



Optimisation et parallélisation d'un algorithme de détection de contact : Potatoes

Rencontres Arcane 2023

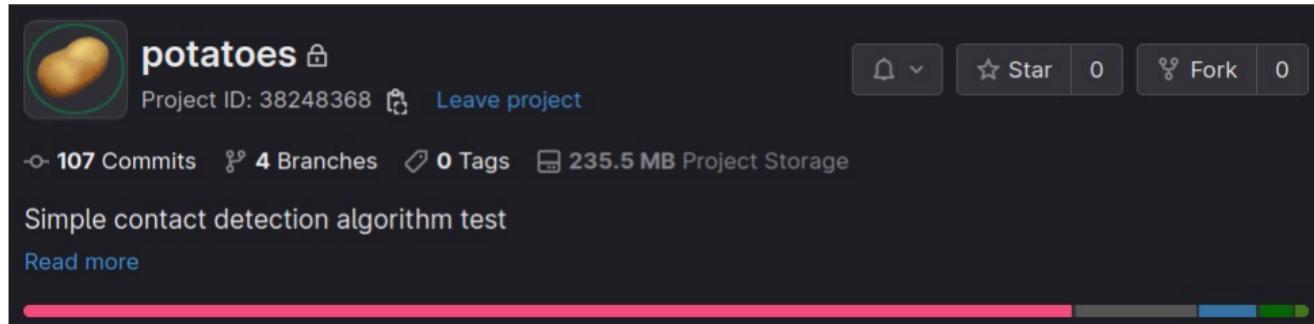
Alexandre l'Héritier
Lundi 17 avril 2023



Table des matières

- 1 Application de base : Potatoes
- 2 Arcanisation de Potatoes : APatoes
- 3 Optimisations et parallélisation
- 4 Temps et conclusion

1 Application de base : ■ Potatoes



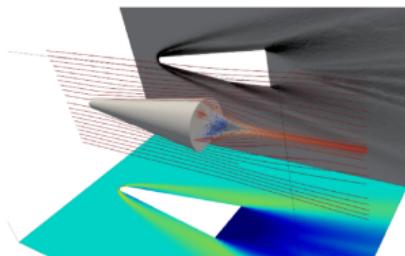
- ▶ Développée par Sixtine Michel au CEA en 2022,
- ▶ Actuellement sur un dépôt Gitlab privé, bientôt disponible en public :
<https://gitlab.com/sixtinemichel/potatoes>

Hydrodynamic numerical simulation

An hydrodynamic numerical simulation in the broad sense can be decompose in two main processes:

- The numerical resolution of the hydrodynamical problems:

Goals: Numerical methods to simulate hyperveloce compressible flow & the displacement or deformation of materials

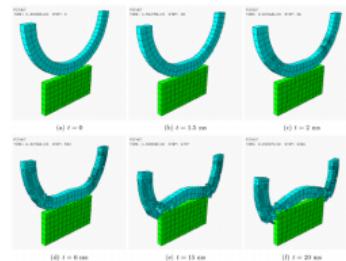


Maneuverable Re-entry Vehicle.

A. Chan et al. (2021)

- The contact treatment (when it occurs)

Goals: Robust and adaptable contact detection procedure & treatment



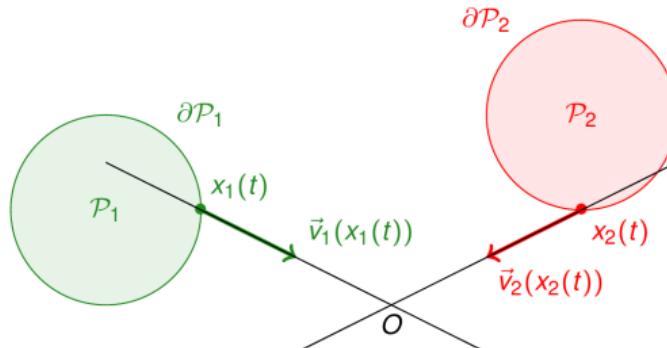
Contact deformation and fracture of materials.

F. Casadei et al. (2016)

Start from a continuous space

Remembering objectives: find a robust contact detection algorithm in terms of accuracy and CPU-time, parallelisable and usable for Lagrangian schemes.

Defintion: A contact will appear at the time $t^* \in \mathbb{R}_+$ if and only if (iff) $\partial\mathcal{P}_1(t^*) \cap \partial\mathcal{P}_2(t^*) \neq \emptyset$.



Representation of two nodes $x_1(t) \in \partial\mathcal{P}_1$ and $x_2(t) \in \partial\mathcal{P}_2$, and their respective velocity fields $\vec{v}_1(x_1(t))$ and $\vec{v}_2(x_2(t))$.

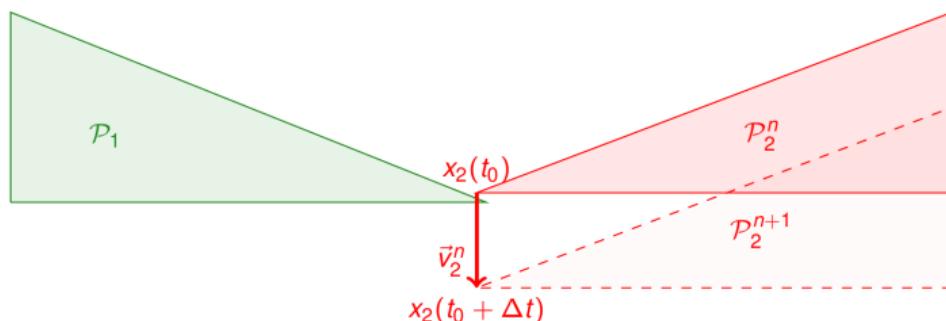
Two-dimensional numerical tests

NTS pathological test

The initial position of x_2 is $x_2(t_0) = (0.98, 0.1)^T$ & $v_2(x_2, t_0) = (0, -1)^T$.

The position of the right corner of \mathcal{P}_1 is $(1, 0)$.

