Réunion ANR Maturation

Marc Chung-To-Sang Mathieu Lobet

Maison de la Simulation

12 Mars 2026

Plan

Objectifs du post-doc

Améliorer l'efficacité de l'algorithme Sparse-PIC

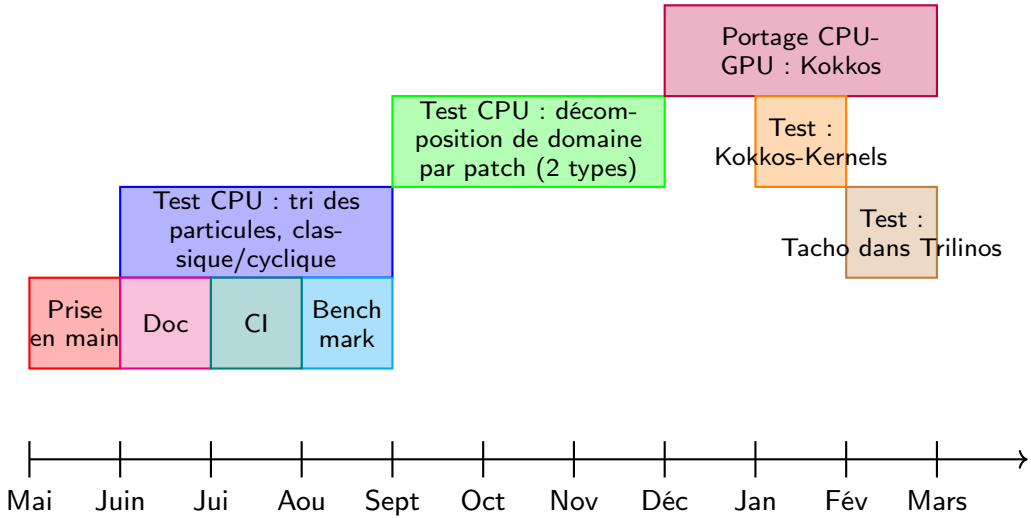
- Tirer profit d'une base de codes variée en Fortran et C++ (deux thèses sur le sujet, travaux entamés)
- Améliorer le parallélisme distribué (MPI/OpenMP), la vectorisation et le portage GPU

Explorer des modèles de programmation pour la portabilité de la performance

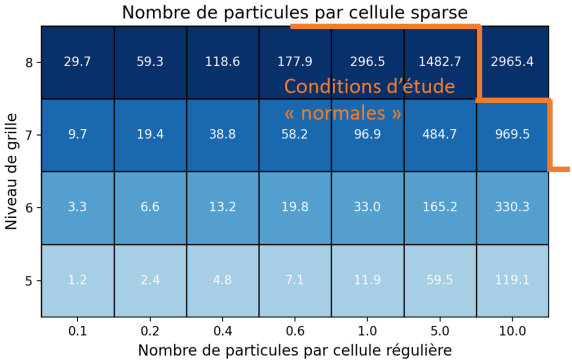
- Tester l'intégration de Kokkos et/ou StarPU
- « Bonnes pratiques logicielles » à définir
- Encadrer un stagiaire, et passage de témoin

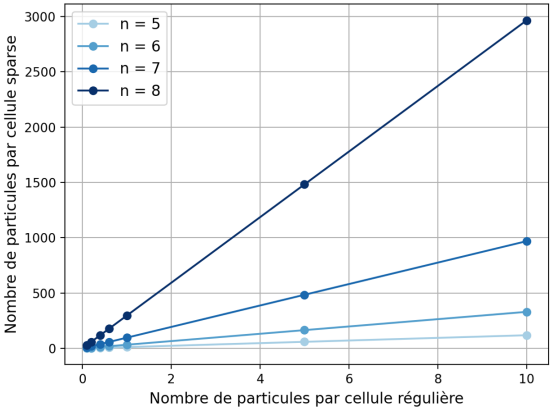
- Évaluer les différentes implémentations sur la performance et la portabilité (Tests sur diverses architectures parallèles, mix CPU/GPU, ...)

Charte chronologique 2025-2026



Comparaison du paramètre de nombre de particules SparsePIC et PIC



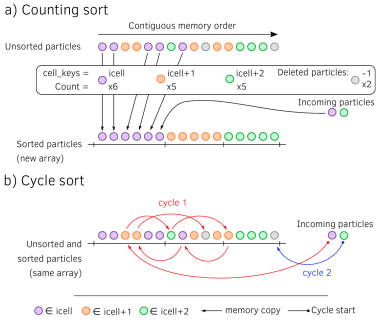


Tri des particules

Trier les particules pour le Sparse-PIC : bonne ou mauvaise idée ?

