CSMA 2024
16ème Colloque National en Calcul des Structures
13-17 Mai 2024, Presqu'île de Giens (Var)

Détection des contacts en mémoire massivement distribuée par un double parcours de Bounding Volume Hierarchies

A. Motte[1,2], B.Prabel[2], C. Bovet[1], V. Chiaruttini[1], O. Jamond[2]

[1] *ONERA Châtillon, DMAS/MS2, {antoine.motte,christophe.bovet,vincent.chiaruttini}@onera.fr*
[2] *CEA Saclay DM2S/SEMT/DYN, {benoit.prabel,olivier.jamond}@cea.fr*

Résumé — La détection des contacts dans les méthodes éléments finis est une opération coûteuse et complexe à optimiser avec la décomposition de domaine des simulations massivement parallèles. Nous proposons une approche utilisant des Bounding Volumes Hierarchies dont le parcours est parallélisé afin de réaliser une pré-détection ultra rapide des contacts dans un cadre massivement parallèle.
Mots clés — Détection du contact, Calcul Haute Performance, Décomposition de domaine, Éléments Finis, Bounding Volumes Hierarchies

1 Introduction

Dans le contexte de la simulation par éléments finis de problèmes de grande dimension, la programmation parallèle avec distribution des données est un paradigme incontournable permettant d'accélérer les phases les plus coûteuses des simulations en répartissant la charge de travail sur de nombreuses unités de calcul. En particulier, dans les nombreux domaines nécessitant la simulation du contact [5], la détection des contacts en mémoire distribuée reste une tâche coûteuse et complexe à paralléliser. Les zones de contact sont imprévisibles dans le cas général, ce qui implique un équilibrage de charge dynamique. Par ailleurs, les calculs de détection à effectuer varient en fonction des hypothèses de la simulation, ce qui aboutit à une grande diversité des algorithmes à optimiser. Ces calculs de détection dépendent notamment de deux composantes, la condition de contact et la discrétisation [6].

Pour commencer, la condition de contact définit entre quels types d'éléments et par quelle opération un contact est détecté. On peut par exemple définir qu'un nœud est en contact s'il a traversé une surface, ou bien s'il est localisé à l'intérieur d'un volume. Cette condition de contact doit s'adapter aux hypothèses du problème de contact (auto-contact, contact frottant) et aux types d'éléments des maillages (éléments massifs, coques, poutres). Ensuite, la discrétisation du contact (node-to-node, node-to-face ou face-to-face[1]) définit quels sont les degrés de liberté à contraindre lorsqu'un contact est détecté. Par exemple, avec une discrétisation node-to-face maître/esclave classique, chaque contrainte lie les degrés de liberté d'un nœud du maillage esclave avec les degrés de liberté d'une face du maillage maître. En pratique, la discrétisation choisie influence le choix de la condition de contact et participe donc à la diversité des algorithmes à traiter (intersections, projections...).

La phase de détection consiste à évaluer la condition de contact sur l'ensemble des couples d'éléments des deux maillages pour identifier les zones de contact. En notant N et M les nombres d'éléments des deux maillages, et c le coût de la condition de contact, le coût de calcul de la phase de détection par une approche directe vaut cNM. Ce coût est inenvisageable pour un problème de grande dimension, car le nombre NM de couples à tester et le coût d'évaluation c de la condition de contact sont trop élevés. C'est pourquoi des structures de données à grande échelle sont nécessaires pour réduire fortement le nombre de couples à tester ainsi que le coût des opérations, grâce à une pré-détection rapide et sans faux négatif (c'est-à-dire qui ne rate pas de contact). La discrétisation peut également ajouter un coût de calcul lors de la formulation des conditions de contact, c'est le cas par exemple pour les méthodes Mortar [1].

1. node-to-face et face-to-face sont parfois appelées node-to-segment et segment-to-segment.