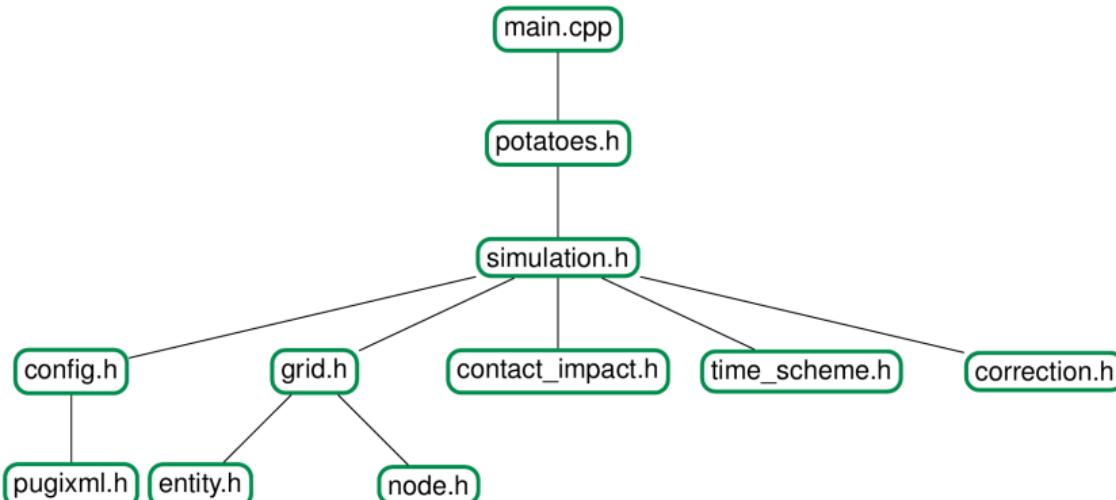
$\Delta x = 0.05$. \mathcal{P}_2 jumps over \mathcal{P}_1 during the first iteration.



Representation of the contact test case.

A word about the code

The *potatoes* library



S. Michel (2022)

potatoes - Potatoid hydrOdynamics and conTAct deTechtiOn procedureES

<https://gitlab.com/sixtinemichel/potatoes>

2. Arcanisation de Potatoes : APatoes

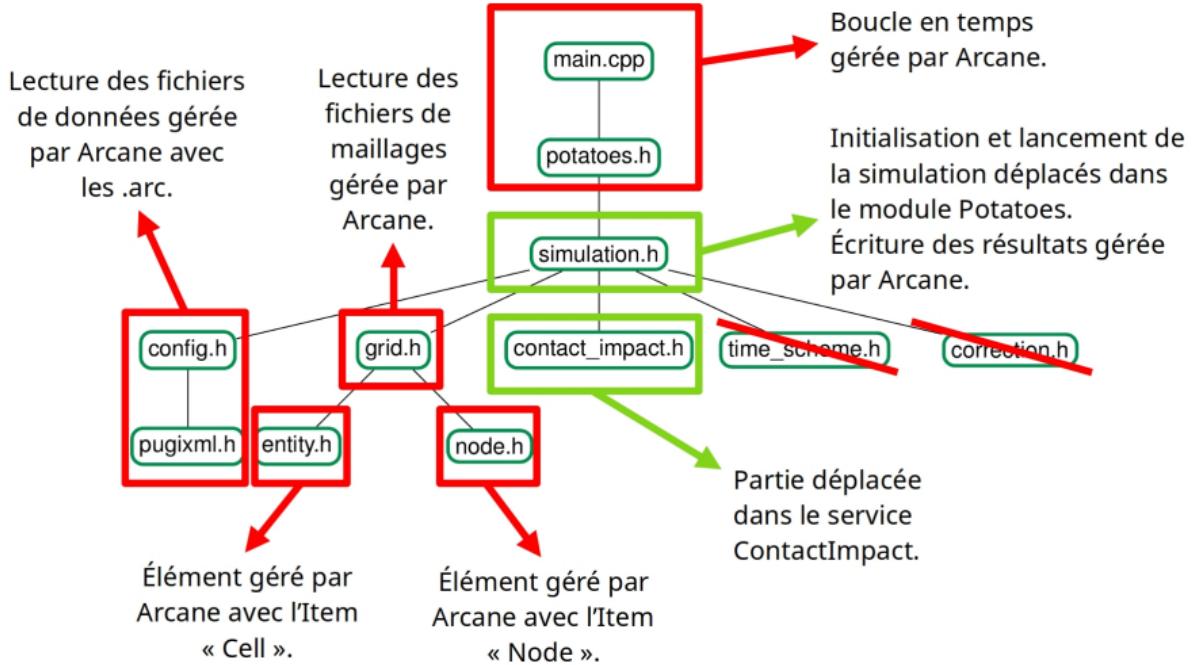


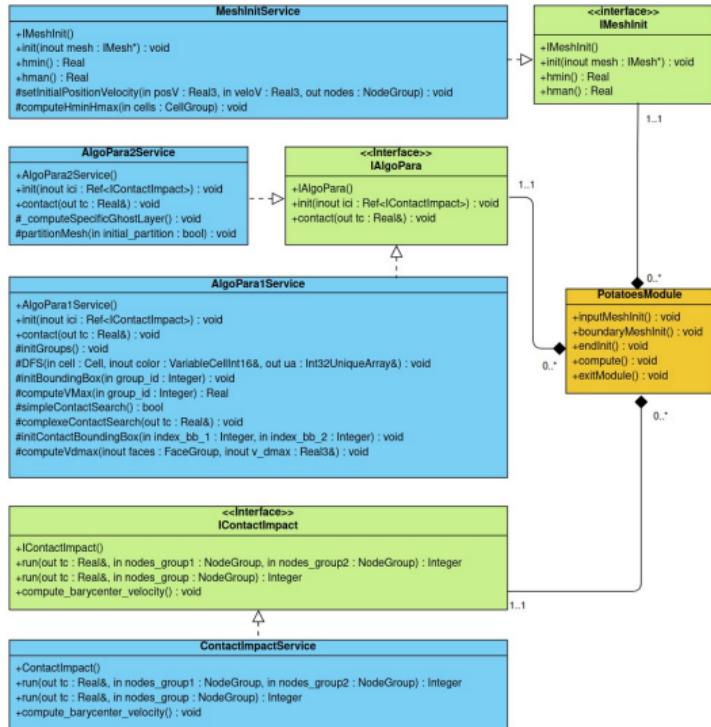
Figure 1 – Doublons entre Potatoes et Arcane



