Projet HPCA: Simulation de propagation d'ondes

Amaury de Wargny, Tayyib Patel

UPMC

April 5, 2016

Plan

Simwave

Domaine 3D modélisé par une grille régulière de points
Propagation d'une onde en chaque point de la grille sur plusieurs pas de temps
Traitement spécifique aux bords du domaine (sauf un)
Utilisation de PML (Perfectly Matched Layers)
Formule de propagation en chaque point de la grille:

$$U_{i,j,k}^{n+1} = \begin{cases} \frac{2}{\Gamma_{i,j,k}+1} U_{i,j,k}^{n} - \frac{\Gamma_{i,j,k}-1}{\Gamma_{i,j,k}+1} U_{i,j,k}^{n-1} + \frac{c_{i,j,k}^2 \Delta t^2}{\Gamma_{i,j,k}+1} \Delta U_{i,j,k}^{n} + c_{i,j,k}^2 \Delta t^2 s^n, & (i,j,k) \in \text{zone PML} \\ 2U_{i,j,k}^{n} - U_{i,j,k}^{n-1} + c_{i,j,k}^2 \Delta t^2 \Delta U_{i,j,k}^{n} + c_{i,j,k}^2 \Delta t^2 s^n, & (i,j,k) \in \text{sinon,} \end{cases}$$

