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

A-set – Outils et méthodes de simulation de nouvelle génération pour les matériaux et structures

J. Besson[1], C. Bovet[2], V. Chiaruttini[3], J. Cortial[3], J.-D. Garaud[2*], A. Geoffre[1], V. Kehr-Candille[2], B. Marchand[4], G. Medghoul[1], R. Messahel[3], J. Rannou[2], N. Rogalski[2], R. Sanchez[2], J.-M. Scherer[1]

[1] *Centre des Matériaux, Versailles Satory, 78000*
[2] *DMAS, ONERA, Université Paris-Saclay, 92320, Châtillon, France*
[3] *Digital Sciences & Technologies, Safran Tech, Magny-Les-Hameaux, 78114, France*
[4] *Anciennement au Centre des Matériaux, Versailles Satory, 78000*
[*] *correspondant : jean-didier.garaud@onera.fr*

Résumé — A-set est une suite d'outils numériques dédiée à la simulation numérique des matériaux et structures, centrée autour des éléments finis. Trois piliers sont au cœur du projet : la simulation massivement parallèle, le remaillage et l'interopérabilité. La bibliothèque de modélisation matériaux est construite de manière à être numériquement efficace, tout en étant facilement extensible pour un utilisateur-développeur de lois de comportement. Elle est déjà riche de modèles plus ou moins classiques : thermo-élasto-visco-plasticité, plasticité cristalline, petites et grandes déformations, zones cohésives. Afin d'étudier les problématiques de fissuration, un outil d'adaptation de maillages dédié et performant est proposé. La bibliothèque éléments finis est construite sur les mêmes principes d'extensibilité et de performances. Ces dernières reposent sur l'utilisation conjointe de plusieurs idées : calculer ce qui peut l'être dès la compilation et exploiter le parallélisme multiniveaux (multithreading, décomposition de domaine). Les excellentes performances ainsi obtenues sont illustrées dans cet article. Le logiciel est distribué sous une double licence Open-source et privative.
Mots clés — Simulation numérique, modélisation, éléments finis, mécanique des matériaux, mécanique des structures, calcul haute performance, maillage, interopérabilité.

1 Introduction

A-set est une suite d'outils numériques centrée sur la méthode des éléments finis et dédiée à la simulation numérique des matériaux et structures. Trois piliers sont au cœur du projet : la simulation massivement parallèle exploitant différents niveaux de parallélisme, le remaillage et l'interopérabilité.

Le périmètre applicatif couvre les simulations de mécaniques non-linéaires, allant jusqu'aux problématiques de fissuration. Le code étant écrit de façon générique, il est aussi adapté à d'autres gammes de problèmes tels que la diffusion, les champs de phase, la dynamique transitoire.

Le code est adapté aux nouvelles technologies (hardware, middleware, bibliothèques) et aux évolutions des méthodes numériques dans ce domaine. Afin d'allier performances et interopérabilité, les langages C++ (norme C++20) et Python sont utilisés ; une attention particulière a notamment été portée aux classes portant les données de simulation, afin que ces données respectent, selon le besoin :

— soit une contiguité mémoire pour être compatibles avec la vectorisation des opérations proposée par le compilateur ou les bibliothèques d'algèbre linéaire tierces, et pour partager des *vues mémoires* sur ces données entre librairies, qu'elles soient écrites en C/C++ ou Python ;
— soit une localité spatiale (*thread-safe* et affinité aux *threads* et *cache*) pour les opérations dynamiques en contexte fortement multithreadé.

Ces deux points seront illustrés sur des cas concrets dans les sections suivantes.

Le développement du projet a été tiré par des cas de validation et défis de simulation : des cas de validation industrielle, démontrant l'efficacité sur des modèles déjà établis d'une part, des défis scientifiques liés à la modélisation multiéchelles et aux très hautes températures d'autre part. Par ailleurs, les outils développés ont vocation à être intégrés dans les différentes chaînes de simulation (pré/post, multiphy-

sique, essais-calculs, optimisation) des partenaires facilitant leur adoption et permettant une meilleure intégration aux processus industriels.

Ce code codéveloppé entre différents centres de recherche répond à plusieurs objectifs et besoins : d'une part ce doit être le support à la recherche en mécanique numérique des partenaires académiques et industriels pour les prochaines années, et un démonstrateur de la faisabilité de ces nouvelles méthodes sur des cas réalistes. D'autre part il doit pouvoir aborder les simulations exceptionnelles nécessaires aux travaux d'expertise et à la R&T. Enfin le code doit pouvoir être utilisé par les étudiants pour l'enseignement de la mécanique et du calcul scientifique.

Ainsi, les développeurs ont la volonté de mettre en Open Source (GPL) une partie significative et attractive, aux fins de collaborations académiques, partenariales recherche-industrie ou d'enseignement. En parallèle, afin de préserver certains savoir-faire des partenaires, le code sera aussi distribué sous une licence privative accordant la possibilité de développer des modules privatifs différenciants et valorisables, de fournir du support et des formations.