Étapes principales suivies pour porter Potatoes sur Arcane

- ▶ À chaque étape, vérifier les résultats.
- ▶ Prendre l'exemple "HelloWorld" disponible dans la documentation Arcane ;
- ▶ Mettre le contenu de la fonction main de Potatoes dans le module "SayHello". Si la fonction "main" récupérait des arguments, les ajouter en dur dans le code ;
- ▶ Réécrire un fichier de données (le plus exhaustif) en .arc, remplacer le lecteur de Potatoes par les "options()" ;
- ▶ Supprimer la boucle globale de Potatoes et la remplacer par une boucle en temps Arcane ;
- ▶ Dupliquer les parties utilisant le maillage ;
- ▶ Remplacer les "Entity" par des "Arcane::Cell" et les "Node" par des "Arcane::Node" en utilisant les variables aux mailles/noeuds pour remplacer les attributs de ces classes ;
- ▶ Ajouter la lecture Arcane des fichiers de maillages et vérifier que les résultats sont identiques ;
- ▶ Supprimer les anciennes entités dupliquées ;
- ▶ Rechercher et remplacer tous les traitements que peut faire Arcane directement.

Arcane Potatooid contAct deTectiOn procedureS



PotatoesModule

S'occupe de créer le "BoundaryMesh", de lancer la simulation et de mettre à jour les positions des noeuds.

MeshInitService

S'occupe d'initialiser la vitesse des noeuds du maillage.

AlgoPara[1/2]Service

S'occupe des structures permettant d'accélérer le calcul.

ContactImpactService

S'occupe de la détection de contact.

Figure 2 – Structure (simplifiée) de APatoes

3. Optimisations et parallélisation



CETT

Cet algorithme a besoin de deux choses :

- ▶ Un noeud
- ▶ Une face

À l'aide de ces deux éléments, il peut nous fournir le **temps restant** avant le contact entre la face et le noeud.

Pour le contact entre deux parties du maillage, il suffit de donner le **groupe de noeuds** d'une partie et le **groupe de faces** de l'autre partie.

Arcane nous permet de récupérer les faces reliées à un noeud. Donc on peut donner **deux groupes de noeuds** (avec au minimum un noeud dans chaque groupe).

Objectif

Ce traitement est long, donc on doit :

- ▶ soit réduire le nombre de noeuds à traiter,
- ▶ soit traiter des sous-groupes de noeuds.

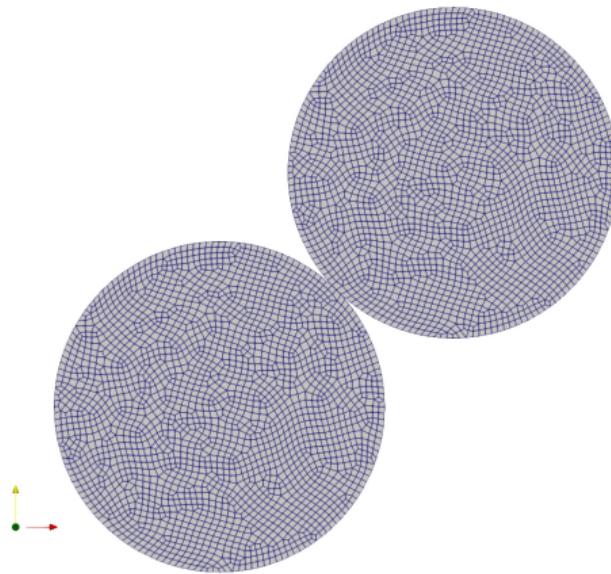


Figure 3 – Maillage d'origine : Nombre de mailles = 4698 /
Nombre de noeuds = 4860

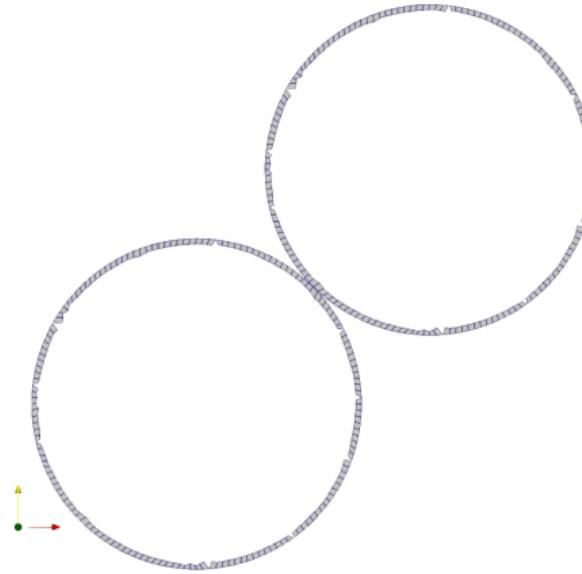


Figure 4 – Maillage de peau (BoundaryMesh) : Nombre de
mailles = 320 / Nombre de noeuds = 658

Essai de parallélisation n°1 (AlgoPara1Service)

