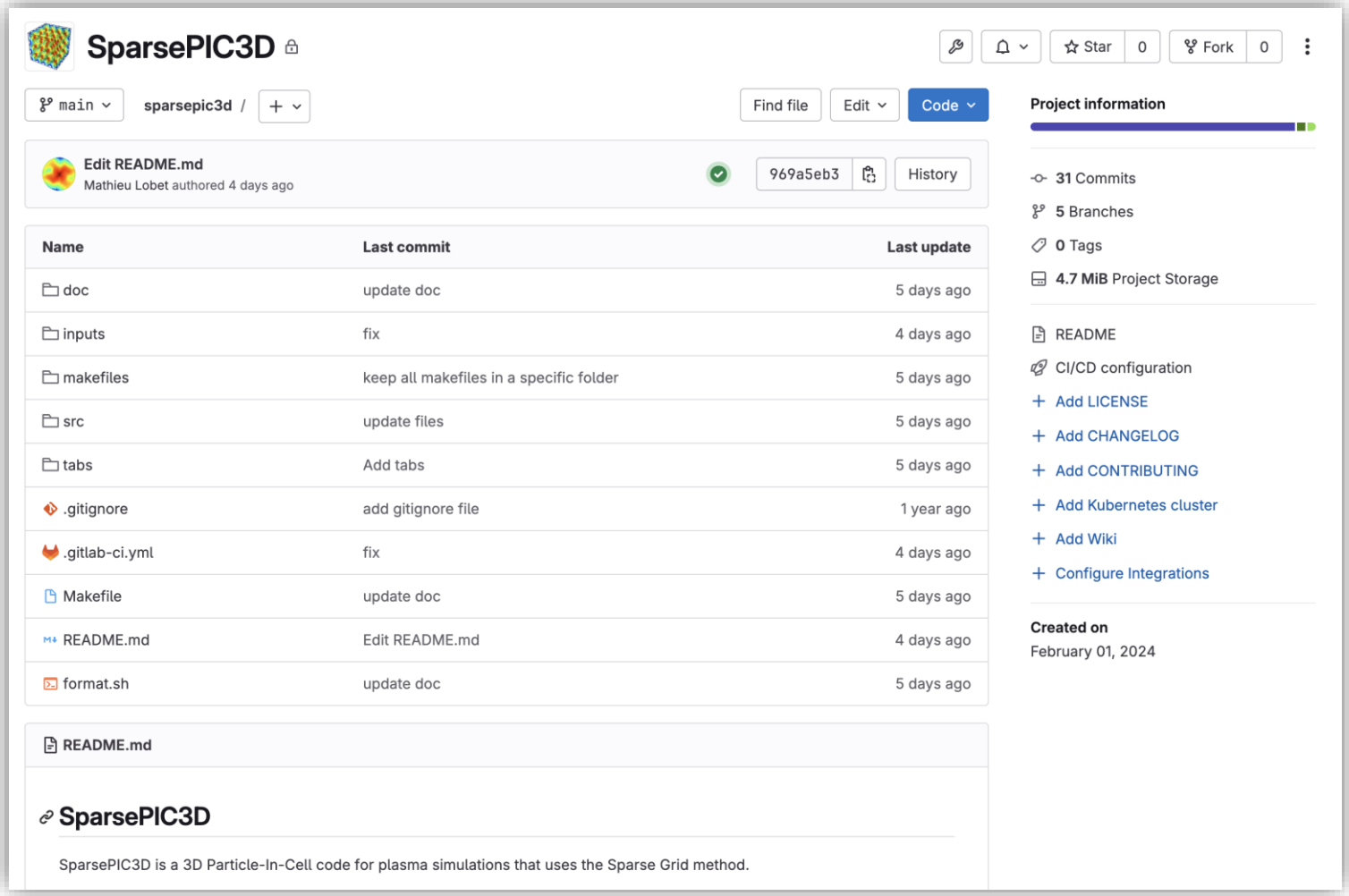
Projet Maturation

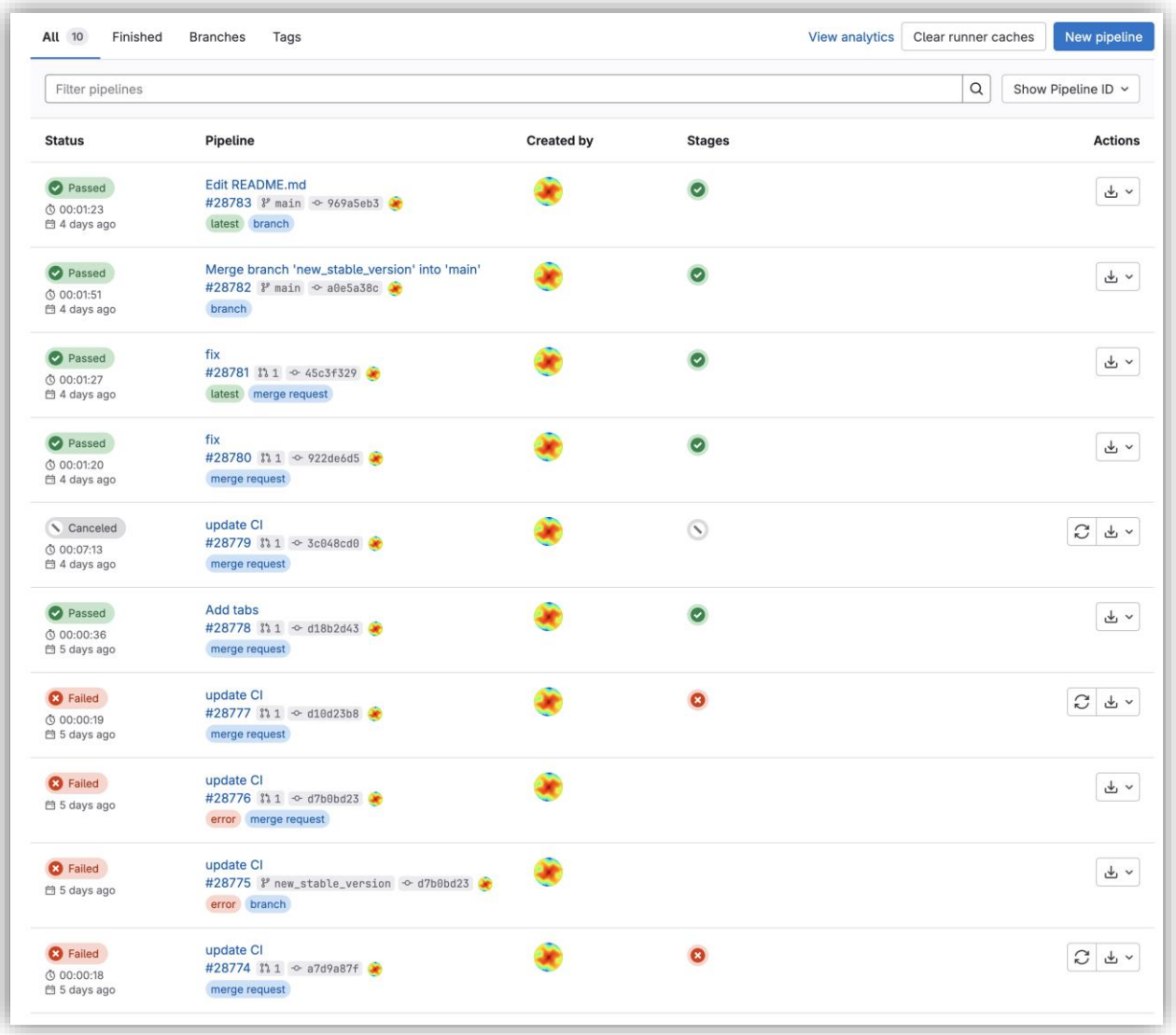
Optimisation du code

Hébergement du code sur le GitLab MdlS



- ✓ Création du repo GitLab (fermé)
  - Groupe : https://gitlab.maisondelasimulation.fr/anr-maturation
  - Code : https://gitlab.maisondelasimulation.fr/anr-maturation/sparsepic3d
  - Benchmarking : https://gitlab.maisondelasimulation.fr/anr-maturation/benchmarking
- ✓ Création d'une doc d'installation en markdown

