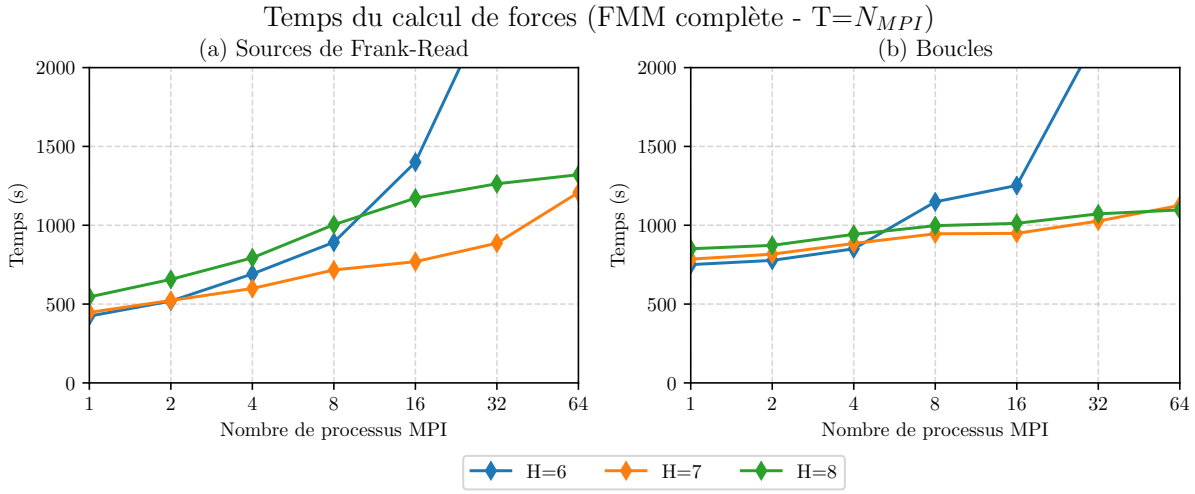


FIGURE IV.30 – Temps d'exécution du calcul des forces en scalabilité faible pour différentes hauteurs d'arbres.

Pour les deux cas-tests, la hauteur d'arbre optimale pour presque toutes les tailles de données est $H = 7$. En dessous, le champ proche prend le dessus pour de grandes tailles de données. Au-dessus le champ lointain est plus coûteux.

Ces résultats nous montrent que le temps de calcul du champ lointain est une part très importante du temps de calcul en FMM complète. Les hauteurs d'arbres optimales sont aussi plus petites qu'en Cutoff. Pour nos études en scalabilité forte à $T = 8$, nous choisirons une hauteur d'arbre $H = 7$.

Des solutions existent pour limiter l'impact du calcul du champ lointain sur la performance au prix d'une perte de précision. Il est possible de changer la fréquence de calcul du champ lointain, ou de changer l'ordre d'interpolation. Les deux types de calculs de force présentés sont deux extrêmes en termes de fréquence de calcul du champ lointain, et il est possible de choisir des valeurs intermédiaires afin de trouver un meilleur équilibre entre le champ proche et le champ lointain. Comme le calcul de champ lointain est environ 10 à 20 fois plus coûteux que le champ proche, ne pas calculer le champ lointain à toutes les itérations peut accélérer énormément le calcul. L'ordre d'interpolation détermine le temps de calcul du champ lointain, mais aussi la précision du calcul. Il est donc possible de l'ajuster pour obtenir un bon compromis entre temps de calcul et précision.

IV.2.2.4 Étude de scalabilité

Lorsque le nombre de processus utilisés pour le calcul augmente, l'efficacité du calcul diminue. Dans cette section, nous verrons comment le simulateur Optidis se comporte lorsque l'on augmente le nombre de processus MPI. L'efficacité a été mesurée en scalabilité faible et en scalabilité forte pour observer l'impact de la taille des données sur les performances.

Les résultats diffèrent en scalabilité faible et en scalabilité forte. Par exemple, la figure IV.31 compare les efficacités obtenues en scalabilité faible et forte pour le cas-test contenant les boucles d'irradiation en utilisant le calcul de forces Cutoff.

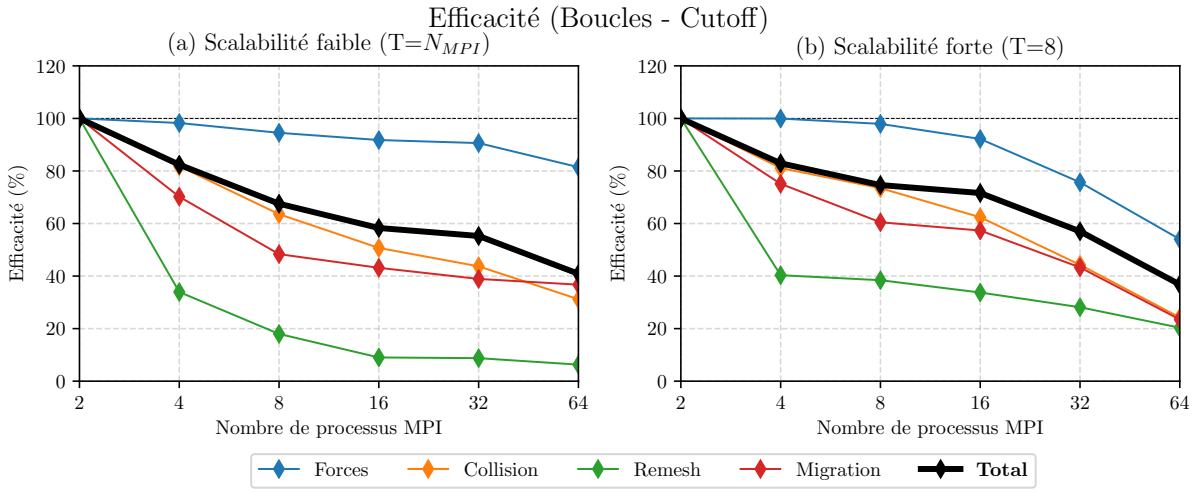


FIGURE IV.31 – Efficacité des différentes étapes du calcul pour les boucles (cas 2) en Cutoff.

L'efficacité totale décroit rapidement pour atteindre 60% pour 32 processus MPI à la fois en scalabilité faible et forte. Si la scalabilité totale est comparable, certaines étapes comme le calcul de forces se comportent très différemment selon le type de scalabilité.

En observant l'efficacité des différentes étapes de la simulation, on constate deux comportements distincts. Pour mettre en avant ces comportements, la figure IV.32 compare les efficacités en scalabilité faible et forte de deux étapes de la simulation. La mesure de référence choisie pour l'efficacité est le temps d'exécution pour 8 processus MPI pour utiliser la même mesure pour toutes les courbes. De cette manière, il est possible de distinguer l'impact du nombre de processus de l'impact de la taille des données.

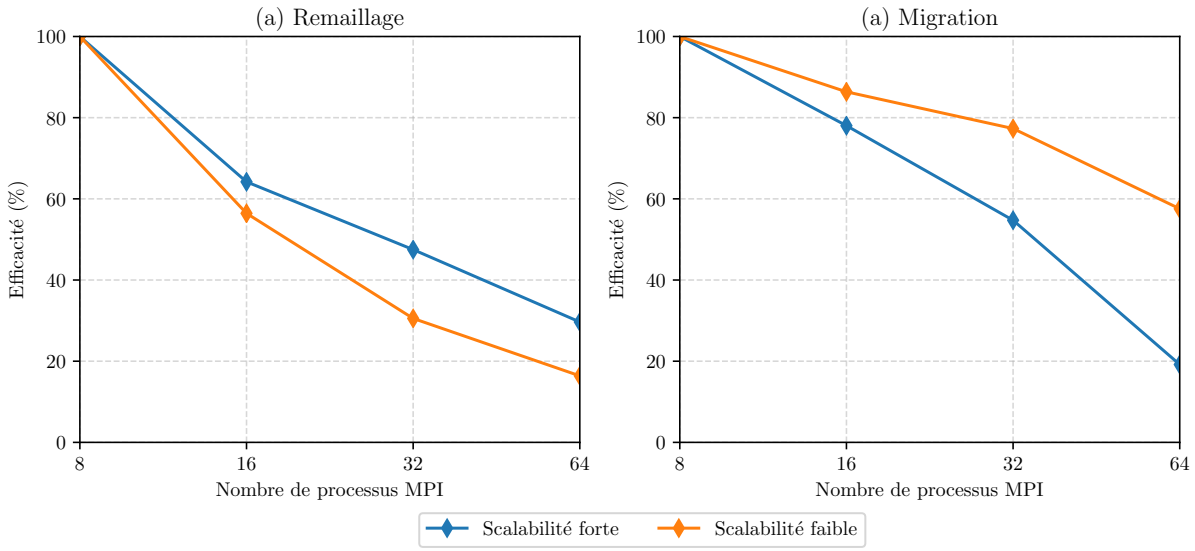


FIGURE IV.32 – Comparaison de l'efficacité en scalabilité forte et faible pour le cas Frank-Read (cas 1).

Le premier type de comportement est observable en figure IV.32a, où l'efficacité est meilleure en scalabilité forte que faible. Dans ce cas, la performance est principalement impactée par l'augmentation de la taille des données. Par exemple, le remaillage utilise les opérations topologiques d'ajout et de suppression d'objet au maillage qui séquentialisent une partie de l'algorithme. Une solution pour améliorer ce type d'algorithme consiste à optimiser les synchronisations, par exemple en réduisant la partie séquentielle du calcul. Pour Optidis, la mise en place d'opérations topologiques non séquentielles permettrait de résoudre certains de ces problèmes.

Le second type de comportement est visible dans la figure IV.32b. Une efficacité supérieure en scalabilité faible qu'en scalabilité forte indique que les performances sont limitées par l'augmentation du nombre de processus. Par exemple l'étape de la migration est une étape où le nombre de communications MPI est important. Une solution pour améliorer la performance de ce type d'algorithme consiste à optimiser les communications MPI.

L'efficacité totale dépend aussi du type de calcul de forces utilisé. La figure IV.33 compare l'efficacité du calcul de forces en Cutoff et en FMM complète.

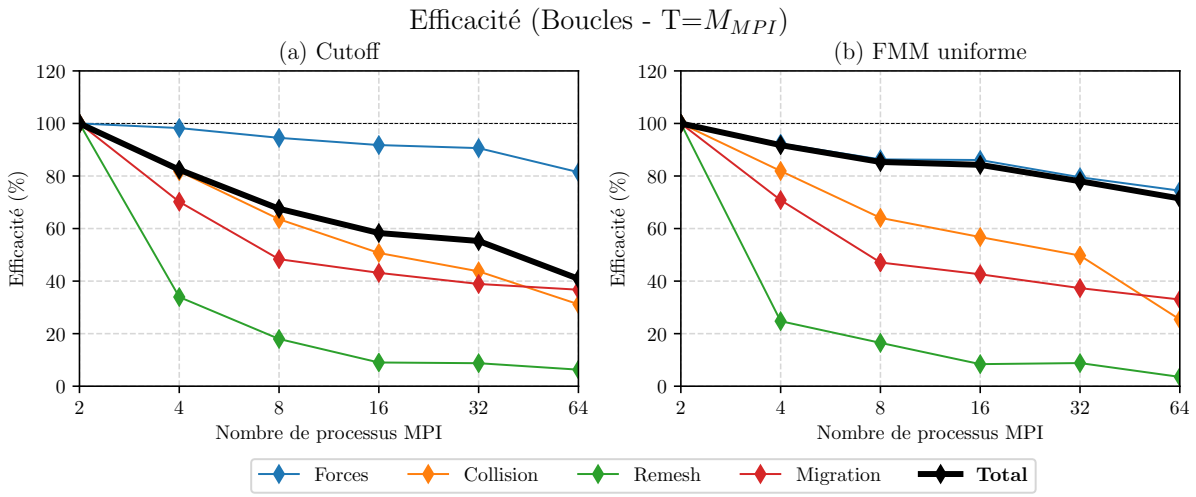


FIGURE IV.33 – Comparaison des efficacités selon le type de calcul de forces pour les boucles (cas 2) en scalabilité faible.

On remarque ici que l'efficacité pour 64 processus peut varier entre 40% et 70% selon le type de calcul de forces utilisé. Comme le calcul du champ lointain domine le temps de calcul en FMM complète, l'efficacité totale de la simulation dans ce cas se rapproche de celle du calcul de forces (fig. IV.33b). Dans le calcul Cutoff (fig. IV.33a) les autres étapes du calcul jouent un rôle plus important dans le temps d'exécution total, ce qui fait chuter l'efficacité.

IV.2.2.5 Synthèse

Les paramètres du calcul de forces influent directement sur la performance du calcul. La hauteur de l'arbre devra être choisie en fonction de la densité de dislocations, du nombre d'objets présents dans la simulation et du type de calcul de forces en utilisant les résultats des sections IV.2.2.2 et IV.2.2.3. Plus le nombre de dislocations est grand, plus la hauteur d'arbre devra être importante afin que le champ proche ne domine pas le calcul. Une plus grande densité de dislocations limitera la hauteur maximale de l'arbre.

Un équilibre entre champ proche et champ lointain doit être trouvé pour obtenir de bonnes performances. La hauteur de l'arbre doit bien sûr être en accord avec la taille des données (sec. IV.2.2.3), mais il est aussi possible de jouer sur la fréquence de calcul du champ lointain afin d'obtenir un bon compromis entre performance et précision. Les résultats de simulations simples (sec. IV.1.2.2) donnent un aperçu de l'impact de la fréquence de calcul du champ lointain sur la précision. Des pistes de réflexion sont présentes dans les travaux de A. Etchevery [40] pour déterminer la fréquence optimale du calcul du champ lointain. Il montre que le champ lointain doit être calculé plus souvent lorsque la simulation est plus dynamique.

Les résultats précédents nous permettent de déduire le nombre de nœuds de calcul et le temps nécessaire à la simulation en fonction du cas test. En section IV.2.2.4, nous avons remarqué que l'efficacité baissait lorsque l'on augmentait le nombre de processeurs de calcul, que l'on soit en scalabilité forte ou faible. Cependant, plusieurs raisons peuvent pousser à augmenter le nombre de nœuds de calcul, comme par exemple une capacité mémoire insuffisante ou un temps d'exécution trop important sur un seul nœud. Le temps de simulation peut être estimé en fonction du nombre de segments, du temps de traitement de chaque segment et du nombre

d'itérations :

$$Temps = N_{segments} \times T_{segment} \times N_{iter} \tag{IV.3}$$

Chacun de ces paramètres peut être estimé selon le cas test considéré ou les résultats de simulations présentés précédemment. Le nombre de segments $N_{segments}$ dépend du cas-test considéré, et peut varier au cours du temps. Par exemple, le cas test des boucles contient un nombre relativement constant au cours du temps de segments $N_{segments} = 70\ 000$ alors que le cas test des sources de Frank-Read contient initialement $N_{segments} = 30\ 000$ dont le nombre varie fortement au cours du temps. Les variations doivent être prises en compte pour le calcul du temps d'exécution. Le temps par segment $T_{segment}$ dépend de tous les paramètres de la simulation étudiés précédemment et peut être estimé à partir des résultats des mesures de performances. Par exemple, les résultats de la figure IV.25 en section IV.2.2.2 nous indiquent le temps nécessaire à chaque processus pour traiter un segment : pour 16 nœuds de calcul en Cutoff pour les boucles d'irradiations, chaque segment est traité en environ 1 µs.

Ces résultats nous montrent que les étapes en dehors du calcul de forces, comme les collisions, le remaillage, et la migration ne sont pas très efficaces. Leur efficacité n'est pas cruciale lorsque l'on utilise le calcul FMM complet, car leur temps d'exécution est négligeable par rapport au calcul de forces. Lorsque l'on effectue un calcul *Cutoff* - par exemple si le champ lointain n'est pas calculé à chaque itération - le temps d'exécution de ces étapes n'est alors plus négligeable.

La performance de certaines de ces étapes est limitée par les opérations topologiques séquentialisées de la structure de données. L'algorithme le plus touché par cette limitation est l'algorithme de remaillage dont l'efficacité chute rapidement. Les opérations topologiques sont aussi un frein à la performance du traitement des collisions : l'efficacité de l'algorithme de collision en scalabilité faible est moins bonne que dans les résultats présentés en partie II (section III.5.3.2) car les opérations topologiques n'étaient pas prises en compte. Les résultats en scalabilité forte, eux, sont identiques. Ce phénomène s'explique par l'augmentation du nombre de collisions à traiter en scalabilité faible, alors qu'il reste constant en scalabilité forte. La solution est de mettre en place des opérations topologiques non-collectives dans la structure de données.

D'autres étapes voient leur scalabilité limitée par les communications MPI, comme la migration ou la préparation avant le calcul des collisions. Les communications *all-to-all* utilisées pour la migration et pour l'échange des sphères fantômes pour les collisions pourraient par exemple être évitées en utilisant une décomposition de domaine qui garantit un nombre de voisins plus limité. Par exemple, il est possible de réduire la hauteur de l'arbre pour la décomposition de domaine décrit en section II.4.2.1. Un nombre de boites plus faible réduit la complexité des frontières des domaines [110], mais limite la capacité de la décomposition de domaine à gérer finement l'équilibrage de charge. Pour améliorer la performance de ces étapes de la simulation, il faut trouver un compromis entre équilibrage de charge et complexité de la géométrie des domaines. Le volume de messages MPI pourrait aussi être réduit en utilisant les caches de données fantômes pour les parcours de la structure décrits en section II.5.3.3.

