CSMA 2026
17ème Colloque National en Calcul des Structures
18-22 Mai 2026, Presqu'île de Giens (Var)

Vers la simulation discrète de grande échelle : implémentation GPU et architecture hybride dans GranOO v4

D. André[1], J. Girardot[2], C. Hubert[3,4]

[1] *Univ. Limoges, IRCER, UMR CNRS 7315, 87000 Limoges, damien.andre@unilim.fr*
[2] *Arts et Métiers, I2M, UMR CNRS 5295, 33400 Talence, jeremie.girardot@ensam.eu*
[3] *UPHF, LAMIH, UMR CNRS 8201, 59313 Valenciennes, cedric.hubert@uphf.fr*
[4] *INSA Hauts-de-France, 59313, Valenciennes*

Résumé — GranOO est une plateforme open source en C++ dédiée à la Méthode des Éléments Discrets pour la mécanique des solides et la simulation multiphysique. La version 4 introduit une refonte de l'architecture et l'intégration d'un moteur de calcul GPU. Le noyau CPU historique est désormais couplé à un backend CUDA permettant le portage parallèle des algorithmes critiques. Cette approche hybride CPU/GPU garantit la compatibilité ascendante tout en offrant d'importants gains de performance. La version modernise également les macro-commandes, la visualisation et l'interface Python. Ces évolutions permettent des simulations de grande taille, illustrées par des cas de fissuration, d'impact et de couplage thermo-mécanique.
Mots clés — MED, open-source, C++, GPU, CUDA, parallélisation, multiphysique, éléments discrets.

1 Introduction

La Méthode des Éléments Discrets (MED) [1] est aujourd'hui largement utilisée pour modéliser des milieux granulaires ou présentant des discontinuités, mais elle constitue également une approche intéressante pour simuler des milieux continus solides lorsque les phénomènes de rupture et de contact deviennent prépondérants [2]. Dans ce cadre, la plateforme libre GranOO (www.granoo.org) [3] offre depuis plus d'une décennie un environnement de développement et de simulation dédié à la MED appliquée à la mécanique des solides continus. Développé conjointement au sein de plusieurs laboratoires français (IRCER, I2M et LAMIH), GranOO se distingue par une architecture orientée objet écrit en C++ et un mécanisme extensible de macro-commandes qui permettent de prototyper rapidement de nouvelles simulations et de nouveaux modèles tout en favorisant la réutilisation et la capitalisation des développements réalisés par des tiers. La plateforme a été utilisée dans de nombreux contextes, allant de la modélisation de la fissuration quasi-fragile à la simulation multiphysique thermo-électro-mécanique [4], en passant par la fragmentation dynamique [5] ou la tribologie [6]. Cependant, la montée en complexité des modèles et l'augmentation du nombre de degrés de liberté imposent des besoins croissants en puissance de calcul. Les versions précédentes de GranOO (jusqu'à la v3) reposaient sur une exécution purement séquentielle ou multi-threadée sur CPU, limitant la taille des simulations accessibles. La version v4 de GranOO marque une étape majeure de son évolution. Elle introduit une refonte du cœur logiciel et l'intégration d'un moteur de calcul compatible GPU, visant à exploiter le parallélisme massif offert par les architectures graphiques modernes. Cette évolution permet d'accélérer significativement certaines opérations critiques de la MED [7] tout en préservant la modularité de GranOO. La présente contribution a pour objectif de présenter cette nouvelle architecture hybride CPU/GPU, d'en décrire les principaux choix d'implémentation et d'illustrer les gains de performance obtenus sur plusieurs cas de simulation originaux.

2 Architecture logicielle de GranOO v4

La version 4 de GranOO repose sur une refonte significative de son architecture interne, visant à la fois à moderniser le code et à préparer son exécution sur GPU. Le projet conserve son principe fondateur : fournir un environnement de simulation libre, orienté objet et extensible. GranOO est structuré

selon une architecture dite « en oignon ». Le noyau central est constitué d'un ensemble de bibliothèques C++ regroupées par modules thématiques, qui implémentent les objets fondamentaux de la simulation : éléments discrets, interactions, liens cohésifs, intégrateurs, entrées/sorties, gestion du temps, etc. Les couches supérieures assurent la communication avec l'utilisateur à travers : (i) des fichiers d'entrée au format `xml` décrivant les simulations (`*.inp`), (ii) un système modulaire de macro-commandes (nommés `PlugIn` dans les classes GranOO) C++ ou Python appelables depuis les fichiers `*.inp`, (iii) des outils internes et externes de pré- et post-traitement (`granoo4-viewer`, `granoo4-cooker`, `paraview`, `pyvista`, etc.).