Notre travail consiste donc à développer une méthode de détection performante et extensible pour simuler des problèmes de contact de grande dimension en mémoire distribuée, mais aussi générique et robuste afin de traiter toute la diversité des opérations géométriques dans la phase de détection des contacts. Nos développements ont été réalisés dans *Parallel Contact Library (PCL)*, une bibliothèque indépendante et OpenSource en C++ moderne avec une interface générique et facile d'utilisation.

2 Choix d'une structure de données pour la pré-détection

Plusieurs structures de données à grande échelle comme les grilles, les Octrees, les KD-trees ou les Bounding Volumes Hierarchies (BVH) permettent d'accélérer les problèmes de géométrie spatiale [2]. Ces structures de données permettent de séparer la détection en deux étapes. D'abord, la pré-détection permet de trier massivement et rapidement les éléments qui ne sont pas en contact de manière évidente. Puis, la détection élémentaire évalue la condition de contact pour un nombre restreint de couples d'éléments.

Une BVH est une structure de données hiérarchique qui approche un maillage de manière itérative avec des primitives géométriques simples appelées Bounding Volumes (BV). Dans notre bibliothèque *PCL*, nous avons utilisé des Axis Aligned Bounding Boxes comme sur la Figure 1. L'utilisation d'autres BVs (sphères, K-DOPs...) reste possible via des développements supplémentaires. De gauche à droite sur la Figure 1, la première approximation la plus grossière simplifie le maillage en un unique BV et forme la racine de l'arbre. Les itérations suivantes multiplient le nombre de BV par deux et forment les étages intermédiaires de l'arbre. Les deux dernières images à droite de la Figure 1 représentent le dernier étage de la BVH où chaque BV englobe un unique élément du maillage, puis le maillage réel.

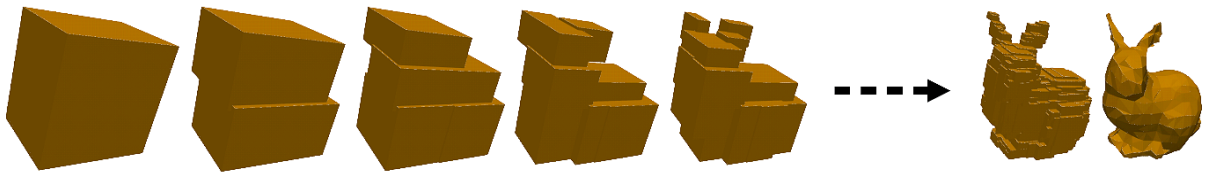


FIGURE 1 – Approximations itératives du maillage par un Bounding Volume Hierarchy

Nous avons choisi d'utiliser les BVHs pour 3 avantages principaux qui répondent aux besoins de généricité et de performance de la phase de détection :

◦ Toutes les topologies d'éléments (nœuds, segments, faces, volumes) peuvent être approchées, il suffit d'englober ces derniers par un BV pour pouvoir construire une BVH. Pour la plupart des autres structures de données à partitionnement spatial, bien que les nuages de points sont faciles à répartir dans des régions de l'espace [3], ce n'est pas le cas pour les éléments non ponctuels. Ces derniers ne se trouvant pas nécessairement dans des régions uniques de l'espace, ils doivent être dupliqués ou découpés. Cela implique des surcoûts à la construction ainsi qu'à l'exploration de la structure.

◦ Les BVHs peuvent être mises à jour sans reconstruire la hiérarchie : il suffit de mettre à jour uniquement les coordonnées des BVs en commençant par les feuilles puis en remontant jusqu'à la racine. Pour les autres structures, soit la mise à jour nécessite une reconstruction totale de la structure, soit la qualité de la structure se dégrade plus rapidement qu'avec une BVH.

◦ Les BVHs peuvent être utilisées des deux côtés du contact puisqu'elles approchent toutes les topologies d'éléments et que leur mise à jour est plus efficace qu'une reconstruction totale. Ceci permet de bénéficier de la structure hiérarchique sur l'ensemble des éléments du problème, là où d'autres structures ne permettent d'approcher que les éléments d'une des deux surfaces de contact.