IV.3 Simulation à grande échelle

Afin d'illustrer l'utilisation d'Optidis pour des simulations massives de dynamique des dislocations, une simulation basée sur le cas-test contenant des sources de Frank-Read présenté en section IV.2.1 a été lancée sur un plus grand nombre d'itérations. L'objectif est de montrer qu'Optidis est capable de simuler plusieurs millions de segments de dislocations en distribuant

le calcul sur plusieurs centaines de cœurs de calcul. La machine utilisée est toujours Poincaré, décrite en section II.5.2.1.

IV.3.1 Paramètres de simulation

La situation initiale est une variante du cas-test présenté en section IV.2.1.1 pour $T \approx 27$ (Domaine de largeur 60 000 Å). La simulation est exécutée sur 25 000 itérations, avec un chargement en contrainte imposée à 100 MPa et un champ lointain calculé toutes les 20 itérations. La figure IV.34 illustre la situation initiale complète de la simulation. Tous les paramètres de la simulation sont décrits dans le tableau IV.5.

Largeur boite de Simulation	60 000 Å
Nombre de lignes	600
Longueur des lignes	5 000 Å
Chargement	Contrainte constante en allongement selon la direction $[0\,0\,1]$
Contrainte	$\sigma_{33} = 100$ MPa
Rayon de capture r_{col}	15 Å
Matériau	Aluminium
Discrétisation	30 Å $> L_{seg} >$ 75 Å
dt	$<$ 0.1 ns (adaptatif)
Periodicité	Périodique

TABLE IV.5 – Paramètres de la simulation longue des sources de Frank-Read dans l'aluminium

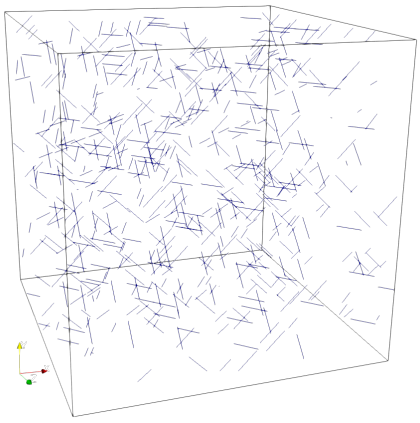
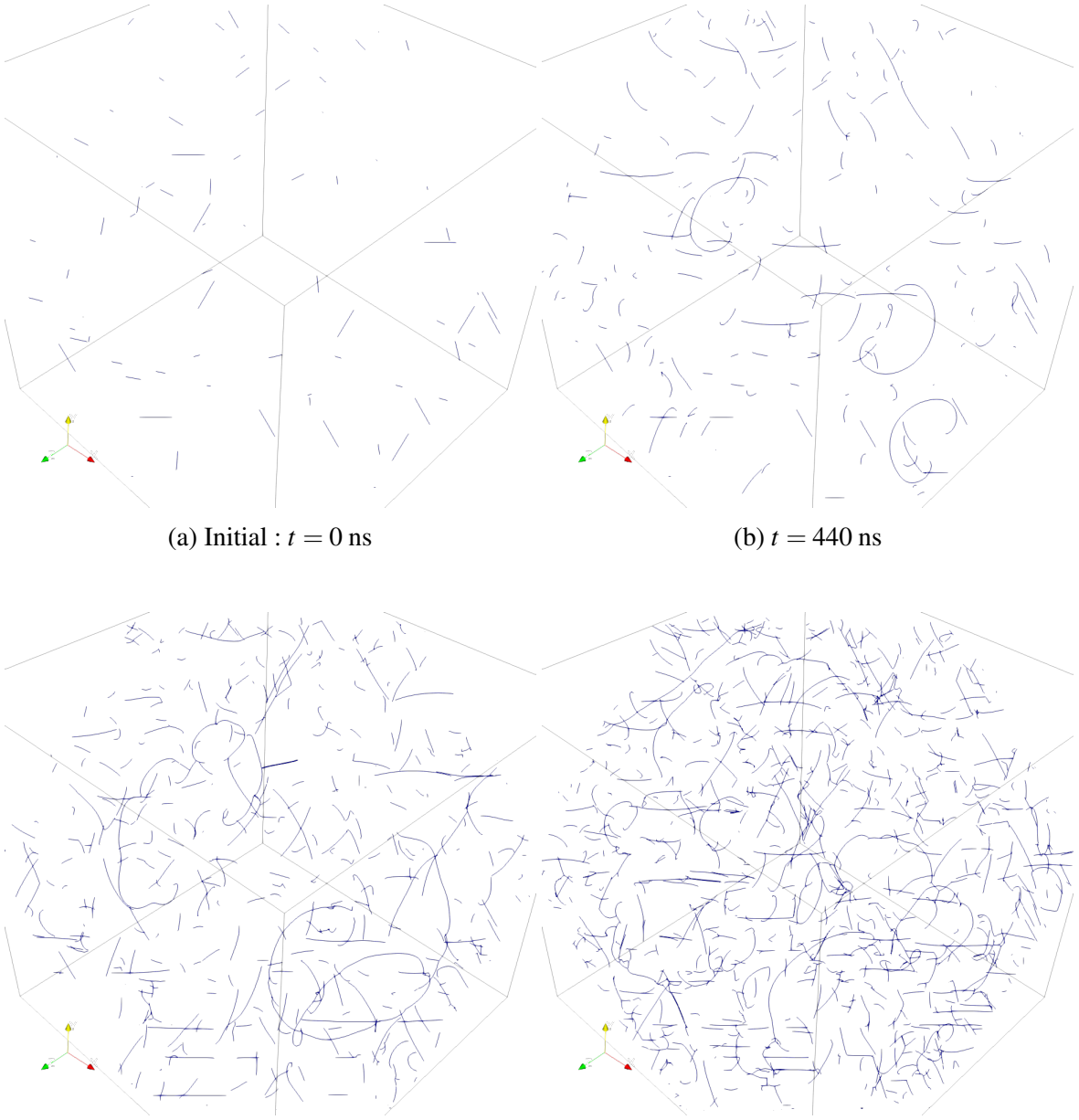


FIGURE IV.34 – Situation initiale de la simulation contenant des Sources de Frank-Read.

IV.3.2 Résultats

Cette simulation a été exécutée avec deux processus par nœud sur 16 nœuds de calcul de Poincaré (128 cœurs) avec une hauteur d'arbre FMM $H = 7$. Les images du déroulement de la simulation sont disponibles en figure IV.35 qui présente une coupe dans le plan $[1\,1\,1]$ d'épaisseur 1000 Å au centre du domaine de simulation.



(a) Initial : $t = 0$ ns

(b) $t = 440$ ns

(c) $t = 825$ ns

(d) Final : $t = 1350$ ns

FIGURE IV.35 – Coupe dans le plan [111] dans les résultats de la simulation longue des sources de Frank-Read.

Au cours de cette simulation 1350 ns sont simulées en 25 000 itérations un pas de temps moyen $\bar{dt} = 0.054$ ns. Le temps de simulation en fonction du nombre d'itérations est tracé en figure IV.36.

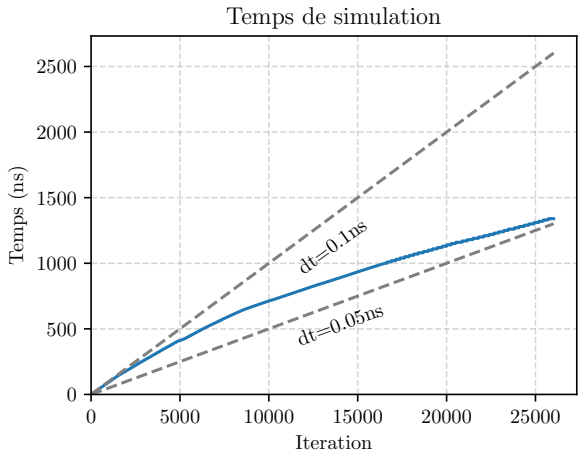


FIGURE IV.36 – Temps de simulation en fonction du nombre d'itérations.

Dans la figure IV.36 on remarque que le pas de temps réduit au fur et à mesure des itérations. L'augmentation du nombre de segments et de jonctions fait augmenter la vitesse maximale des nœuds, ce qui réduit le pas de temps acceptable dans le cadre du schéma d'intégration d'Euler.

La déformation plastique atteint $3.5 \cdot 10^{-3}$, comme le montre la figure IV.37. Pour comparer les performances, la valeur de la déformation plastique atteinte a plus de sens que le temps de simulation. En effet, selon la viscosité choisie pour le calcul des vitesses les pas de temps pourront être différents, mais le déplacement des dislocations restera la même. On retiendra donc qu'Optidis est capable de simuler une déformation plastique de $3.5 \cdot 10^{-3}$ en 20 heures sur 16 nœuds de calcul pour ce cas test.

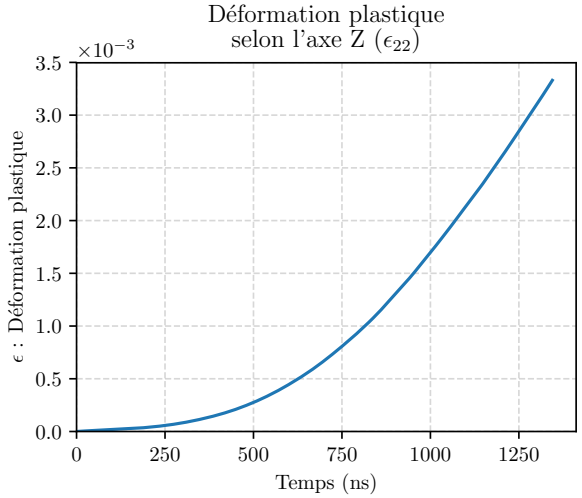


FIGURE IV.37 – Déformation plastique.

Les sources de Frank-Read sont très dynamiques, et le nombre de segments de simulation varie fortement au cours du temps. Comme on peut l'observer dans la figure IV.38a, le nombre de segments initial est d'environ 57 000 pour terminer à plus de 1 320 000. La densité de dislocation passe alors de $\rho_1 \approx 1.5 \cdot 10^{12}$ m^{-2} à $\rho_2 \approx 3 \cdot 10^{13}$ m^{-2}. La figure IV.38b donne le déséquilibre de

charge au cours du temps. Il s'agit de l'écart relatif du nombre de segments entre le processus qui en possède le plus S_{max}, et celui qui en possède le moins S_{min}. Les données brutes affichent les extremums du déséquilibre de charge, une courbe lissée avec un filtre gaussien ($\sigma = 100$ iter) donne la valeur moyenne du déséquilibre.

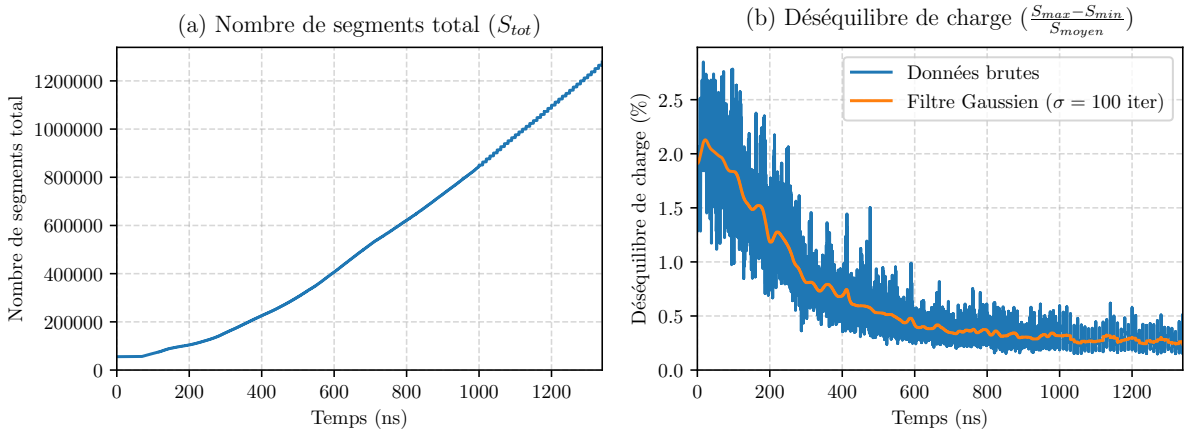


FIGURE IV.38 – Distribution des segments. (a) Le nombre de segments total simulé sur tous les processus MPI. (b) La différence relative maximale du nombre de segments locaux aux processus MPI.

Le nombre de collisions varie fortement d'une itération à l'autre. C'est pour cette raison que la performance du traitement des collisions n'a pas été mesurée en partie III. On observe bien cette variation en figure IV.39 qui présente le nombre de collisions détectées à chaque itération. Si le nombre de collisions détectées peut atteindre un nombre important, par exemple 100 col/iter vers 1200ns, sa valeur moyenne lisible sur la courbe lissée ne dépasse que rarement 30 col/iter. Le nombre moyen de collisions par itération est donc relativement faible par rapport au nombre de segments simulés.

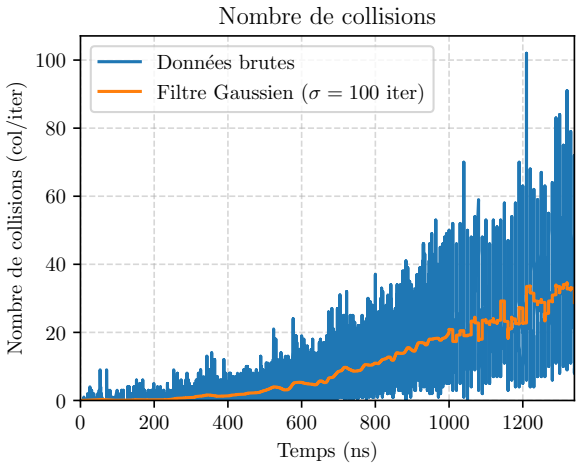


FIGURE IV.39 – Nombre de collisions.

Le temps d'exécution total est de 20 heures sur 16 nœuds de calcul de Poincaré avec une hauteur d'arbre $H = 7$. Les temps d'exécution de chaque étape du calcul à chaque itération sont

synthétisés dans la figure IV.40. Les temps de calcul par itération (a) et par segments (b) y sont présentés. Les valeurs ont été lissées pour effacer les variations locales entre itérations, notamment pour moyenner le temps de calcul des forces afin de prendre en compte le calcul périodique du champ lointain.

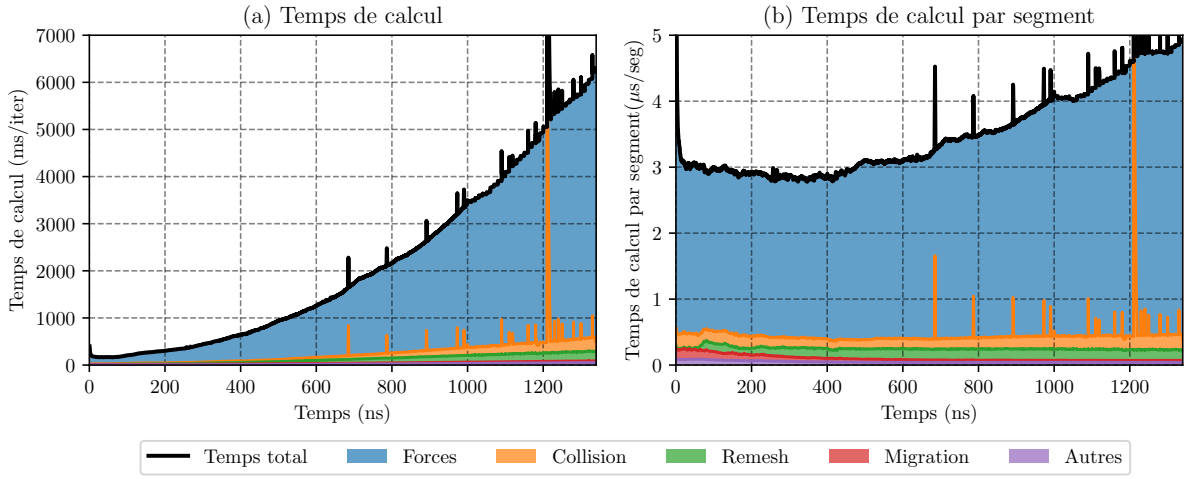


FIGURE IV.40 – Temps d'exécution des différentes étapes de calcul à chaque itération (filtre gaussien $\sigma = 10$iter). (a) Temps d'exécution par itération. (b) Temps de traitement de chaque segment.

Dans la figure IV.40a, on observe l'augmentation de temps de traitement en accord avec l'augmentation du nombre de segments. La figure IV.40b nous confirme que le temps de calcul évolue quasiment linéairement en fonction du nombre de segments. On remarque tout de même que le temps d'exécution par segment du calcul de forces augmente à partir d'environ 600ns, soit environ S_{tot}=400 000 segments ou encore S_{moy}=12 500 segments par processus.

On remarque dans la figure IV.40 des pics sur le temps d'exécution de l'algorithme de collision. La re-détection des collisions peut parfois mener à des situations où la géométrie ne converge pas : une collision re-détectée peut inverser la modification topologique précédente par exemple. Afin d'éviter les modifications topologiques cycliques, le nombre de modifications topologiques est borné. Cela permet de sortir rapidement d'un cycle. Par exemple, pour cette simulation le temps de gestion des collisions est toujours inférieur à 15s par itération en limitant le nombre d'opérations topologiques à 1000 par pas de temps.