GitLab CI



- ✓ Mise en place d'une première CI sur le Mesocentre Ruche (compilation et run mais pas de validation des résultats) sur un cas réduit issu de RF Mag

- □ Amélioration à prévoir :
  - Validation des résultats par comparaison avec des résultats de référence ou analytiques 🔥
  - Ajout de cas supplémentaires pour valider tous les aspects du code 🔥
  - Tests unitaires

Compilation

✓ Amélioration du Makefile pour s'adapter aux calculateurs et faciliter l'usage aux débutants

```
make FC=mpiifort MODE=release ARCH=cascadelake EXEC=sparsepic3d_csk
```

- -O3 : optimisation automatique
- -qopenmp : active les directives OpenMP

Vectorisation :
- -qopenmp-simd : active uniquement les pragmas simd (pas nécessaire avec –qopenmp)
- -ip : optimisations inter-procéduralesau sein d'un seul fichier source.
- -ipo : pareil que ip mais à l'échelle du projet (peut être lent)
- -qopt-zmm-usage=high : Cette option active l'utilisation intensive des registres ZMM pour les instructions AVX-512.

Architecture :
- -xCORE-AVX512 : Skylake et Cascadelake

⚠ Attention :
- -Ofast : induit -fp-model=fast
- -fp-model=fast par défaut dans ifc et icc à remplacer par -fp-model=precise ou strict

Première phase de vectorisation

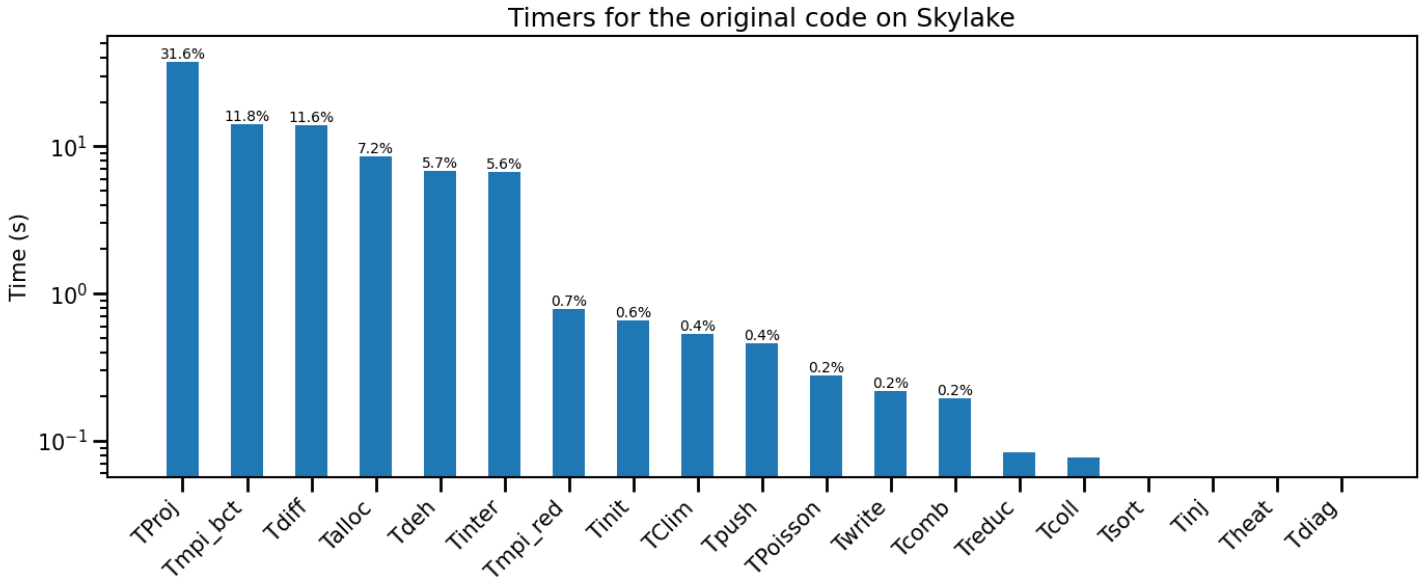
- Architectures :

Processeur	Calculateur	Compilateur	MPI
Intel Cascade Lake (2x20 coeurs)	Mésocentre Paris-Saclay (Ruche)	ifort 19.1.3.304 20200925	mpiifort
Intel Skylake (2x24 coeurs)	TGCC	ifx 2024.2.0 20240602	mpiifx
AMD Rome (2 x 64 coeurs)	TGCC	ifx 2024.2.0 20240602	mpiifx

- Paramètres :

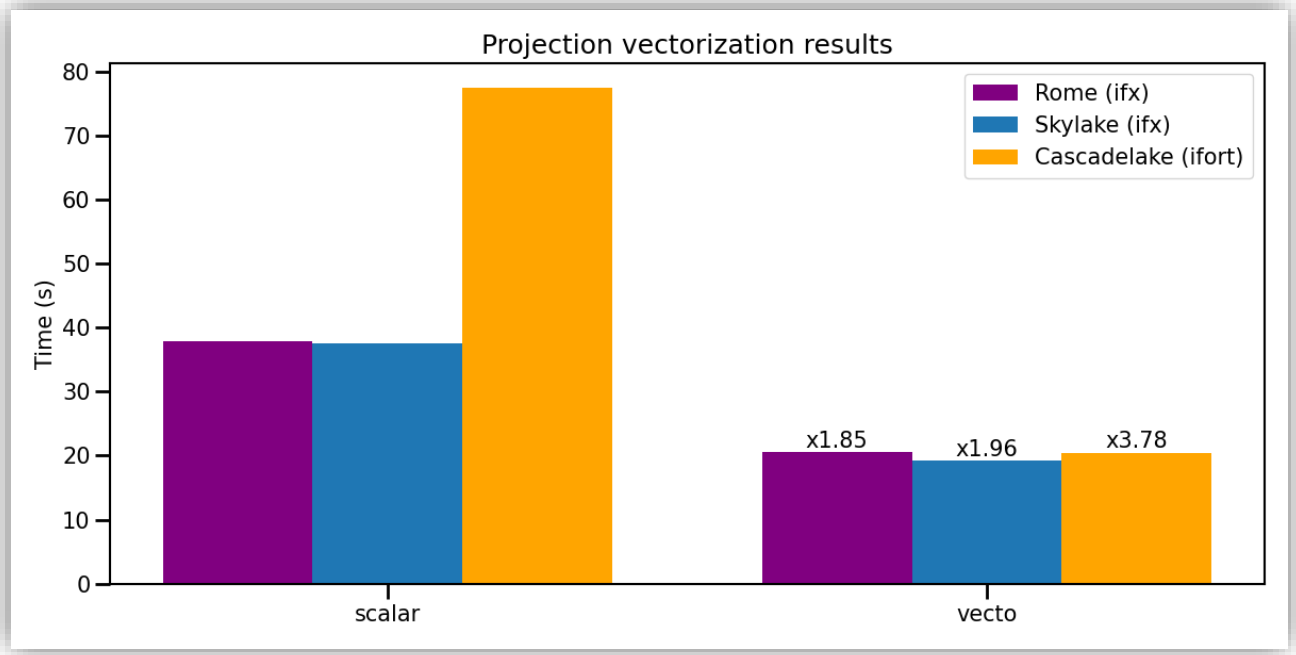
Processeur	Nombre de noeuds	MPI par noeud	Coeurs par MPI
Intel Cascade Lake	8	2	16
Intel Skylake	8	2	16
AMD Rome	2	4	16