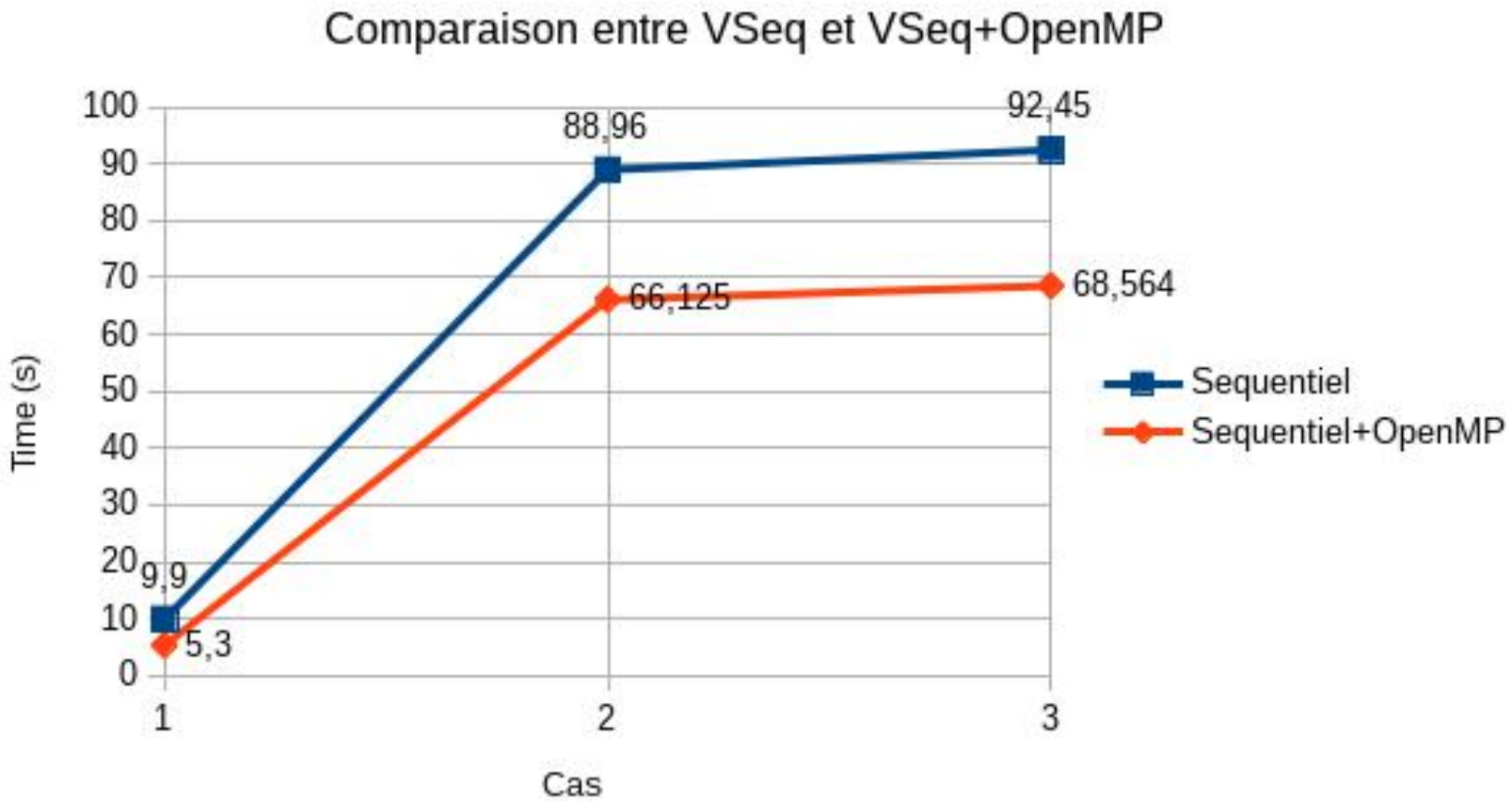
Objectif

Simulation HPC de cette propagation
Couramment utilisée en pratique (effets d'un tremblement de terre, RTM..)

Retouches du code séquentiel

Utilisation de directives OpenMP : parallèlisation de la boucle sur z dans pwave-update-fields