IV.3.3 Synthèse

La simulation des sources de Frank-Read dans l'aluminium sur un plus grand nombre de pas de temps nous en apprend plus sur le comportement d'Optidis.

On observe une diminution du pas de temps au cours du temps, lorsque le réseau de dislocations se complexifie. Des segments proches qui ne fusionnent pas peuvent générer des vitesses nodales importantes qui réduisent le pas de temps. En observant de plus près le réseau de dislocation au cours de la simulation, on s'aperçoit que les nœuds les plus rapides sont en fait ceux proches des nœuds physiques. Ce phénomène est lié à celui observé pour la jonction de Lomer-Cottrell : le calcul local des forces pourrait générer des instabilités aux abords des nœuds physiques.

Le temps d'exécution est dominé par le calcul de forces, même lorsque le champ lointain n'est calculé que toutes les 20 itérations. Dans cette simulation, le temps de traitement de chaque segment reste relativement constant au cours du temps. On observe des irrégularités dans le temps de traitement des collisions qui ne posent cependant pas de réels problèmes de performance, puisque le nombre de collisions est faible par rapport au nombre de segments simulés.

Bilan

Cette partie consacrée à la validation et aux mesures de performances a permis de mettre en avant l'apport des contributions présentées dans ce manuscrit, mais permet aussi de les mettre en perspectives en identifiant les points à traiter pour améliorer différents aspects du code Optidis.

Dans un premier temps, des comparaisons avec NUMODIS ont été effectuées en quantifiant les différences avec la comparaison des déformations plastiques, mais aussi en observant les différences de géométrie. Ces résultats nous montrent qu'il y a un bon accord entre NUMODIS et Optidis, mais que quelques différences sont tout de même présentes. Le calcul local des forces, notamment, semble problématique aux alentours des nœuds physiques.

Des mesures de performances sur un nombre faible d'itérations, mais en faisant varier plusieurs paramètres de simulation ont permis d'observer l'impact de ces paramètres sur la performance. Les paramètres de calcul des forces ont un impact important sur les performances. Les mesures de FMM avec et sans calcul du champ lointain permettent de montrer qu'un temps de calcul non négligeable peut être économisé en ne calculant pas le champ lointain à chaque itération. Ces mesures permettent aussi d'étudier la performance MPI d'Optidis dont l'efficacité totale varie entre 40% et 70% selon les paramètres utilisés.

Enfin, une simulation complète sur 128 cœurs de calculs pendant 20 heures a permis d'illustrer le fonctionnement d'Optidis sur 25 000 pas de temps. Cette simulation contient des sources de Frank-Read, jusqu'à 1 300 000 segments, pour une densité de dislocations de $3 \cdot 10^{13}$ m^{-2} dans un grain périodique d'aluminium cubique de 6 µm de côté. En plus de démontrer la faisabilité de telles simulations, cette expérience nous permet d'observer le comportement du code afin d'analyser les aspects à améliorer.

Conclusion

Au cours de cette thèse, j'ai présenté des solutions à plusieurs problèmes informatiques que posent les simulations de dynamique des dislocations sur architectures parallèles et distribuées. J'ai notamment traité les problématiques de stockage des données en proposant une nouvelle structure de données pour la dynamique des dislocations. J'ai aussi traité les problématiques liées aux collisions entre dislocations en proposant un nouvel algorithme de collision.

Après avoir présenté les dislocations et leur rôle dans la plasticité des matériaux cristallins, nous avons vu comment la dynamique des dislocations s'inscrivait à l'échelle mésoscopique dans la modélisation multi échelle des matériaux. Les simulations de dynamique des dislocations se tournent vers le calcul intensif pour répondre aux besoins de plus en plus importants en puissance de calcul des études menées pour effectuer la transition de l'échelle microscopique vers la mécanique des milieux continus. Des méthodes permettent d'accélérer le calcul afin d'augmenter le nombre de segments simulés, comme la FMM ou la parallélisation. Le recours aux architectures parallèles et distribuées pose cependant plusieurs problèmes informatiques dont il est question tout au long de cette thèse.

Une des spécificités des simulations de dynamique des dislocations est le caractère dynamique du réseau de dislocations. La topologie du maillage et le nombre de segments de dislocations peuvent varier fortement au fil des remaillages et des collisions. La première problématique traitée est celle du stockage du maillage dynamique dont les données doivent être accessibles rapidement dans les phases de calcul intensif, et dont la topologie doit aussi pouvoir être modifiée lors des remaillages et collisions. Après avoir étudié les différentes manières de stocker un maillage dynamique, je propose une structure de données qui permet un accès fiable et rapide aux données de la simulation dans un contexte parallèle et distribué. Une interface abstraite a été mise en place et permet de découpler l'écriture des algorithmes physiques de l'optimisation des accès aux données. Je décris comment cette structure a été implémentée avec des tableaux dynamiques, et comment j'ai traité les problématiques de fragmentation mémoire et de distribution des données au sein d'Optidis. J'utilise notamment une méthode de nettoyage périodique pour réduire la fragmentation mémoire et pour augmenter la localité spatiale des objets du maillage. Les données sont distribuées en utilisant une décomposition de domaines selon la Z-curve, et je propose une méthode pour équilibrer la charge entre les processus. Des mesures de performance ont montré que l'interface d'abstraction et son implémentation permettent un parcours performant des données portées par le réseau de dislocation.

La seconde problématique traitée concerne la gestion fiable des collisions entre dislocations, responsables, notamment, de la formation des jonctions qui sont impliquées dans le phénomène d'écrouissage. Les modifications topologiques effectuées lors d'une collision peuvent rapidement être source d'instabilités et limiter le pas de temps. J'ai étudié les différentes méthodes utilisées pour détecter et traiter les collisions dans la dynamique des dislocations, mais aussi

dans d'autres types de simulations pour aboutir à quatre algorithmes que j'ai implémentés et comparés. À partir de ces comparaisons, j'ai choisi un algorithme de détection et de traitement des collisions que j'ai ensuite intégré dans Optidis. Il s'agit d'un algorithme de gestion dynamique des collisions qui prend en compte le mouvement des dislocations au cours du pas de temps pour n'oublier aucune intersection. Cet algorithme est plus fiable grâce à une re-détection des collisions et l'utilisation de la nouvelle interface d'accès aux données. Il est aussi performant grâce à un découpage en grille uniforme de l'espace, qui permet de réduire le nombre de primitives de collision à calculer, ainsi que l'utilisation de sphères englobantes qui réduit le coût de chaque primitive. Des mesures de performances nous permettent de valider le fonctionnement de cet algorithme en mémoire partagée et distribuée. L'efficacité en mémoire distribuée en scalabilité faible de cet algorithme a été mesurée à environ 70% pour 256 cœurs de calcul et 50% pour 512 cœurs avec plusieurs millions de segments de dislocations.

Optidis a été validé en vérifiant sa précision par rapport au simulateur séquentiel Numodis sur des simulations simples. On observe une bonne adéquation entre Numodis et Optidis, et la différence relative des déformations plastiques entre les deux codes est d'environ 10^{-2} pour les cas-tests comparés, à savoir la source de Frank-Read et la jonction de Lomer-Cottrell. L'apport de l'algorithme de collision a été validé sur une simulation complète impliquant une ligne de dislocation et une boucle d'irradiation et montre que ce nouvel algorithme permet l'utilisation de pas de temps plus grands.

La performance et la scalabilité d'Optidis ont été mesurées sur des simulations à plus grande échelle contenant des boucles d'irradiation et des sources de Frank-Read. Ces mesures ont été menées sur un nombre faible d'itérations en faisant varier plusieurs paramètres de simulation afin d'observer leur impact sur la performance. Ces simulations peuvent contenir jusqu'à plusieurs millions de segments sur 512 cœurs de calcul avec des efficacités en scalabilité faible allant de 40% à 70% selon les paramètres de la simulation. On note que les paramètres du calcul de forces ont un impact particulièrement important sur la performance et la scalabilité de la simulation complète. En effet, le calcul des forces est le plus long, mais aussi le plus efficace. Réduire la fréquence de calcul du champ lointain diminue le temps de calcul total, mais réduit aussi l'efficacité en augmentant la proportion du temps d'exécution passé dans des algorithmes moins efficaces, comme le remaillage, la migration ou les collisions.

Une simulation de sources de Frank-Read dans l'aluminium a été exécutée sur un nombre plus important de pas de temps. Les sources de Frank-Read étant des objets très dynamiques, le nombre de segments de dislocations varie entre 57 000 et 1 320 000 au cours des 25 000 pas de temps simulés pendant 20 heures sur 128 cœurs de calcul. Différentes métriques qui définissent la performance ont été analysées, comme le pas de temps utilisé, le nombre de collisions traitées et le temps d'exécution au cours de chaque pas de temps.

Ces résultats montrent l'apport de mes différentes contributions sur la performance d'Optidis. Bien que le nombre de dislocations soit multiplié par un facteur 20 entre le début et la fin de la simulation, le temps de calcul par segment reste relativement constant, ce qui montre bien que la dynamicité des données est gérée correctement par la structure de données. De plus, les données sont correctement réparties entre les processus grâce à la méthode de répartition de charge mise en place. Le nombre de collisions est relativement faible au cours d'un pas de temps, ce qui justifie le choix fait pour l'algorithme de gestion des collisions de privilégier la fiabilité du traitement à sa performance. Les mesures montrent bien que l'algorithme de collisions ne pénalise pas la performance de la simulation complète.

Perspectives

Les résultats de cette étude nous montrent que la simulation de plusieurs millions de segments de dislocations sur plusieurs centaines de cœurs de calcul est possible avec Optidis. Les simulations menées au cours de cette thèse ont cependant comme unique finalité de montrer la performance d'Optidis, mais ne sont pas utilisées pour effectuer des études réelles sur la plasticité. Si les résultats de performance et de scalabilité présentés ici montrent que la puissance de calcul disponible dans Optidis permet d'effectuer des simulations à une échelle relativement grande, les études sur le comportement mécanique des matériaux restent à mener par la communauté des métallurgistes.

Le phénomène le plus accessible à modéliser avec Optidis est probablement l'écrouissage, pour lequel des simulations très similaires à celle présentée en section IV.3 sont déjà utilisées dans des codes tels que MicroMégas/Tridis [80, 84] ou ParaDiS [19]. Reproduire les simulations utilisées dans ces études permettrait de comparer la précision et la performance d'Optidis avec les autres codes.

Les différentes mesures et observations effectuées tout au long de cette thèse pointent aussi vers un certain nombre d'aspects à améliorer pour augmenter la performance et permettre des simulations à plus grande échelle. La performance et la scalabilité de chaque itération peuvent encore être améliorées en allant plus loin dans certains aspects des travaux présentés ici. Il est possible d'améliorer la performance de la structure de données en proposant une interface non-collective pour les modifications topologiques, en groupant les modifications ou en limitant le nombre de processus impliqués dans chaque opération. Une interface non-collective permettrait aussi de mettre œuvre l'algorithme de collision parallèle proposé en section III.3.1.4 mais non implémenté.

La performance MPI des différents algorithmes peut aussi être améliorée en réduisant le volume de données fantômes. L'optimisation de la décomposition de domaine permet de diminuer la surface partagée entre les processeurs, mais d'autres facteurs comme la répartition de charge doivent être pris en compte. Il est aussi possible de diminuer le nombre de processus MPI en exploitant mieux le parallélisme en mémoire partagée. Une solution possible est d'adapter la structure de données aux architectures NUMA pour n'utiliser qu'un seul processus par nœud de calcul au lieu d'un processus par nœud NUMA. Il faut alors travailler sur l'allocation des données pour les distribuer sur les différents bancs mémoire. Une autre solution est d'utiliser des bibliothèques qui fournissent des moyens optimisés d'accéder aux données pour l'implémentation de la structure, comme par exemple Kokkos [38], qui permet aussi une transition vers des architectures plus complexes.

La performance de la simulation dans sa globalité peut aussi être améliorée, non pas en calculant plus vite, mais en optimisant les aspects numériques. Le nouvel algorithme de collision ouvre des portes pour mettre en place de nouvelles méthodes d'intégration. Il est possible d'augmenter le pas de temps, et donc de réduire le nombre d'itérations nécessaires, en utilisant des méthodes d'ordre supérieur. Optimiser plus finement la fréquence du champ lointain et l'ordre d'interpolation permet aussi d'accélérer la simulation en diminuant le coût du calcul des forces. Il faut ici trouver le compromis optimal entre vitesse et précision.

Enfin, comme nous l'avons vu, un enjeu majeur pour permettre des simulations à plus grande échelle est l'optimisation du calcul des forces qui représente une majeure partie du temps d'exécution. Les travaux que nous avons présentés ici utilisent la méthode des multipôles rapides ainsi

que les architectures distribuées pour accélérer le calcul. L'utilisation d'autres architectures parallèles émerge, avec par exemple l'utilisation de GPUs pour la dynamique des dislocations [41], ou pour la FMM [57, 3]. Agréger ces techniques et utiliser les GPUs pour accélérer les calculs FMM pour la dynamique des dislocations permettrait d'accéder à un nouvel ordre de grandeur de parallélisme.

Annexe A

Partie 2 - Structure de données

A.1 Interface C++ pour la structure de données

A.1.1 Types

NodeInfo et SegmentInfo
sont des structures contenant les données portées respectivement par un nœud et un segment. Elles sont détaillées en figure A.1. Elles permettent d'initialiser un objet lors de sa création, ou de lire les informations qui sont associées à un objet, mais elles ne permettent pas de modifier l'information stockée dans la structure de données.

NodeInfo_ref et SegmentInfo_ref
sont des références vers les données portées respectivement par un nœud et un segment. Elles permettent l'accès et la modification des informations contenues dans la structure de données.

NodeIndex_local et SegmentIndex_local
sont des indices locaux qui pointent vers un nœud ou un segment du processus MPI local.

NodeIndex_global et SegmentIndex_global
sont des indices globaux qui pointent vers un nœud ou un segment d'un processus MPI local ou distant. Ces structures contiennent le rang du processus MPI qui possède la donnée ainsi qu'un indice local de la donnée sur le processus distant. Ces indices globaux permettent d'indicer les objets en mémoire distribuée.

```
1  struct Mesh_NodeInfo
2  {
3      double x,y,z;
4      double fx, fy, fz;
5      double vx, vy, vz;
6      [...]
7
8  };
```

(a) Nœuds.

```
1  struct Mesh_SegmentInfo
2  {
3      Mesh_NodeIndex_global n1, n2;
4      GlideSystem gs;
5      double fx_n1, fy_n1, fz_n1;
6      double fx_n2, fy_n2, fz_n2;
7      [...]
8  };
```

(b) Segments.

FIGURE A.1 – Données portées par les nœuds/segments telles qu'elles apparaissent dans la structure de données.

```
struct Mesh_NodeInfo_ref
{
    double &x,&y,&z;
    double &fx, &fy, &fz;
    double &vx, &vy, &vz;
    [...]

};
```

(a) Nœuds.

```
struct Mesh_SegmentInfo_ref
{
    Mesh_NodeIndex_global n1, n2;
    GlideSystem &gs;
    double &fx_n1, &fy_n1, &fz_n1;
    double &fx_n2, &fy_n2, &fz_n2;
    [...]
};
```

(b) Segments.

FIGURE A.2 – Interface d'accès aux données dans la structure qui permet leur modification.

```
struct NodeIndex_local
{

    int index;

};
```

(a) Index local pour l'implémentation à base de tableaux simples.

```
struct NodeIndex_global
{
   int getRank();
   NodeIndex_local getLocalIndex();
   [...]
};
```

(b) Indice global.