- Intel Advisor + timers internes pour l'analyse du code



Vectorisation du projecteur

- Implémentation de la méthode testée dans PICSAR : VINCENTI, Henri, LOBET, Mathieu, LEHE, Rémi, *et al.* An efficient and portable SIMD algorithm for charge/current deposition in Particle-In-Cell codes. *Computer Physics Communications*, 2017, vol. 210, p. 145-154.
- Cette méthode ne marche bien que si le poids de chaque grille parcimonieuse multiplié par la taille des registres vectoriels rentre en cache L2

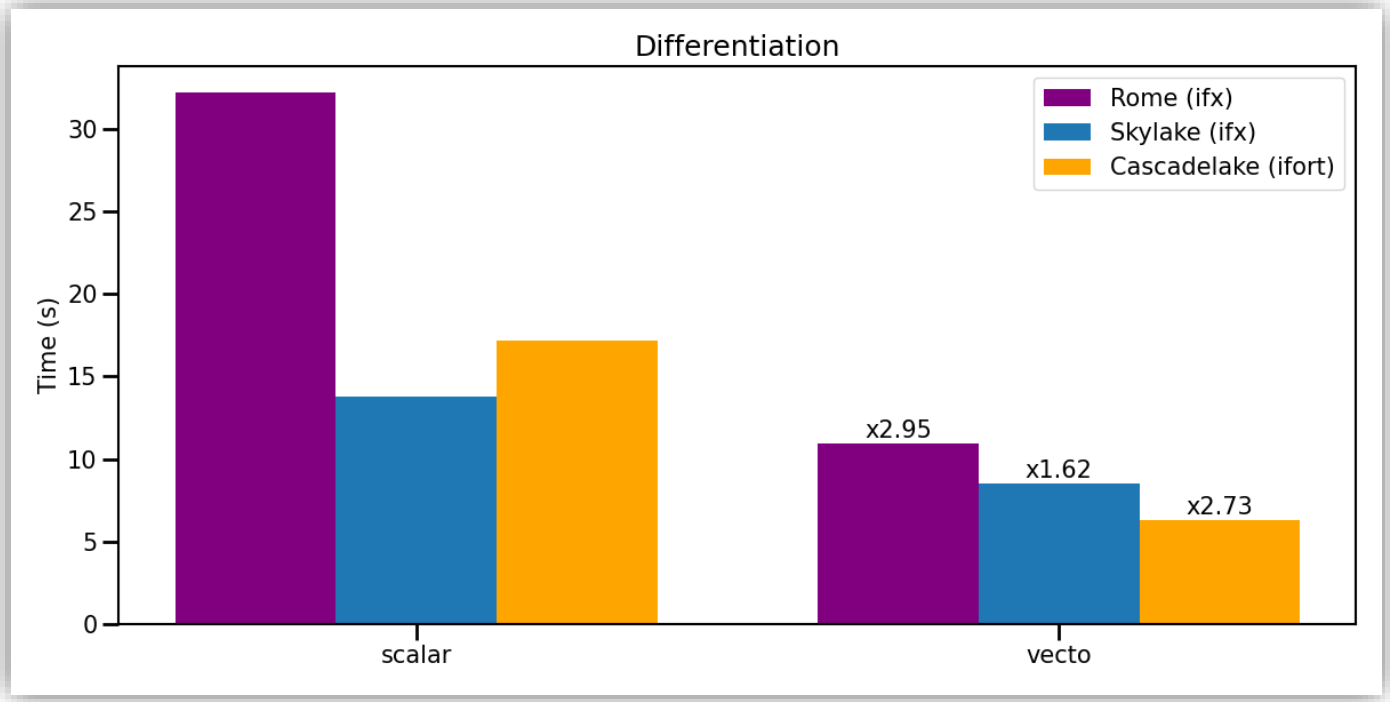


Différentiation

- Dans la version d'origine : partiellement parallèle via OpenMP mais pas vectorisée
- Changement du layout mémoire pour favorisé la vecto : E(1, ix, iy, iz) -> Ex(i,j,k)
- Une seule boucle parallèle sur la direction h

```
! $OMP DO
do h = 0 ,2** nstar
do j = 0 ,2** nstar
do i = 1 ,2** nstar -1
Egrid (1 , i ,j , h ) = -( phi ( i +1 , j , h ) - phi (i -1 , j , h ) ) * dxx
end do
end do
end do
! $OMP END DO
```

```
! $OMP DO
do h = 0 ,2** nstar
do j = 0 ,2** nstar
!$OMP SIMD
do i = 1 ,2** nstar -1
Exgrid (i ,j , h ) = -( phi ( i +1 , j , h ) - phi (i -1 , j , h ) ) * dxx
end do
end do
end do
! $OMP END DO
```