2 Module d'intégration pour les lois de comportement A-mat

Le projet A-set comporte son propre module pour l'intégration des lois matériaux, nommé A-mat. Ce module est indépendant du moteur éléments finis A-solve, et peut donc être associé à n'importe quel autre solveur. Le code A-mat a été pensé pour le calcul haute performance depuis son architecture de base. Les appels à la loi de comportement étant généralement nombreux et indépendants, la structure du code a été conçue pour accueillir nativement le calcul parallèle, de manière efficace, en garantissant notamment la *thread-safety*. L'approche globale repose ainsi sur un tableau de données supérieur, partagé par les points d'intégration répondant au même matériau, et stockant les données nécessaires à l'intégration de la loi. Chacun des points d'intégration accède aux données le concernant via ce tableau commun, sans problème d'accès concurrents dans un contexte de calcul parallèle. Ceci permet à A-mat de supporter naturellement le parallélisation, que ce soit le calcul *multithread* voire GPU. L'approche est représentée en Figure 1.

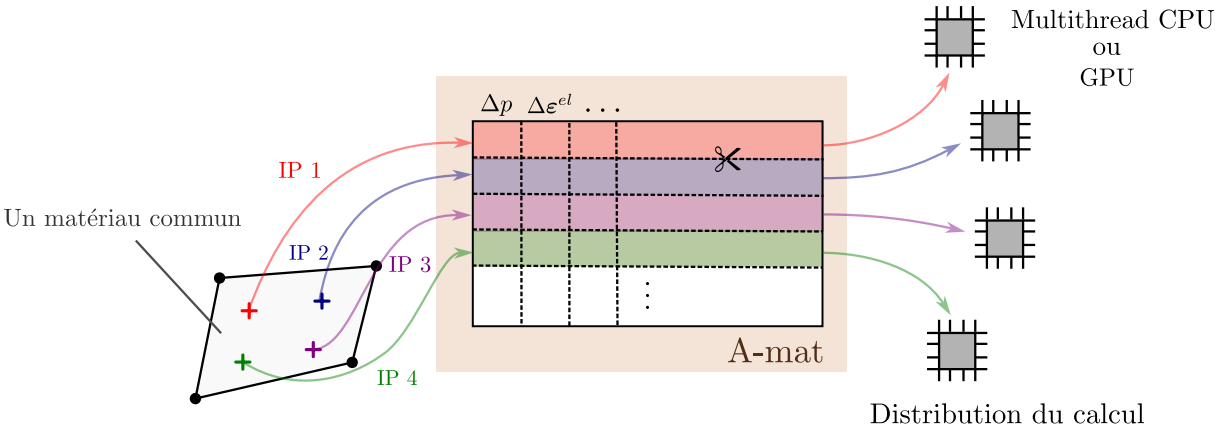


FIGURE 1 – Schéma de principe de la stratégie de placement des données et du *calcul haute performance* du code A-mat, calcul par bloc de points d'intégration.

L'implémentation d'A-mat repose en grande partie sur la bibliothèque de calcul scientifique Eigen. Cette bibliothèque permet notamment de manipuler des objets de type *Tensor*, similaires aux tenseurs du mécanicien. Ainsi, les implémentations sont simplifiées et le code se présente au plus proche des équations. Le formalisme tensoriel permet notamment de s'affranchir de la notation de Voigt pour laquelle la disparité de convention peut entraîner des ambiguïtés d'interprétation pour les résultats. Les implémentations peuvent être testées de façon directe à l'aide d'un simulateur pouvant travailler avec une loi quelconque, tout en considérant des trajets de chargement complexes.

L'utilisateur dispose d'une interface YAML permettant de piloter simplement ses choix de matériaux et de paramètres de résolution. Le code présente les algorithmes de résolution usuels :

— Explicite : Runge-Kutta, Runge-Kutta adaptatif, ...

— Implicite : Euler, θ-méthode, ...
— Spéciaux : retour radial, implicite à partir d'une estimation rapide explicite, ...

A-mat propose des lois de comportement fréquemment rencontrées en mécanique des matériaux. Nous pouvons notamment indiquer le modèle *GenEVP* permettant de donner un cadre élasto-visco-plastique modulaire à l'utilisateur. Ce modèle, hérité du logiciel Z-set [1, 2], permet le choix arbitraire d'une élasticité, d'un critère de plasticité, d'un écoulement plastique ainsi que d'écrouissages isotrope et cinématique, permettant une grande flexibilité en termes de modélisation (Figure 2). En outre, d'autres modèles avancés sont aussi présents dans la bibliothèque A-mat, parmi lesquels des modèles de plasticité cristalline, des modèles hypo et hyper-élastiques, des comportements de zones cohésives, ...