FIGURE A.3 – Indices des nœuds dans la structure de données.

```
        class MeshHelper{
            //Parcours des noeuds
            //f de type void(*)( Mesh::NodeInfo_ref&& n)
            template<typename Function>
            static void fastIterate_nodes(Mesh& m, const Function& f);
            //Parcours des segments avec noeuds connectés
            //f de type void(*)( Mesh::SegmentInfo_ref&& s_info,
            //                   const Mesh::NodeInfo_ref& n1_info,
            //                   const Mesh::NodeInfo_ref& n2_info)
            template<typename Function>
            static void fastIterate_segments_withNodes( Mesh& m, const Function& f);
            //Parcours des noeuds avec segments connectés
            //f de type void(*)( Mesh::NodeInfo_ref&& n,
            //                   const std::vector<MeshHelper::Connexion>& connexions)
            template<typename Function>
            static void fastIterate_nodes_withSegments(Mesh& m, const Function& f);
        };
```

FIGURE A.4 – Fonctions de parcours de la structure de données.

```
class Mesh{
    //Opérations globales
    void clean();
    void migrate();
    void reserve( size_t size );
    //Opérations sur les noeuds
    struct Nodes{
        //Modifications topologiques
        NodeIndex_global insert(const NodeInfo&  n, int insert_rank = -1);
        void             erase (const NodeIndex_global& n);
        //Accès à la topologie
        SegmentIndex_list
            getConnectedSegments (const NodeIndex_local& n) const;
        SegmentIndex_list
            getConnectedSegments (const NodeIndex_global& n) const;
        //Accès aux données des objets
        size_t size() const;
        NodeInfo get(const NodeIndex_global& n) const;
        void     set(const NodeIndex_global& n, const NodeInfo& n_info);
        NodeInfo_ref at         (const NodeIndex_local& n);
        NodeInfo_ref operator[] (const NodeIndex_local& n);
        NodeIterator begin      () ;
        NodeIterator end        () ;
    };
    //Opérations sur les segments
    struct Segments{
        //Modifications topologiques
        SegmentIndex_global insert(const SegmentInfo& s, int insert_rank = -1);
        void                erase (const SegmentIndex_global& s);
        //Accès à la topologie
        SegmentIndex_global find (const NodeIndex_global& n1,
                                  const NodeIndex_global& n2) const;
        //Accès aux données des objets
        size_t size() const;
        SegmentInfo get(const SegmentIndex_global& s) const;
        void        set(const SegmentIndex_global& s, const SegmentInfo& s_info);
        SegmentInfo_ref at        (const SegmentIndex_local& s);
        SegmentInfo_ref operator[](const SegmentIndex_local& s);
        SegmentIterator begin     ();
        SegmentIterator end         ();
    };
};
```

FIGURE A.5 – Structure de données contenant le réseau de dislocations.

```
double dt;
[...]
MeshHelper::fastiterate_nodes( mesh,
[dt](Mesh::NodeInfo_ref& n)
{
    n.x_old() = n.x();
    n.y_old() = n.y();
    n.z_old() = n.z();
    n.x() += dt*n.vx();
    n.y() += dt*n.vy();
    n.z() += dt*n.vz();
});
[...]
```

(a) Déplacement des nœuds.

```
std::vector<double> vstrain(6);

container::MeshHelper::fastIterate_segments_withNodes(mesh,
[&](const Mesh::SegmentInfo_ref s_info,
const Mesh::NodeInfo_ref& n1,
const Mesh::NodeInfo_ref& n2)
{
    Vect3D x1   {n1.x()    , n1.y()     , n1.z()};
    Vect3D X1   {n1.x_old(), n1.y_old() , n1.z_old()};
    Vect3D x2   {n2.x()    , n2.y()     , n2.z()};
    Vect3D X2   {n2.x_old(), n2.y_old() , n2.z_old()};

    const Vect3D X1x1 = x1 - X1;
    const Vect3D X1X2 = x2 - X2;

    // compute swept surface
    Vect3D e = x2 -X1;
    Vect3D h = X1x1-X1X2;
    Vect3D n = 0.5 * linalg::cross(e,h);

    Vect3D burgers = s_info.gs().burgers();

    // calculate strain rate increment
    vstrain[0]+=n.x*burgers.x;
    vstrain[1]+=n.y*burgers.y;
    vstrain[2]+=n.z*burgers.z;
    vstrain[3]+=0.5*(n.x*burgers.y+n.y*burgers.x);
    vstrain[4]+=0.5*(n.x*burgers.z+n.z*burgers.x);
    vstrain[5]+=0.5*(n.y*burgers.z+n.z*burgers.y);
});
```

(b) Calcul de l'incrément de déformation plastique.

FIGURE A.6 – Exemple d'utilisation des fonctions de parcours.

Le code de la figure A.6a calcule le déplacement des nœuds :

$$x_i^{old} \leftarrow x_i$$
$$x_i \leftarrow x_i + dt \cdot v_i$$

Le code de la figure A.6b calcule l'incrément de déformation plastique :

$$\Delta\epsilon = \sum_{s \in segments s} \frac{b_s \otimes n_s + n_s \otimes b_s}{2V} \Delta A_s \tag{A.1}$$

avec V le volume de la boite de simulation, avec b_s le vecteur de Burgers, n_s la normale au plan de glissement et ΔA_s l'aire balayée par le segment.

A.2 Performances de la structure de données : résultats complets

Ces graphes sont expliqués et une sélection de ces résultats est commentée en section II.5

A.2.1 Mémoire partagée OpenMP

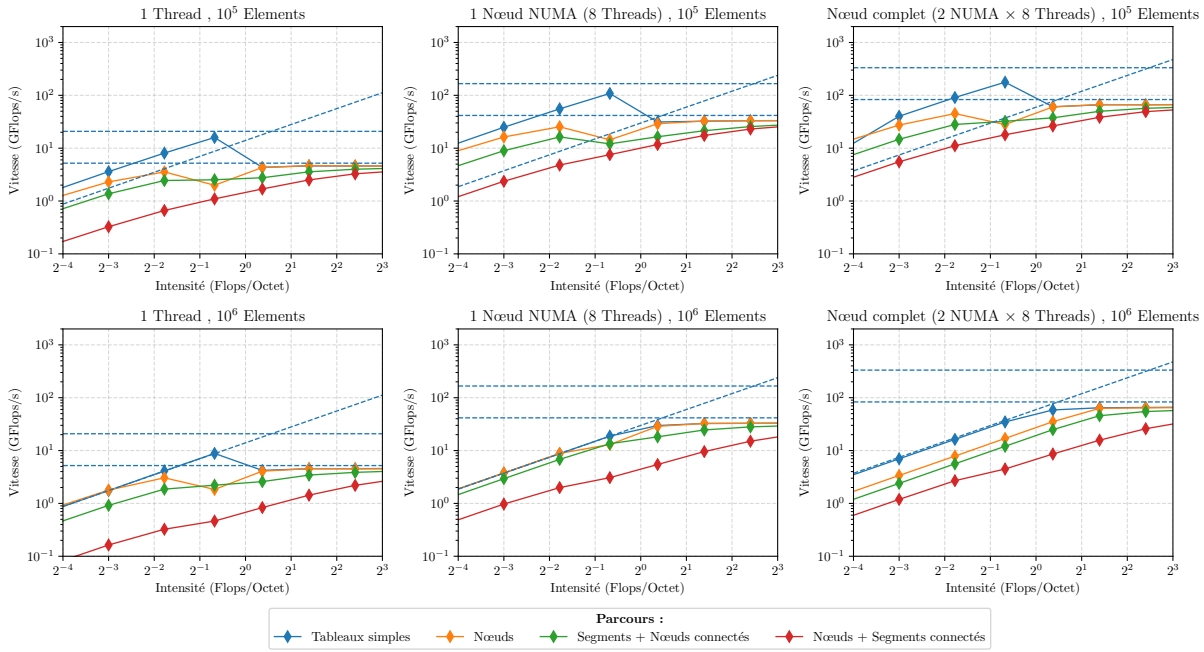


FIGURE A.7 – Résultats complets des mesures de performances de la structure de données en OpenMP (Roofline).

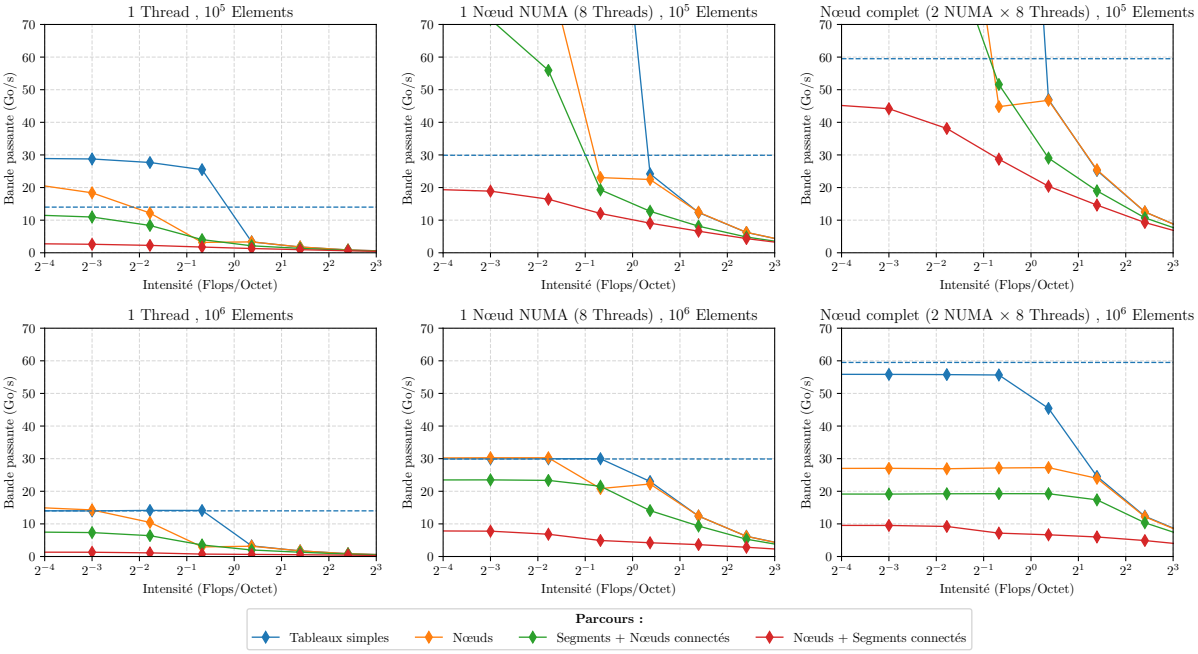


FIGURE A.8 – Résultats complets des mesures de performances de la structure de données en OpenMP (Bande-passante).

A.2.2 Mémoire distribuée MPI

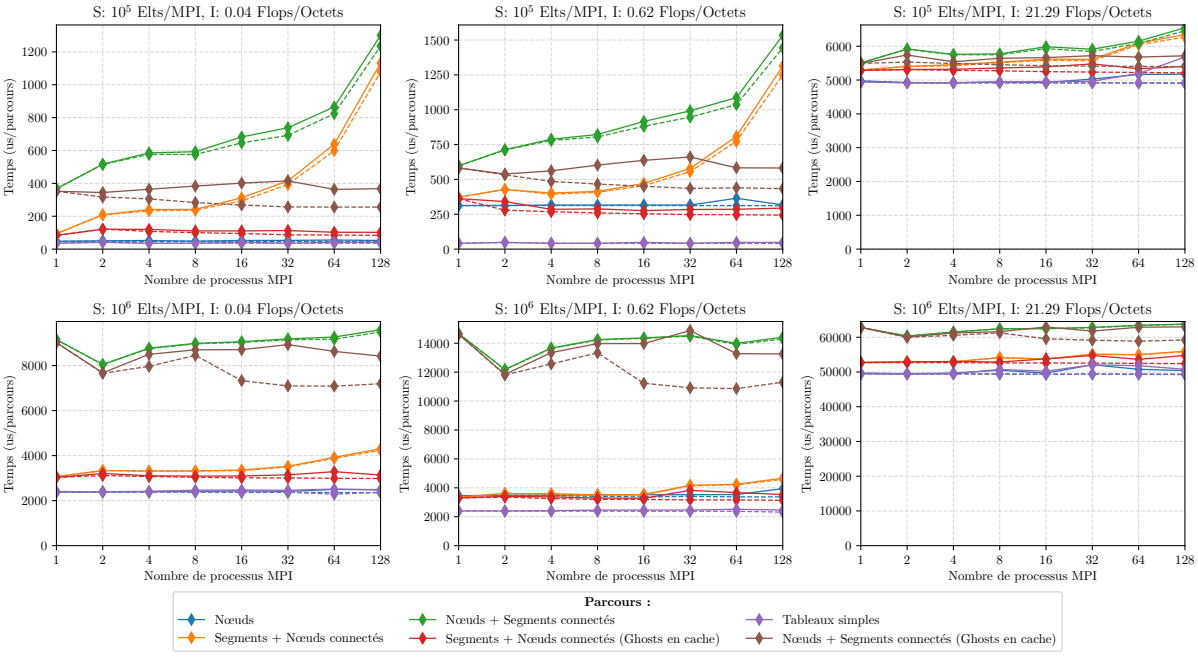


FIGURE A.9 – Résultats complets des mesures de performances de la structure de données en MPI (Temps d'exécution).

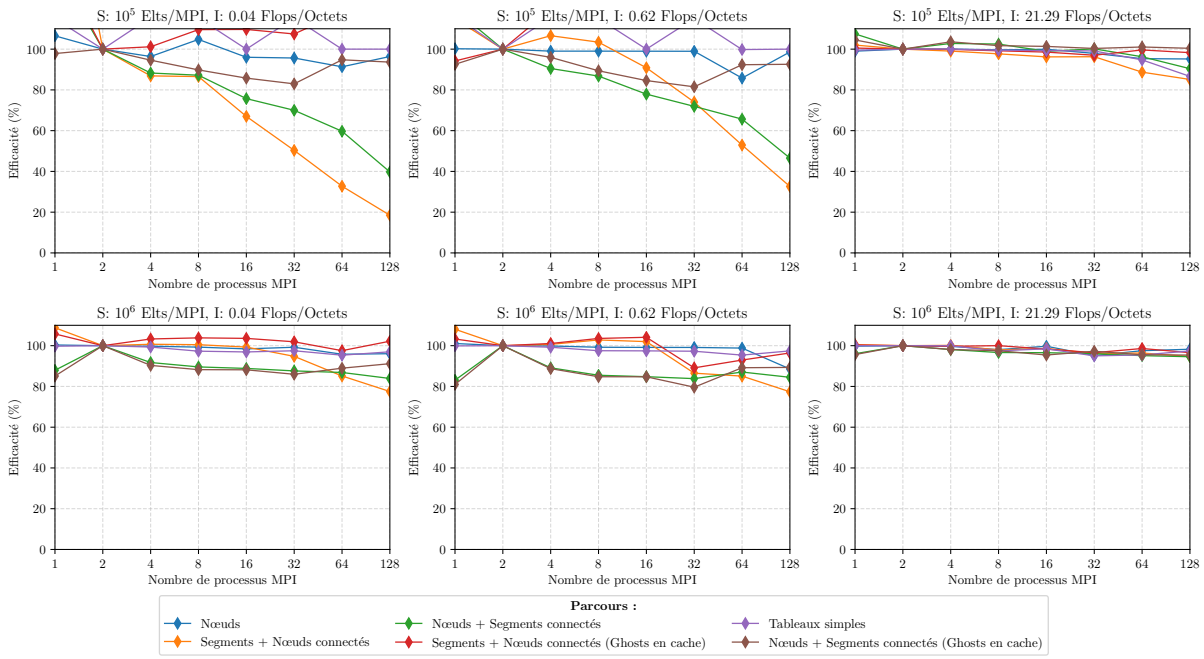


FIGURE A.10 – Résultats complets des mesures de performances de la structure de données en MPI (Efficacité).

Annexe B

Partie 3 - Collisions

Les primitives de détection et de traitement des collisions ont été réécrites au cours de cette thèse afin d'améliorer la fiabilité de l'algorithme de collision.

B.1 Détail des primitives de détection de collision

B.1.1 Collision entre nœuds

La primitive la plus simple détecte la collision entre 2 nœuds du réseau de dislocation. Les nœuds sont représentés par des sphères de rayon ε ayant pour positions initiales $\boldsymbol{p}_1$ et $\boldsymbol{p}_2$, et se déplaçant à vitesse constante $\boldsymbol{v}_1$ et $\boldsymbol{v}_2$, comme l'illustre la figure B.1.

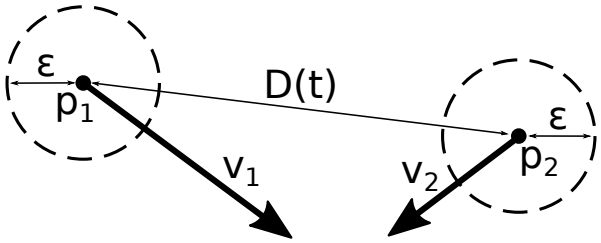


FIGURE B.1 – Distance entre deux nœuds du réseau de dislocations.

La distance entre les deux points au cours du temps se calcule de la manière suivante :

$$D(t) = \|(\boldsymbol{p}_2 + t * \boldsymbol{v}_2) - (\boldsymbol{p}_1 + t * \boldsymbol{v}_1)\| \tag{B.1}$$

$$D^2(t) = at^2 + bt + c \tag{B.2}$$

avec

$$a = \|\boldsymbol{v}_2 - \boldsymbol{v}_1\|^2 \tag{B.3}$$

$$b = 2(\boldsymbol{v}_2 - \boldsymbol{v}_1\boldsymbol{p}_2 - \boldsymbol{p}_1) \tag{B.4}$$

$$c = \|\boldsymbol{p}_2 - \boldsymbol{p}_1\|^2 \tag{B.5}$$

Comme $D^2 > 0$, le polynôme de degré 2 est convexe, le minimum de la fonction est atteint en $t_{min} = \frac{-b}{2a}$. Si t_{min} n'est pas dans $[0, dt]$, alors on le ramène dans l'intervalle [1], comme le montre la figure B.2.

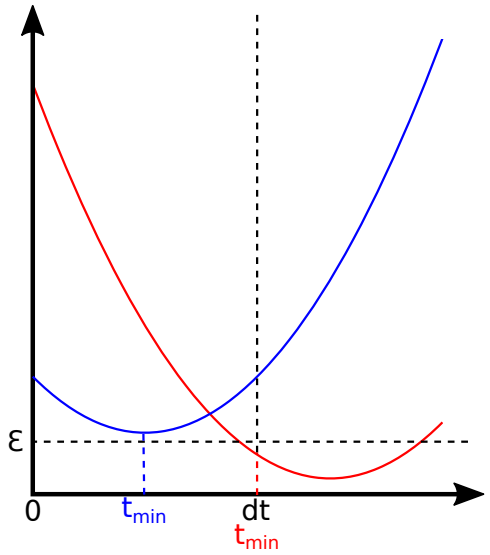


FIGURE B.2 – Distance entre deux nœuds du réseau de dislocations. Sur la courbe rouge t_{min} est après dt : on le ramène dans l'intervalle ($t_{min} = dt$).