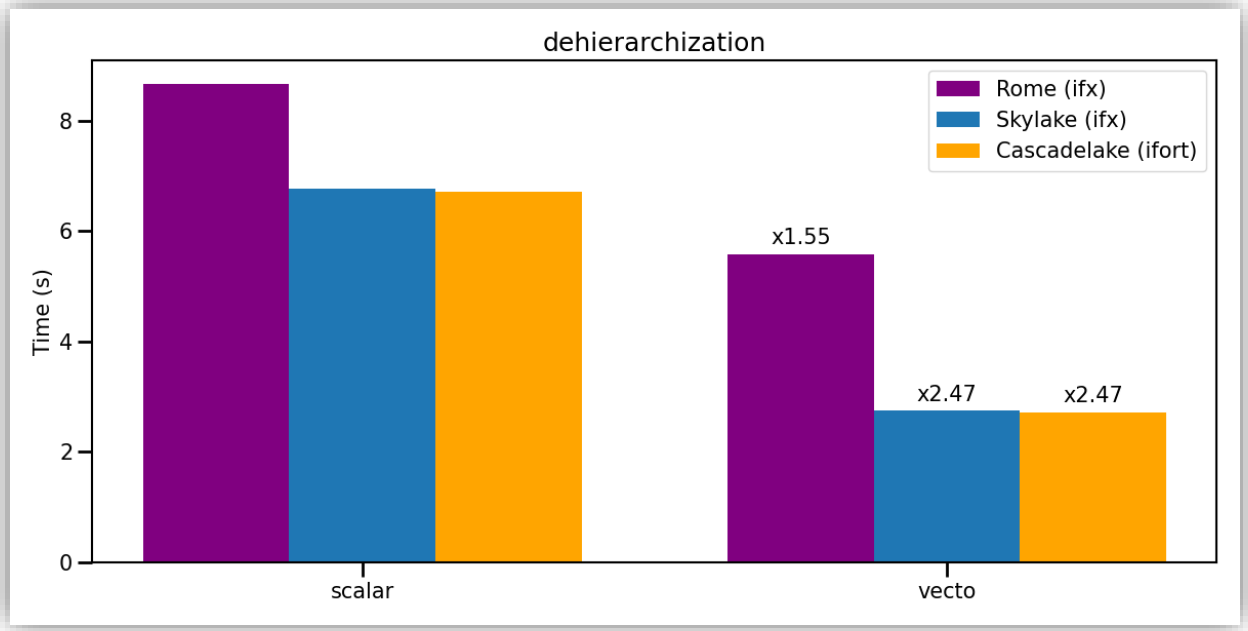


Dehierarchisation

- Changement de l'ordre des boucles pour avoir toujours la boucle sur i au niveau le plus profond et donc favoriser la vectorisation
- Regroupement des boucles parallèles

```
!$OMP DO
do j=0,2**l
do i=0,2**k
do p=m,1,-1
hz=2**(p-1)
do h=hz,2**m-hz,2*hz
grid(i,j,h)=grid(i,j,h)+(grid(i,j,h-hz)+grid(i,j,h+hz))/2
end do
…
```

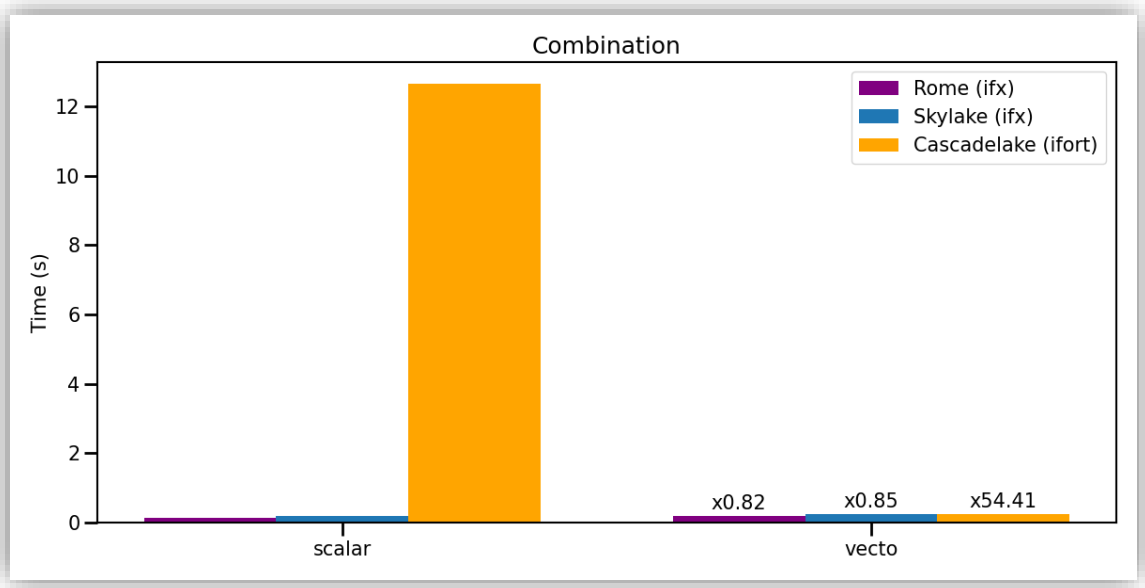
```
!$OMP DO
do j=0,2**l
do p=m,1,-1
hz=2**(p-1)
do h=hz,2**m-hz,2*hz
!$OMP SIMD
do i=0,2**k
grid(i,j,h)=grid(i,j,h)+(grid(i,j,h-hz)+grid(i,j,h+hz))/2
end do
…
```



Combinaison

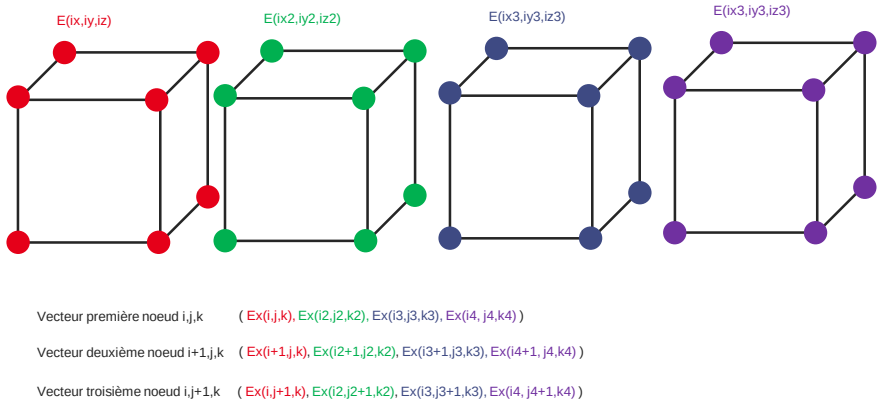
- Appel à de l'atomicité
- Phase très couteuse avec ifort / optimisée par le compilateur avec ifx
- Découplage avec la phase de hiérarchisation
- Changement du niveau de parallélisme pour supprimer l'atomicité