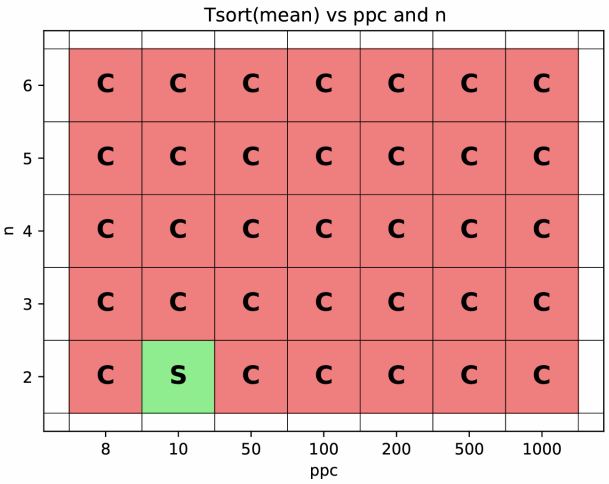
Réponse courte : mieux vaut éviter, le plus souvent, il n'y a pas assez de particules

Générée par Emmy Strale AI.



Adaptive SIMD optimizations in particle-in-cell codes with fine-grain particle sorting, Beck & al., 2019, Hellsevier

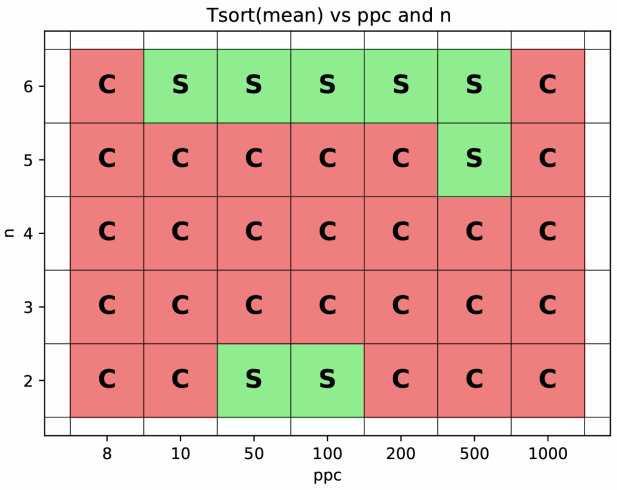
Tri classique vs cyclique : tableau déjà trié



C : cyclique ; S : classique ; tableau déjà trié

Générée par Emmy Strale AI.

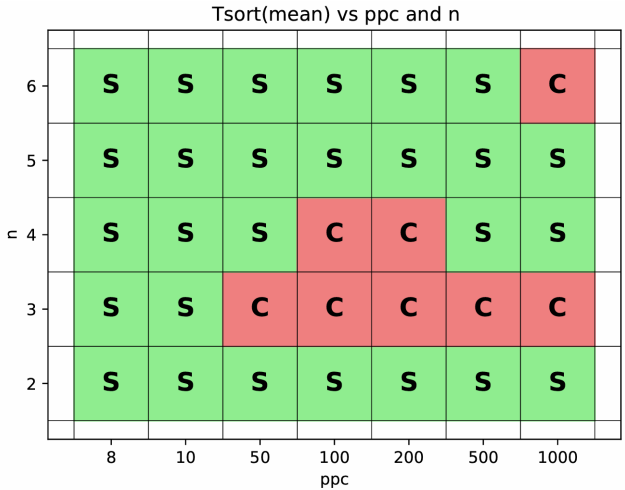
Tri classique vs cyclique : 25% de désordre



C : cyclique ; S : classique ; 25% de désordre

Générée par Emmy Strale AI.

Tri classique vs cyclique : 50% de désordre



C : cyclique ; S : classique ; 50% de désordre

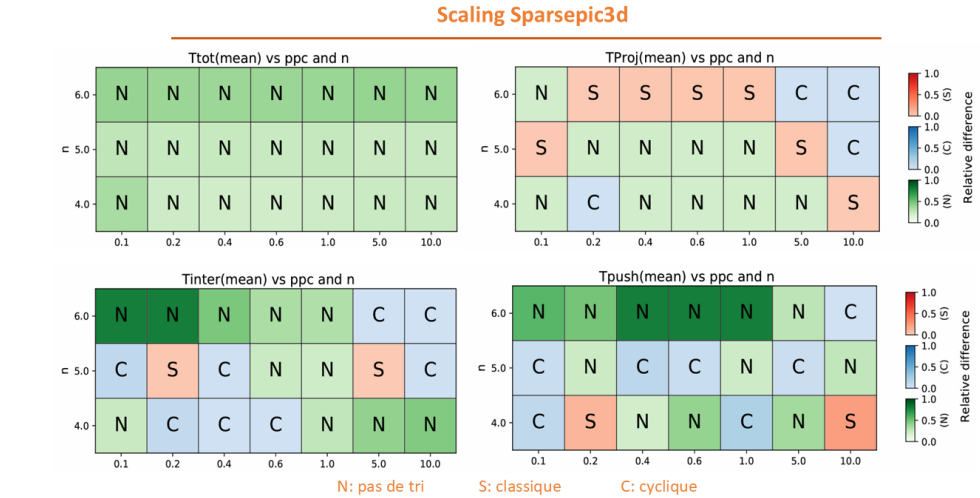
Générée par Emmy Strale AI.