Une collision a lieu si la distance est inférieure à ε, et le temps de la collision est t_{min} :

$$\text{collision}(n_1, n_2) \equiv D^2(t_{min}) < \varepsilon \tag{B.6}$$

B.1.2 Collision entre un point et un segment

La seconde primitive présentée détecte la collision entre un nœud et un segment. Le nœud est représenté par une sphère de rayon ε, et le segment par un cylindre de rayon ε. Le segment $s = [n_1, n_2]$ et le nœud n_3 ont pour positions et vitesses respectives $\boldsymbol{p}_i$ et $\boldsymbol{v}_i$, comme l'illustre la figure B.3.

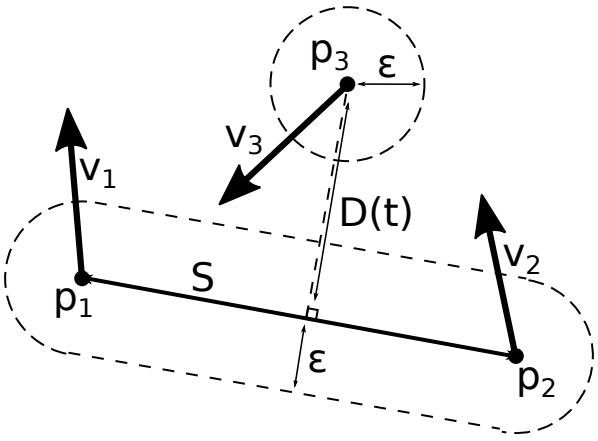


FIGURE B.3 – Distance entre un nœud et un segment du réseau de dislocations.

1. l'opération peut être délicate si $a = 0$: la distance est constante, et $t_{min} = 0$

La distance entre la droite (s) et le point n_3 se calcule de la manière suivante :

$$D(s,n_3) = \frac{\|\boldsymbol{P}_{12}(t) \wedge \boldsymbol{P}_{13}(t)\|}{\|\boldsymbol{P}_{12}(t)\|} \tag{B.7}$$

$$= \frac{\|(\boldsymbol{p}_{12} + t\boldsymbol{v}_{12}) \wedge (\boldsymbol{p}_{13} + t\boldsymbol{v}_{13})\|}{\|\boldsymbol{P}_{12}(t)\|} \tag{B.8}$$

$$= \frac{\|\boldsymbol{a}t^2 + \boldsymbol{b}t + \boldsymbol{c}\|}{\|\boldsymbol{P}_{12}(t)\|} \tag{B.9}$$

avec

$$\boldsymbol{a} = \boldsymbol{v}_{12} \wedge \boldsymbol{v}_{13} \tag{B.10}$$

$$\boldsymbol{b} = \boldsymbol{v}_{12} \wedge \boldsymbol{p}_{13} + \boldsymbol{p}_{12} \wedge \boldsymbol{v}_{13} \tag{B.11}$$

$$\boldsymbol{c} = \boldsymbol{p}_{12} \wedge \boldsymbol{p}_{13} \tag{B.12}$$

En passant la distance au carré, on obtient :

$$D^2(s,n_3) = \frac{\|\boldsymbol{a}t^2 + \boldsymbol{b}t + \boldsymbol{c}\|^2}{\|\boldsymbol{P}_{12}(t)\|^2} \tag{B.13}$$

$$= \frac{at^4 + bt^3 + ct^2 + dt + e}{\|\boldsymbol{P}_{12}(t)\|^2} \tag{B.14}$$

avec

$$a = \|\boldsymbol{a}\|^2 \tag{B.15}$$

$$b = 2(\boldsymbol{a} \cdot \boldsymbol{b}) \tag{B.16}$$

$$c = \|\boldsymbol{b}\|^2 + 2(\boldsymbol{a} \cdot \boldsymbol{c}) \tag{B.17}$$

$$d = 2(\boldsymbol{b} \cdot \boldsymbol{c}) \tag{B.18}$$

$$e = \|\boldsymbol{c}\|^2 \tag{B.19}$$

Comme pour la collision point / point, il s'agit maintenant de minimiser la distance entre le segment et le point, et de comparer cette distance minimale à ε. Cependant, la fonction distance dans ce cas est difficile à minimiser analytiquement car il n'existe pas de formule simple pour calculer les racines d'un polynôme de degré supérieur à 4. Deux possibilités s'offrent à nous :

— Minimiser numériquement la distance (Dichotomie, Méthode de Newton, ...) ;
— Approximer la distance par une fonction qu'il est possible de minimiser analytiquement.

La méthode choisie consiste à approximer la distance $||\boldsymbol{P}_{12}(t)||$ par une constante. Cette approximation est justifiée car si $||\boldsymbol{P}_{12}(t)||$ diminue trop, la collision sera détectée comme une collision entre les points n_1 et n_2. Le calcul de la distance devient :

$$D^2(t) = \frac{\|\boldsymbol{P}_{12}(t) \wedge \boldsymbol{P}_{13}(t)\|^2}{\|\boldsymbol{P}_{12}(t)\|^2} \approx \frac{P(t)}{C^2} \tag{B.20}$$

avec P un polynôme de degré 4.

Minimiser D^2 revient à minimiser le polynôme P : nous cherchons les racines de P' pour lesquelles P'' est positif. $P > 0$, donc il possède au plus deux minima locaux, comme le montre la figure B.4.

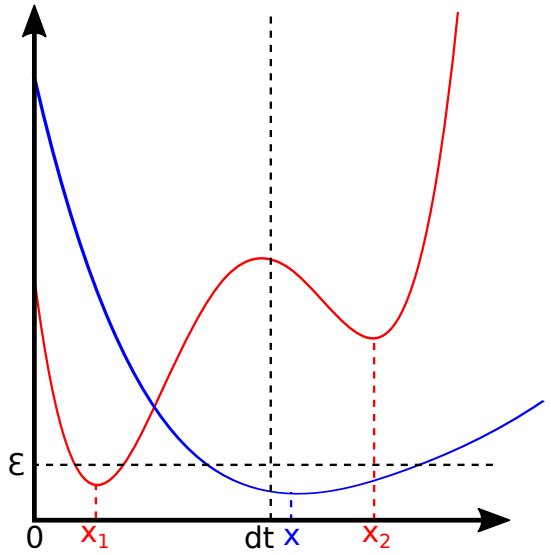


FIGURE B.4 – Minimisation de la distance entre un nœud et un segment.

Pour les minimas x_i trouvés, il faut tester si $D(x_i) < \varepsilon$, mais aussi si le point entre en collision avec la droite à l'intérieur du segment. Le calcul de distance pour la comparaison avec ε doit être effectué à partir des positions des points, et non de l'expression approximée (B.20). Le test d'inclusion au segment peut se faire en vérifiant que les produits scalaires $< \boldsymbol{P}_{12}, \boldsymbol{P}_{13} >$ et $< \boldsymbol{P}_{12}, \boldsymbol{P}_{23} >$ sont de signes différents.

Le cas où la collision survient au bout du segment est déjà géré lors des collisions point / point. Cela permet d'éviter des instabilités numériques lors du calcul de la distance.

B.1.3 Collision entre segments

Pour la détection de la collision entre deux segments, les segments sont représentés par des cylindres de rayon ε. Les segments $s_1 = [n_1, n_2]$ et $s_2 = [n_3, n_4]$ ont pour positions et vitesses respectives $\boldsymbol{p}_i$ et $\boldsymbol{v}_i$, comme l'illustre la figure B.5

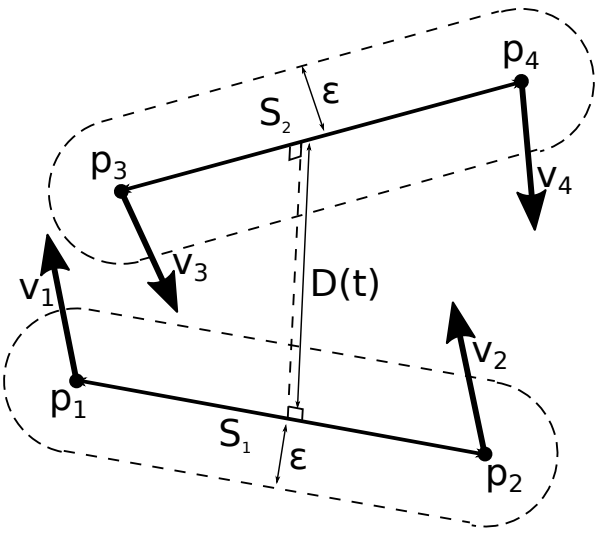


FIGURE B.5 – Distance entre deux segments du réseau de dislocations.

À ce point, il est important de comprendre que les cas particuliers sont déjà gérés par les primitives précédentes. Par exemple, le cas où les segments sont parallèles est déjà géré par la primitive de détection de collision point / segment, et le cas où l'un des segments serait de longueur nulle est pris en compte par le cas point / point. Il ne nous reste plus qu'à gérer la collision entre segments dans le cas général.

La distance se calcule de la manière suivante :

$$D(s_1,s_2) = \frac{\boldsymbol{P}_{13}(t)\cdot(\boldsymbol{P}_{12}(t)\wedge\boldsymbol{P}_{34}(t))}{\|\boldsymbol{P}_{12}(t)\wedge\boldsymbol{P}_{34}(t)\|} \tag{B.21}$$

$$= \frac{det(\boldsymbol{P}_{13}(t),\boldsymbol{P}_{12}(t),\boldsymbol{P}_{34}(t))}{\|\boldsymbol{P}_{12}\wedge\boldsymbol{P}_{34}\|} \tag{B.22}$$

Comme les primitives précédentes couvrent les cas où $\|\boldsymbol{P}_{12}\wedge\boldsymbol{P}_{34}\| = 0$, nous n'avons plus qu'à trouver les racines du déterminant :

$$P(t) = \det(\boldsymbol{P}_{13}(t),\boldsymbol{P}_{12}(t),\boldsymbol{P}_{34}(t)) \tag{B.23}$$

$$= \det(\boldsymbol{p}_{13}+t\ \boldsymbol{v}_{13},\ \boldsymbol{p}_{12}+t\ \boldsymbol{v}_{12},\ \boldsymbol{p}_{34}+t\ \boldsymbol{v}_{34})) \tag{B.24}$$

Par linéarité du déterminant :

$$P(t) = a\cdot t^3 + b\cdot t^2 + c\cdot t + d \tag{B.25}$$

$$a = \det(\boldsymbol{v}_{13},\boldsymbol{v}_{12},\boldsymbol{v}_{34}) \tag{B.26}$$

$$b = \det(\boldsymbol{p}_{13},\boldsymbol{v}_{12},\boldsymbol{v}_{34}) + \det(\boldsymbol{v}_{13},\boldsymbol{p}_{12},\boldsymbol{v}_{34}) + \det(\boldsymbol{v}_{13},\boldsymbol{v}_{12},\boldsymbol{p}_{34}) \tag{B.27}$$

$$c = \det(\boldsymbol{v}_{13},\boldsymbol{p}_{12},\boldsymbol{p}_{34}) + \det(\boldsymbol{p}_{13},\boldsymbol{v}_{12},\boldsymbol{p}_{34}) + \det(\boldsymbol{p}_{13},\boldsymbol{p}_{12},\boldsymbol{v}_{34}) \tag{B.28}$$

$$d = \det(\boldsymbol{p}_{13},\boldsymbol{p}_{12},\boldsymbol{p}_{34}) \tag{B.29}$$

Contrairement aux expressions précédentes, cette distance est algébrique. Ici nous souhaitons trouver les minimums de $|D|$ qui sont les racines de D mais aussi certains des ses extremums locaux. On résout donc 2 équations :

$$P(t) = 0 \tag{B.30}$$
$$P'(t) = 0 \text{ et } P''(t) \text{ de même signe que } P(t) \tag{B.31}$$

Pour chacun de ces minimums, il faut alors tester si $D < \varepsilon$, et si le point de collision est bien à l'intérieur des segments. Le temps retenu pour la collision, est la première de ces valeurs.

B.2 Détail des opérations topologiques avec déplacement

Cette partie détaille les modifications topologiques à effectuer pour chaque type de collision rencontrée. Ces opérations topologiques sont aussi décrites dans le livre de V. Bulatov et al. (10.4) [18].

B.2.1 Collision entre points

Lorsque deux points entrent en collision, ces deux points fusionnent comme le montre la figure B.6.

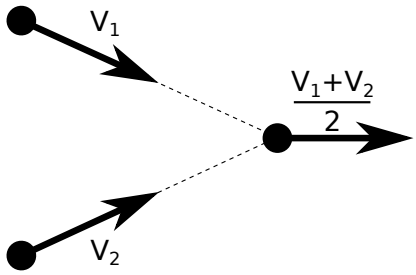


FIGURE B.6 – Collision de deux nœuds.

Position et vitesse du point de collision
Les deux points fusionnent en un unique point :

- sa position est la position des deux points au temps de collision. Comme les deux points peuvent ne pas être exactement confondus, on choisit le milieu du segment formé par les deux points au temps de collision. Cela permet de minimiser la distance aux trajectoires initiales.
- sa vitesse est la moyenne des vitesses des deux points.

Topologies exceptionnelles
Les segments précédemment connectés aux deux nœuds fusionnés sont transférés au nouveau nœud. Dans le cas particulier illustré dans la figure B.7, les nœuds qui entrent en collision sont connectés au même nœud.

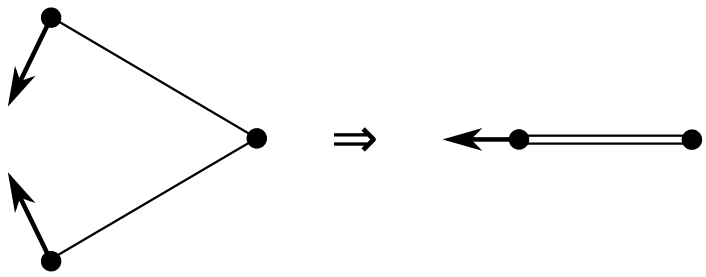


FIGURE B.7 – Fusion d'un segment.

Après la collision, le segment double doit être fusionné en un unique segment dont les propriétés sont les suivantes :

— Vecteur de Burgers : Sommation de Burgers [2]. Si le vecteur de Burgers résultant est nul, le nouveau segment doit être supprimé ;

— Plan de glissement : Doit être déduit en utilisant le vecteur directeur et le vecteur de Burgers

Dans le cas où le segment est annihilé (vecteur de Burgers nul), une autre situation exceptionnelle peut se produire : le nœud nouvellement crée peut être orphelin. Dans le cas où aucun segment n'est connecté au nouveau nœud, il faut le supprimer.

Plans de glissement

Pour que le résultat soit en accord avec la physique des dislocations, il faut que les segments connectés au nouveau nœud restent inclus dans leurs plans de glissement. Pour cela, les segments connectés doivent être inclus dans les plans de glissement après la modification topologique, mais il faut aussi que leur mouvement ultérieur reste inclus dans ces plans :

— La position du nouveau nœud a été choisie afin que la distance du nouveau point à chaque plan de glissement des segments connectés soit inférieure à ε. En effet, le point de collision est à une distance inférieure à ε de la trajectoire des points. De plus, la trajectoire des points respecte les plans de glissement des segments connectés.

— Pour que tous les segments se déplacent dans leurs plans de glissement respectifs, il faut projeter la vitesse du nouveau nœud sur chacun des plans de glissement des segments connectés.

B.2.2 Collision entre un point et un segment

Lorsqu'un point entre en collision avec un segment loin des bords de celui-ci, le point est fusionné avec le segment comme l'illustre la figure B.8.

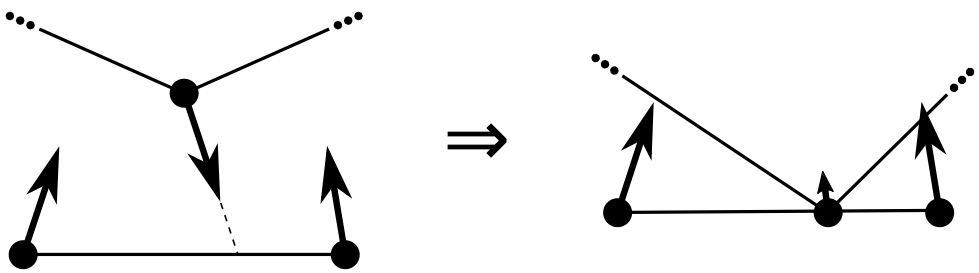


FIGURE B.8 – Collision d'un nœud avec un segment.

2. Attention au sens de parcours de la dislocation

Position et vitesse du point de collision

La position du nouveau nœud est en théorie la position du nœud au temps de la collision. La méthode de calcul de la position est détaillée plus tard dans cette section, dans le paragraphe sur les plans de glissement.

La vitesse du nœud fusionné avec le segment doit être déterminée en fonction des vitesses des objets qui entrent en collision. L'idée la plus simple consiste à prendre comme nouvelle vitesse la moyenne entre la vitesse du nœud n_3 et la vitesse du segment au point de collision :

$$\boldsymbol{v}_3' = \frac{(\alpha \boldsymbol{v}_1 + (1-\alpha)\boldsymbol{v}_2) + \boldsymbol{v}_3}{2} \tag{B.32}$$

avec $\alpha \in [0,1]$ le paramètre de la position du point de collision sur le segment $S = [1-2]$.