Le choix a été fait de limiter la compatibilité ascendante entre la version 4 et la version 3 : les fichiers XML `*.inp` restent rétrocompatibles, mais les API C++ ont été modifiées. En effet, le cœur C++ de GranOO v4 a été profondément réorganisé afin de préparer la parallélisation et d'améliorer la gestion mémoire. Cette évolution repose notamment sur une uniformisation des structures de données selon une représentation de type *Structure of Arrays* (SoA), mieux adaptée au calcul parallèle et à l'exploitation du GPU. Ce choix technique constitue un compromis délicat entre performance et souplesse logicielle : il permet de bénéficier d'un accès mémoire contigu pour les calculs massivement parallèles tout en conservant le caractère « orienté objet »du code. GranOO v4 parvient à maintenir cette double compatibilité grâce à une conception modulaire originale où les objets de haut niveau (comme `Body`, `Node` ou `Interaction`) exposent une interface classique, tandis que les données internes sont stockées dans des structures SoA centralisées. Cette approche garantit des performances comparables à une implémentation procédurale (non orientée objet) tout en préservant la lisibilité, la maintenabilité et l'héritabilité du code d'une architecture orientée objet.

De plus, le mécanisme de macro-commandes `PlugIn`, introduit dans les versions précédentes, a été étendu pour supporter des traitements exécutés sur GPU. Une macro-commande `PlugIn` peut désormais déclarer explicitement sa zone d'exécution (CPU ou GPU), ce qui permet à l'utilisateur de composer librement ses séquences de calculs dans le fichier d'entrée. Cette évolution ouvre la possibilité de définir des macro-commandes hybrides, capables d'orchestrer des traitements mixtes entre les deux espaces mémoire. Certaines macro-commandes `PlugIn` élémentaires assurent, par exemple, la synchronisation explicite entre le CPU et le GPU ou, au contraire, la désynchronisation temporaire afin de tirer parti de l'exécution asynchrone. Ce mécanisme est particulièrement utile lors de l'écriture de fichiers de résultats : le GPU peut poursuivre la simulation pendant que le CPU effectue les opérations d'entrée/sortie. De la même manière, d'autres macro-commandes simples permettent d'effectuer des *dump* ciblés de zones mémoire du GPU vers le CPU pour analyse, visualisation ou reprise de calcul. Ces opérations sont intégrées de manière transparente dans la séquence de calculs décrite dans le fichier `*.inp`, sans nécessiter de manipulation directe du code CUDA par l'utilisateur. Ce système conserve ainsi la philosophie initiale de GranOO : séparer clairement le cœur numérique des aspects de pilotage et de gestion de simulation.

3 Implémentation GPU

L'objectif principal du portage GPU de GranOO v4 est de tirer parti du parallélisme massif offert par les architectures graphiques afin d'accélérer les étapes les plus coûteuses, tout en conservant la souplesse et la lisibilité du code orienté objet historique.

Le moteur GPU de GranOO v4 repose sur un modèle de données « miroir » : chaque entité (élément discret, joint cohésif, etc.) dispose d'une représentation sur le CPU et sur le GPU. Les synchronisations CPU–GPU sont explicites et réalisées par une macro-commande dédiée `GPU-DUMP` uniquement lorsque cela est nécessaire, afin de minimiser les transferts mémoire. Cette macro-commande dédiée permet de forcer les synchronisations du GPU vers le CPU à des instants précis du calcul ou de les suspendre temporairement pour autoriser des traitements asynchrones. L'architecture CUDA a été conçue pour rester compatible avec une exécution séquentielle : en l'absence de GPU, les mêmes macro-commandes peuvent fonctionner sur CPU, ce qui assure la portabilité et la vérification croisée des résultats. Une attention particulière a été portée à la maximisation du code partagé entre le CPU et le GPU. Dans la mesure du possible, les mêmes algorithmes sont exécutés lors d'un calcul CPU ou bien lors d'un calcul GPU. Seuls les algorithmes de détection des contacts font exception. Ces modules ont été entièrement réécrits et utilisent des algorithme différents selon si ils sont exécutés dans un contexte GPU ou bien CPU.