Résultats sur ces premières optimisations



Version 1

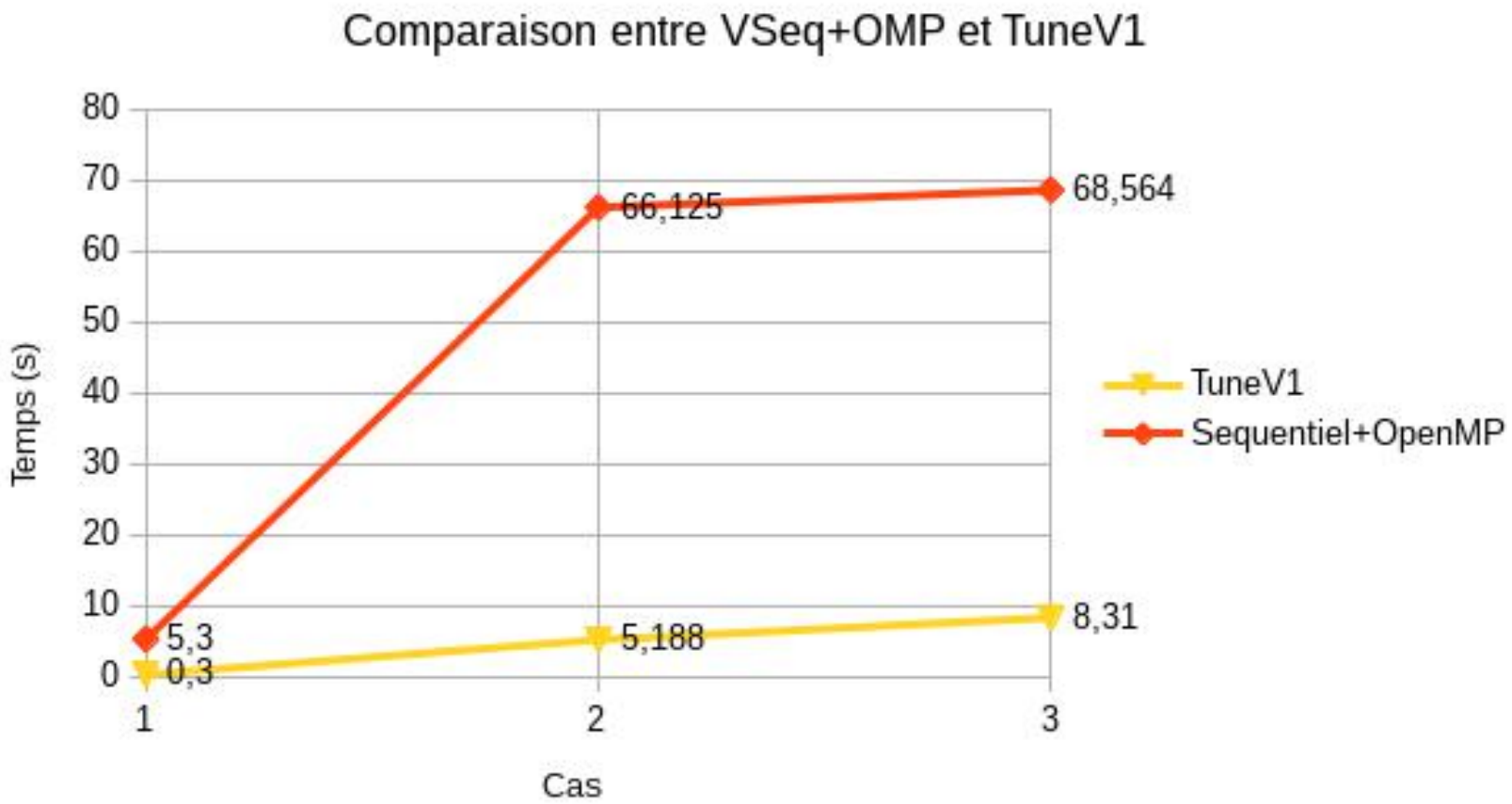
Calcul des données de la vague sur GPU, U0 renvoyé sur CPU
Taille de blocs par défaut: 16, 16, 1
Un kernel pour mettre à jour les données à chaque étape
Vérification des résultats avec l'option check de simwave