Atténuation de l'effet dents-de-scie

Le calcul des vitesses proposé précédemment pose un problème, car il dépend de l'ordre des collisions. Dans l'exemple de la figure B.9, deux lignes de dislocations entrent en contact. Dans cet exemple, tous les nœuds entrent en contact au même moment avec les segments donc il faut choisir un ordre pour gérer ces collisions. Selon l'ordre choisi, le résultat est différent. Par exemple dans la figure, on fusionne les nœuds de la ligne du bas en premier : deux nœuds de la ligne ont une vitesse non nulle. Si on avait fusionné la ligne du haut en premier, les nœuds encore mobiles ne seraient pas les mêmes.

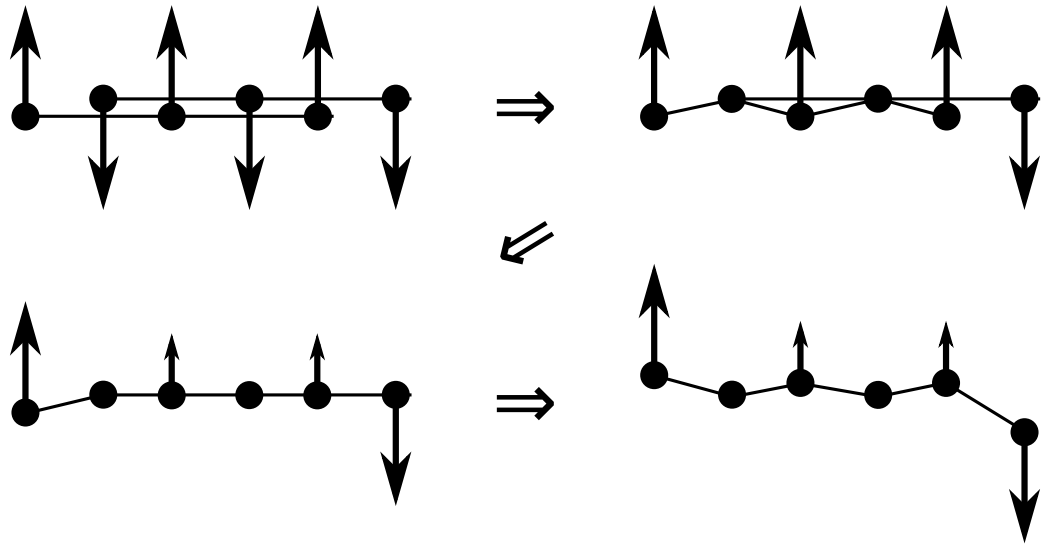


FIGURE B.9 – Collision de deux lignes de dislocation.

Une solution à ce problème est d'utiliser toujours les mêmes valeurs pour $\boldsymbol{v}_1$ et $\boldsymbol{v}_2$ au cours du pas de temps. On peut par exemple utiliser la valeur de la vitesse au début du pas de temps ($\boldsymbol{V0}$). De cette manière, l'effet dents-de-scie est atténué :

$$\boldsymbol{v}_3' = \frac{(\alpha \boldsymbol{V0}_1 + (1-\alpha)\boldsymbol{V0}_2) + \boldsymbol{V0}_3}{2} \tag{B.33}$$

Topologies exceptionnelles et plans de glissement
Comme pour la fusion des nœuds (section B.2.1), il faut gérer les topologies exceptionnelles et vérifier que le mouvement respecte bien les plans de glissement.

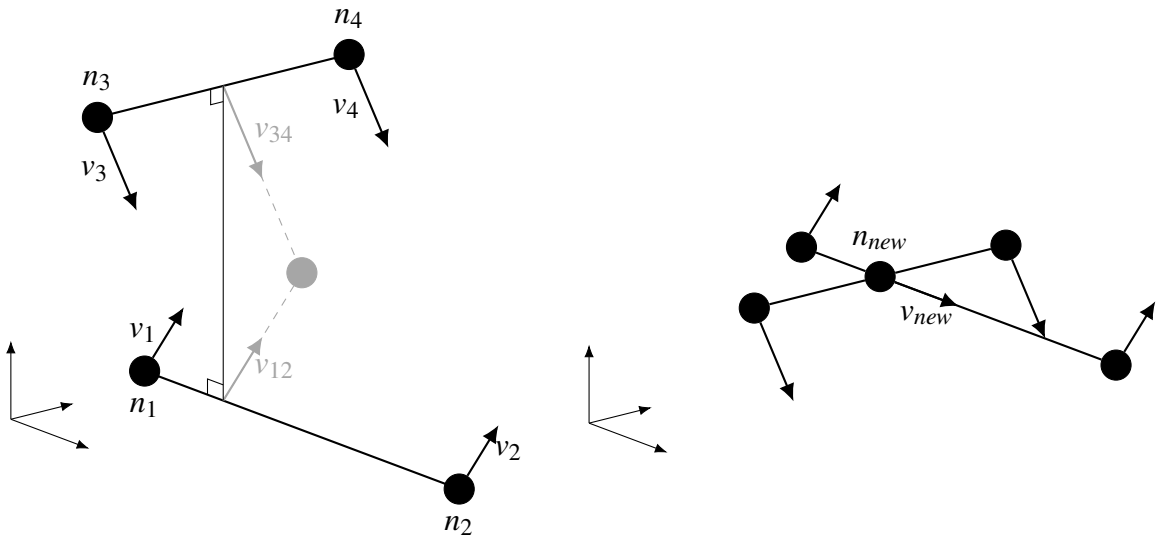
— certaines situations nécessitent de gérer la fusion des segments et l'apparition de nœuds orphelins ;

— la projection des vitesses sur les plans de glissement se déroule comme pour la fusion des nœuds.

La position du nouveau nœud doit faire l'objet d'un traitement particulier pour que les segments connectés appartiennent bien à leurs plans de glissement respectifs. Des imprécisions dans le calcul du temps de collision sont introduites par la détection de collision à ε près, ajoutées aux approximations décrites en section B.1.2. En conséquence, le calcul de la position de collision peut être inexact. La méthode choisie pour réduire l'erreur consiste à calculer la position du nœud au temps de collision, puis de projeter successivement cette position sur les différents plans de glissement afin de s'en rapprocher le plus possible.

B.2.3 Collision entre segments

Lorsque deux segments entrent en contact, ils collisionnent en un unique point, comme le montre la figure B.10.



(a) État initial du réseau.

(b) Réseau après l'opération topologique.

FIGURE B.10 – Collision de 2 segments en 3 dimensions.

Unicité du point de collision
L'hypothèse d'unicité du point de collision est possible car le cas où les segments entrent en contact à plusieurs endroits simultanément sont traités comme des collisions Point/Point ou Point/Segment. En effet, lorsque les segments sont dans le même plan la collision s'effectue forcément par l'extrémité d'un des segments. Par exemple la figure B.9 décrit le traitement des collisions pour deux lignes parallèles.

Position et vitesse du point de collision
La position du nouveau point est la position du point d'intersection des segments au moment de la collision. La vitesse du nouveau point est la moyenne des vitesses des points de collision sur chaque segment.

Topologies exceptionnelles et plans de glissement

Les topologies particulières ayant été traitées lors des collisions Point/Point ou Point/Segment, il n'y a pas de cas particulier à traiter au niveau de la topologie. La vitesse du nouveau point doit tout de même être projetée sur les plans de glissement des segments connectés.

B.3 Détail des opérations topologiques sans déplacement

B.3.1 Collision entre points

Lorsque deux nœuds du réseau de dislocations entrent en collision, ils fusionnent pour former un unique point, comme le montre la figure B.11.

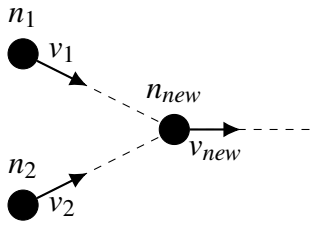


FIGURE B.11 – Fusion de deux nœuds du réseau de dislocations.

Le nouveau nœud est placé où a lieu la collision, c'est-à-dire à la position que prennent n_1 et n_2 au temps de collision t_{col} :

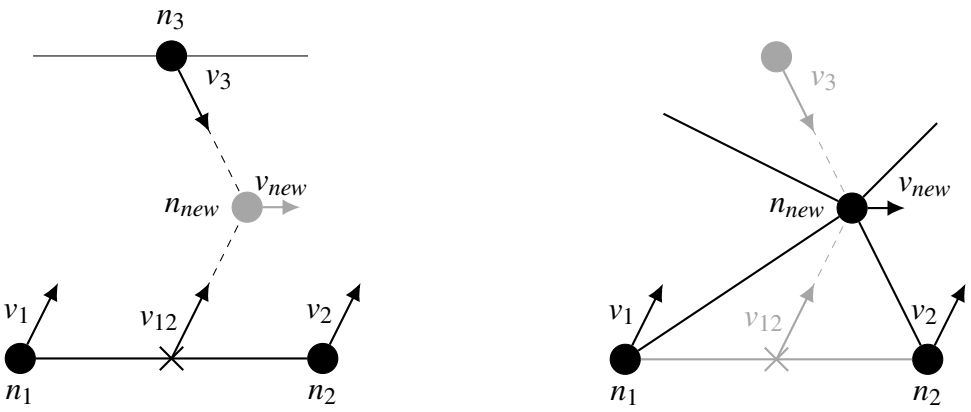
$$\begin{aligned} \boldsymbol{n}_{new} &= \boldsymbol{n}_1 + t_{col} \cdot \boldsymbol{v}_1 \\ &= \boldsymbol{n}_2 + t_{col} \cdot \boldsymbol{v}_2 \end{aligned} \tag{B.34}$$

La nouvelle vitesse est calculée pour que le nœud soit positionné à la fin du pas de temps ($t = dt$) comme s'il s'était déplacé à la vitesse $\frac{\boldsymbol{v}_1+\boldsymbol{v}_2}{2}$ pendant le reste du pas de temps. Cela permet de compenser le déplacement déjà effectué :

$$\boldsymbol{v}_{new} = \frac{\boldsymbol{v}_1 + \boldsymbol{v}_2}{2} \frac{dt - t_{col}}{dt} \tag{B.35}$$

B.3.2 Collision entre un point et un segment

Lors de la collision entre un nœud et un segment, le nœud qui entre en collision est fusionné avec le segment, comme le montre la figure B.12.



(a) État initial du réseau.

(b) Réseau après l'opération topologique.

FIGURE B.12 – Opération topologique effectuée lors de la collision d'un nœud avec un segment.

Seul le point n_3 est modifié pour se transformer en n_{new} : les autres nœuds du réseau restent inchangés. La topologie du réseau est modifiée en remplaçant le segment (n_1, n_2) par les segments (n_1, n_{new}) et (n_2, n_{new}).

Le nouveau nœud est placé où a lieu la collision :

$$\boldsymbol{n}_{new} = \boldsymbol{n}_3 + t_{col} \cdot \boldsymbol{v}_3 \tag{B.36}$$

La vitesse du nouveau point doit s'apparenter à la moyenne entre la vitesse $\boldsymbol{v}_3$ du nœud n_3 et la vitesse $\boldsymbol{v}_{12}$ du point de collision sur le segment (n_1, n_2). Comme auparavant, la vitesse est ajustée pour compenser le déplacement déjà effectué :

$$\boldsymbol{v}_{new} = \frac{\boldsymbol{v}_3 + \boldsymbol{v}_{12}}{2} \frac{dt - t_{col}}{dt} \tag{B.37}$$

$\boldsymbol{v}_{12}$ peut être calculée en déplaçant le segment (n_1, n_2) au temps de collision et en effectuant un calcul de barycentre :

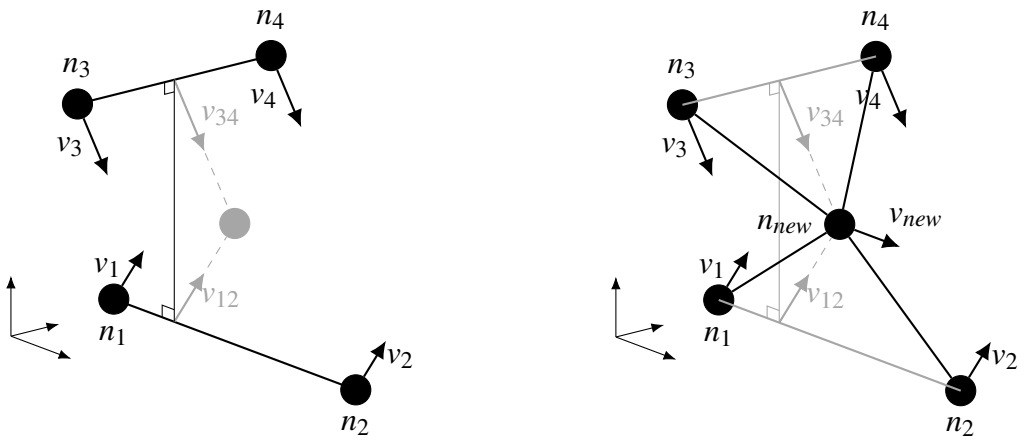
$$\begin{aligned} \boldsymbol{v}_{12} &= \frac{||n_1(t_{col})n_{new}|| \cdot \boldsymbol{v}_1 + ||n_2(t_{col})n_{new}|| \cdot \boldsymbol{v}_2}{||n_1(t_{col})n_2(t_{col})||} \\ \text{avec } n_i(t) &= n_i + t \cdot \boldsymbol{v}_i \end{aligned} \tag{B.38}$$

Si n_1 est n_2 ont déjà été modifiés par d'autres opérations topologiques auparavant, il est préférable (si possible) d'utiliser les vitesses que ces nœuds avaient initialement afin d'éviter la formation de dents de scie [3].

B.3.3 Collision entre segments

Lors de la collision entre deux segments, les deux segments sont fusionnés en un point à l'endroit où a lieu la collision, comme le montre la figure B.13.

3. Voir section B.2.2



(a) État initial du réseau.

(b) Réseau après l'opération topologique.

FIGURE B.13 – Opération topologique effectuée lors de la collision entre deux segments.

Les nœuds existants préalablement ne se déplacent pas, mais la topologie du réseau est modifiée en remplaçant le segment (n_1, n_2) par les segments (n_1, n_{new}) et (n_2, n_{new}) , et le segment (n_3, n_4) par (n_3, n_{new}) et (n_4, n_{new}).

Le nouveau nœud est placé où a lieu la collision. Pour déterminer le lieu de la collision, il faut calculer les paramètres sur les droites du lieu de collision. Les nœuds sont déplacés au temps de la collision, et la formule inspirée de [78] est utilisée :

$$\begin{aligned} a_{12} &= \frac{(\boldsymbol{P}_{31}\cdot\boldsymbol{P}_{34})(\boldsymbol{P}_{34}\cdot\boldsymbol{P}_{12}) - (\boldsymbol{P}_{31}\cdot\boldsymbol{P}_{12})(\boldsymbol{P}_{34}\cdot\boldsymbol{P}_{34})}{(\boldsymbol{P}_{21}\cdot\boldsymbol{P}_{21})(\boldsymbol{P}_{34}\cdot\boldsymbol{P}_{34}) - (\boldsymbol{P}_{34}\cdot\boldsymbol{P}_{12})(\boldsymbol{P}_{34}\cdot\boldsymbol{P}_{12})} \\ a_{34} &= \frac{(\boldsymbol{P}_{31}\cdot\boldsymbol{P}_{34}) + (\boldsymbol{P}_{34}\cdot\boldsymbol{P}_{21})\cdot a_{12}}{||\boldsymbol{P}_{34}||^2} \end{aligned} \tag{B.39}$$

On obtient la position de collision par la formule suivante :

$$\begin{aligned} \boldsymbol{n}_{new} &= a_{12}\cdot\boldsymbol{n}_1(t) + (1-a_{12})\cdot\boldsymbol{n}_2(t) \\ &= a_{34}\cdot\boldsymbol{n}_3(t) + (1-a_{34})\cdot\boldsymbol{n}_4(t) \end{aligned} \tag{B.40}$$

La vitesse du nouveau point est la moyenne des vitesses des points de collision sur les segments. Cette vitesse est ajustée pour compenser le mouvement déjà effectué :

$$\boldsymbol{v}_{new} = \frac{\boldsymbol{v}_{12} + \boldsymbol{v}_{34}}{2}\frac{dt - t_{col}}{dt} \tag{B.41}$$

$\boldsymbol{v}_{12}$ et $\boldsymbol{v}_{34}$ sont calculées en calculant le barycentre en utilisant les paramètres a_{12} et a_{34} :

$$\boldsymbol{v}_{ij} = a_{ij}\boldsymbol{v}_i + (1-a_{ij})\boldsymbol{v}_j \tag{B.42}$$

Annexe C

Partie 4 - Validation et Performances

C.1 Mesures de performances MPI : résultats complets

C.1.1 Temps de calcul en scalabilité faible

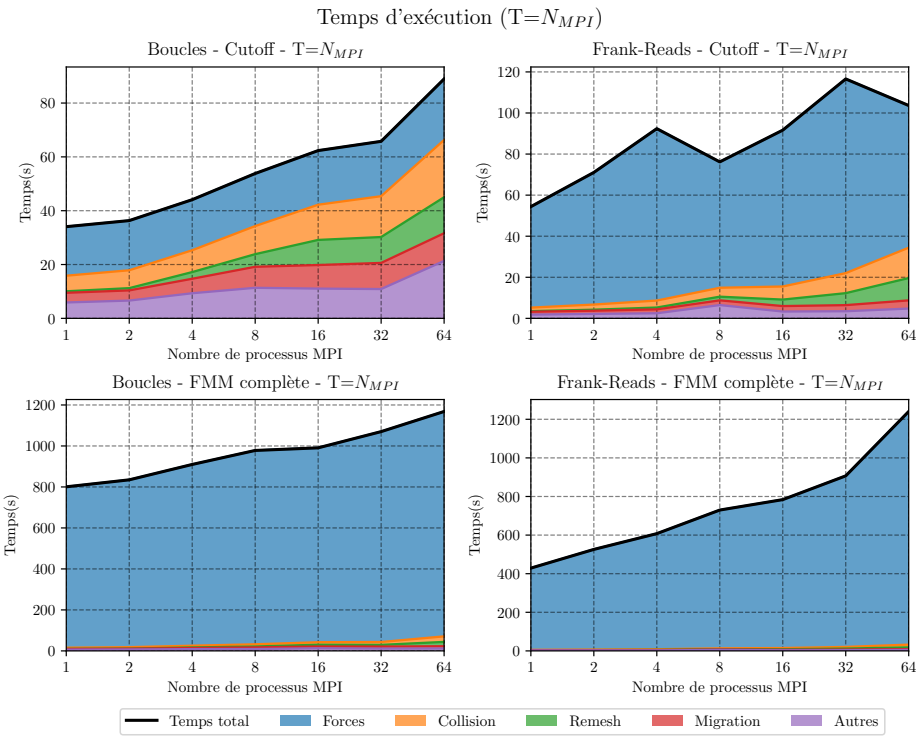


FIGURE C.1 – Temps de calcul en scalabilité faible.