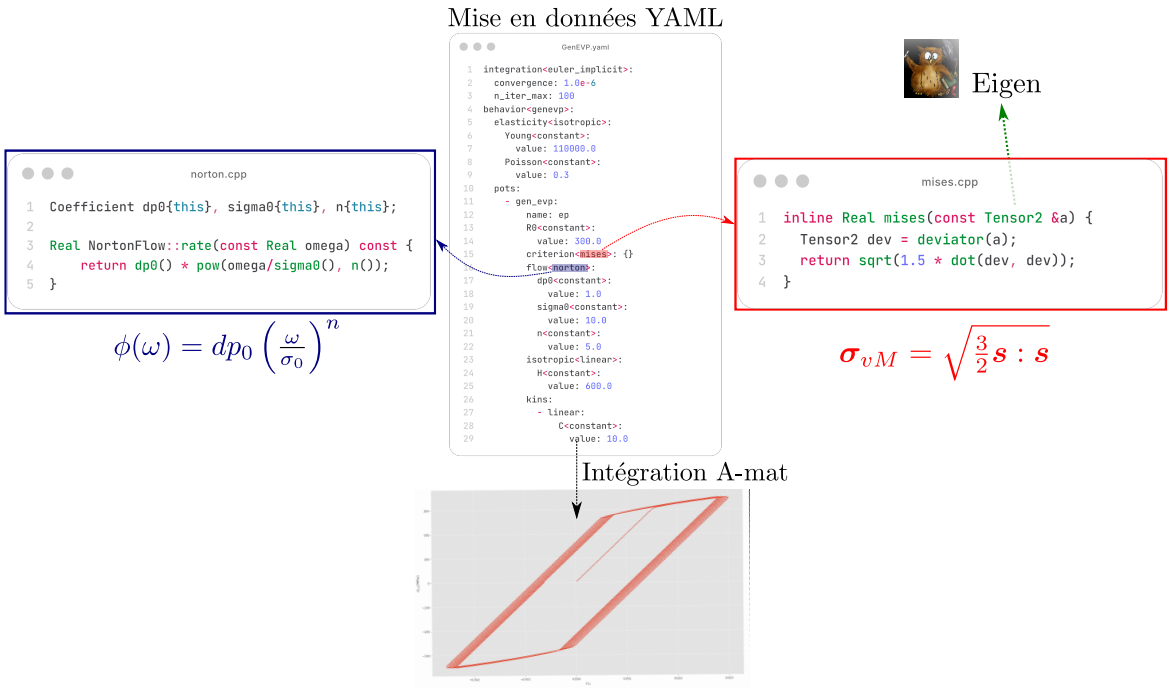


FIGURE 2 – Mise en données YAML et implémentations du critère de von Mises et l'écoulement plastique de Norton dans A-mat.

3 Spécificités de la structure du code

3.1 Éléments finis génériques

A-set dispose d'un mécanisme générique de construction d'éléments finis s'appuyant sur une structure fortement « template » du C++. L'intérêt principal est que tout ce qui est connu dès la compilation est exploité et optimisé par le compilateur tout en préservant la généricité. Une fois choisis le support géométrique (topologie, nombre de noeuds), le type d'interpolation, la règle de quadrature et la nature et le nombre des degrés de liberté (champ scalaire ou vectoriel), on est en mesure de précalculer un certain nombre de tableaux tels que ceux contenant les valeurs des fonctions de forme ou de leurs gradients à chaque point d'intégration dans l'élément de référence isoparamétrique.

Les éléments sont des classes templatées par une ou plusieurs « règles » basées sur des classes de politique, c'est à dire des classes définissant des comportements quasiment orthogonaux. Les principales politiques définies pour chaque règle sont l'interpolation, la quadrature, la topologie du support géométrique et la nature des degrés de liberté (scalaire, vectorielle). La plupart des éléments continus classiques se construisent de manière générique avec une unique règle comme illustré sur la Figure 3. Les politiques ne sont cependant pas totalement orthogonales : on doit par exemple choisir une règle de quadrature compatible avec la topologie du support.

À partir d'une seule implémentation des fonctions de construction d'un opérateur tangent, de calcul des forces internes, etc., il est ainsi possible de générer la plupart des éléments finis classiques (éléments continus, éléments d'interface, etc.) en combinant des politiques compatibles entre elles.

Une originalité de notre démarche est l'utilisation de concepts de métaprogrammation template qui permet de généraliser cette structure à plusieurs « règles ». On peut ainsi générer des éléments hybrides à plusieurs champs comportant des types d'interpolation ou de quadrature différenciées ; ces formulations hybrides [3] sont utiles par exemple pour les formulations non locales, les matériaux incompressibles ou encore pour certaines formulations coque.