TuneV1

Détermination de la taille optimale des blocs de threads pour chaque cas

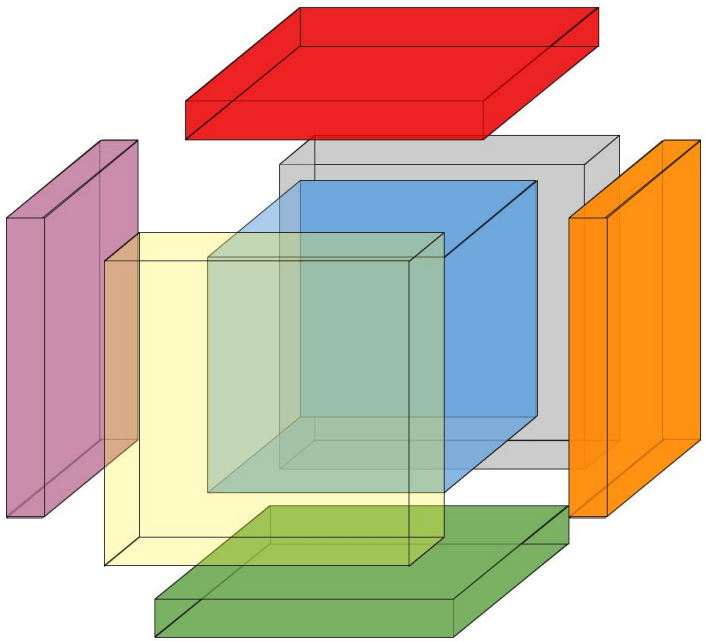
16, 4, 1

TuneV1
Résultats



Version 2

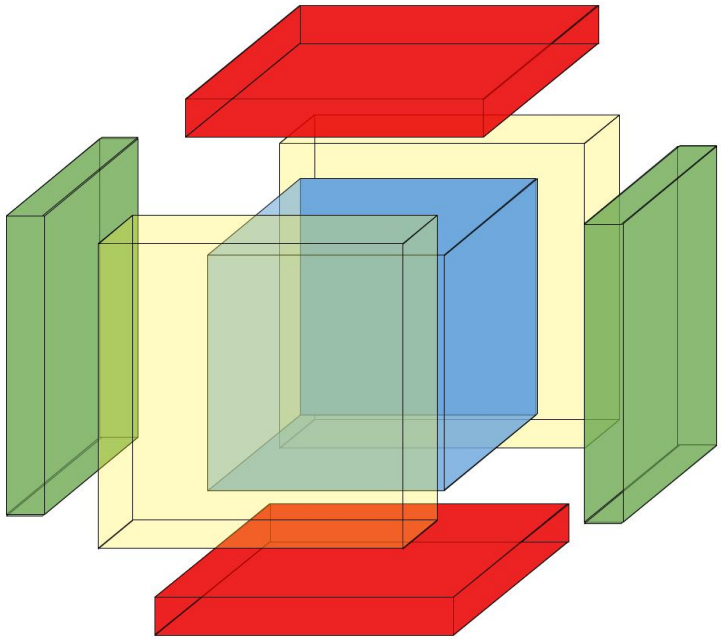
Utilisation de 7 kernels pour calculer les données à chaque étape (bas, haut, gauche, droite, avant, arrière, milieu)
Temps de calcul plus long que la version précédente, pour des tailles de blocs optimales