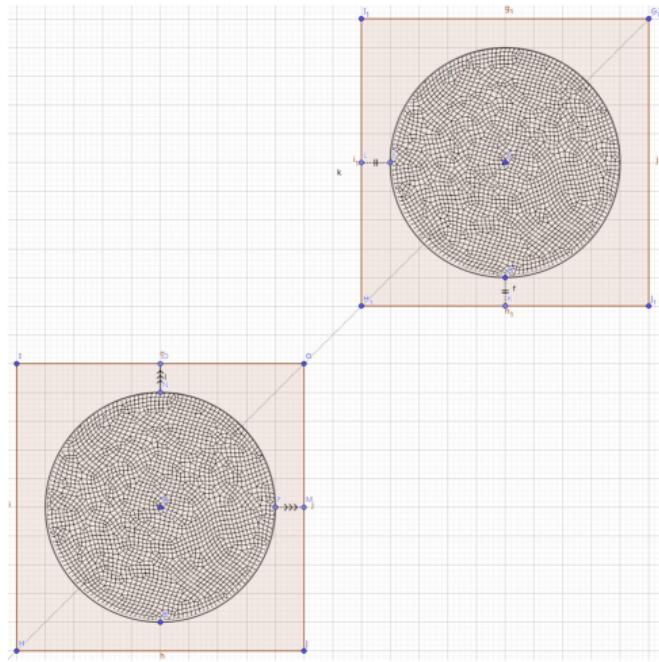


Figure 5 – Maillage avant le contact

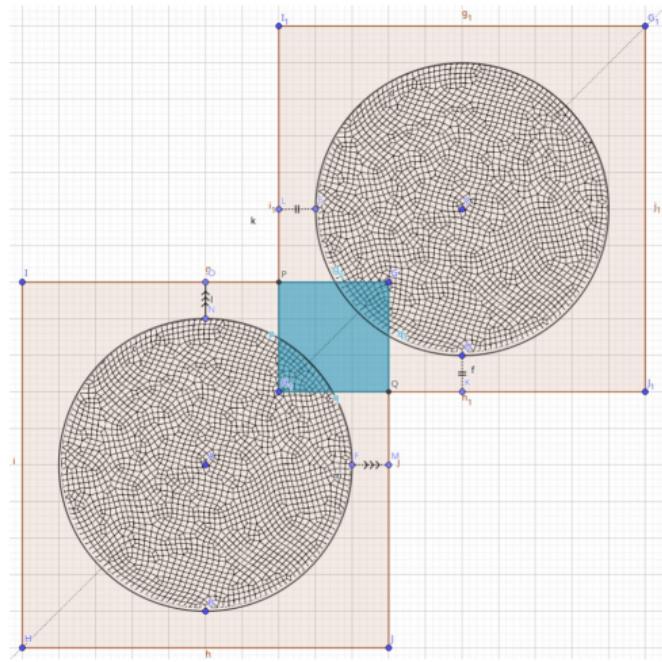


Figure 6 – Collision des BoundingBox

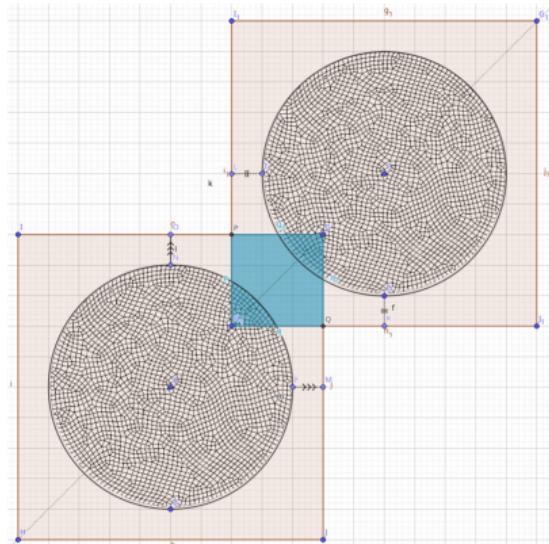


Figure 7 – Collision des BoundingBox

BoundingBox

À chaque pas de temps :

- ▶ On construit les BoundingBox autour des deux parties du maillage (en orange). On agrandit la BoundingBox en ajoutant la distance parcourue en un pas de temps ($vitesseMax * DeltaT$).
- ▶ On regarde s'il y a collision entre les BoundingBox. Si non, pas de collision.
 - ▶ Si oui, on construit la **ContactBox** (en bleu).
 - ▶ On regarde quels noeuds sont dedans.
 - ▶ On lance CETT avec ces noeuds.

La **ContactBox** est définie comme étant la zone d'intersection de **deux BoundingBox**.

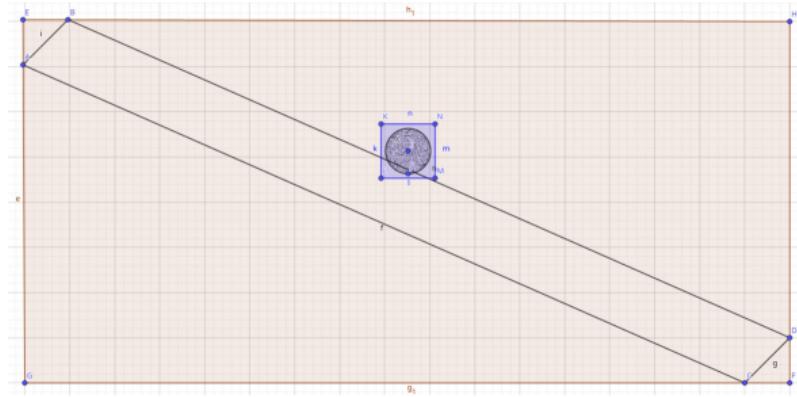


Figure 8 – Cas extrême : aucune collision détectée

Problème

La **ContactBox** n'inclut aucun noeud de la face avec laquelle rentre en collision le cercle.

Note

La **BoundingBox** du cercle est, ici, confondue avec la **ContactBox** (en bleu).

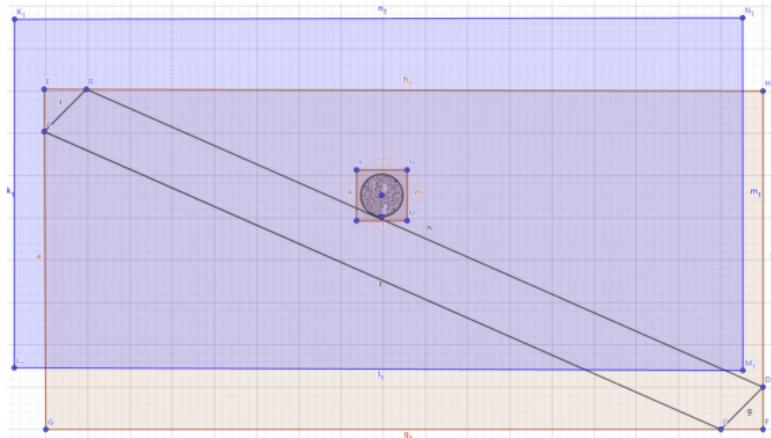


Figure 9 – La collision est bien détectée ici

Solution

Il faut agrandir la **ContactBox** pour inclure au moins un noeud de la face impliquée dans la collision. On utilise pour cela la longueur de face maximale (en x, y, z) de chaque partie du maillage pour agrandir la **ContactBox**.