Le second ensemble d'algorithmes portés sur GPU concerne les joints cohésifs reliant les éléments discrets. Ces joints cohésifs constituent l'une des originalités majeures de GranOO, qui propose plusieurs types de joints permettant de reproduire le comportement mécanique d'un solide élastique fragile selon une approche dite *lattice*. Cette méthode consiste à discrétiser le milieu continu par un réseau d'éléments discrets reliés entre eux par des liaisons cohésives, chacune suivant une loi de comportement locale et un critère de rupture spécifique. Les principaux modèles de joints cohésifs (ressorts linéaires `Spring`, poutre d'Euler-Bernoulli `Beam` [8] et joints *lattice spring* `FlatBond` [9]) ainsi que leurs critères de rupture associés ont été portés sur GPU. Ainsi, chaque joint est calculé indépendamment dans un kernel CUDA, les résultats étant ensuite agrégés dans des tableaux de forces et de moments de réaction. Un soin particulier a été apporté à la gestion des accès concurrents en écriture, en minimisant le recours aux opérations atomiques et en privilégiant des stratégies de réduction locales afin de limiter les goulots d'étranglement.

Enfin, l'intégration temporelle explicite de type Velocity Verlet a également été transférée sur GPU. Les mises à jour des positions, vitesses et orientations sont effectuées en parallèle, ce qui permet de maintenir un haut degré de vectorisation sur des centaines de milliers d'éléments discrets.

4 Validation GPU

La validation du moteur GPU de GranOO v4 a été conduite en comparant systématiquement les résultats obtenus avec les versions CPU et GPU sur un ensemble de cas tests représentatifs (aussi disponibles en exemples standards de GranOO). L'objectif est de vérifier la cohérence numérique entre les deux modes de calcul et de quantifier les éventuels écarts. Ce travail a été réalisé sur onze cas tests inclus dans la distribution standard de GranOO v4, que l'utilisateur peut aisément reproduire. Ces exemples couvrent à la fois des simulations de milieux continus (avec différents modèles de joints cohésifs) et des simulations de milieux granulaires (avec divers modèles de contact). Les comparaisons ont été effectuées à deux niveaux : les validations locales et les validations globales.

Les validations locales concernent les comparaisons des résultats issus d'interactions élémentaires (déplacements, forces, moments, etc.) entre particules ou paires de particules ciblées, tandis que la validation globale porte sur le suivi de grandeurs intégrales ou moyennes (contraintes moyennes, déformations moyennes, contraintes à rupture) au cours d'une simulation. Pour les deux architectures, les calculs ont été réalisés en double précision et montrent une adéquation parfaite pour les validations globales. Les validations locales présentent parfois des écarts, liées à des différences d'ordonnancement des calculs entre CPU et GPU, ce qui peut conduire à des variations numériques. Ces écarts n'ont toutefois aucune influence sur les grandeurs globales ni sur l'évolution mécanique de la structure et peuvent être éliminés en forçant un ordonnancement strictement identique enntre CPU et GPU.

Les résultats de ces validations ne sont pas détaillés ici : l'ensemble des cas testés présentent une très bonne correspondance entre les implémentations CPU et GPU, confirmant la robustesse du portage. La section suivante se concentre donc sur les benchmarks de performance, destinés à évaluer les gains en temps de calcul apportés par le moteur GPU en termes de temps d'exécution entre un CPU mono-thread INTEL(R) CORE(TM) ULTRA 7 165H et un GPU NVIDIA RTX 2000 ADA GENERATION. Le tableau 1 résume les résultats obtenus sur quatre simulations standards de GranOO. Dans ce tableau,

- **Ref** indique le code interne de l'exemple (dans l'arborescence `GranOO/Examples`) ;
- **Milieu** précise le type de milieu étudié (granulaire avec contacts ou continu avec joints cohésifs) ;
- **Modèle** correspond au modèle micromécanique utilisé ;
- **ED** donne le nombre d'éléments discrets ;
- **It** indique le nombre d'itérations de la boucle explicite ;
- **CPU** et **GPU** rapportent respectivement le temps d'exécution sur CPU et sur GPU ;
- **Gain** présente le facteur d'accélération obtenu.