Sans tri vs tri

Trier les particules pour le Sparse-PIC : bonne ou mauvaise idée ?

- On a comparé les deux tris l'un par rapport à l'autre, les résultats dépendent du "désordre" dans le tableau de particules.
- Le tri cyclique performe mieux à désordre faible, le tri classique à désordre élevé.
- On compare maintenant le code SparsePIC sans tri (option "N" pour "No sort") et avec tri (options "S" pour classic "sorting", et "C" pour "cyclic sorting").
- On propose une comparaison N, S, C en mettant en valeur la lettre correspondant au meilleur résultat, l'intensité de la couleur dépend de la différence relative par rapport à l'option la plus proche.

Sans tri vs tri

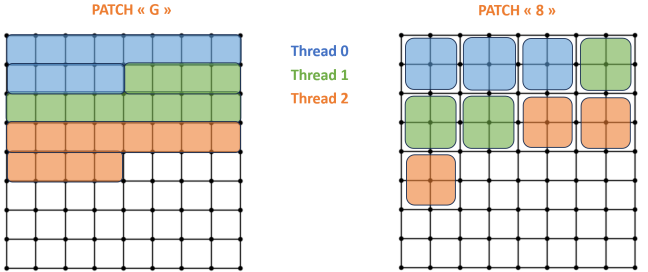


Décomposition de domaine sur CPU : intra-noeud

Diviser pour mieux "cacher"

Tests de deux types de patch différents répartis au sein d'un noeud de calcul CPU.

Générée par Emmy Strale AI.

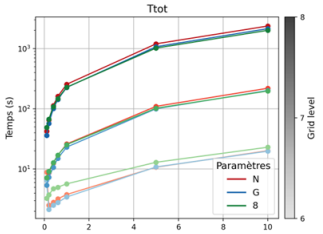


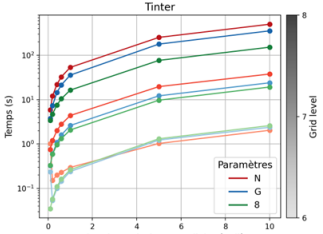
Patch par morceau de ligne, en suivant l'indiçage Fortran

Patch composé de plusieurs « petits » parallépipèdes rectangles

Décomposition de domaine sur CPU : intra-noeud

Scaling Sparsepic3d: 1 nœud, 1 task, 40 cpus/task

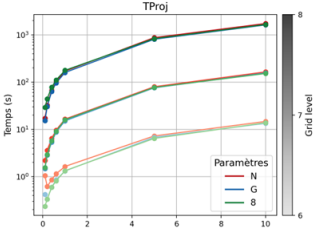


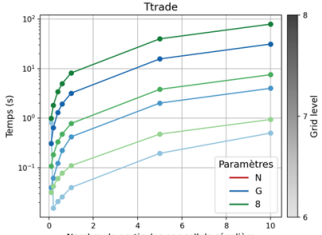


N: pas de tri/decomp

G: « gros patch »

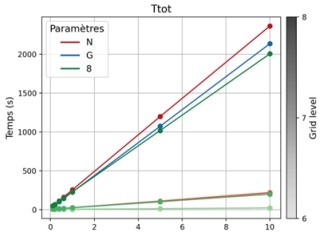
8: plusieurs « petits patch »

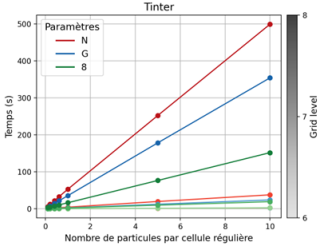




Décomposition de domaine sur CPU : intra-noeud

Scaling Sparsepic3d : 1 nœud, 1 task, 40 cpus/task

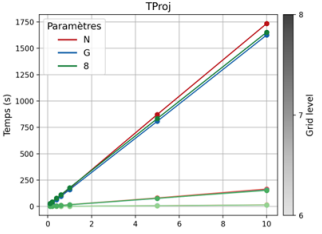


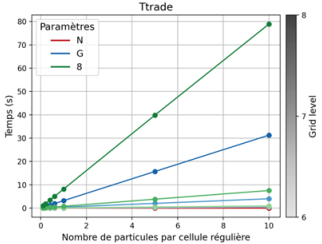


N: pas de tri/decomp

G: « gros patch »