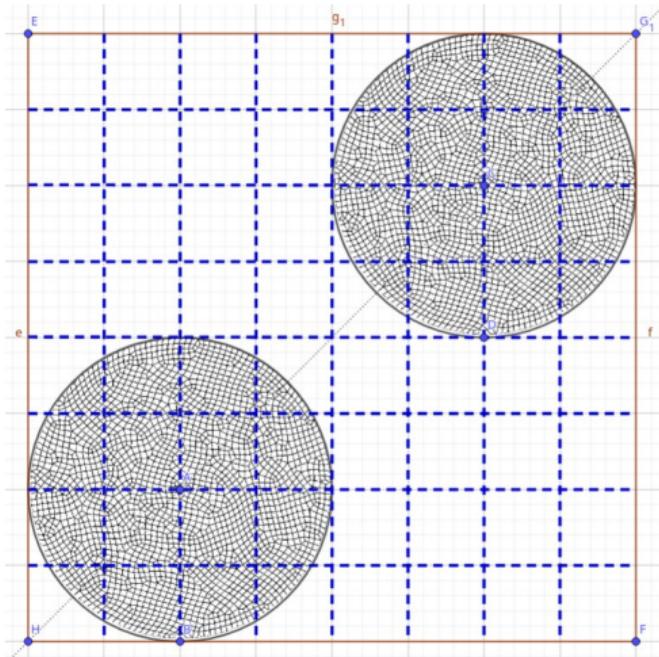


Figure 10 – Découpage du maillage en plusieurs blocs

Algorithme

À chaque pas de temps :

- ▶ On construit la **BoundingBox** englobant tout le maillage (en orange).
- ▶ On découpe la **BoundingBox** en plusieurs blocs (en bleu).
- ▶ On répartit les **blocs** entre les processus (pas d'équilibrage pour l'instant).
- ▶ Chaque processus regarde si les mailles qu'il possède sont dans ses blocs. Un partage des mailles est réalisé collectivement.
- ▶ On envoie les mailles fantômes là où c'est nécessaire.
- ▶ Chaque processus applique CETT sur les mailles de ces blocs.

Partage des mailles

Plusieurs étapes à faire à **chaque itération**, pour **chaque maille** :

- ▶ On détermine le barycentre de la maille (en vert), ce qui va nous donner le propriétaire de la maille,
- ▶ On envoie la maille à son propriétaire (si nécessaire).

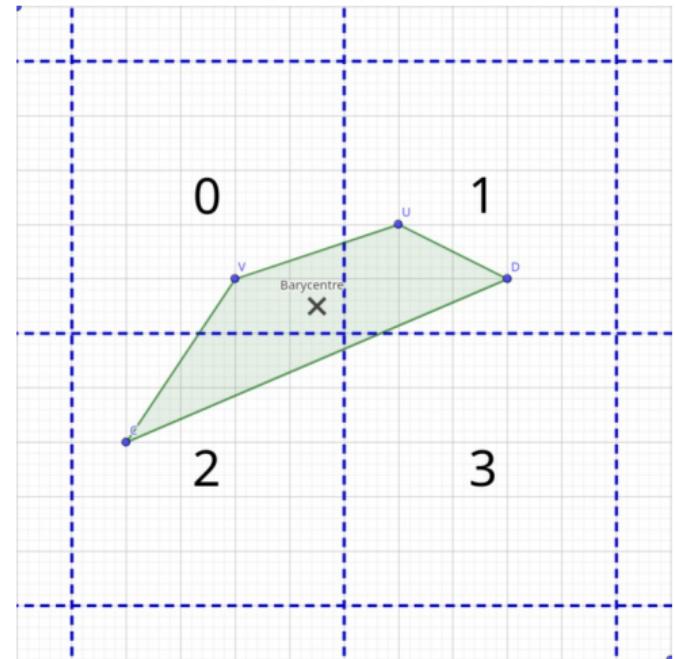


Figure 11 – Zoom sur une des mailles

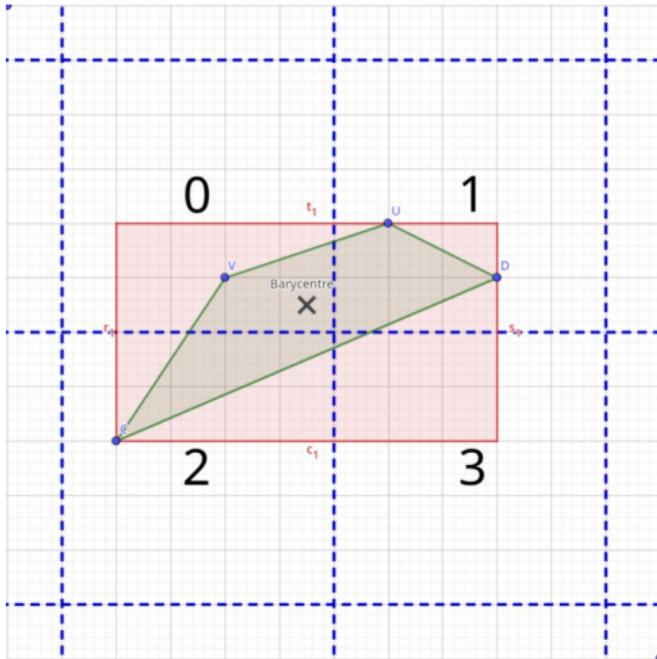


Figure 12 – Zoom sur une des mailles, avec sa BoundingBox

Mailles fantômes

- ▶ On crée la BoundingBox de la maille (en orange),
- ▶ On détermine les processus ayant des blocs intersectant la BoundingBox de la maille,
- ▶ On envoie la maille en tant que maille fantôme dans ces processus.