Ces résultats mettent en évidence un gain moyen de l'ordre de $\times 40$ sur les cas testés, confirmant l'efficacité du portage GPU. La baisse de performance constatée pour les milieux granulaires entre les cas tests `00100` et `00101` s'explique par une densité de contact beaucoup plus élevée dans le cas `00101`, correspondant à un milieu granulaire dense, tandis que le cas `00100` représente un milieu plus pulvérulent. En effet, dans l'architecture GPU, le choix a été fait de conserver une pile d'appel supplémentaire par rapport à l'architecture CPU lors de la gestion des contacts dans GranOO, afin de maintenir une structure

de code souple et compatible avec la logique du fichier `*.inp`. Ce compromis, légèrement coûteux en performance dans les cas de forte densité de contact, a été retenu pour préserver la cohérence logicielle et la flexibilité du moteur de calcul. Il reflète aussi le fait que le cœur de GranOO n'est pas spécifiquement dédié à la simulation de milieux granulaires, mais bien à la modélisation de milieux continus via des joints cohésifs où un gain plus significatif ($\times$38-$\times$50) est constaté.

Ref	Type	Modèle	ED	It	CPU	GPU	Gain
`00010`	continu	joint cohésif `Beam`	10k	10k	278,1 s	6,6 s	$\times$42
`00011`	continu / fragile	joint cohésif `Beam`	10k	10k	159,4 s	3,8 s	$\times$42
`00012`	continu	joint cohésif `FlatBond`	10k	6k	551,5 s	11,0 s	$\times$50
`00013`	continu / fragile	joint cohésif `FlatBond`	10k	10k	849,6 s	22,3 s	$\times$38
`00100`	granulaire	contact `Standard2`	100k	50k	2910,0 s	65,2 s	$\times$45
`00101`	granulaire	contact `HertzMindlin`	100k	10k	1246,8 s	65,0 s	$\times$19

TABLE 1 – Benchmark de temps d'exécution CPU/GPU

5 Application

Afin d'illustrer les capacités de la version GPU de GranOO v4, trois applications représentatives de la fissuration dans des matériaux fragiles sont présentées. Ces exemples, issus de travaux originaux, mettent en évidence la diversité des contextes étudiés : fissuration dynamique, fissuration quasi-statique à l'échelle microstructurale et rupture thermo-mécanique multiphysique. Dans tous les cas, le recours au GPU permet de simuler des domaines bien plus étendus qu'avec une exécution CPU classique, tout en réduisant les temps de calcul d'un ordre de grandeur.

5.1 Simulation multiphysique : choc thermique

Cette étude s'inspire des travaux expérimentaux de Jiang *et al.* [10], qui ont analysé la formation et l'évolution de réseaux de fissures dans des plaques minces de céramique soumises à un choc thermique sévère. Dans leurs expériences, des plaques d'alumine (Al_2O_3) de $50 \times 10 \times 1$ mm étaient portées à une température initiale T_0 comprise entre 300 et 600 °C, puis brutalement trempées dans un bain d'eau à 20 °C. Les motifs de fissuration observés présentent une structure régulière, quasi périodique et hiérarchique, dont la densité augmente avec la sévérité du choc thermique (voir Fig. 1b).

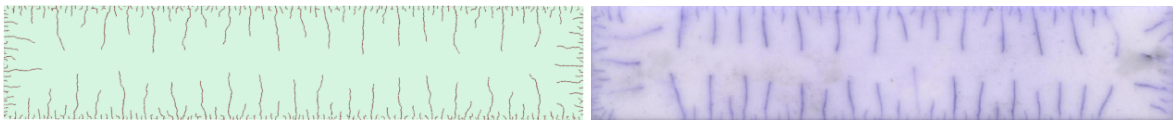
Dans le présent travail, la simulation de ce phénomène a été réalisée à l'aide de GranOO v4 dans une configuration bidimensionnelle comprenant environ 200k éléments discrets. Le domaine correspond à une plaque céramique soumise à un refroidissement brutal sur ses quatre faces latérales, conformément au protocole expérimental. Les propriétés thermiques et mécaniques sont celles de l'alumine dense (voir Tableau 2). Les liens entre éléments discrets sont modélisés par des joints cohésifs de type `FlatBond` [9], capables de représenter la fissuration fragile du matériau sans calibration préalable. La modélisation est réalisée en pseudo-2D, c'est-à-dire avec une seule couche d'éléments discrets suivant l'épaisseur (axe $\vec{z}$), tout en interdisant les déplacements selon cette même direction.