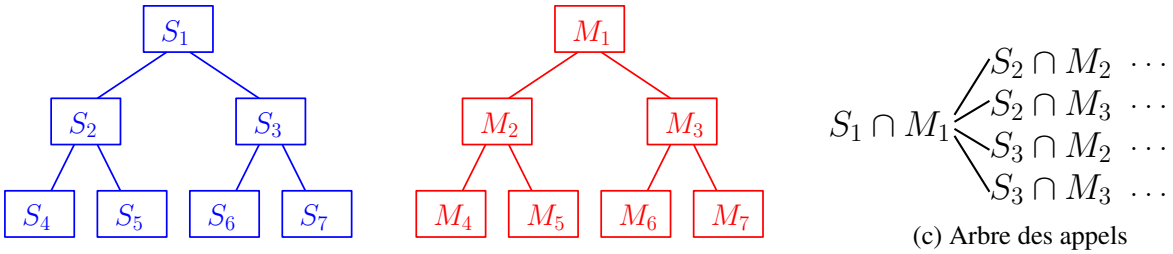
La méthode de détection que nous avons développé consiste à constuire deux BVHs en début de simulation, une de chaque côté du contact (pour un couple de surfaces en contact). À chauqe pas de temps, seules les coordonnées des BVs sont mises à jour avant pour la phase de détection. Pour explorer simultanément les deux BVHs, nous avons développé un algorithme de double parcours détaillé dans la section suivante.

3 Algorithme de double parcours de BVHs pour la pré-détection

L'algorithme présenté dans cette section réalise un double parcours d'arbres afin d'identifier les paires d'éléments de deux maillages qui vérifient une condition de contact (voir section 1). On se place ici dans l'hypothèse d'un unique couple de maillages slave et master[2]. La généralisation à plusieurs couples slave/master et à l'auto-contact d'un unique maillage est en cours de développement. Pour commencer, on présente un noyau séquentiel qui sera ensuite utilisé dans les versions parallèles ensuite.

3.1 Approche séquentielle

Nous avons apporté une modification mineure à l'algorithme de double parcours proposé par Laursen et Yang [4], afin d'explorer les deux arbres simultanément et non alternativement. L'algorithme consiste à intersecter les deux arbres en commençant par l'intersection des deux racines (voir Figure 2). Si ces racines s'intersectent, l'exploration des arbres doit se poursuivre en intersectant deux à deux les BVs du prochain étage de chaque arbre. Le schéma d'appel des intersections prend alors la forme d'arbre quaternaire[3] (voir Figure 2c).



(a) BVH slave (b) BVH master (c) Arbre des appels (intersections de BVs)

FIGURE 2 – Algorithme de double parcours de BVHs

3.2 Approche parallèle

Les développements présentés dans cette section se placent dans le cadre d'une simulation par éléments finis en mémoire distribuée utilisant MPI. On hérite d'une distribution des données imposée par le partitionnement de maillage du code appelant. Chaque tâche MPI n'a donc qu'une vision partielle des données, elle n'a accès qu'à un sous-domaine des maillages de la simulation. En se plaçant dans le cadre d'un unique couple de maillages slave et master, chaque sous-domaine possède une sous-partie du maillage slave et une sous-partie du maillage master, dont les tailles sont variables sur l'ensemble des sous-domaines[4].

L'approche parallèle consiste à construire deux BVHs distribuées slave et master. Sur chaque tâche MPI, la partie locale de ces deux BVHs est construite et mise à jour sur le sous-domaine local avec un algorithme séquentiel indépendant des autres tâches MPI. En revanche, lors de la phase de détection, l'intersection des arbres slave avec les arbres master entres toutes les tâches MPI nécessite un algorithme parallèle. En effet, pour chaque tâche MPI, seul le couple d'arbres slave/master local peut être directement intersecté grâce à l'algorithme séquentiel présenté en section 3.1. Pour tous les autres couples, il faut d'abord ramener les deux arbres slave et master sur le même processeur avant d'utiliser l'algorithme d'intersection séquentiel. Pour simplifier les premiers développements de l'algorithme, nous avons choisi de façon arbitraire de ramener les arbres master sur les processeurs possédant les arbres slave. Il est nécessaire de trouver un compromis entre le nombre de communications à effectuer et le volume de données envoyées. Pour minimiser le nombre de communications, les arbres master pourraient être communiqués en totalité dès le début de l'algorithme. À l'opposé, pour minimiser le volume de données, seuls les nœuds des arbres master dont on a immédiatement besoin lors de l'exploration pourrait être envoyés.