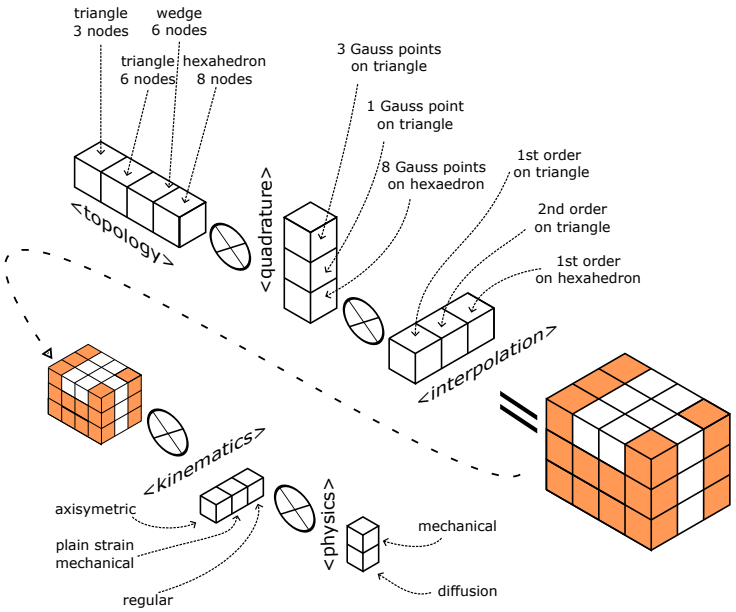


FIGURE 3 – Schéma de principe de la construction générique d'éléments finis à partir de « classes de politique » (interpolation, quadrature, hypothèses cinématiques, ...).

3.2 Interfaces : Usine à objets, langage de mise en données, Python

L'extensibilité requiert la possibilité de pouvoir développer des plugins. À la base se trouve donc classiquement une *usine à objets*. Un aspect moins classique de notre démarche réside dans le fait de la jumeler avec une usine des *paramètres* de ces objets. Celle-ci permet une généricité complète au niveau du format des données d'entrée utilisateur. À titre d'exemple, l'utilisateur a actuellement à sa disposition trois formats de fichier d'entrée : JSON, YAML et Python.

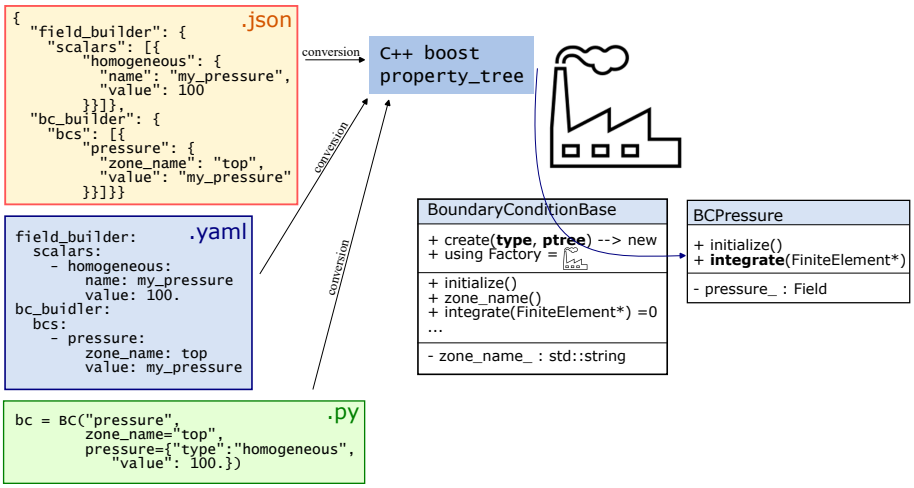


FIGURE 4 – Uniformité des mises en données : à partir de la déclaration des objets C++ et de leurs paramètres, l'*object factory* réalise la création et l'initiatisation à partir de multiples langages, du moment que ce dernier peut être convertit en `boost::property_tree`.

La bibliothèque *Nanobind* [4] est utilisée pour faire la passerelle entre les classes C++ et leur homologue Python. Très compacte d'utilisation, elle offre la possibilité de piloter entièrement un calcul depuis

un script python, ouvrant ainsi de nombreuses opportunités.

4 Maillages et adaptations

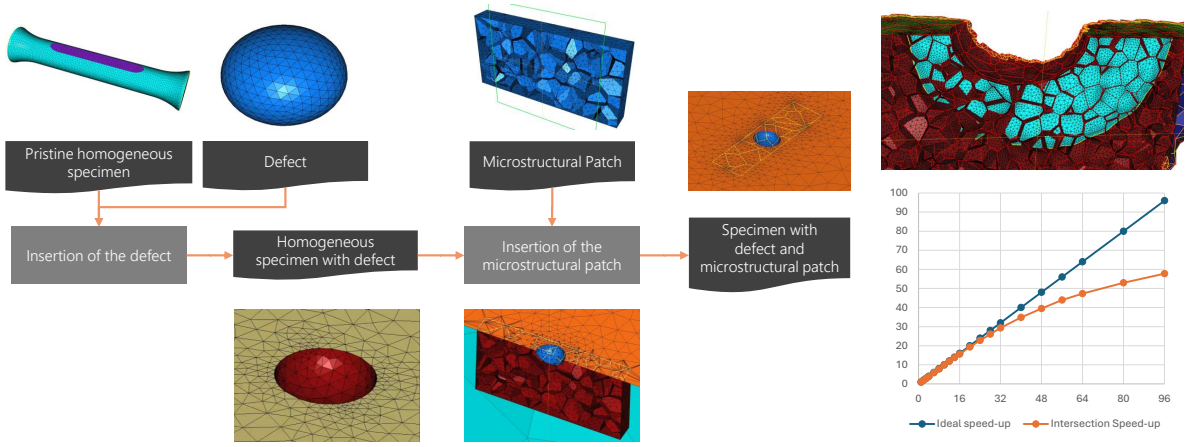
`DynamicMesh` est la bibliothèque de remaillage dynamique intégrée au code A-set. Développée en C++ et parallélisée avec *Intel TBB* [5], elle permet des opérations rapides et robustes de modification de maillages conformes, nécessaires aux simulations non-linéaires de grande taille.