Algorithme n° 1

Avantages :

- ▶ La création des BoundingBox et la détection de leurs collisions est assez rapide, ce qui permet de gagner beaucoup de temps lorsque les parties du maillage sont éloignées,
- ▶ Le principe de ContactBox permet de réduire grandement le nombre de noeuds à envoyer à CETT.

Inconvénients :

- ▶ Traitement uniquement en séquentiel (pour l'instant), il n'y a que CETT qui est multithreadé,
- ▶ Pas de collision intra-maillage possible,
- ▶ Découpage du maillage en plusieurs parties obligatoires (réduction du coût de l'algorithme grâce au maillage de peau).

Algorithme n° 2

Avantages :

- ▶ Support du multiprocessus et possibilité de faire de l'équilibrage de charge,
- ▶ Toutes les collisions sont détectées,
- ▶ Pas besoin de découper le maillage en plusieurs parties.

Inconvénients :

- ▶ Pas de BoundingBox par partie (car pas de découpage) donc l'algorithme est toujours exécuté entièrement, même si les différentes parties sont éloignées.

4. ■ Temps et conclusion

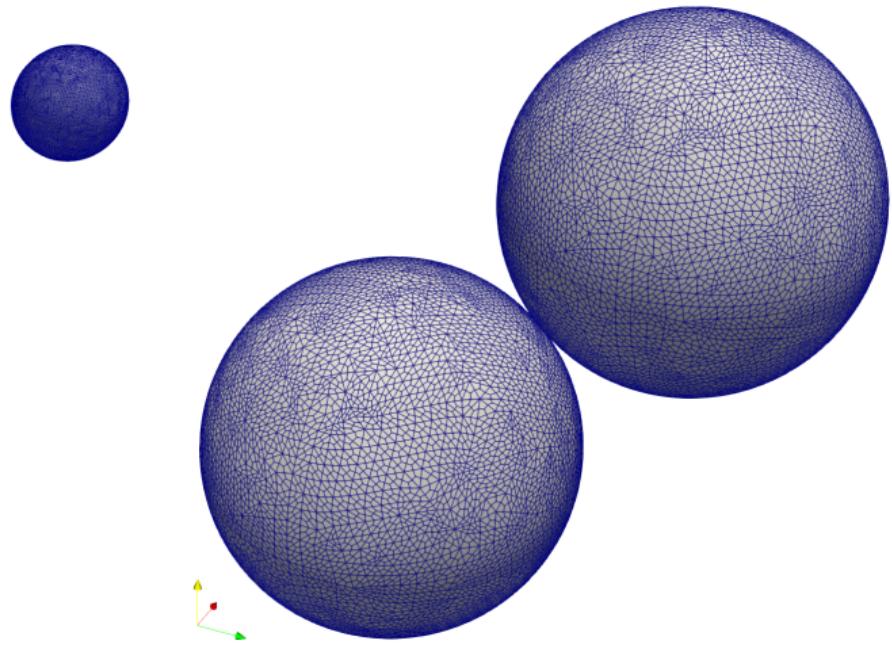


Figure 13 – Maillage à T=0 (sphere008)

Figure 14 – Maillage à T=3.42 (sphere008)



Évaluation 2D (maillage entier)

Finesse	0,04	0,02	0,01	0,005
Nb mailles	4698	18644	72366	288844
Nb noeuds	4860	18966	73000	290110

Évaluation 2D (maillage de peau)

Finesse	0,04	0,02	0,01	0,005
Nb mailles	320	640	1264	2528
Nb noeuds	658	1290	2572	5154

Évaluation 3D (maillage entier)

Finesse	0,64	0,32	0,16	0,08
Nb mailles	2192	7488	29176	207192
Nb noeuds	3026	9554	35346	238770

Évaluation 3D (maillage de peau)

Finesse	0,64	0,32	0,16	0,08
Nb mailles	1188	2760	7404	29352
Nb noeuds	2592	6464	17816	72054

Environnement d'exécution

Arcane 3.9.4.0 / GCC 11.2 / CPU AMD Rome (2x64 coeurs)

Nombre de couples [noeud, face] traités par CETT

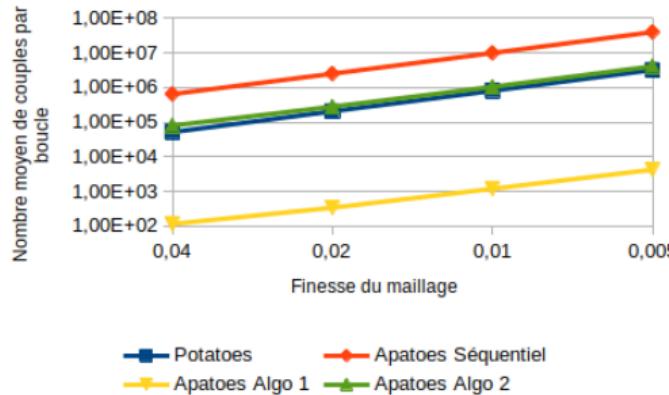


Figure 15 – Nombre moyen de couples [noeud, face] traités par CETT (2D)

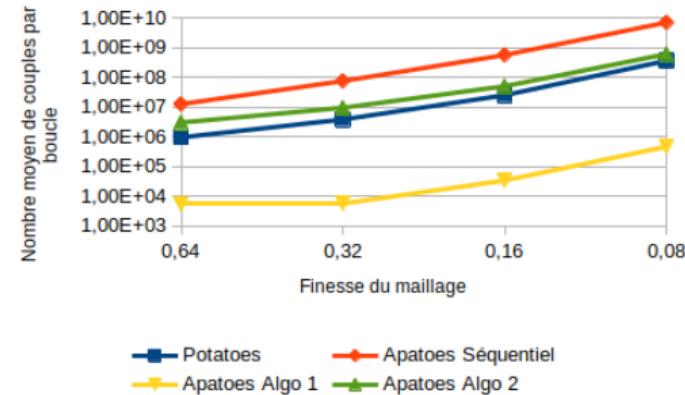


Figure 16 – Nombre moyen de couples [noeud, face] traités par CETT (3D)

Speedup en 2D

En moyenne, par rapport à "**Potatoes**", "**APatoes Séquentiel**" traite 12x plus de couples, "**APatoes Algo 1**" traite 720x moins de couples et "**APatoes Algo 2**" traite 29% plus de couples.