2. L'algorithme s'applique aussi bien aux discrétisations symétriques qu'aux discrétisations asymétriques. Les noms slave et master identifient les deux côtés du contact par analogie avec les discrétisations asymétriques, sans perte de généralité.

3. Un arbre quaternaire est un arbre dont chaque nœud possède au plus 4 nœuds fils.

4. Les sous-parties des maillages slave et master de chaque sous-domaine sont éventuellement vides. Plusieurs sous-domaines peuvent ne posséder que des éléments slave, que des éléments master, ou aucun des deux.

Nous avons cherché à trouver un compromis entre ces deux approches en développant un algorithme permettant d'envoyer les arbres master en plusieurs communications contenant chacune un nombre d d'étages des arbres (voir Figure 3), d étant un paramètre de l'algorithme. Lorsque qu'un BV M_i doit être communiqué pour faire une intersection, un sous-arbre de profondeur d partant de M_i est envoyé. Ainsi, l'exploration pourra continuer sans interruption jusqu'à atteindre les feuilles de ce sous-arbre. On réitère l'opération avec les nouveaux BVs nécessaires à la poursuite de l'exploration. De cette manière, on ne paie qu'une seule fois la latence pour envoyer d étages, et le coût mémoire de l'algorithme reste maîtrisé.

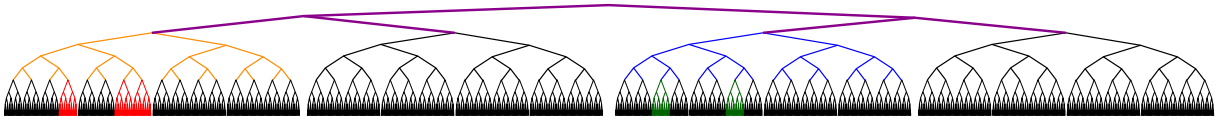


FIGURE 3 – Exemple de schéma de communication d'un arbre master :
Première communication des 3 premiers étages à tous les procs du réseau (violet)
Seconde communication de sous-arbres de 4 étages à deux procs X (orange) et Y (bleu)
Troisième communication des étages restants aux deux procs X (rouge) et Y (vert)
De nombreuses branches n'ont pas eu besoin d'être communiquées (noir)

L'algorithme réalise donc un parcours itératif, chaque itération comporte une phase de passage de commandes, une phase de communication de sous-arbres de d étages, puis une phase d'exploration de ces sous-arbres. La première communication est extrêmement dense, puisque chaque tâche MPI doit envoyer les premiers étages de son arbre master local à toutes les autres tâches MPI. Pour éviter un coût trop élevé de cette première communication, on définit spécifiquement une profondeur d_0, généralement strictement inférieure à d, qui permet d'envoyer un faible nombre d'étages pour démarrer l'exploration. Dans le cas général, cette première exploration de d_0 étages va réduire fortement la densité de la communication suivante car les arbres slave et master éloignés dans l'espace terminent immédiatement leur intersection. Finalement, on obtient l'algorithme 1 qui est lancé sur chaque tâche MPI de rang i avec les arbres locaux slave et master S_i et M_i. On utilise la fonction séquentielle *ParcoursPartiel*(**input :** S_i, M_j, d, **output :** newNeededNodes, masterLeavesReached) qui réalise une intersection entre S_i et M_j de façon partielle sur une profondeur d à partir de la fin de l'exploration précédente. Cette intersection crée une nouvelle liste masterNeededNodes contenant les nœuds internes (non feuilles) atteints dans l'arbre M_j, et complète la liste masterLeavesReached contenant les nœuds feuilles atteints dans l'arbre M_j.