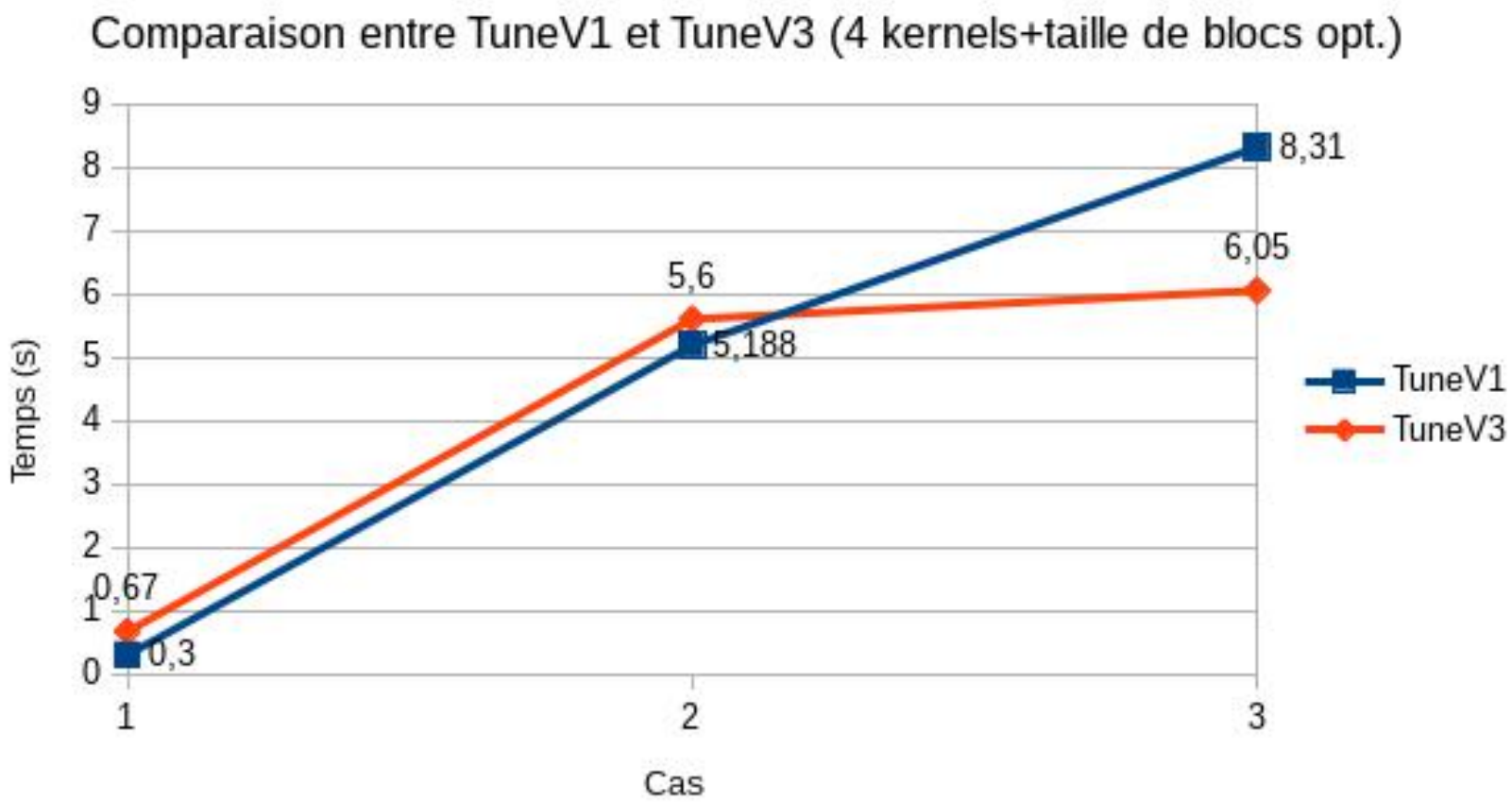
Utiliser trop de kernels fait perdre l'efficacité gagnée par la parallèlisation sur GPU