Speedup en 3D

En moyenne, par rapport à "**Potatoes**", "**APatoes Séquentiel**" traite 19x plus de couples, "**APatoes Algo 1**" traite 768x moins de couples et "**APatoes Algo 2**" traite 70% plus de couples.

Potatoes et APatoes en séquentiel

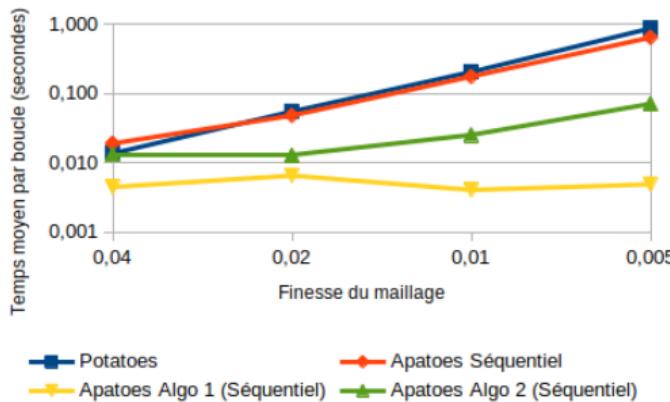


Figure 17 – Potatoes / APatoes en séquentiel (2D)

Speedup en 2D

En moyenne, par rapport à "**Potatoes**",
"APatoes Séquentiel" est 30% plus rapide,
"APatoes Algo 1" est 58 fois plus rapide et
"APatoes Algo 2" est 9 fois plus rapide.

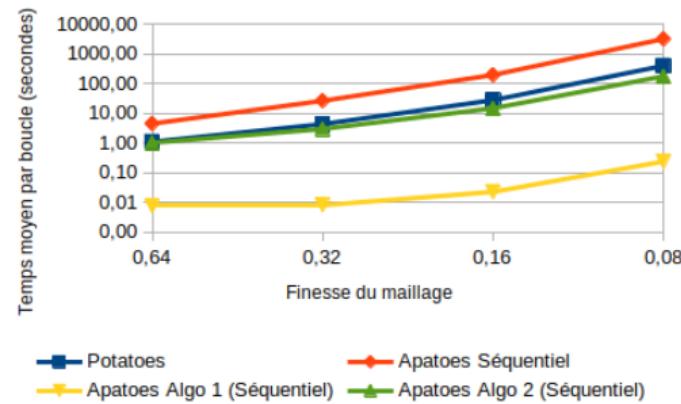


Figure 18 – Potatoes / APatoes en séquentiel (3D)

Speedup en 3D

En moyenne, par rapport à "**Potatoes**",
"APatoes Séquentiel" est 8 fois plus lent,
"APatoes Algo 1" est 1561 fois plus rapide et
"APatoes Algo 2" est 2 fois plus rapide.

Les deux algorithmes de APatoes en parallèle

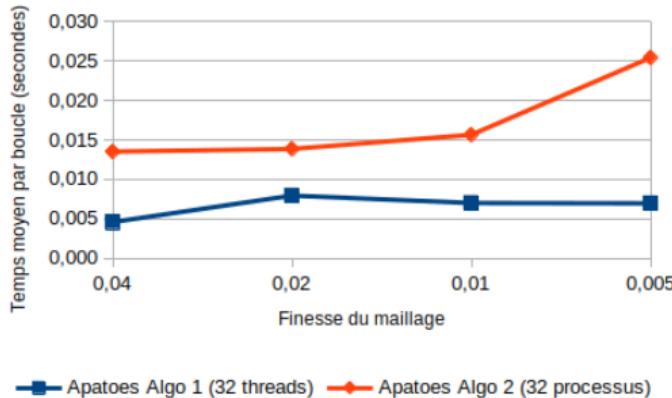


Figure 19 – Algorithmes de APatoes en parallèle (2D)

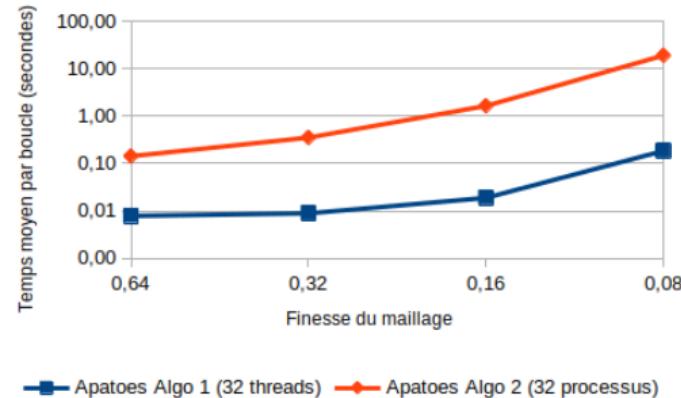


Figure 20 – Algorithmes de APatoes en parallèle (3D)

Speedup en 2D

En moyenne, "APatoes Algo 1 (32 threads)" est 2.6 fois plus rapide que "APatoes Algo 2 (32 processus)".

Speedup en 3D

En moyenne, "APatoes Algo 1 (32 threads)" est 97 fois plus rapide que "APatoes Algo 2 (32 processus)".