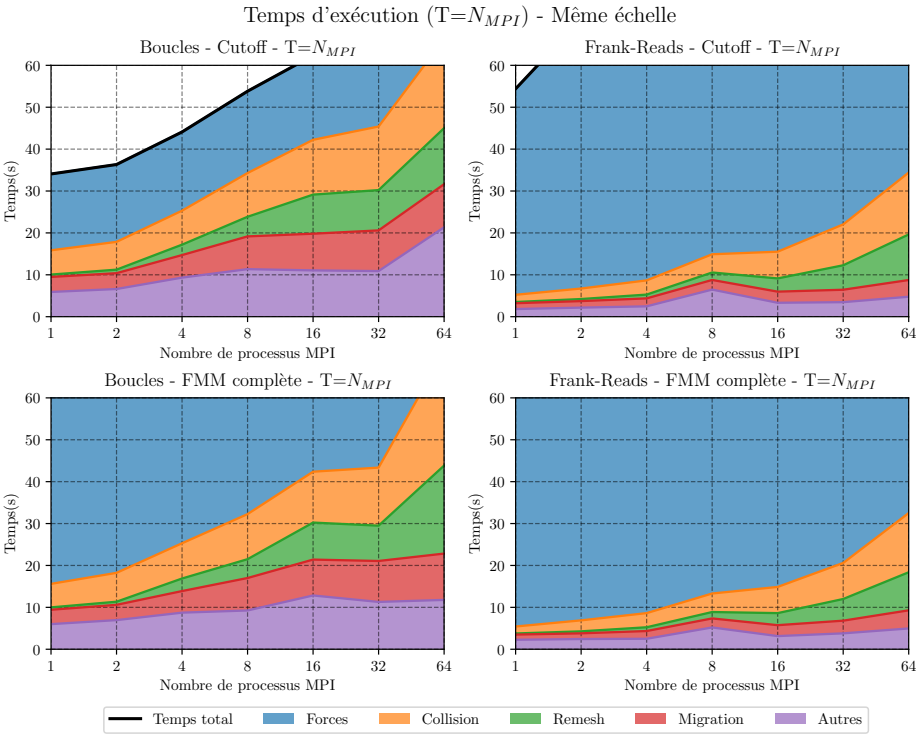


FIGURE C.2 – Temps de calcul en scalabilité faible (tous à la même échelle).

C.1.2 Temps MPI cumulé en scalabilité forte

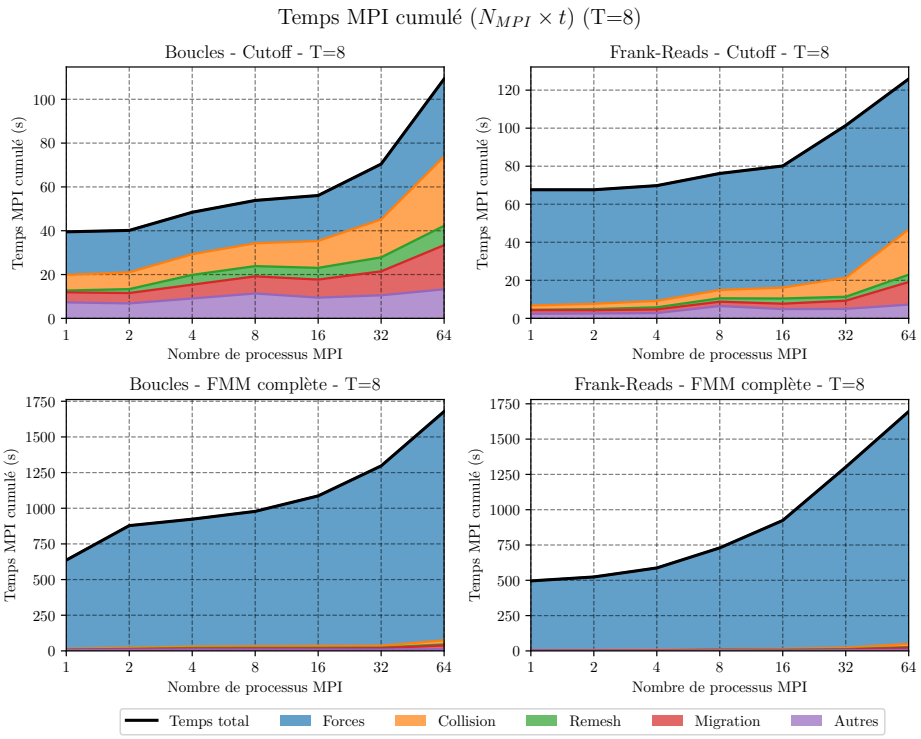


FIGURE C.3 – Temps de calcul en scalabilité forte.

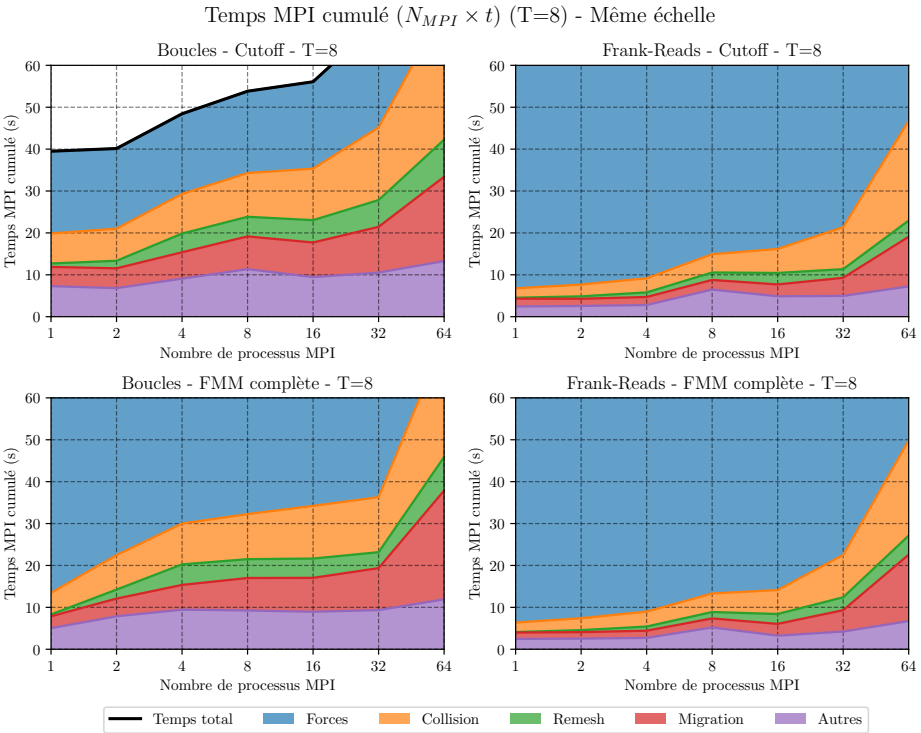


FIGURE C.4 – Temps de calcul en scalabilité forte.

C.1.3 Efficacité

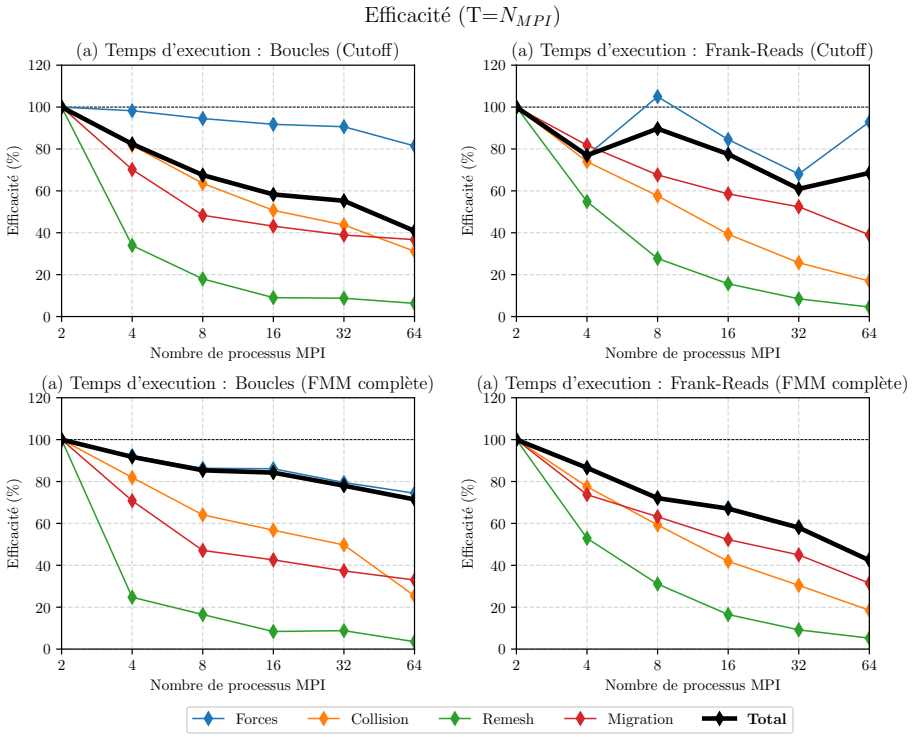


FIGURE C.5 – Efficacité en scalabilité faible.

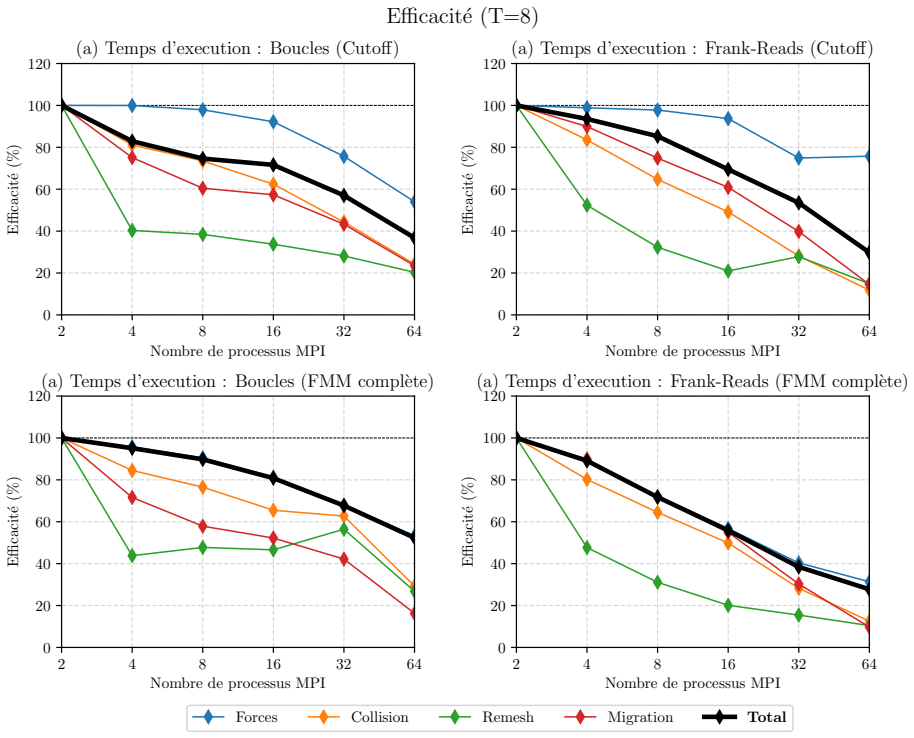


FIGURE C.6 – Efficacité en scalabilité forte.

Bibliographie

[1] Top500.org. https://www.top500.org/. Accessed : 10-2018.

[2] Description de poincaré. https://groupes.renater.fr/wiki/poincare/public/description_de_poincare, 2015. Accessed : 09-2018.

[3] AGULLO, E., BRAMAS, B., COULAUD, O., DARVE, E., MESSNER, M., AND TAKAHASHI, T. Task-based parallelization of the fast multipole method on nvidia gpus and multicore processors. In *GPU Technology Conference* (2013).

[4] AGULLO, E., BRAMAS, B., COULAUD, O., DARVE, E., MESSNER, M., AND TAKAHASHI, T. Task-based fmm for heterogeneous architectures. *Concurrency and Computation : Practice and Experience 28*, 9 (2016), 2608–2629.

[5] ALDER, B. J., AND WAINWRIGHT, T. E. Studies in molecular dynamics. i. general method. *The Journal of Chemical Physics 31*, 2 (1959), 459–466.

[6] ANDERSON, P. M., HIRTH, J. P., AND LOTHE, J. *Theory of dislocations*. Cambridge University Press, 2017.

[7] ARSENLIS, A., CAI, W., TANG, M., RHEE, M., OPPELSTRUP, T., HOMMES, G., PIERCE, T. G., AND BULATOV, V. V. Enabling strain hardening simulations with dislocation dynamics. *Modelling and Simulation in Materials Science and Engineering 15*, 6 (2007), 553.

[8] ARSENLIS, A., RHEE, M., HOMMES, G., COOK, R., AND MARIAN, J. A dislocation dynamics study of the transition from homogeneous to heterogeneous deformation in irradiated body-centered cubic iron. *Acta Materialia 60*, 9 (2012), 3748–3757.

[9] BACON, D. J., AND OSETSKY, Y. N. Multiscale modelling of radiation damage in metals : from defect generation to material properties. *Materials Science and Engineering : A 365*, 1-2 (2004), 46–56.

[10] BARAFF, D. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *ACM SIGGRAPH Computer Graphics* (1989), vol. 23, ACM, pp. 223–232.

[11] BARAFF, D. Dynamic simulation of non-penetrating rigid bodies. Tech. rep., Cornell University, 1992.

[12] BEALL, M. W., AND SHEPHARD, M. S. A general topology-based mesh data structure. *International Journal for Numerical Methods in Engineering 40*, 9 (1997), 1573–1596.

[13] BECKER, E. B., CAREY, G. F., AND ODEN, J. T. Finite elements, an introduction : Volume i. *., 258* (1981), 1981.

[14] BERTI, G. Gral—the grid algorithms library. *Future Generation Computer Systems 22*, 1-2 (2006), 110–122.

[15] BIK, A. J. The software vectorization handbook. *Intel, Hillsboro* (2004).

[16] BLANCHARD, P. *Fast hierarchical algorithms for the low-rank approximation of matrices, with applications to materials physics, geostatistics and data analysis*. PhD thesis, Université de Bordeaux, 2017.

[17] BRECHT, T. On the importance of parallel application placement in numa multiprocessors. In *Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV)* (1993), pp. 1–18.

[18] BULATOV, V., AND CAI, W. *Computer simulations of dislocations*, vol. 3. Oxford University Press on Demand, 2006.

[19] BULATOV, V. V., HSIUNG, L. L., TANG, M., ARSENLIS, A., BARTELT, M. C., CAI, W., FLORANDO, J. N., HIRATANI, M., RHEE, M., HOMMES, G., ET AL. Dislocation multi-junctions and strain hardening. *Nature 440*, 7088 (2006), 1174–1178.

[20] BURGERS, J. M. Physics.—some considerations on the fields of stress connected with dislocations in a regular crystal lattice. i. In *Selected Papers of JM Burgers*. Springer, 1995, pp. 335–389.

[21] CAI, W., ARSENLIS, A., WEINBERGER, C. R., AND BULATOV, V. V. A non-singular continuum theory of dislocations. *Journal of the Mechanics and Physics of Solids 54*, 3 (2006), 561–587.

[22] CALLISTER, W. D., AND RETHWISCH, D. G. *Fundamentals of materials science and engineering*, vol. 21. Wiley New York, 2013.

[23] CANOVA, G., BRÉCHET, Y., KUBIN, L., DEVINCRE, B., PONTIKIS, V., AND CONDAT, M. 3d simulation of dislocation motion on a lattice : application to the yield surface of single crystals. In *Solid State Phenomena* (1993), vol. 35, Trans Tech Publ, pp. 101–106.