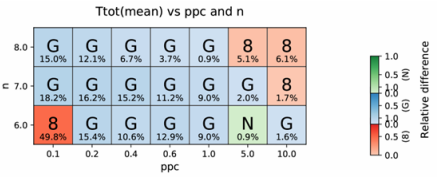
8: plusieurs « petits patch »

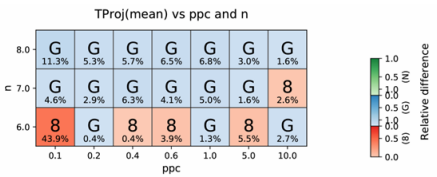


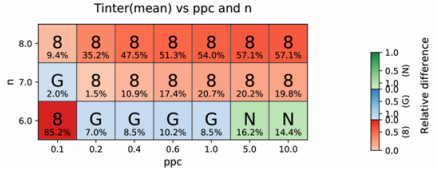


Décomposition de domaine sur CPU : intra-noeud

Comparaison N – patch G – patch 8: 1 nœud, 1 task, 40 cpus/task

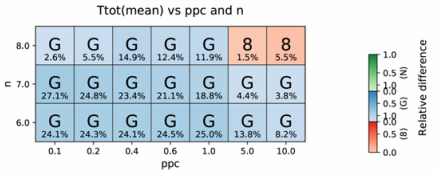


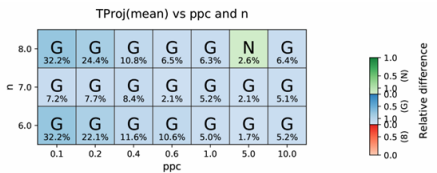


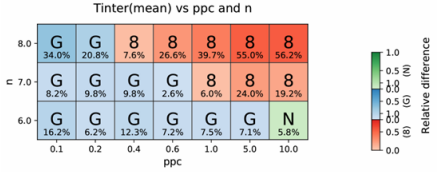


Décomposition de domaine sur CPU : intra-noeud

Comparaison N – patch G – patch 8: 4 nœuds, 4 tasks, 40 cpus/task







Décomposition de domaine sur CPU : intra-noeud

Conclusions

- Performant pour les paramètres Sparse-PIC et PIC.
- Plus adapté au Sparse qu'un tri classique.
- Patch "G" : gros patchs pour des petits paramètres SparsePIC
- Patch "8" : les petits patchs fonctionnent mieux à grands paramètres n et ppc. Optimum pour une taille de 512 ou 1024.
- La limite entre les deux fluctue en fonction des ressources disponibles.
- Bonus : augmenter le nombre de tasks au sein d'un noeud sous-performe sur Ruche, à éviter !
- Bonus : scaling efficiency (nombre_processeurs_ref x speedup / nombre_processeurs) supérieure à 90% pour n=8, et ppc=0.6 (pour 2 noeuds), ppc=1 (3 noeuds), ppc=5 (4 noeuds).

Kokkos : la librairie portable pour GPU

Porter avec un outil portable

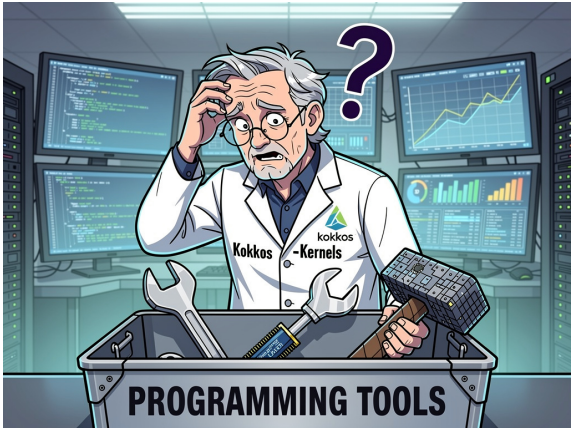
On profite de la proximité de l'équipe CExA pour utiliser Kokkos et GPUifier le code progressivement.

Générée par Emmy Strale AI.

Kokkos-kernels : la librairie math de Kokkos ? ? ?

Kokkos-kernels est déjà agrégé par Trilinos

La librairie dispose de trop peu d'outils "déjà prêts".



Générée par Emmy Strale AI.