Algorithme n° 1 (BoundingBox)

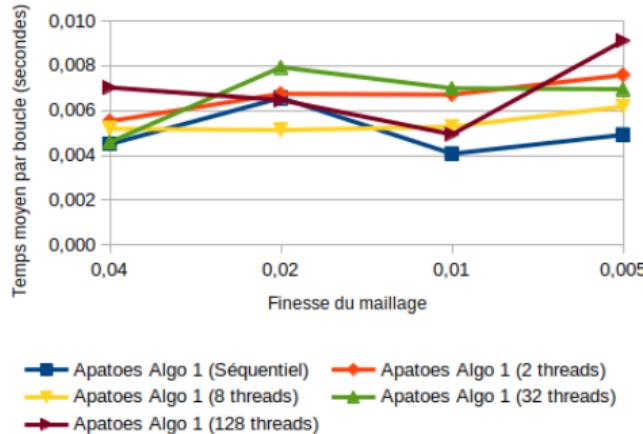


Figure 21 – Algorithme n° 1 de APatoes (2D)

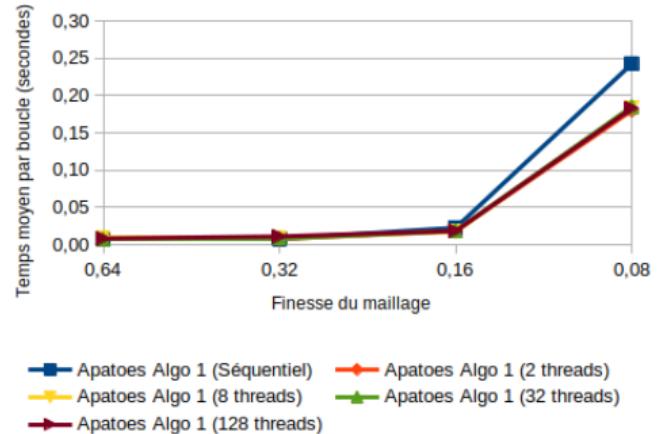


Figure 22 – Algorithme n° 1 de APatoes (3D)

Nombre moyen de couples [noeud, face] traités en 2D

Algorithme	Nombre moyen de couples traités
Potatoes	1 063 748
APatoes Algo 1	1 476

Nombre moyen de couples [noeud, face] traités en 3D

Algorithme	Nombre moyen de couples traités
Potatoes	100 016 572
APatoes Algo 1	130 137

Algorithme n° 2 (Blocs)

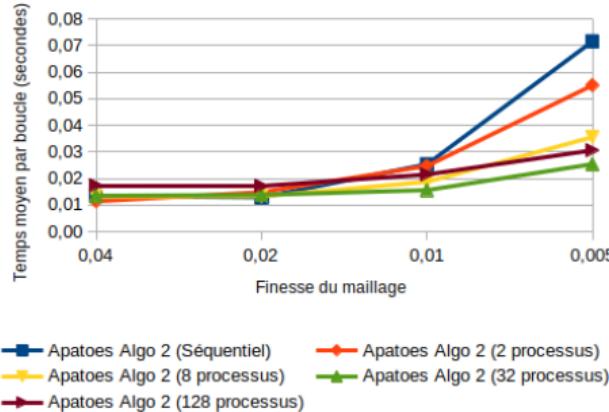


Figure 23 – Algorithme n° 2 de APatoes (2D)

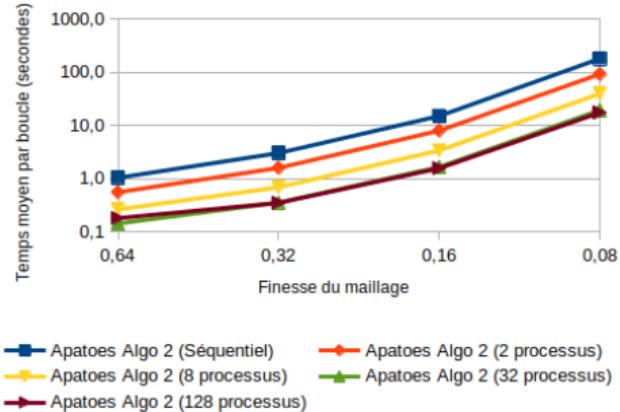


Figure 24 – Algorithme n° 2 de APatoes (3D)

Speedup en 2D

En moyenne, par rapport au "**Séquentiel**",
"2 processus" est 16% plus rapide,
"8 processus" est 52% plus rapide,
"32 processus" est 80% plus rapide et
"128 processus" est 42% plus rapide.

Speedup en 3D

En moyenne, par rapport au "**Séquentiel**",
"2 processus" est 2 fois plus rapide,
"8 processus" est 4.5 fois plus rapide,
"32 processus" est 9 fois plus rapide et
"128 processus" est 10 fois plus rapide.



Arcanisation

Développer sur Arcane permet de **gagner du temps** au niveau du développement mais aussi au niveau de la maintenance. Le portage Potatoes -> APatoes a pris **15 jours**. L'écriture de l'algorithme n° 1 a pris **10 jours**. L'écriture de l'algorithme n° 2 a pris **7 jours**. Potatoes version Arcane se résume aujourd'hui aux algorithmes de parallélisation et de contact, le reste étant géré par Arcane.

Les deux algorithmes

- ▶ L'algorithme de test n° 1 permet, dans les cas présentés ici, de diviser, en moyenne, par **700** le nombre de couples [noeud, face] calculés (en 3D / par rapport à Potatoes).
- ▶ L'algorithme de test n° 2 permet, dans les cas présentés ici, de diviser, en moyenne, par **10** le temps de calcul (en 3D / par rapport à Potatoes).

Travaux futurs

- ▶ Étudier la possibilité de mélanger les deux algorithmes : découper les **BoundingBox** de l'algorithme de test n° 1 en **blocs** de l'algorithme de test n° 2,
- ▶ Ajouter un **équilibrage de charge** dans l'algorithme de test n° 2,
- ▶ Faire un portage sur **GPU**.



Merci de votre attention !

Optimisation et parallélisation d'un algorithme de détection de contact : Potatoes

Alexandre l'Héritier

Crédits supplémentaires :

Commissariat à l'énergie atomique et aux énergies alternatives
Centre de Bruyères-le-Châtel | 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00 | F. +33 (0)1 69 26 40 00

Établissement public à caractère industriel et commercial – RCS Paris B 775 685 019