[24] CASTRO, M., FERNANDES, L. G., POUSA, C., MÉHAUT, J.-F., AND DE AGUIAR, M. S. Numa-ictm : A parallel version of ictm exploiting memory placement strategies for numa machines. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on* (2009), IEEE, pp. 1–8.

[25] CELES, W., PAULINO, G. H., AND ESPINHA, R. A compact adjacency-based topological data structure for finite element mesh representation. *International journal for numerical methods in engineering 64*, 11 (2005), 1529–1556.

[26] CHENG, D. R., SHAH, V., GILBERT, J. R., AND EDELMAN, A. Fast sorting on a distributed-memory architecture. *http ://hdl.handle.net/1721.1/7418* (2005).

[27] COHEN, J. D., LIN, M. C., MANOCHA, D., AND PONAMGI, M. I-collide : An interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics* (1995), ACM, pp. 189–ff.

[28] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. *Introduction to algorithms*. MIT press, 2009.

[29] COTTRELL, A. Lx. the formation of immobile dislocations during slip. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 43*, 341 (1952), 645–647.

[30] CULLEY, R., AND KEMPF, K. A collision detection algorithm based on velocity and distance bounds. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on* (1986), vol. 3, IEEE, pp. 1064–1069.

[31] DALE, N., AND WALKER, H. M. *Abstract data types : specifications, implementations, and applications*. Jones & Bartlett Learning, 1996.

[32] DENNING, P. J. The locality principle. In *Communication Networks And Computer Systems : A Tribute to Professor Erol Gelenbe*. World Scientific, 2006, pp. 43–67.

[33] DÉPRÉS, C. *Modélisation physique des stades précurseurs de l'endommagement en fatigue dans l'acier inoxydable austénitique 316L*. PhD thesis, Grenoble INPG, 2004.

[34] DEVINCRE, B., AND KUBIN, L. Scale transitions in crystal plasticity by dislocation dynamics simulations. *Comptes Rendus Physique 11*, 3-4 (2010), 274–284.

[35] DROUET, J., DUPUY, L., ONIMUS, F., AND MOMPIOU, F. A direct comparison between in-situ transmission electron microscopy observations and dislocation dynamics simulations of interaction between dislocation and irradiation induced loop in a zirconium alloy. *Scripta Materialia 119* (2016), 71–75.

[36] DUPUY, L. Numodis. http://www.numodis.fr/. Accessed : 09-2018.

[37] DUPUY, L., AND FIVEL, M. A study of dislocation junctions in fcc metals by an orientation dependent line tension model. *Acta Materialia 50*, 19 (2002), 4873–4885.

[38] EDWARDS, H. C., TROTT, C. R., AND SUNDERLAND, D. Kokkos : Enabling many-core performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing 74*, 12 (2014), 3202–3216.

[39] ERICSON, C. *Real-time collision detection*. CRC Press, 2004.

[40] ETCHEVERRY, A. *Simulation de la dynamique des dislocations à très grande échelle*. PhD thesis, Université de Bordeaux, 2015.

[41] FERRONI, F., TARLETON, E., AND FITZGERALD, S. Gpu accelerated dislocation dynamics. *Journal of Computational Physics 272* (2014), 619–628.

[42] FIVEL, M. *Plasticite Cristalline Et Transition D'echelle Cas Du Monocristal*. Ed. Techniques Ingénieur, 2004.

[43] FOREMAN, A., AND MAKIN, M. Dislocation movement through random arrays of obstacles. *Philosophical magazine 14*, 131 (1966), 911–924.

[44] FOREMAN, A., AND MAKIN, M. Dislocation movement through random arrays of obstacles. *Canadian Journal of Physics 45*, 2 (1967), 511–517.

[45] FRANCIOSI, P., BERVEILLER, M., AND ZAOUI, A. Latent hardening in copper and aluminium single crystals. *Acta Metallurgica 28*, 3 (1980), 273–283.

[46] FRANK, F., AND READ JR, W. Multiplication processes for slow moving dislocations. *Physical Review 79*, 4 (1950), 722.

[47] FRANK, F. C., AND READ, W. T. Multiplication processes for slow moving dislocations. *Phys. Rev. 79* (Aug 1950), 722–723.

[48] FRIAS, L., PETIT, J., AND ROURA, S. Lists revisited : Cache conscious stl lists. In *International Workshop on Experimental and Efficient Algorithms* (2006), Springer, pp. 121–133.

[49] FRIEDEL, J. *Dislocations : International Series of Monographs on Solid State Physics*, vol. 3. Elsevier, 2013.

[50] FUCHS, H., KEDEM, Z. M., AND NAYLOR, B. F. On visible surface generation by a priori tree structures. In *ACM Siggraph Computer Graphics* (1980), vol. 14, ACM, pp. 124–133.

[51] GEORGE, P. L. *Automatic mesh generation : applications to finite element methods*. John Wiley & Sons, Inc., 1992.

[52] GHONIEM, N. M. Curved parametric segments for the stress field of 3-d dislocation loops. *Journal of engineering materials and technology 121*, 2 (1999), 136–142.

[53] GILBERT, E. G., JOHNSON, D. W., AND KEERTHI, S. S. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation 4*, 2 (1988), 193–203.

[54] GISSLER, M., IHMSEN, M., AND TESCHNER, M. Efficient uniform grids for collision handling in medical simulators. *GRAPP 11* (2011), 79–84.

[55] GOODRICH, M. T., AND TAMASSIA, R. *Algorithm design and applications*. Wiley Publishing, 2014.

[56] GREENGARD, L., AND ROKHLIN, V. A fast algorithm for particle simulations. *Journal of computational physics 73*, 2 (1987), 325–348.

[57] GUMEROV, N. A., AND DURAISWAMI, R. Fast multipole methods on graphics processors. *Journal of Computational Physics 227*, 18 (2008), 8290–8313.

[58] GÉLÉBART, L., AND DEROUILLAT, J. Amitex-fftp. http://www.maisondelasimulation.fr/projects/amitex/html/, 2016. Accessed : 09-2018.

[59] HOMANN, H., AND LAENEN, F. Soax : A generic c++ structure of arrays for handling particles in hpc codes. *Computer Physics Communications 224* (2018), 325–332.

[60] HOPCROFT, J. E., SCHWARTZ, J. T., AND SHARIR, M. Efficient detection of intersections among spheres. *The International Journal of Robotics Research 2*, 4 (1983), 77–80.

[61] HULL, D., AND BACON, D. J. *Introduction to dislocations*, vol. 37. Elsevier, 2011.

[62] HYDE, R. The fallacy of premature optimization. *Ubiquity 2009*, February (2009), 1.

[63] INTEL. Intel 64 and ia-32 architectures optimization reference manual, 2018.

[64] ISO. *ISO/IEC 14882 :2011 Information technology — Programming languages — C++*, third ed. Sept. 2011.

[65] JAOUL, B. *Etude de la plasticité et application aux métaux*. Presses des MINES, 2008.

[66] JOSUTTIS, N. M. *The C++ standard library : a tutorial and reference*. Addison-Wesley Professional, 2012.

[67] KANTER, D. Intel's sandy bridge microarchitecture. https://www.realworldtech.com/sandy-bridge/, 2010. Accessed : 09-2018.

[68] KARLIK, M., NEDBAL, I., AND SIEGL, J. Microstructure of a reactor pressure vessel steel close to the zones of ductile tearing and cleavage. *Materials Science and Engineering : A 357*, 1-2 (2003), 423–428.

[69] KENNEDY, K., AND MCKINLEY, K. S. Maximizing loop parallelism and improving data locality via loop fusion and distribution. In *International Workshop on Languages and Compilers for Parallel Computing* (1993), Springer, pp. 301–320.

[70] KIRK, B. S., PETERSON, J. W., STOGNER, R. H., AND CAREY, G. F. libmesh : a c++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers 22*, 3-4 (2006), 237–254.

[71] KLOSOWSKI, J. T., HELD, M., MITCHELL, J. S., SOWIZRAL, H., AND ZIKAN, K. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization & Computer Graphics*, 1 (1998), 21–36.

[72] KUBIN, L. *Dislocations, mesoscale simulations and plastic flow*, vol. 5. Oxford University Press, 2013.

[73] KUBIN, L. P., CANOVA, G., CONDAT, M., DEVINCRE, B., PONTIKIS, V., AND BRÉCHET, Y. Dislocation microstructures and plastic flow : a 3d simulation. In *Solid State Phenomena* (1992), vol. 23, Trans Tech Publ, pp. 455–472.

[74] LEBON, C. *Etude expérimentale et simulation numérique des mécanismes de plasticité dans les alliages de zirconium*. PhD thesis, Université de La Rochelle, 2011.

[75] LÉPINOUX, J., MAZIÈRE, D., PONTIKIS, V., AND SAADA, G. *Multiscale phenomena in plasticity : from experiments to phenomenology, modelling and materials engineering*, vol. 367. Springer Science & Business Media, 2012.

[76] LESAR, R., AND RICKMAN, J. Multipole expansion of dislocation interactions : application to discrete dislocations. *Physical Review B 65*, 14 (2002), 144110.

[77] LOMER, W. A dislocation reaction in the face-centred cubic lattice. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 42*, 334 (1951), 1327–1331.

[78] LUMELSKY, V. J. On fast computation of distance between line segments. *Information Processing Letters 21*, 2 (1985), 55–61.

[79] MACNEICE, P., OLSON, K. M., MOBARRY, C., DE FAINCHTEIN, R., AND PACKER, C. Paramesh : A parallel adaptive mesh refinement community toolkit. *Computer physics communications 126*, 3 (2000), 330–354.

[80] MADEC, R., DEVINCRE, B., AND KUBIN, L. From dislocation junctions to forest hardening. *Physical review letters 89*, 25 (2002), 255508.

[81] MADEC, R., DEVINCRE, B., AND KUBIN, L. On the use of periodic boundary conditions in dislocation dynamics simulations. In *IUTAM Symposium on Mesoscopic Dynamics of Fracture Process and Materials Strength* (2004), Springer, pp. 35–44.

[82] MCCALPIN, J. D. Stream benchmark. http://www.cs.virginia.edu/stream/, 1995. Accessed : 09-2018.

[83] MIRTICH, B. Efficient algorithms for two-phase collision detection. *Practical motion planning in robotics : current approaches and future directions* (1997), 203–223.

[84] MONNET, G., VINCENT, L., AND DEVINCRE, B. Dislocation-dynamics based crystal plasticity law for the low-and high-temperature deformation regimes of bcc crystal. *Acta Materialia 61*, 16 (2013), 6178–6190.

[85] MURA, T. *Micromechanics of defects in solids*. Springer Science & Business Media, 2013.

[86] NABARRO, F. R. N., BASINSKI, Z. S., AND HOLT, D. The plasticity of pure single crystals. *Advances in Physics 13*, 50 (1964), 193–323.

[87] NOGARET, T. *Approche multiéchelle des mécanismes de plasticité dans les aciers austénitiques irradiés*. PhD thesis, CEA Saclay, Direction des systèmes d'information, 2008.

[88] OROWAN, E. Zur kristallplastizität. i. *Zeitschrift für Physik 89*, 9-10 (1934), 605–613.

[89] PALMER, I. J., AND GRIMSDALE, R. L. Collision detection for animation using sphere-trees. In *Computer Graphics Forum* (1995), vol. 14, Wiley Online Library, pp. 105–116.

[90] PAN, X., HAN, C. S., DAUBER, K., AND LAW, K. H. A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations. *Ai & Society 22*, 2 (2007), 113–132.

[91] PARUCHURI, P., PULLALAREVU, A. R., AND KARLAPALEM, K. Multi agent simulation of unorganized traffic. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 1* (2002), ACM, pp. 176–183.

[92] PATAKI, N. Safe iterator framework for the c++ standard template library. *Acta Electrotechnica et Informatica 12*, 1 (2012), 17.

[93] PEACH, M., AND KOEHLER, J. The forces exerted on dislocations and the stress fields produced by them. *Physical Review 80*, 3 (1950), 436.

[94] PHILIBERT, J., VIGNES, A., BRECHET, Y., AND COMBRADE, P. *Métallurgie : du minerai au matériau*. Ed. Techniques Ingénieur, 1997.

[95] POLANYI, M. Über eine art gitterstörung, die einen kristall plastisch machen könnte. *Zeitschrift für Physik 89*, 9-10 (1934), 660–664.

[96] QUEYREAU, S., MONNET, G., AND DEVINCRE, B. Slip systems interactions in α-iron determined by dislocation dynamics simulations. *International Journal of Plasticity 25*, 2 (2009), 361–377.

[97] REQUICHA, A. A., AND ROSSIGNAC, J. R. Solid modeling and beyond. *IEEE Computer Graphics and Applications 12*, 5 (1992), 31–44.

[98] SAGAN, H. *Space-filling curves*. Springer Science & Business Media, 2012.

[99] SAMET, H., AND WEBBER, R. E. Hierarchical data structures and algorithms for computer graphics. i. fundamentals. *IEEE Computer Graphics and applications*, 3 (1988), 48–49.

[100] SERRA, A., AND BACON, D. Atomic-level computer simulation of the interaction between dislocations and interstitial loops in α-zirconium. *Modelling and Simulation in Materials Science and Engineering 21*, 4 (2013), 045007.

[101] SHENOY, V., KUKTA, R., AND PHILLIPS, R. Mesoscopic analysis of structure and strength of dislocation junctions in fcc metals. *Physical Review Letters 84*, 7 (2000), 1491.

[102] SHI, X. *Etude par simulations de dynamique des dislocations des effets d'irradiation sur la ferrite à haute température*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2014.

[103] SHI, X., DUPUY, L., DEVINCRE, B., TERENTYEV, D., AND VINCENT, L. Interaction of< 100> dislocation loops with dislocations studied by dislocation dynamics in α-iron. *Journal of Nuclear Materials 460* (2015), 37–43.

[104] SHIN, C.-S., FIVEL, M. C., VERDIER, M., AND KWON, S. Numerical methods to improve the computing efficiency of discrete dislocation dynamics simulations. *Journal of Computational Physics 215*, 2 (2006), 417–429.

[105] SILLS, R. B., AGHAEI, A., AND CAI, W. Advanced time integration algorithms for dislocation dynamics simulations of work hardening. *Modelling and Simulation in Materials Science and Engineering 24*, 4 (2016), 045019.

[106] STEINHAUSER, M. O. *Computational Multiscale Modeling of Fluids and Solids*. Springer, 2017.

[107] STUKOWSKI, A., BULATOV, V. V., AND ARSENLIS, A. Automated identification and indexing of dislocations in crystal interfaces. *Modelling and Simulation in Materials Science and Engineering 20*, 8 (2012), 085007.

[108] TAYLOR, G. I. The mechanism of plastic deformation of crystals. part i.—theoretical. *Proc. R. Soc. Lond. A 145*, 855 (1934), 362–387.

[109] TERENTYEV, D., GRAMMATIKOPOULOS, P., BACON, D., AND OSETSKY, Y. N. Simulation of the interaction between an edge dislocation and a< 100> interstitial dislocation loop in α-iron. *Acta Materialia 56*, 18 (2008), 5034–5046.

[110] TIRTHAPURA, S., SEAL, S., AND ALURU, S. A formal analysis of space filling curves for parallel domain decomposition. In *Parallel Processing, 2006. ICPP 2006. International Conference on* (2006), IEEE, pp. 505–512.

[111] TOURNADRE, L. *Vers une meilleure compréhension des mécanismes de déformation par croissance libre sous irradiation des alliages de zirconium*. PhD thesis, La Rochelle, 2012.

[112] VALSALAM, V., AND SKJELLUM, A. A framework for high-performance matrix multiplication based on hierarchical abstractions, algorithms and optimized low-level kernels. *Concurrency and Computation : Practice and Experience 14*, 10 (2002), 805–839.

[113] VAN DEN BERGEN, G. J. A. *Collision detection in interactive 3D computer animation*. University Press Facilities, 1999.

[114] VOLTERRA, V. Sur l'équilibre des corps élastiques multiplement connexes. In *Annales scientifiques de l'École normale supérieure* (1907), vol. 24, pp. 401–517.

[115] WANG, Z., GHONIEM, N., SWAMINARAYAN, S., AND LESAR, R. A parallel algorithm for 3d dislocation dynamics. *Journal of computational physics 219*, 2 (2006), 608–621.

[116] WEIKUM, G., AND VOSSEN, G. *Transactional information systems : theory, algorithms, and the practice of concurrency control and recovery*. Elsevier, 2001.

[117] WEISER, M. Program slicing. In *Proceedings of the 5th international conference on Software engineering* (1981), IEEE Press, pp. 439–449.

[118] WEST, D. B., ET AL. *Introduction to graph theory*, vol. 2. Prentice hall Upper Saddle River, 2001.

[119] WEYGAND, D., FRIEDMAN, L., VAN DER GIESSEN, E., AND NEEDLEMAN, A. Aspects of boundary-value problem solutions with three-dimensional dislocation dynamics. *Modelling and Simulation in Materials Science and Engineering 10*, 4 (2002), 437.

[120] WILLIAMS, S., WATERMAN, A., AND PATTERSON, D. Roofline : an insightful visual performance model for multicore architectures. *Communications of the ACM 52*, 4 (2009), 65–76.