Version 3

Utilisation de 4 kernels pour calculer les données à chaque étape (bas/haut, gauche/droite, avant/arrière, milieu)



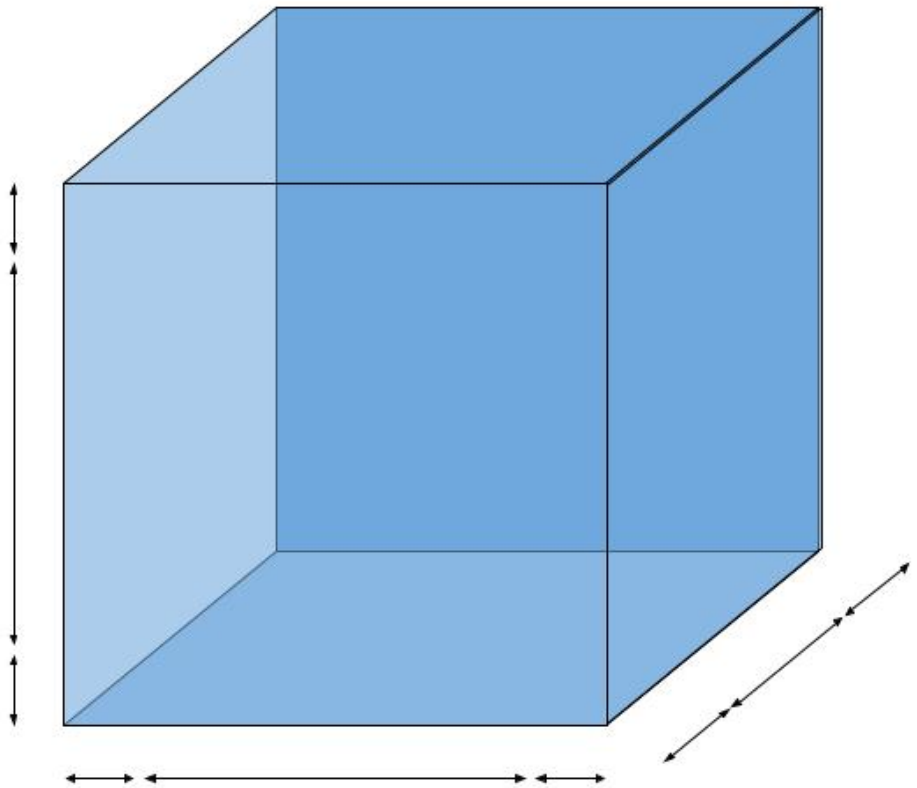
Version 3
Résultats

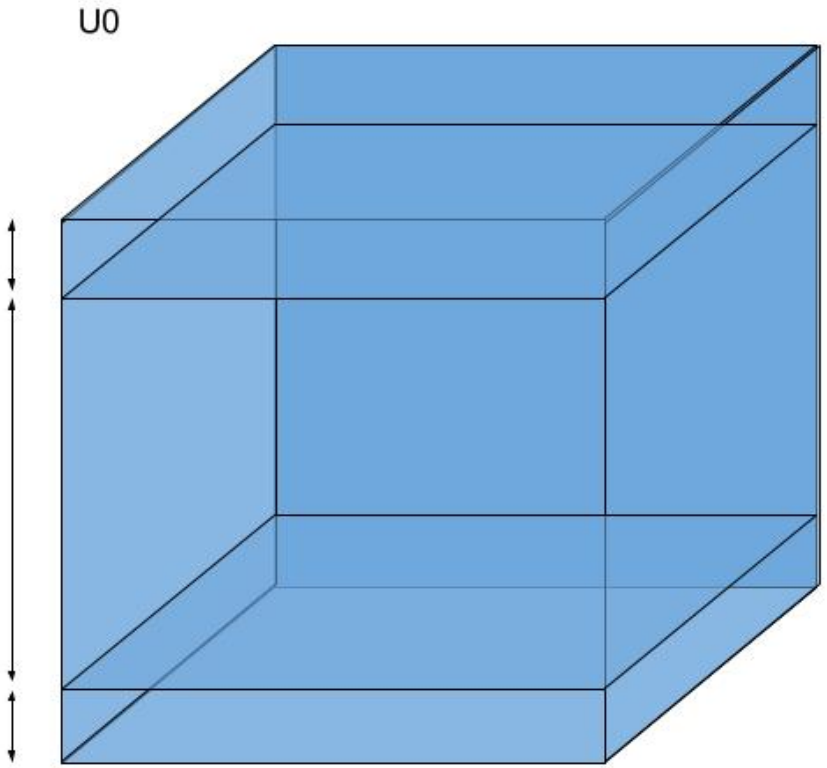


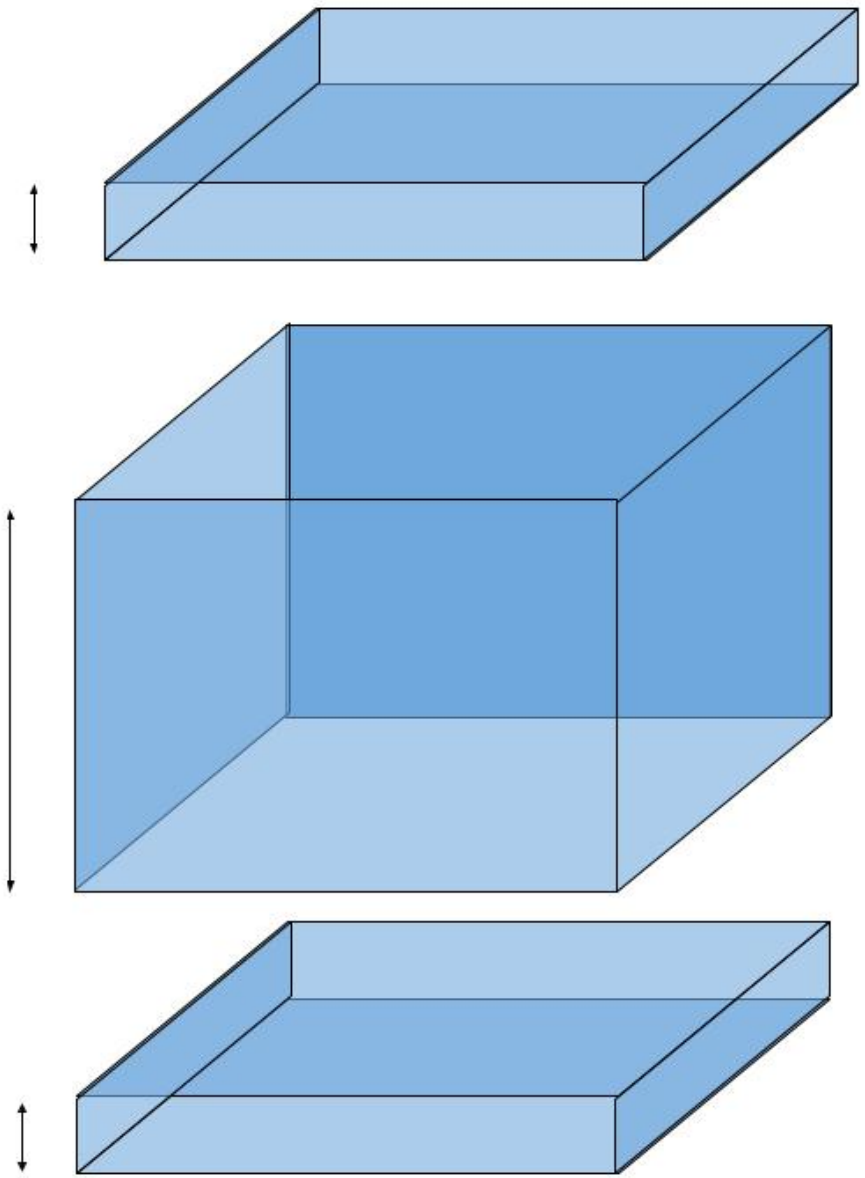