Le calcul repose sur un schéma couplé thermo-mécanique explicite. À chaque itération, la conduction thermique est résolue sur GPU via la macro-commande `GPU-CONDUCT-HEAT`, puis la dilatation thermique est appliquée au moyen de la macro-commande `GPU-APPLY-THERMAL-EXPANSION`. Le chargement thermique correspond à une chute de température imposée sur les quatre faces de l'échantillon (macro-commande `GPU-APPLY-TEMPERATURE`), tandis que l'évolution mécanique est intégrée à l'aide d'un schéma de type Velocity-Verlet modifié (macro-commande `GPU-INTEGRATE-ACCELERATION`). Les rotations locales ne sont pas prises en compte dans ce modèle. Deux pas de temps distincts sont employés : un pas de temps thermique ($\Delta t_{\text{th}} = 1 \times 10^{-7}$ s) et un pas de temps mécanique ($\Delta t_{\text{mec}} = 2{,}4 \times 10^{-10}$ s). Les aspects dynamiques n'étant pas étudiés, un fort coefficient d'amortissement visqueux est introduit dans le calcul mécanique afin d'accélérer la convergence vers un état quasi-statique.

Au cours de la simulation, les fissures (voir Fig. 1a) apparaissent naturellement, sous l'effet du gradient de température et de la dilatation thermique, par rupture des liens cohésifs. Leur évolution est suivie grâce à des *dump* réguliers (`GPU-DUMP`) permettant de transférer vers le CPU les champs thermiques et

Paramètre thermo-mécanique	Symbole	Valeur	Unité
Masse volumique	ρ	3980	$kg.m^{-3}$
Conductivité	λ	30	$W.m^{-1}.K^{-1}$
Capacité calorifique	C_p	800	$J.K^{-1}.kg^{-1}$
Température initiale	T_0	500	°C
Coefficient de dilatation	α	7,5	$10^{-6}.K^{-1}$
Module de Young	E	370	GPa
Coefficient de Poisson	ν	0,22	-
Contrainte à rupture (mode I)	σ_f	350	Mpa
Paramètre numérique		**Valeur**	
Nombre d'éléments discrets		200k	
Nombre d'itérations de calcul		5000k	

TABLE 2 – Paramètres de la simulation de choc thermique



(a) Modélisation DEM (b) Expérience [10]

FIGURE 1 – Choc thermique descendant sur céramique (Alumine) à $T_0 = 500°C$

mécaniques calculés sur GPU. La comparaison entre les images expérimentales et les résultats numériques (voir Fig. 1) montre que la simulation reproduit de manière qualitative et quantitative la formation d'un réseau de fissures thermiques périodiques et hiérarchiques, similaire à celui observé par Jiang *et al.* sur l'alumine. L'image 1a correspond à la dernière itération de la simulation, lorsque les contraintes thermiques maximales atteignent 320 MPa, soit une valeur environ 10 % inférieure à la contrainte critique de rupture, ce qui garantit que les fissures ne se propageront plus significativement. L'exécution complète de la simulation (soit 5×10^9 itérations) sur GPU nécessite environ 15 heures, tandis qu'une exécution CPU est estimée à plus de 25 jours, illustrant pleinement le gain de performance apporté par l'architecture parallèle GPU de GranOO v4.

5.2 Fissuration quasi-statique de l'os cortical

Cet exemple traite de la simulation numérique d'une éprouvette de traction d'os cortical, issue des travaux de thèse de Roothaer [11]. L'échantillon prélevé a été usiné à l'aide d'un mini-tour numérique pour obtenir une éprouvette de traction, qui a ensuite été scannée à l'aide d'un micro-tomographe afin de pouvoir en étudier la structure interne. La finesse de la résolution du scanner, ici 2.94 µm, permet notamment d'extraire l'enveloppe des zones les plus denses de l'éprouvette (Figure 2a), utilisée ici comme domaine d'étude. Cette enveloppe sera remplie d'éléments discrets sphériques (Figure 3b), interconnectés par des liaisons cohésives de type Euler-Bernoulli `Beam`. Le domaine discret ainsi construit contient environ 100k éléments et environ 290k liaisons cohésives.