4.1 Description de la bibliothèque de maillage

La bibliothèque offre des mécanismes efficaces d'insertion, suppression et remplacement de cellules polyédriques, ainsi que de mise à jour cohérente de la connectivité. Elle s'appuie sur une représentation unifiée des entités (cellules, sommets) et sur une gestion structurée des ensembles géométriques (sets).

Les cellules (types `P1`/`P2`) et sommets sont identifiés par un type topologique et un indice unique. La qualité élémentaire, liée au conditionnement géométrique, sert de critère aux opérations adaptatives. Les ensembles géométriques correspondent aux frontières où peuvent s'appliquer des chargements et contacts, aux domaines disposant de comportements matériaux distincts, ou aux fronts de fissure par exemple ; ils sont indispensables à la résolution efficace avec des approches par remaillage adaptatif.

Deux représentations sont disponibles pour les groupes d'entités : (i) explicite et (ii) par étiquettes (tags). Une opération de *color mapping* permet de passer de l'une à l'autre, garantissant une mise à jour efficace lors d'opérations massives de remaillage, interne ou effectué par des librairies externe (`CDT`, `MMG`, `MeshGems`, etc.). Les suppressions sont différées puis compactées via un *garbage collection*. L'implémentation repose sur des conteneurs *thread-safe*, une gestion des alignements sur mémoire cache et des boucles parallèles, assurant une forte extensibilité avec par, exemple, sur l'algorithme d'intersection de maillages surfaciques une mise à jour parallèle des entités, permettant des accélérations de facteur > 50 entre 1 et 96 cœurs sur des problèmes cibles (cf. Figure 5).



(a) Chaîne de génération du modèle multiéchelle : insertion d'un défaut puis d'un patch microstructural pour la simulation haute-fidélité de fissures courtes.

(b) Extensibilité forte jusqu'à 96 cœurs de l'algorithme d'intersection.

FIGURE 5 – `DynamicMesh` appliqué aux discrétisations complexes en fissuration.

`DynamicMesh` est déjà utilisé dans des travaux de recherche en cours, pour la génération automatique de maillages conformes combinant géométrie macroscopique, défaut de surface et patch microstructural, dans le cadre de l'étude des fissures courtes (thèse de M. Bouyx, Centrale Nantes–ONERA–Safran, Figure 5). Les performances du code rendent possible la réalisation de dizaines de calculs haute-fidélité de propagation de fissure en quelques heures, ouvrant la voie à des analyses statistiques de l'impact de la variabilité microstructurale sur composants réels.

Cette capacité est un élément clé des stratégies de modélisation multiéchelle visant à prédire l'influence conjointe des défauts de surface et de la microstructure locale sur la tenue en fatigue.

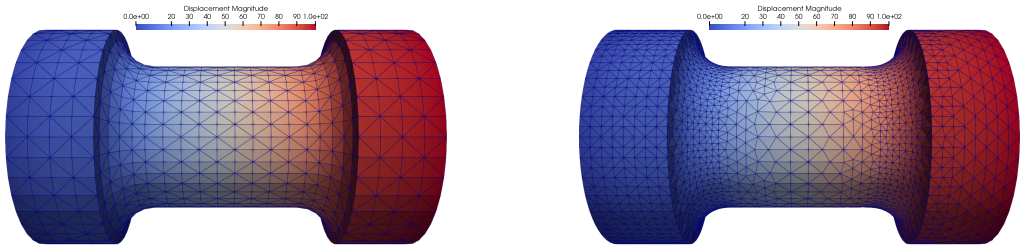
4.2 Remaillage adaptatif

`DynamicMesh` constitue l'une des trois briques élémentaires du processus de remaillage adaptatif dans `A-set`. La deuxième brique réside dans la définition de critères de déclenchement de ce processus. Ces critères sont généralement basés sur des indicateurs d'erreurs de discrétisation et servent par ailleurs à définir les nouvelles cartes de taille de maille pour le (dé)raffinement. Les indicateurs basés sur les travaux de Zienkiewicz & Zhu (ZZ1, ZZ2) [6, 7], exploitant les défauts de régularité, ont été implémentés.