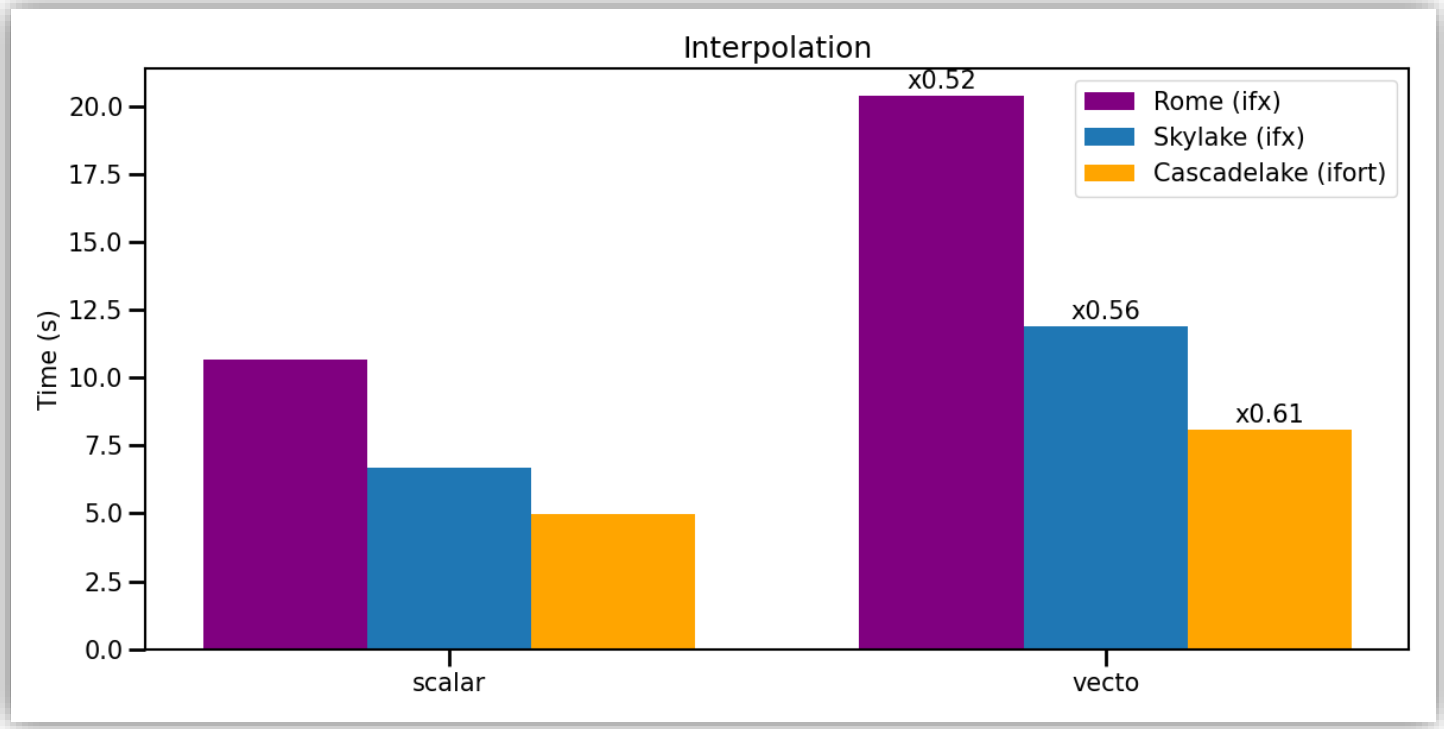
```
do isg = 1, Nsg
k = idx(isg, 1)
l = idx(isg, 2)
m = idx(isg, 3)
sig = idx(isg, 4)
ik = 2._RP**(n - k)
il = 2._RP**(n - l)
im = 2._RP**(n - m)
! call hierarchize(k,l,m,phiTab(isg)%grid)
!$OMP PARALLEL DO shared(phiTab,phiView) private(i,j,h)
do h = 0, 2**m
do j = 0, 2**l
do i = 0, 2**k
!!$OMP ATOMIC
phiView(i*ik, j*il, h*im) = phiView(i*ik, j*il, h*im) + sig*phiTab(isg)%grid(i, j, h)
! write(6,*) i,j,h,gridView(i,j,h)
end do
end do
end do
!$OMP END PARALLEL DO
end do
```



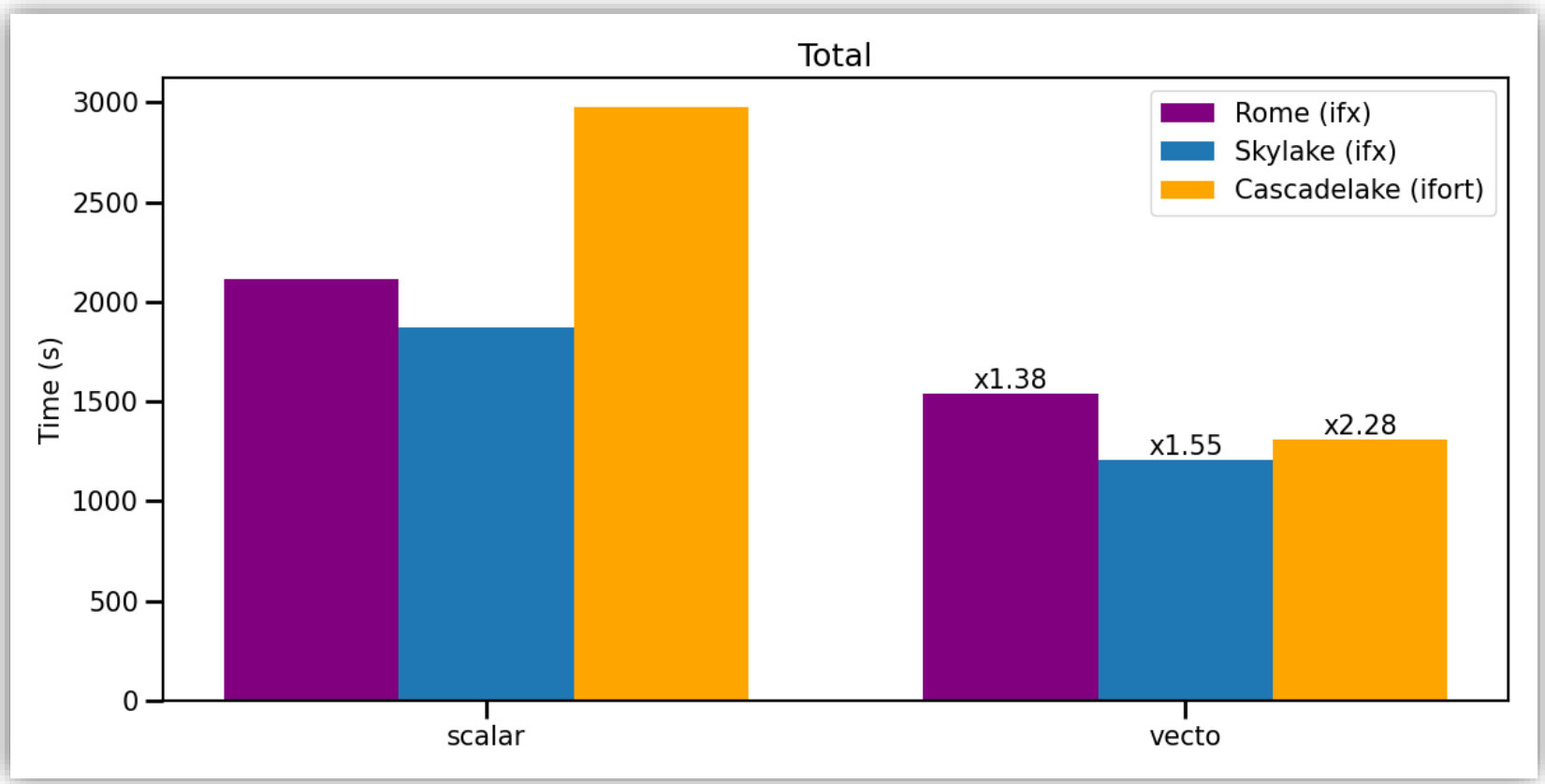
✕ Impact négatif sur l'interpolation

- Ralentissement inattendu d'un facteur 2
- Conséquence du changement de structure de données pour les champs électriques et magnétiques notamment le passage d'un tableau Egrid(3, nx, ny, nz) à trois tableaux Exgrid, Eygrid et Ezgrid
- Problème d'accès irrégulier aux tableaux Egrid pour la vectorisation (construction du vecteur de champs pour chaque particule
- En l'état la vectorisation de l'interpolation est possible mais pas efficiace