Algorithm 1 Double parcours de BVHs en mémoire distribuée

1: **procedure** PARALLELPREDETECTION(S_i, M_i, d_0, d) ▷ S_i et M_i ont les arbres slave et master locaux
Initialisation
2: **for** $j \neq i$ **do** neededInternalNodes$_{ij} = \emptyset$, masterLeavesReached$_{ij} = \emptyset$
3: masterLeavesReached$_{ii}$ = $S_i \cap M_i$ ▷ intersection séquentielle des arbres slave/master locaux
4: Communication des d_0 premiers étages master
5: ⇒ Réception des d_0 premiers étages de tous les arbres M_j, $j \neq i$
6: **for** $j \neq i$ **do** ParcoursPartiel(S_i, M_j, d_0, neededInternalNodes$_{ij}$, masterLeavesReached$_{ij}$)
Début du parcours itératif
7: **while** neededInternalNodes est non vide **do**
Passage des commandes
8: Communication des neededInternalNodes$_{ij}$ aux tâches MPI $j \neq i$
9: ⇒ Réception des neededInternalNodes$_{ji}$ des autres tâches MPI $j \neq i$
Exécution des commandes
10: Communication des sous-arbres de profondeur d à partir des neededInternalNodes$_{ji}$
11: ⇒ Réception des sous-arbres de profondeur d demandés dans M_j $\forall j \neq i$
Poursuite du parcours
12: **for** $j \neq i$ **do** ParcoursPartiel(S_i, M_j, d, neededInternalNodes$_{ij}$, masterLeavesReached$_{ij}$)
13: **return** masterLeavesReached$_{ij}$ $\forall j$ ▷ nœuds feuilles atteints dans tous les M_j par $S_i \cap M_j$

3.3 Stratégies de communication

L'algorithme de pré-détection parallèle présenté en section 3.2 présente de nombreuses phases de communication qui peuvent être réalisées par différentes fonctions de MPI. Ainsi, nous avons développé 3 variantes pour le schéma de communication employé dans l'algorithme 1 :

◦ La première variante utilise des collectives denses : les communications de la phase d'initialisation utilisent les fonctions dérivées de MPI_Allgather, celles de la boucle du parcours itératif utilisent des fonctions dérivées de MPI_Alltoall. Chaque communication nécessite au préalable la connaissance de la taille des messages envoyés, impliquant à chaque fois une communication supplémentaire. On a ainsi deux collectives appelées dans l'initialisation, puis deux collectives appelées pour le passage des commandes, et enfin deux collectives pour l'exécution des commandes (voir algorithme 1).

◦ La deuxième variante utilise un graphe de communication et des neighborhood collectives : les communications de la phase d'initialisation utilisent les fonctions dérivées de MPI_Neighbor_allgather, celles de la boucle du parcours itératif utilisent des fonctions dérivées de MPI_Neighbor_alltoall. Ceci expose explicitement à MPI la structure du graphe de communication, ce qui permet a priori de bénéficier d'optimisations spécifiques lorsque ce graphe devient creux. En effet, au fur et à mesure de l'exploration, la localisation des zones de contact s'affine et seules quelques paires de processeurs ont besoin de continuer à communiquer des sous-arbres afin d'identifier exactement où ont lieu les contacts. Cela signifie que la densité du graphe de communication décroit au cours du temps, jusqu'à atteindre un niveau de densité minimal en fin d'exploration qui dépend de la répartition des zones de contact sur les processeurs. La réduction du graphe s'effectue avant le passage des commandes (voir algorithme 1).