La dernière brique est la méthode de transfert de champs respectant certaines conditions essentielles pour la poursuite du calcul (compatibilité entre les champs nodaux et les champs aux points d'intégration, équilibre, réduction de la diffusion numérique). À ce titre les transferts par collocation et par point le plus proche sont disponibles.

Les choix architecturaux et *l'object factory* facilitent l'ajout de nouveaux critères et/ou méthodes de transfert. L'ensemble de ces briques exploitent nativement le parallélisme à mémoire partagée.

Une illustration du processus de remaillage adaptatif est donnée sur la Figure 6. On considère une éprouvette, de matériau élastique, dont une de ses têtes est soumise à un déplacement imposé. On fixe à 5% le seuil maximal de l'erreur sur le champ de contrainte, évaluée via l'estimateur ZZ2. Comme attendu, les zones à forte variation sont bien raffinées. Dans cet exemple, l'ensemble du domaine est remaillé. Toutefois, grâce aux fonctionnalités de `DynamicMesh`, il est également possible de restreindre ce processus sur une zone d'intérêt particulière.



(a) Champ sur maillage initial.

(b) Champ après remaillage et transfert.

FIGURE 6 – Remaillage adaptatif d'une éprouvette sous sollicitation uniaxiale (la norme du déplacement est représentée).

5 Un code nativement haute performance

Le code est structuré de manière à exploiter nativement le parallélisme multi-niveaux, en mémoire partagée (*multithreading*) et en mémoire distribuée (`MPI`). Le *multithreading* intervient à différents niveaux. En plus du parallélisme fourni par les opérations BLAS 3 du noyau BLAS utilisé et du parallélisme hybride fourni par les solveurs directs comme Mumps [8], nous utilisons un parallélisme par tâche pour certaines étapes du calcul. Ce parallélisme par tâche repose sur la bibliothèque *Intel TBB*. Ainsi la parallélisation de la construction de l'opérateur éléments finis – via l'appel de la loi de comportement, le calcul du profil et le remplissage de la matrice creuse – a été particulièrement soignée. Différents algorithmes ont été implémentés comme le traditionnel *mutex* par colonnes, ou un algorithme *mutex-free* fondé sur un coloriage du graphe des éléments.

5.1 Parallélisme à mémoire partagée

Les performances tirées de la vectorisation et du parallélisme à mémoire partagée dépendent fortement d'un bon équilibrage de la charge entre cœurs et d'une gestion efficace des accès à la mémoire. Dans A-set, le vectorisation est gérée en grande partie par :

— le choix judicieux du *placement des données*, de *l'organisation de la mémoire* (*memory layout*), de *l'alignement des données*, mais aussi par

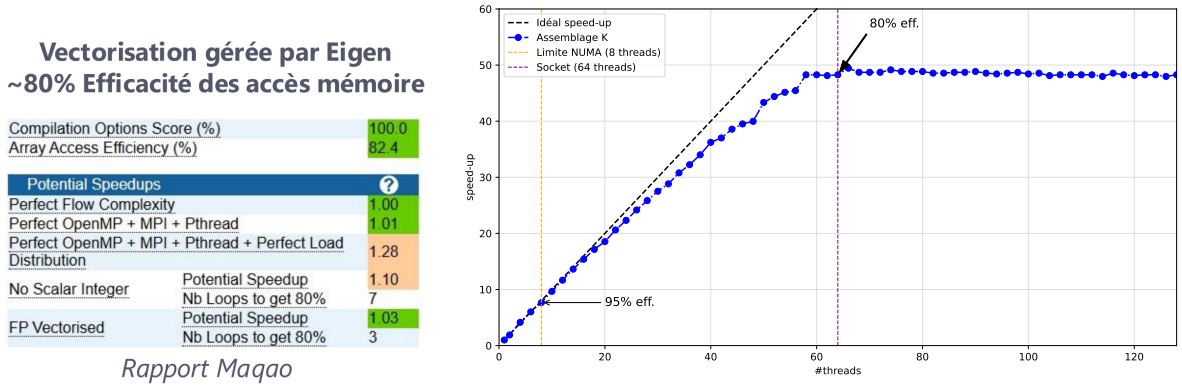
— la délégation des opérations vectorielles BLAS 3 à la librairie d'algèbre linéaire *Eigen*.

Le parallélisme à mémoire partagée repose sur la bibliothèque `OneAPI TBB` [5], privilégiant le parallélisme par tâches pour son algorithme efficace de rééquilibrage de charge par vol de tâche et sa gestion des *pools* de threads avec une surcharge minimale, particulièrement adapté aux maillages non structurés et hybrides qui induisent un déséquilibre naturel des charges de calcul.