(a) Enveloppe de l'éprouvette de traction (b) Domaine discret généré avec GranOO

FIGURE 2 – Préparation du domaine discret de l'éprouvette d'os cortical

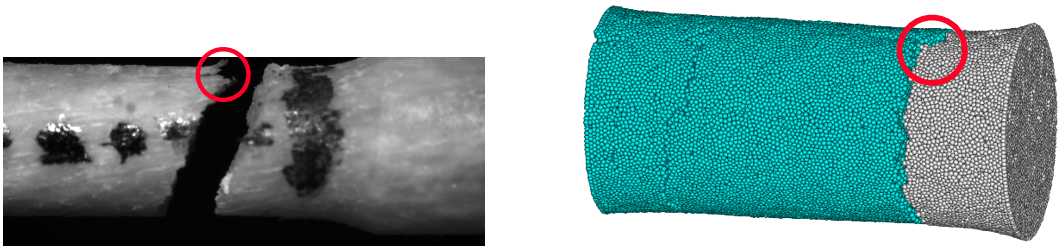
Contrairement à l'exemple précédent, il est ici nécessaire de calibrer les paramètres mécaniques des

liaisons cohésives vis-à-vis de la réponse physique de l'éprouvette, en particulier le module d'Young des poutres E_μ et leur section, via le rapport de leur rayon sur celui des éléments discrets, noté R_r. Cette calibration est effectuée grâce à la force de traction mesurée par la machine ainsi qu'au suivi de mires pointées sur l'échantillon, qui permettent de mesurer son allongement. Les paramètres calibrés sont indiqués dans le Tableau 3. Le domaine ainsi calibré est enfin sollicité en traction jusqu'à la rupture, pour validation de la localisation et du faciès de rupture.

Module d'Young E_μ	Rapport des rayons R_r	Contrainte à rupture $\sigma_{f\mu}$
6.56×10^{10} Pa	0.63	5.9×10^{8} Pa

TABLE 3 – Paramètres calibrés du domaine discret

Le résultat de la simulation montre que l'initiation de la rupture se fait au même endroit, dans la zone entourée sur la Figure 3. Cette zone présente en effet une faiblesse dans la structure de l'éprouvette, conduisant ainsi à une concentration de contrainte.



(a) Éprouvette expérimentale (b) Éprouvette éléments discrets

FIGURE 3 – Comparaison des profils de rupture expérimental et numérique

Cependant, la propagation ne se fait pas de la même manière, et reste perpendiculaire à l'axe de traction, contrairement à l'éprouvette expérimentale. Plusieurs hypothèses sont émises à ce stade, notamment la densité d'éléments discrets utilisés dans le domaine, $\sim$ 100k, qui est insuffisante pour représenter certaines zones fines de l'éprouvette. Celles-ci ne contiennent que quelques éléments dans leur section et ne représentent pas la tenue mécanique effective du matériau. D'autre part, des questions se posent sur la représentation numérique de l'os cortical, en particulier la présence de fluides supposés incompressibles emprisonnés dans les canaux. Dans cette simulation, les canaux sont vides et donc compressibles. Tenir compte, a minima, de ces deux hypothèses conduira rapidement à des modèles conséquents, de l'ordre de $\sim$ 500k éléments discrets. L'utilisation de la version GPU de GranOO 4 prendra alors tout son sens.

5.3 Impact ballistique

Ce dernier exemple traite de la simulation numérique d'un impact à 245 m/s sur un matériau tissé représenté à l'échelle dite mésoscopique, c'est-à-dire celle des fils (voir [12] pour plus de détails sur l'implémentation). De telles structures sont la plupart du temps modélisées à l'aide d'éléments finis, ce qui entraîne des calculs très longs, notamment en raison du traitement des contacts. Dans cette simulation, chaque fil est modélisé comme un corps déformable et discrétisé à l'aide de multiples barres élastiques. Les éléments discrets aux extrémités sont répartis uniformément dans la section du fil. Ces éléments sont reliés entre eux selon un motif de grille qui représente le comportement orthotrope du fil. Le même diamètre et la même densité sont adoptés pour chaque élément, qui est maintenu sphérique pour le traitement de la détection de contacts. Le comportement orthotrope des fils est modélisé à l'aide d'un motif de ressorts longitudinaux et transversaux, et deux rigidités différentes sont prises en compte : une pour les ressorts longitudinaux dans la direction du fil, identifiée directement via le module élastique mesuré, et une pour les ressorts transversaux, supposés dix fois plus faibles.

Dans une précédente étude ([12]), des premiers résultats très encourageants avaient montré que cette description naïve du comportement de cette structure permettait néanmoins de très bien prédire le profil de vitesse au cours du temps pour des vitesses de l'ordre de la centaine de mètres par seconde. Chaque calcul, intégrant notamment la coûteuse phase de détection de contacts mais aussi la nécessité de représenter de nombreux fils (la dimension du carré de tissu à simuler est ici de 10x10cm), aboutissait à une

simulation contenant plusieurs centaines de milliers de nœuds et d'éléments. Ce calcul a été relancé avec la dernière version GPU de GranOO et contient 300k éléments et 500k liens. La figure 4 présente, pour trois instants au cours de l'impact, le champ de vitesse hors plan suivant $\vec{z}$ ainsi que le champ de vitesse plan-transverse suivant l'axe $\vec{x}$.