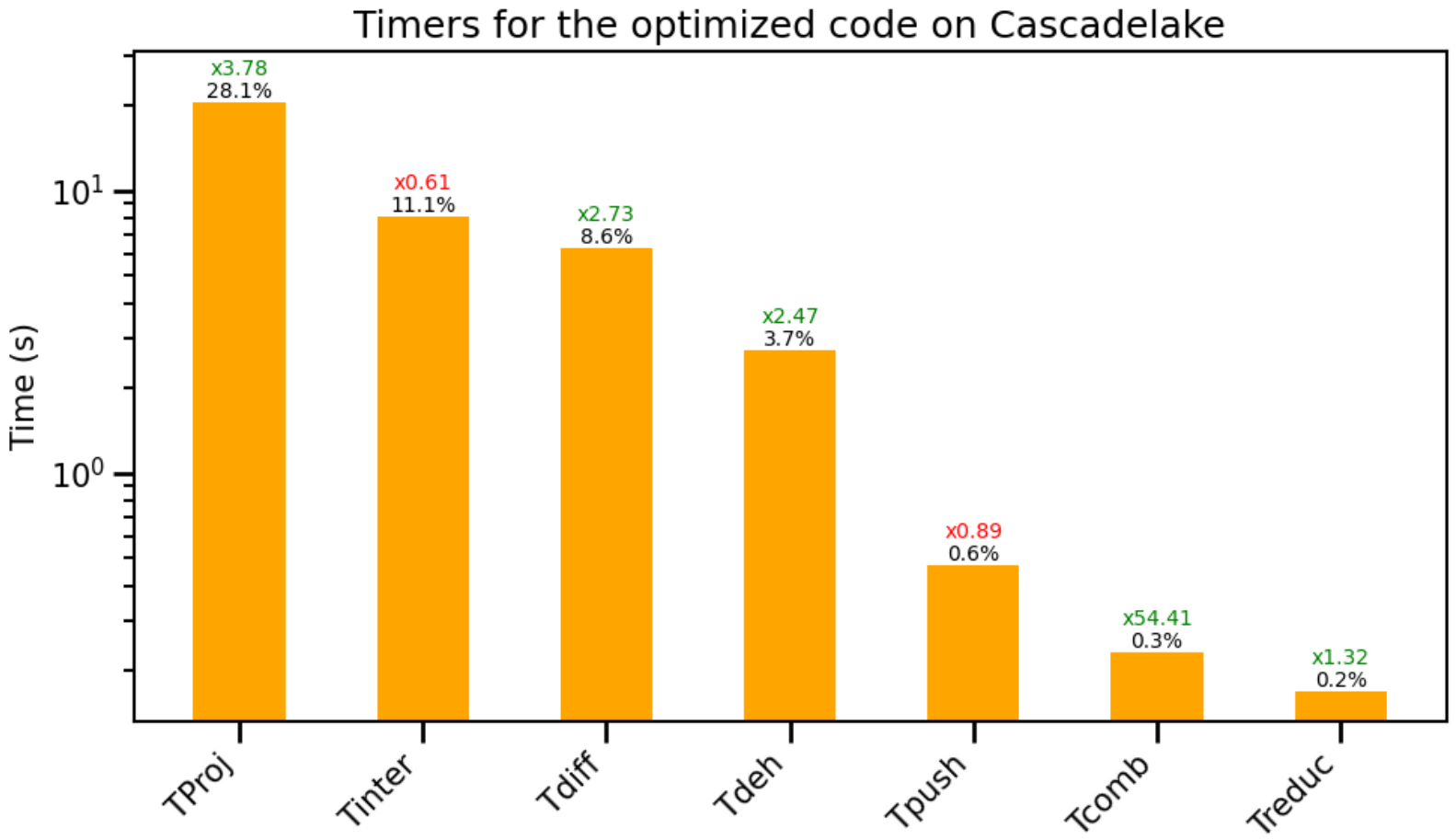




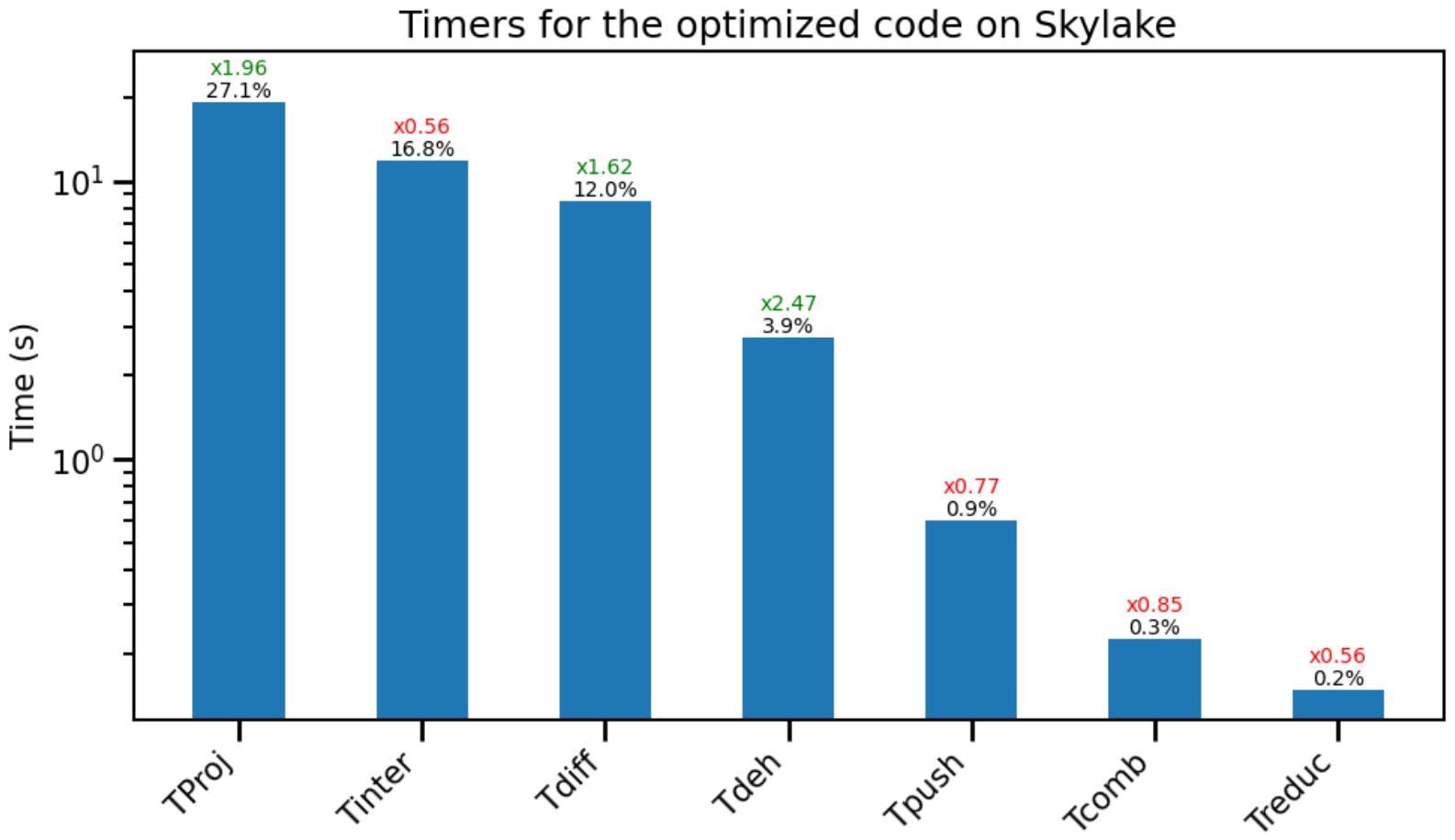
Performance globale



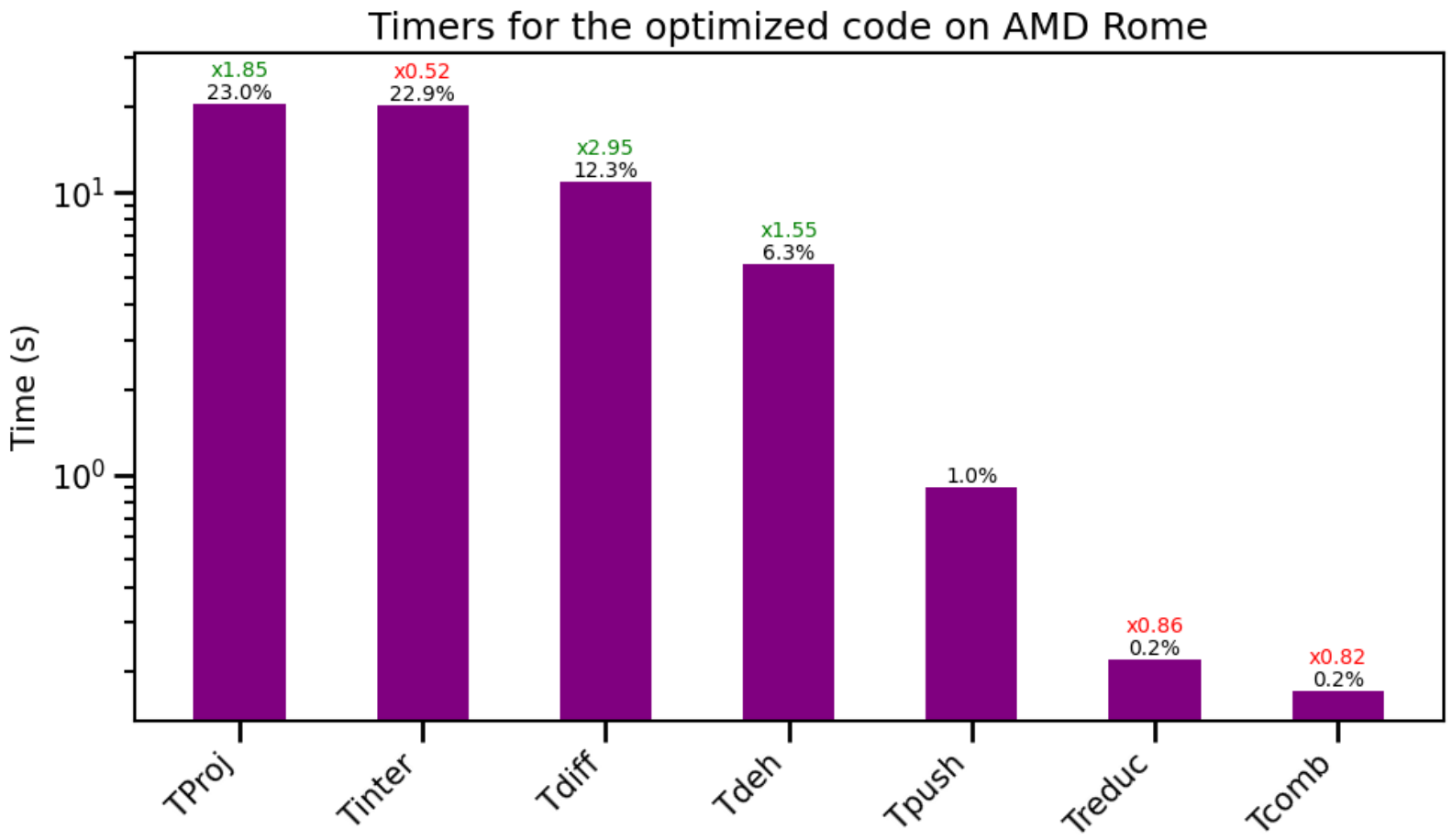
Performance globale (Cascadelake, ifort)