◦ La troisième variante utilise des One-Sided Communications (OSC) qui communiquent par Remote Direct Memory Access (RDMA[5]) : il n'y a plus d'étape de passage de commandes, les sous-arbres sont directement lus à distance avec MPI_Get. Contrairement aux deux variantes précédentes, les OSC ne nécessitent pas de synchronisation entre les tâches MPI, ce qui permet de supprimer les temps d'attente et de libérer plus rapidement les tâches MPI faiblement chargées. À la construction d'une BVH, chaque tâche MPI fournit un accès à distance à ses données locales (MPI_Window). Aucun problème d'accès concurrent à la mémoire ne peut se produire, car l'algorithme 1 n'a besoin que d'un accès en lecture seule. Cependant, les OSC ajoutent des contraintes aux structures de données. La localisation des données dans la mémoire distante doit être connue[6] et pour assurer la performance de la lecture à distance, leur répartition doit être la plus contiguë possible[7]. Pour cela, la numérotation des nœuds a été choisie telle que tous les sous-arbres d'intérêt[8] sont contigus en mémoire.

Les deux premières approches utilisant des collectives denses et des neighborhood collectives ont été développées dans *PCL*. La troisième approche utilisant le RDMA est en cours de développement.

4 Analyse des performances et de l'extensibilité

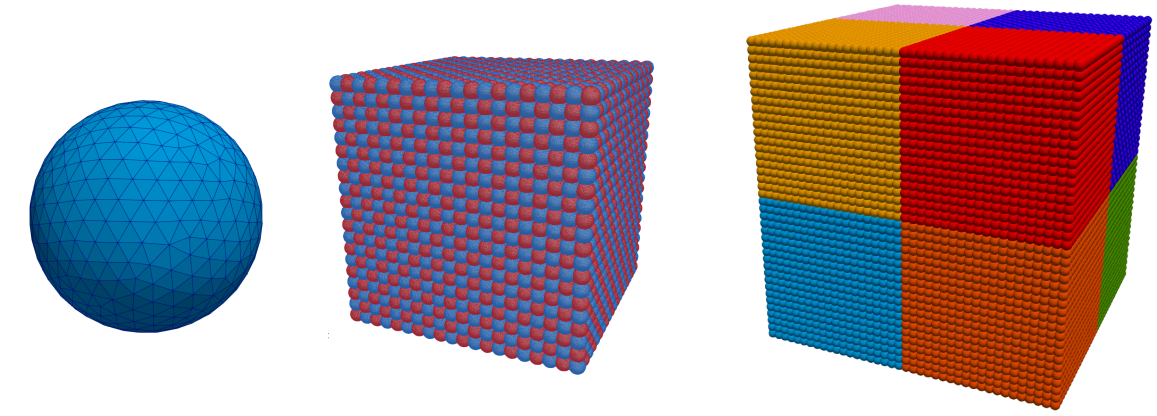
L'analyse présentée ici porte sur un problème de pré-détection seule, sans détection élémentaire et sans simulation mécanique. Les calculs sont lancés sur le Très Grand Centre de Calcul (TGCC) du CEA, sur les nœuds Skylake d'Irene, chacun possédant 2x24 cœurs Intel Skylake@2.7GHz (AVX512) et 180Go de RAM. Le problème étudié comporte un motif par tâche MPI composé de sphères disposées sur une grille cartésienne 3D régulière et espacées de façon à entrer en contact avec leurs voisines (hors diagonales, voir Figure 4b). Ce motif en forme de cube est lui-même placé sur une grille cartésienne 3D régulière pour former un grand cube de motifs avec toutes les autres tâches MPI (voir Figure 4c). Chaque sphère contient 1038 triangles, chaque motif est un cube de $20^3 = 8000$ sphères. C'est donc un problème à 8×10^6 éléments par tâche MPI, volontairement chargé pour tester l'algorithme en conditions difficiles.

5. La disponibilité du RDMA dépend des protocoles supportés par la carte réseau : les protocoles RDMA over Converged Ethernet (RoCE), internet Wide Area RDMA Protocol (iWARP) et InfiniBand supportent le RDMA.