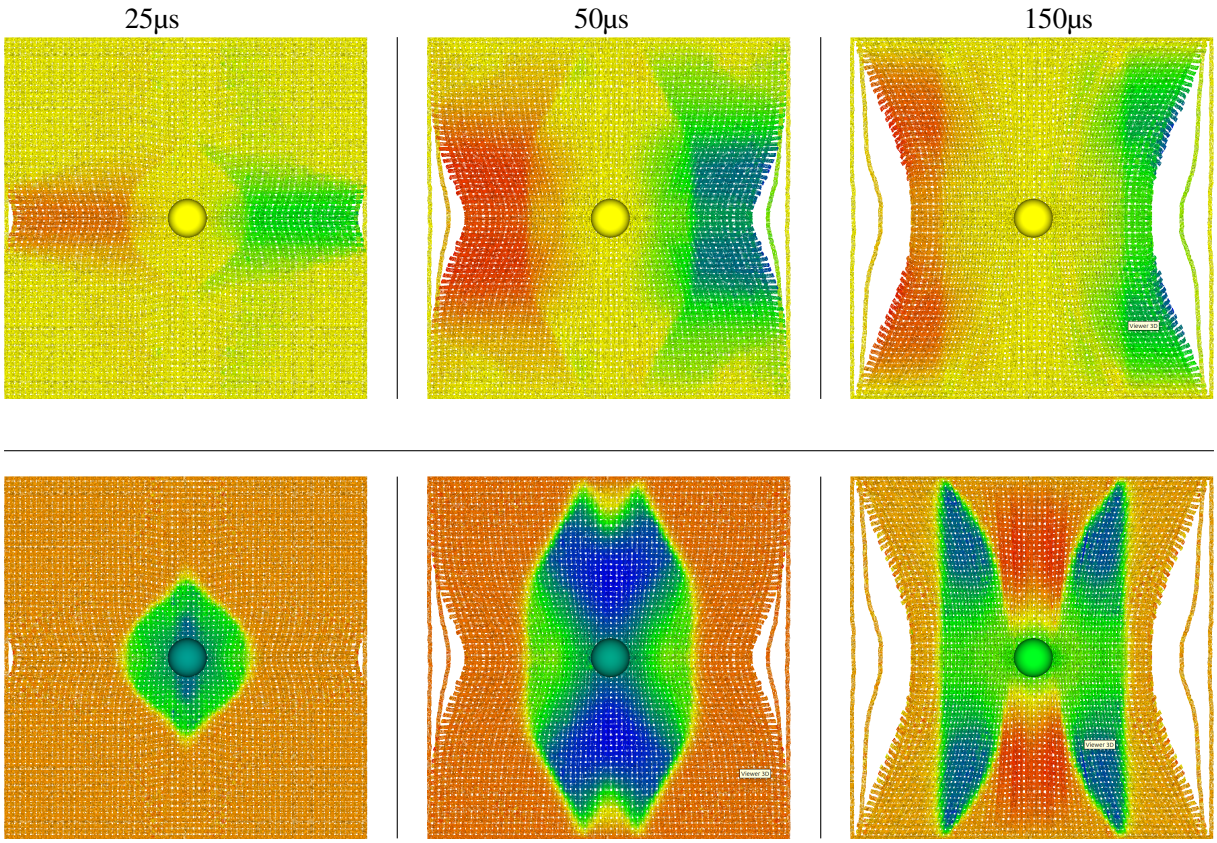


FIGURE 4 – Vue du dessus et évolution des champs de vitesses pour trois instants différents au cours de l'impact à 245 m/s. La ligne du haut représente le champ de vitesse suivant l'axe horizontal dans le plan, la ligne du bas représente le champ de vitesse hors plan dans la direction de l'impact.

Ces résultats illustrent les capacités de la DEM à traiter l'impact sur les tissus secs à l'échelle mésoscopique, comme déjà discuté dans de précédents travaux. La vitesse de calcul de cette simulation lancée sur la dernière version de GranOO est en revanche bien plus élevée, passant d'un temps de calcul de 20h sur l'ancienne version monothread à 55 minutes sur le même ordinateur (station portable, processeur i9, carte graphique NVIDIA RTX 3000). La résolution d'une algorithmie identique sur un environnement GPU permet alors de se projeter vers de futures simulations d'impacts sur textiles intégrant de nouvelles contraintes de conception, comme la présence de plusieurs plis mais aussi des tissages plus complexes et 3D, à l'instar du tissage interlock.