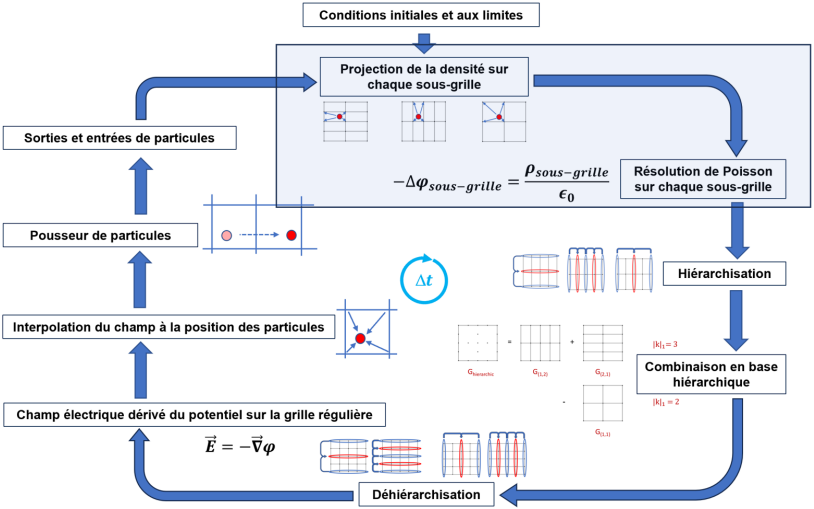
Trilinos : Trouver un équivalent à Pardiso : y'a qui ? Tacho

Générée par Emmy Strale AI, prompt soumis à droit d'auteur.

TRILINOS

- Trilinos, une bibliothèque logicielle open-source basée sur Kokkos
- Read the ~~docs~~, the code !
- Tacho : un solver direct pour GPU, testé mais pas encore bien parallélisé !

État du portage du code sur GPU



Partie du code passé sous GPU avec Kokkos

Conclusions

- Pas de ~~répit~~ tri pour le Sparse-PIC
- Une décomposition de domaine intéressante pour l'algo SparsePIC
- Un portage GPU en cours avec Kokkos + Tacho
- Bonus : plongée dans Kokkos ou Trilinos
- Bonus : plongée dans le code Fortran-C++-Kokkos
- Prochaine étape : encadrement d'un stagiaire pour le portage GPU

Résultats avec MiniPIC sur GPU

Objectif de MiniPIC

MiniPIC est une mini-application PIC standard destinée à tester différents modèles de programmation et d'optimisation pour les codes PIC sur CPU et GPU.

- Sur CPU : vectorisation, parallélisation par tâche, décomposition de domaine
- Sur GPU : Portabilité de la performance (Kokkos, Thrust, StdPar et SYCL) et optimisation de la performance
- Code source : https://github.com/Maison-de-la-Simulation/miniPIC
- Anticiper la réécriture de `Smilei` avec Kokkos

Développement de mini-apps spécifiques

- travailler sur des morceaux de code spécifiques avec plus de flexibilité
- permettre une exploration plus rapide de différentes approches d'optimisation
- préparer la rédaction du livre avec des exemples de code plus simples et plus accessibles que le code de production

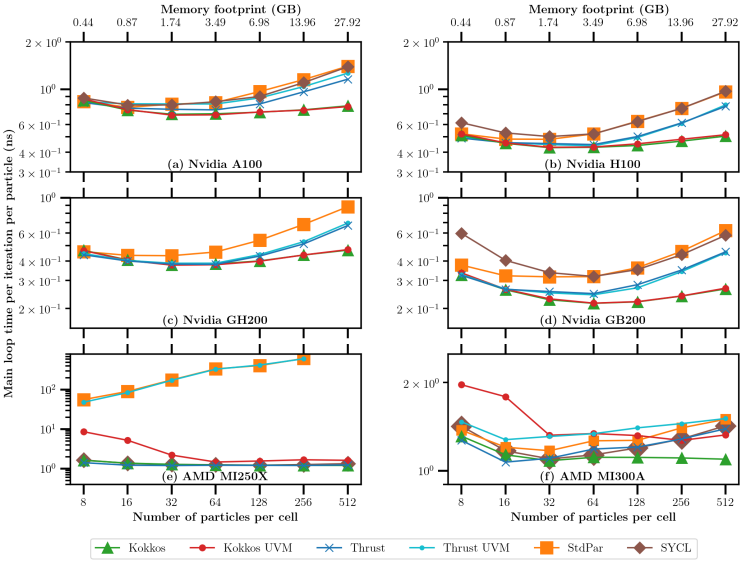
Comparaison des modèles de programmation sur GPU