6. Le RDMA accède aux données distantes via la carte réseau sans impliquer le CPU, ce qui ne permet pas de suivre un chemin de pointeurs ou d'indices à distance. Ainsi, pour accéder à des sous-arbres distants sans interruption, il faut que les nœuds de ces sous-arbres aient une localisation calculable à l'avance (par rapport à la racine du sous arbre par exemple).

7. Les OSC génèrent une instruction de lecture par bloc contigu en mémoire auquel il faut accéder. Si la mémoire est très morcelée, le nombre important d'instructions réduit fortement la performance de la lecture des données.

8. Les sous-arbres d'intérêt sont ceux de profondeur d démarrant à une profondeur $d_0 + kd$, $k \in \mathbb{N}^*$ (voir section 3.2)



(a) Sphère de base du motif (≈ 1000 triangles)

(b) Motif (slave/master) de sphères sur chaque tâche MPI

(c) Maillage distribué sur toutes les tâches MPI (ici 8 sous-domaines, un par couleur)

FIGURE 4 – Cas test d'extensibilité faible

4.1 Extensibilité et influence des paramètres d_0 et d

Les arbres envoyés dans le réseau sont envoyés de façon partielle. Les paramètres d_0 et d présentés en section 3.2 définissent la profondeur des sous-arbres envoyés dans l'algorithme, respectivement à l'initialisation de celui-ci puis dans le reste du parcours. Afin de tester l'influence de ces deux paramètres, le cas test d'extensibilité faible de la Figure 4 a été réalisé avec de nombreux couples (d_0, d). La Figure 5 ci-dessous montre le temps CPU, obtenu en faisant la moyenne sur toutes les tâches MPI et sur 10 exécutions consécutives de l'algorithme de pré-détection parallélisé, ainsi que le speed-up et l'efficacité, tous deux évalués par rapport au temps de calcul de 8 tâches MPI. Les différents couples (d_0, d) dont notés $d_0 + n \times d$ où n est le nombre d'itérations dans la boucle principale de l'algorithme. Les tests sont réalisés avec des collectives standardes. Les tests menés avec des neighborhood collectives n'ont pas montré d'amélioration de performance significative pour le moment et sont actuellement en cours d'approfondissement.

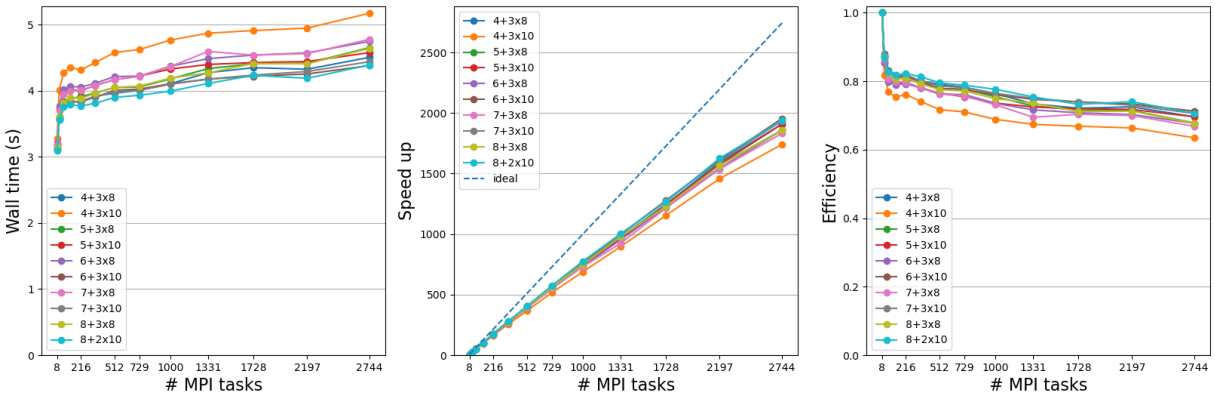


FIGURE 5 – Extensibilité faible de l'algorithme 1 sur le cas test présenté en Figure 4

Un second cas test nécessitant davantage de communication a été testé en prenant un motif de taille 40^3 en retirant les sphères intérieures, seules les sphères à la bordure du motif (voir Figure 4b) sont conservées. Ainsi, chaque motif est composé de 9128 sphères[9] et donc d'environ 9×10^6 triangles. Cette taille est comparable au premier cas test, mais le volume de données échangées est bien plus important car toutes les sphères sont sur le bord du motif.