6 Conclusion

La version 4 de GranOO marque une évolution majeure de la plateforme, tant par la réorganisation profonde de son architecture logicielle que par l'intégration d'un moteur de calcul GPU performant et flexible. Cette refonte permet désormais d'exécuter des simulations de grande échelle auparavant inaccessibles, tout en conservant la philosophie d'un environnement orienté objet, modulaire et extensible. Les validations numériques montrent une très bonne cohérence entre les calculs CPU et GPU, tandis que les benchmarks mettent en évidence des gains de performance pouvant dépasser un facteur 40 selon les cas étudiés. Les exemples applicatifs illustrent la capacité de GranOO v4 à traiter des phénomènes complexes tels que la fissuration fragile ou les couplages multiphysiques avec une précision et une rapidité accrues. Enfin, fidèle à son engagement *open source*, GranOO v4 est librement accessible sur la

plateforme GitLab du CNRS [1], permettant à la communauté scientifique de contribuer, d'étendre et de déployer aisément cette nouvelle génération d'outils de simulation discrète.

Références

[1] Peter A Cundall and Otto DL Strack. A discrete numerical model for granular assemblies. *geotechnique*, 29(1) :47–65, 1979.

[2] Shengqiang Jiang, Chao Tang, Xu Li, Yuanqiang Tan, Ruitao Peng, Dongmin Yang, and Sisi Liu. Discrete element modeling of the machining processes of brittle materials : Recent development and future prospective. *The International Journal of Advanced Manufacturing Technology*, 109(9) :2795–2829, 2020.

[3] Damien André, Jean-Luc Charles, and Ivan Iordanoff. *3D Discrete Element Workbench for Highly Dynamic Thermo-mechanical Analysis : GranOO*. John Wiley & Sons, 2015.

[4] Cédric Hubert, Damien André, Laurent Dubar, Ivan Iordanoff, and Jean-Luc Charles. Simulation of continuum electrical conduction and joule heating using dem domains. *International Journal for Numerical Methods in Engineering*, 110(9) :862–877, 2017.

[5] Vincent Longchamp, Jérémie Girardot, Damien André, and Frédéric Malaise. Development of a numerical protocol for the very high strain rate dynamic fragmentation of porous-brittle materials at the microstructure scale. *Comptes Rendus. Mécanique*, 353(G1) :1085–1109, 2025.

[6] T. Pazmiño, I. Pombo, J. Girardot, L. Godino, and J.A. Sánchez. Multiscale simulation of volumetric wear of vitrified alumina grinding wheels. *Wear*, 530–531 :205020, October 2023.

[7] Guang-Yu Liu, Wen-Jie Xu, Nicolin Govender, and Daniel N Wilke. Simulation of rock fracture process based on gpu-accelerated discrete element method. *Powder Technology*, 377 :640–656, 2021.

[8] Damien André, Ivan Iordanoff, Jean-luc Charles, and Jérôme Néauport. Discrete element method to simulate continuous material by using the cohesive beam model. *Computer methods in applied mechanics and engineering*, 213 :113–125, 2012.

[9] Damien André, Jérémie Girardot, and Cédric Hubert. A novel DEM approach for modeling brittle elastic media based on distinct lattice spring model. *Computer Methods in Applied Mechanics and Engineering*, 350 :100–122, June 2019.

[10] CP Jiang, XF Wu, Jia Li, Fan Song, YF Shao, XH Xu, and Peng Yan. A study of the mechanism of formation and numerical simulations of crack patterns in ceramics subjected to thermal shock. *Acta Materialia*, 60(11) :4540–4550, 2012.

[11] Xavier Roothaer. *Approche multi-échelle du comportement mécanique des os porteurs et non-porteurs : vers une personnalisation des modèles numériques EF de l'être humain*. PhD thesis, 2019. Université Polytechnique Hauts-de-France 2019.

[12] J. Girardot and F. Dau. A mesoscopic model using the discrete element method for impacts on dry fabrics. *Matériaux & Techniques*, 104(4) :408, 2016.

1. https://src.koda.cnrs.fr/damien.andre.2/granoo.git