Performance globale (Skylake, ifx)

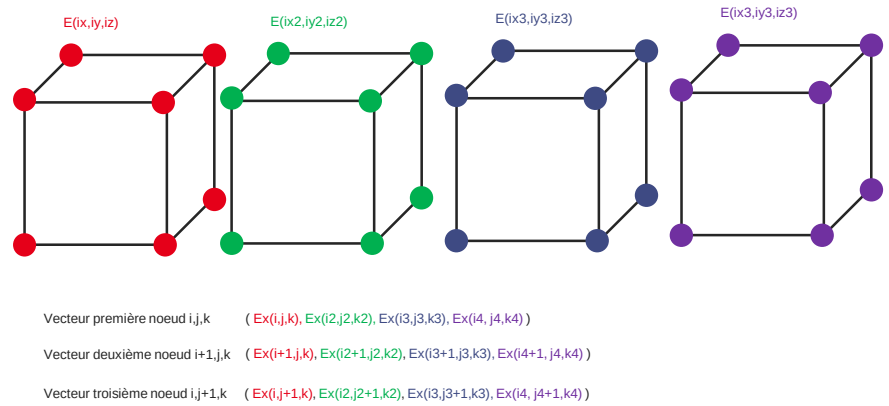


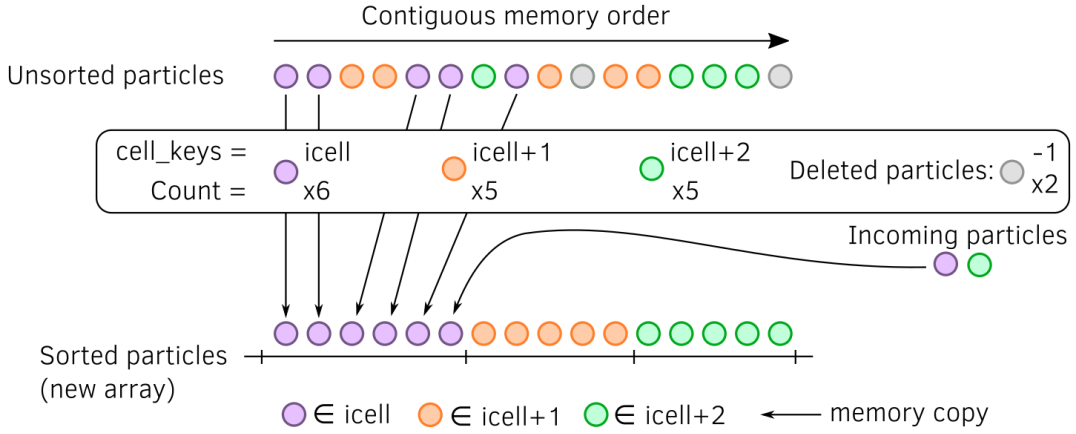
Performance globale (AMD Rome, ifx)