Objectif

- Comparer les performances de différents modèles de programmation (Kokkos, Thrust, StdPar, SYCL) pour une implémentation PIC naïve sur différents modèles de GPU

À retenir

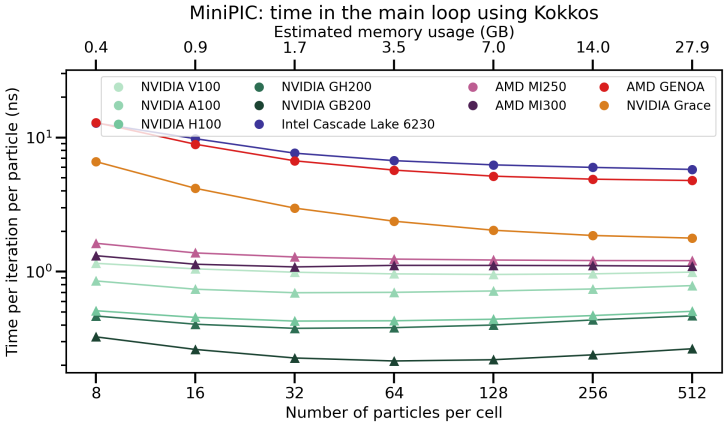
- Kokkos donne de manière générale de meilleures performances que les autres modèles de programmation sans nécessiter d'optimisation spécifique



Comparaison des GPUs

À retenir

- Les GPU NVIDIA offrent généralement de meilleures performances que les autres modèles de GPU pour l'implémentation PIC naïve testée
- Même avec une implémentation naive, le gain de performance par rapport au CPU est suffisant pour justifier le portage GPU



Description de l'étude sur le tri des particules sur GPU

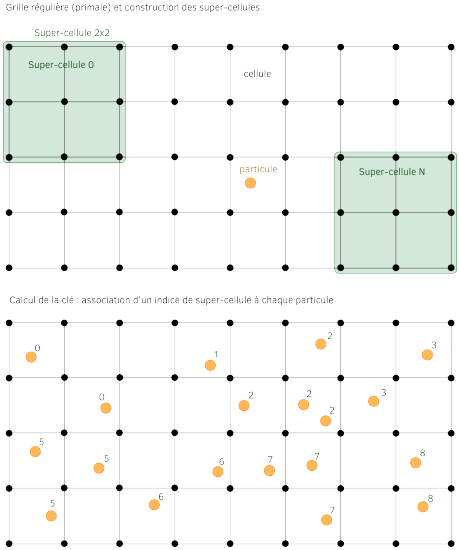
Objectif de cette étude

- Étudier l'intérêt du tri de particules sur GPU associé à des algorithmes de projection et d'interpolation adaptés
- Comparer les performances de différentes approches de tri sur GPU
- Tester la performance et la portabilité du tri avec Kokkos (comparaison avec Thrust utilisé dans `Smilei`)
- Création d'une mini-app **mini-sort** dédiée

Principe du tri de particules sur GPU

Approche de type bucket sort

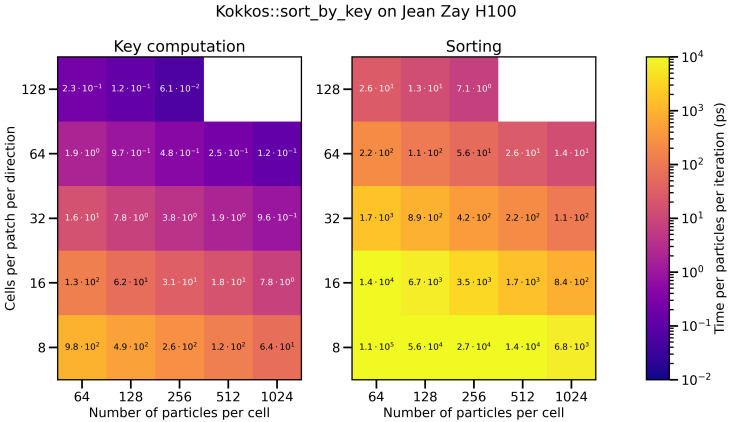
- Définition de la taille des super-cellules (ex : 2x2x2 cellules)
- Calcul de l'indice de super-cellule pour chaque particule en fonction de sa position
- Ordonnancement des particules (chaque tableau si grandeurs dissociées) en fonction de leur indice de super-cellule par permutation (ex : avec `sort_by_key` de Kokkos)
- Coût élevé potentiel même lorsque le niveau de désordre est faible / duplication temporaire de la mémoire