Pour évaluer les performances du parallélisme à mémoire partagée dans A-set, notamment dans les modules A-mat et A-solve, nous avons mené une étude d'extensibilité forte sur l'assemblage de la matrice globale de raideur. Cette étude inclut à la fois l'appel à l'intégration de la loi de comportement dans A-mat et les calculs ainsi que l'assemblage des matrices élémentaires dans A-solve. Il est appliqué à un cube élastique linéaire comportant 200 éléments par direction, soit 24 millions de degrés de liberté. Les simulations ont été effectuées sur un nœud équipé de deux processeurs AMD EPYC 9534 64-Core (architecture Zen 4), soit un total de 128 cœurs répartis sur deux sockets.

L'analyse effectuée à l'aide de l'outil MAQAO [9], illustrée à la Figure 7(a), montre que la *vectorisation* dans A-set est très performante. En effet, l'efficacité des accès mémoire atteint un score d'adaptation du mapping des données à l'architecture cible de 82,4%, pour un maximum possible de 100% (voir [9]).

Les performances de l'assembleur multithreadé sont présentées sur la Figure 7(b), où l'on constate une bonne scalabilité jusqu'à la limite d'un socket (64 cœurs). On note ainsi des niveaux d'efficacité en multithreading atteignant 95% pour 8 threads, et 80% pour 64 threads, cette dernière valeur correspondant à la taille d'un socket.



(a) Rapport MAQAO des performances de la vectorisation.

(b) Extensibilité forte jusqu'à 128 cœurs de l'assemblage de la matrice globale de rigidité.

FIGURE 7 – Performances de (a) la vectorisation et (b) du multithreading pour l'assemblage de la matrice globale de rigidité, jusqu'à 128 cœurs, pour un cube élastique linéaire de $24 \cdot 10^6$ DDL.

Il est important de rappeler que la cible en production consiste à exploiter A-set conjointement avec la bibliothèque *Scalable Interface Algebra* (présentée dans la section 5.2) en mode hybride, combinant tous les niveaux de parallélisme sur des configurations à 8 ou 16 threads par rang MPI (8 correspondant à la taille d'un nœud NUMA sur les processeurs AMD Zen 4 utilisés pour les tests). Au regard de cette cible, les performances mesurées ici sont excellentes.

5.2 Parallélisme à mémoire distribuée

Le parallélisme à mémoire distribué repose sur la bibliothèque open source *Scalable Interface Algebra*, développée à l'Onera, qui implémente de manière efficace les méthodes de décomposition sans recouvrement. Ces méthodes consistent à distribuer le maillage global sur différents processeurs, les équilibres locaux sont résolus avec des solveurs directs (typiquement Mumps [8, 10] ou Rugged [11]) et un solveur itératif cherche à obtenir l'équilibre de l'interface entre sous-domaines. Les méthodes classiques BDD [12] et FETI [13] sont disponibles ainsi que les méthodes Adaptive Multipreconditionned FETI [14, 15] et Adaptive Multipreconditionned BDD [16]. Celles-ci permettent de résoudre, sur plusieurs milliers de cœurs, des systèmes très mal conditionnés caractéristiques des cas industriels.

Pour illustrer les performances du solveur, une étude d'extensibilité faible a été réalisée sur le calculateur Sator de l'Onera (Cascade Lake à 96 cœurs). L'étude est classiquement réalisée sur un cube élastique linéaire homogène en acier. Elle utilise un ratio $H/h = 40$, ce qui conduit à approximativement

des sous-domaines de 200k *ddls*, trois cœurs sont affectés à chaque sous-domaine. Les performances obtenues avec la méthode FETI sont données sur la Figure 8. Il faut noter que la méthode est générique, elle n'exploite pas les spécificités du problème éléments finis considérés (homogénéité etc.). Ainsi, il faut moins d'une minute pour résoudre un problème éléments finis de 190 millions d'inconnues, sur 3000 cœurs.

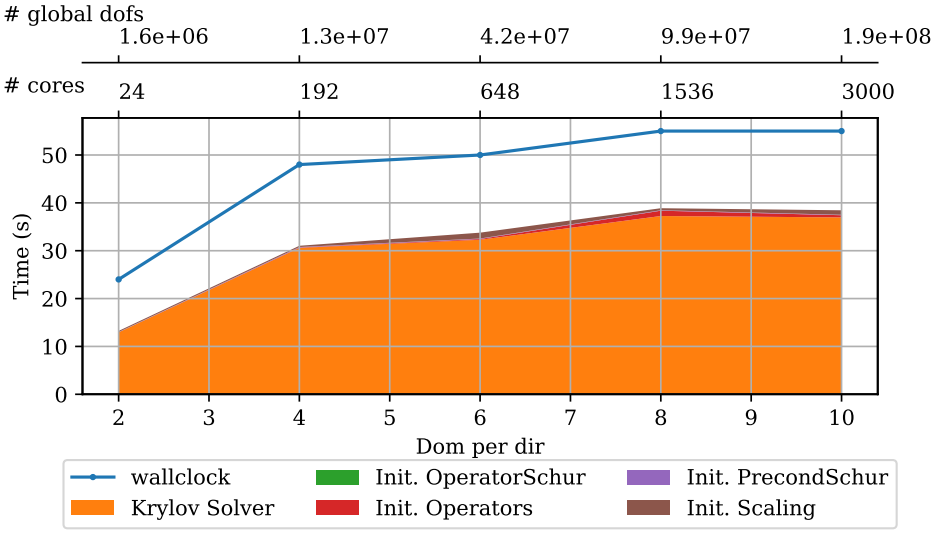


FIGURE 8 – Étude d'extensibilité faible sur un cube élastique homogène ($H/h = 40$, 3 cœurs/domaine).