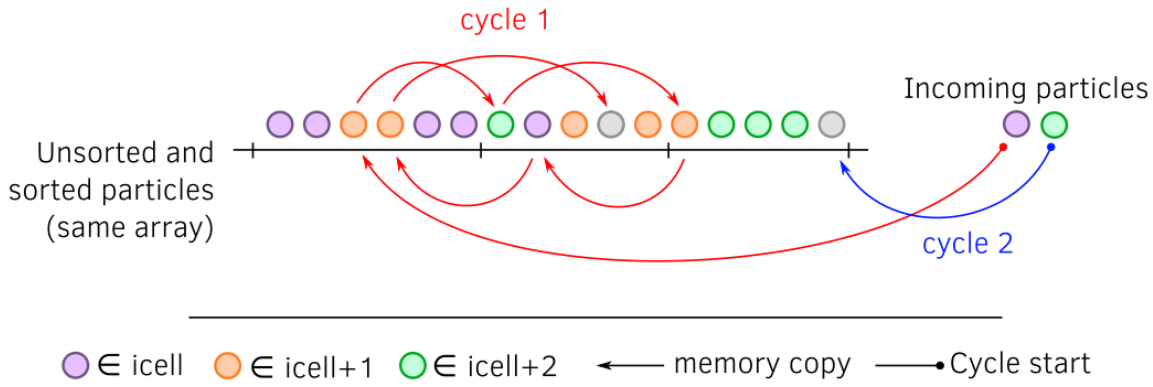


Aller plus loin : tri sur les particules

- Adopter une stratégie proche de SMILEI : tri optimisé sur les particules + algorithmes d'interpolation et de projection capable de tirer parti du tri
- Tri de type Cycle sort qui part du principe que les particules se mélangent peu entre chaque pas de temps (plus efficace que le bucket sort)





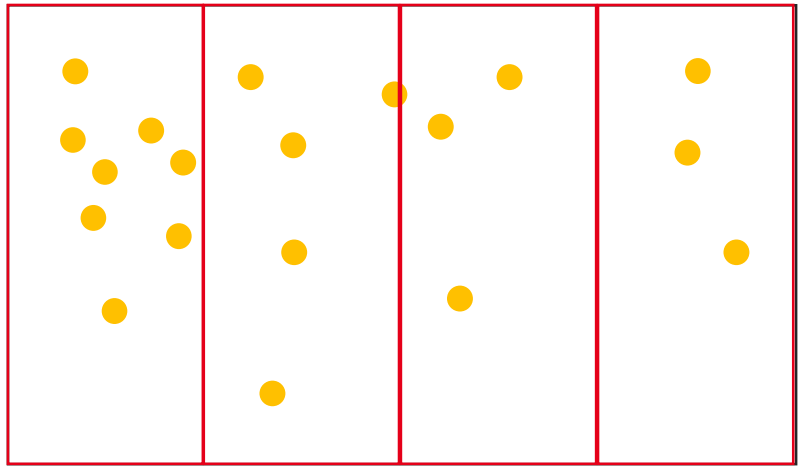


Aller plus loin : tri sur les particules

Question de la parallélisation du tri :

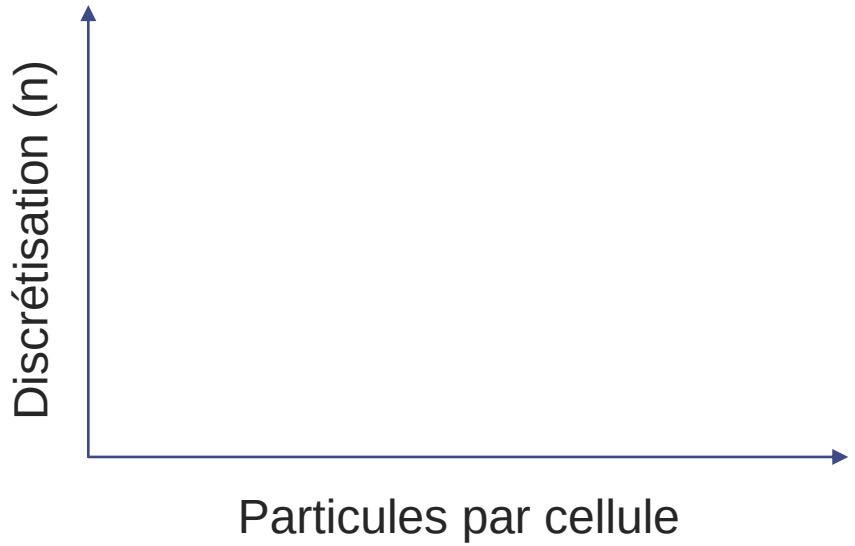
- Tri sur chaque paquet indépendant de particules
  - Simple à implémenter dans la version actuelle
  - Moins de particules par cellule
  - Buffer de la taille du domaine
  - Bien pour l'équilibrage de charge

- Création de paquet de particules géographiquement indépendant (similaire à la notion de tile dans PICSAR)
  - Plus de particules par cellule (bien pour la vectorisation et le GPU)
  - Des buffers plus petits car adaptés à la taille du paquet (bien pour le cache)
  - Un déséquilibrage de charge potentiel
  - Procédure d'échange de particules à ajouter
  - Meilleur passage à l'échelle au niveau du noeud (du projecteur notamment)



Aller plus loin : validation des optimisations

- Définir plusieurs cas tests physiques pour valider les optimisations 🔥
- Définir un jeu de paramètres critiques pour la performance (par exemple nombre de particules par cellules, discrétisation du domaine) 🔥
- Automatisation des tests en Python
- Tester les architectures ARM (A64FX, Grace, Graviton)
- Temps dans les diags en mode production ?

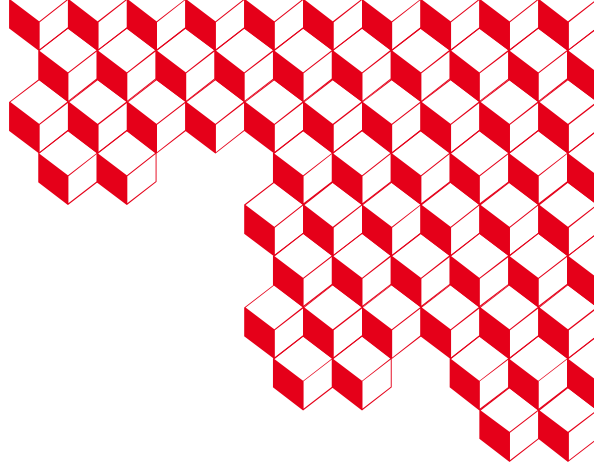


Aller plus loin : génie log and devOps

- Formatage automatique du code (fprettify par exemple) 🔥
- Eviter l'ouverture et la fermeture des régions parallèles
- Favoriser les structures plutôt que les gros tableaux monoblock pour les conteneurs de données
- Noms de variable plus explicites
- Doc utilisateur et développeur
- Cmake plutôt Makefile ?
- Compilation du code avec Gfortran : implémentation de solveur autre que Pardiso 🔥

Aller plus loin : GPU

- Version Kokkos pour la partie GPU ?



Merci

CEA SACLAY
91191 Gif-sur-Yvette Cedex
France
nathalie.guillaume@cea.fr
Standard. + 33 1 69 08 60 00