Tri des particules sur GPU

À retenir

- Les fonctions de tri implémentées dans Kokkos, comme `sort_by_key`, donnent des performances comparables à celles de Thrust sur NVIDIA H100
- Rentable pour des tailles de grilles importantes et un nombre de particules par cellule élevé, ce qui correspond à des paramètres plus proches du PIC standard que du Sparse-PIC



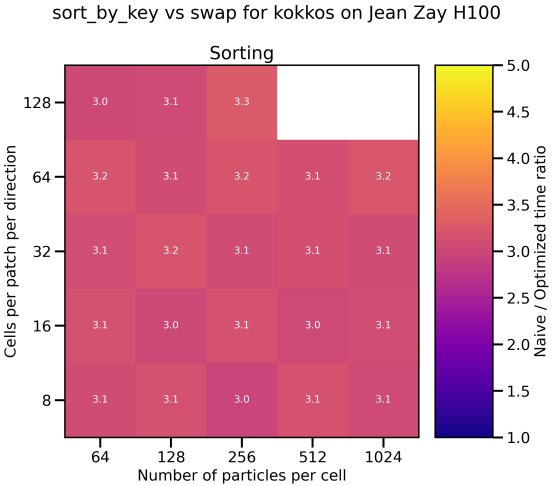
Tri des particules sur GPU

À retenir

- Stratégie d'optimisation du tri déjà intéressante en réexploitant les idées de `Smilei` avec `Thrust`
- Recherche d'optimisation du tri à poursuivre pour améliorer les performances
- Résultats sur architecture AMD à faire pour confirmer les tendances observées sur NVIDIA

Attention

Résultats préliminaires à prendre encore avec des pincettes



Projection (déposition du courant) optimisée sur GPU

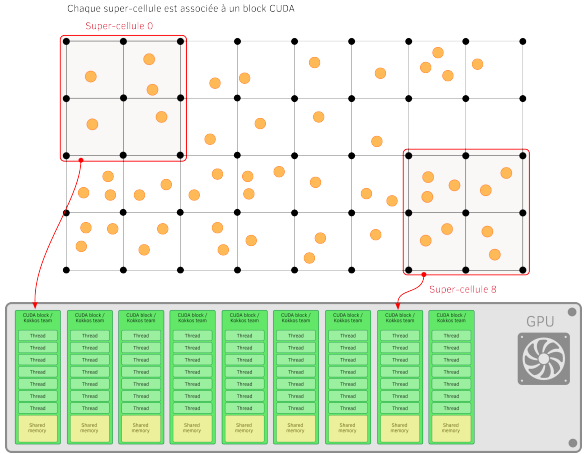
Objectif de cette étude

- Étudier des algorithmes de projection optimisés, notamment grâce à un tri de particules, sur GPU
- Comparer les performances sur divers modèles de GPU
- Tester la performance et la portabilité avec Kokkos
- Expériences pour la rédaction du livre (partie GPU)
- Création d'une mini-app **mini-proj** dédiée

Projection (déposition du courant) optimisée sur GPU

Description

- Chaque groupe de particules appartenant à la même super-cellule est traité par un bloc de threads dédié

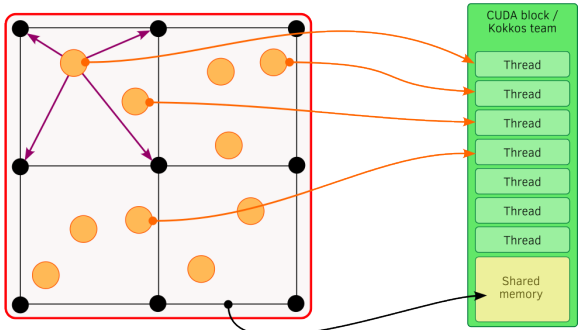


Projection (déposition du courant) optimisée sur GPU

Description

- La projection de chaque particule est effectuée par un thread dédié au sein du bloc associé à la super-cellule correspondante
- L'atomicité est gérée au niveau du bloc, ce qui permet d'éviter les conflits d'accès à la mémoire globale
- Une copie locale de la grille est faite en mémoire partagée pour chaque bloc, ce qui permet de réduire les accès à la mémoire globale
- Une réduction est effectuée à la fin de chaque bloc pour combiner les contributions de chaque thread à la grille globale