9. Le motif est la surface d'un cube de 40 sphères de côté. Il contient donc 8×1 sphères dans les coins, 12×38 sphères sur les arêtes, et 6×38^2 sphères sur les faces du cube. $8 + 12 \times 38 + 6 \times 38^2 = 9128$.

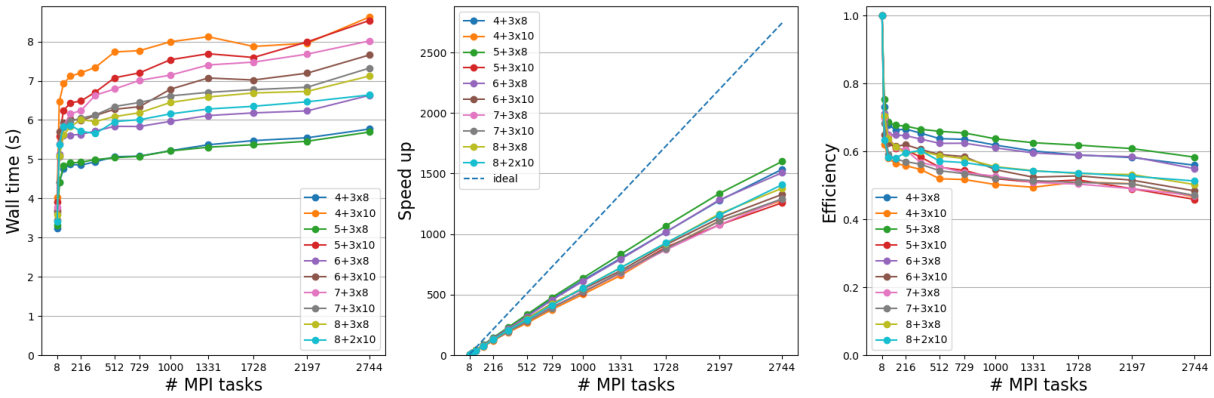


FIGURE 6 – Extensibilité faible sur un cas avec davantage de communications

Ces premiers résultats montrent que les paramètres d_0 et d ont une influence de l'ordre de 20% à 40% sur le temps de calcul total de la pré-détection. L'efficacité du meilleur couple (d_0, d) est de l'ordre de 70 à 80% (Figure 5) pour le cas présenté en Figure 4, et de 60 à 70% pour le cas sans sphères intérieures, plus lourd en communications (Figure 6). La chute de l'efficacité de 8 à 64 tâches MPI s'explique en partie par l'augmentation des échanges de données par tâche MPI (à 8 coeurs, chacun n'a que 3 voisins), et par la nécessité de passer par le réseau à partir de 48 coeurs. Plusieurs pistes sont développées pour améliorer les temps de calcul et l'efficacité : une analyse des volumes de données strictement nécessaires à l'exploration par rapport au volume de données envoyées dans le réseau, l'impact de la latence et de la bande passante sur le choix optimal de d_0 et d, un meilleur paramétrage des profondeurs des sous-arbres envoyés, et enfin la faisabilité et l'optimisation de l'algorithme utilisant des RDMA. Tous ces points seront approfondis dans nos travaux à venir.

Références

[1] B. R. Akula, *Extended mortar method for contact and mesh-tying applications* (2019)

[2] C. Ericson, *Real-Time Collision Detection* (2005)

[3] W. Fan et al., *An octree-based proxy for collision detection in large-scale particle systems* (2013)

[4] Tod A. Laursen et al., *A contact searching algorithm including bounding volume trees applied to finite sliding mortar formulations* (2008)

[5] P. Wriggers, *Computational contact mechanics* (2006)

[6] V. A. Yastrebov et al., *Numerical Methods in Contact Mechanics* (2013)