6 Conclusion

A-set constitue une plateforme de simulation numérique de nouvelle génération, conçue pour concilier généricité des méthodes, calcul haute performance, discrétisation adaptative, et interopérabilité. Son architecture modulaire, articulée autour des briques A-mat, A-solve et DynamicMesh, permet d'aborder efficacement des problématiques complexes de mécanique non linéaire, de fissuration et de modélisation multiéchelle, tout en maintenant une excellente scalabilité sur des architectures HPC modernes, en mémoire partagée comme distribuée.

Les résultats présentés démontrent la capacité d'A-set à exploiter efficacement les différents niveaux de parallélisme, rendant accessibles des simulations de très grande taille, représentatives des cas industriels à haute-fidélité incluant, par exemple, la prise en compte de la variabilité des processus de production. La conception générique des éléments finis, des lois de comportement et des stratégies de remaillage garantit par ailleurs la forte extensibilité du code.

Enfin, A-set a vocation à s'inscrire dans une dynamique ouverte et collaborative. Sa diffusion sous double licence vise à favoriser les collaborations académiques et recherche–industrie, tout en assurant un cadre pérenne de développement. Les perspectives incluent l'extension vers les architectures hybrides CPU/GPU, le couplage avec des approches d'intelligence artificielle, et la structuration de futurs projets collaboratifs et consortiums autour de la simulation numérique avancée des matériaux et des structures.

Remerciements

Ce travail a reçu un soutien financier de la DGAC, de France relance et de l'Union européenne NextGenerationEU [subvention n°2021-08, projet ARIZE].

Références

[1] Jacques Besson, Georges Cailletaud, Jean-Louis Chaboche, and Samuel Forest. *Mécanique non linéaire des matériaux*. Hermès, 2001.

[2] Jean-Didier Garaud, Johann Rannou, Christophe Bovet, Sylvia Feld-Payet, Vincent Chiaruttini, Basile Marchand, Laurent Lacourt, Vladislav Yastrebov, Nikolay Osipov, and Stéphane Quilici. Z-set-suite logicielle pour la simulation des matériaux et structures. In *14ème Colloque National En Calcul Des Structures*, 2019.

[3] Yi Zhang, Eric Lorentz, and Jacques Besson. Ductile damage modelling with locking-free regularised gtn model. *International Journal for Numerical Methods in Engineering*, 113(13) :1871–1903, 2018.

[4] Wenzel Jakob. nanobind : tiny and efficient c++/python bindings, 2022. https ://github.com/wjakob/nanobind.

[5] James Reinders. *Intel Threading Building Blocks*. O'Reilly, 2007.

[6] Olgierd C. Zienkiewicz and J. Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. part 1 : The recovery technique. *International Journal for Numerical Methods in Engineering*, 33(7) :1331–1364, 1992.

[7] Olgierd C. Zienkiewicz and J. Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. part 2 : Error estimates and adaptivity. *International Journal for Numerical Methods in Engineering*, 33(7) :1365–1382, 1992.

[8] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1) :15–41, 2001.

[9] Cédric Valensi, William Jalby, Mathieu Tribalat, Emmanuel Oseret, Salah Ibnamar, and Kevin Camus. Using MAQAO to analyse and optimise an application. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 423–424, 2019.

[10] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures. *ACM Transactions on Mathematical Software*, 45(1) :1–23, 2019.

[11] Christophe Bovet. On the use of graph centralities to compute generalized inverse of singular finite element operators : Applications to the analysis of floating substructures. *International Journal for Numerical Methods in Engineering*, 124(9) :1933–1964, 2023.

[12] Jan Mandel. Balancing domain decomposition. *Communications in Numerical Methods in Engineering*, 9(3) :233–241, 1993.

[13] Charbel Farhat and Francois-Xavier Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering*, 32(6) :1205–1227, 1991.

[14] Christophe Bovet, Augustin Parret-Fréaud, Nicole Spillane, and Pierre Gosselet. Adaptive multipreconditioned FETI : Scalability results and robustness assessment. *Computers & Structures*, 193 :1–20, 2017.

[15] Christophe Bovet, Augustin Parret-Fréaud, and Pierre Gosselet. Two-level adaptation for adaptive multipreconditioned feti. *Advances in Engineering Software*, 152 :102952, 2021.

[16] Christophe Bovet, Théodore Gauthier, and Pierre Gosselet. On the Use of Block Low Rank Preconditioners for Primal Domain Decomposition Methods. *International Journal for Numerical Methods in Engineering*, 126(3) :e7623, 2025.