Chaque particule est traité en parallèle par un thread au sein d'un bloc



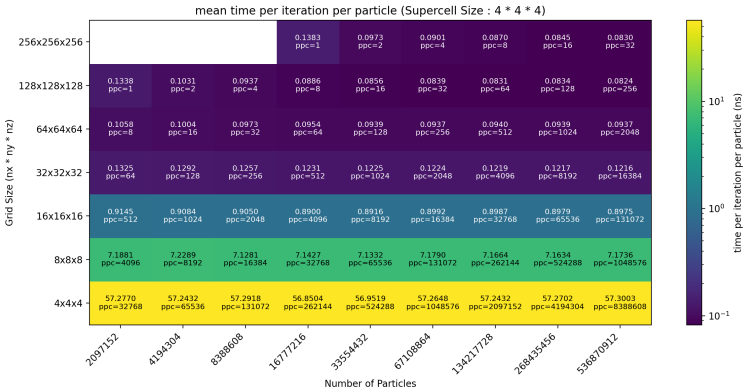
Projection (déposition du courant) optimisée sur GPU

Description

- Accélération jusqu'à 10x pour de larges tailles de grilles et un nombre de particules par cellule élevé
- Accélération suffisante pour compenser le coût du tri dans ce cas

Pas adapté quand :

- Gros déséquilibre de charge entre super-cellules
- Peu de particules par cellule



Interpolation (champs aux particules) optimisée sur GPU

Objectif de cette étude

- Étudier des algorithmes d'interpolation optimisés, notamment grâce à un tri de particules, sur GPU
- Comparer les performances sur divers modèles de GPU
- Tester la performance et la portabilité avec Kokkos
- Expériences pour la rédaction du livre (partie GPU)
- Création d'une mini-app **mini-interp** dédiée

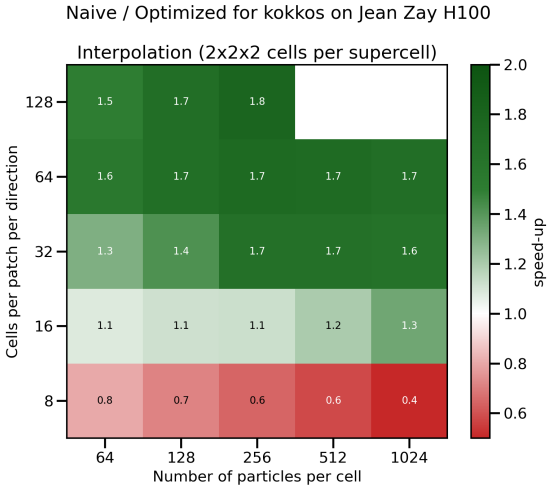
Méthode d'optimisation

- On applique le même principe que pour la projection optimisée

Interpolation (champs aux particules) optimisée sur GPU

À retenir

- Effet positif sur l'interpolation pour de grandes tailles de grilles et un nombre de particules par cellule élevé
- Inversion de tendance avec un ralentissement pour de petites tailles de grilles
- Intéressant pour le Sparse-PIC pour un n élevé
- Possibilité de jouer sur la taille des super-cellules pour avoir suffisamment de particules par super-cellule dans le cas du Sparse-PIC
- Résultats similaires sur diverses architectures AMD et NVIDIA



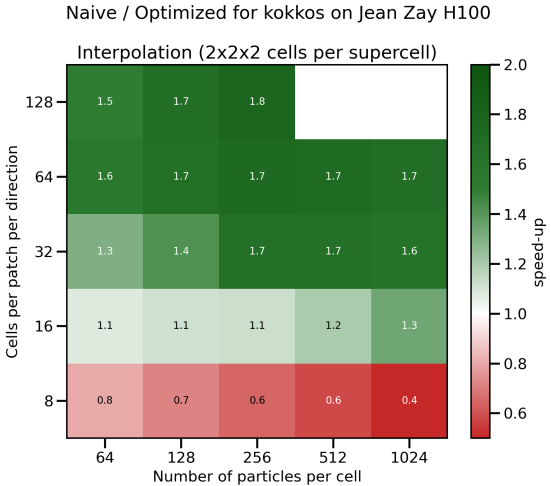
Interpolation (champs aux particules) optimisée sur GPU

Peu adapté quand :

- Gros déséquilibre de charge entre super-cellules
- Peu de particules par cellule

Amélioration possible

- Méthode adaptative comme pour la vectorisation pour choisir le meilleur algorithme d'interpolation en fonction du nombre de particules par cellule



Retour d'expérience de miniPIC pour le Sparse-PIC

- Kokkos est une bonne option pour le portage GPU du code de production
- Une optimisation de la projection et de l'interpolation possible grâce à ce tri mais uniquement pour des paramètres plus proches du PIC standard que du Sparse-PIC
- A confirmer pour des paramètres plus proches du Sparse-PIC en jouant sur la taille des super-cellules tout en gardant des tailles de grilles suffisamment grandes et un nombre de particules par cellule suffisamment élevé
- La technique présentée par Clément reste à privilégier pour les paramètres du Sparse-PIC

Merci !