Version 4

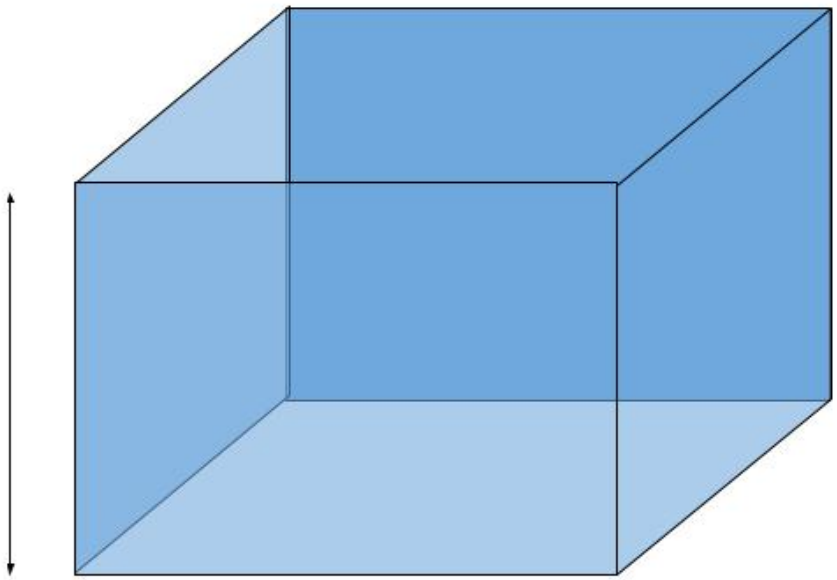
Amélioration de l'ensemble "calcul sur GPU + rapatriement des snapshots sur CPU" (en prenant les meilleurs blocs de thread)
Utilisation de 4 cudaStreams

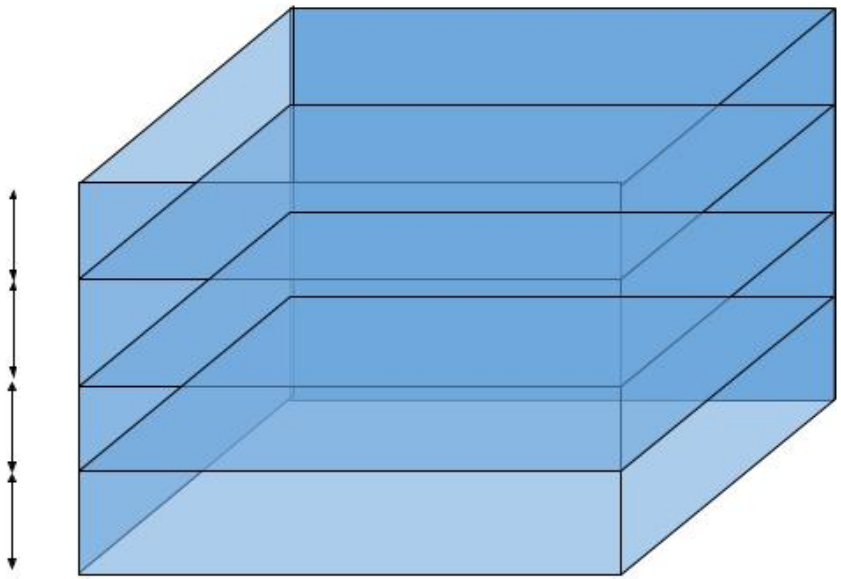
U0

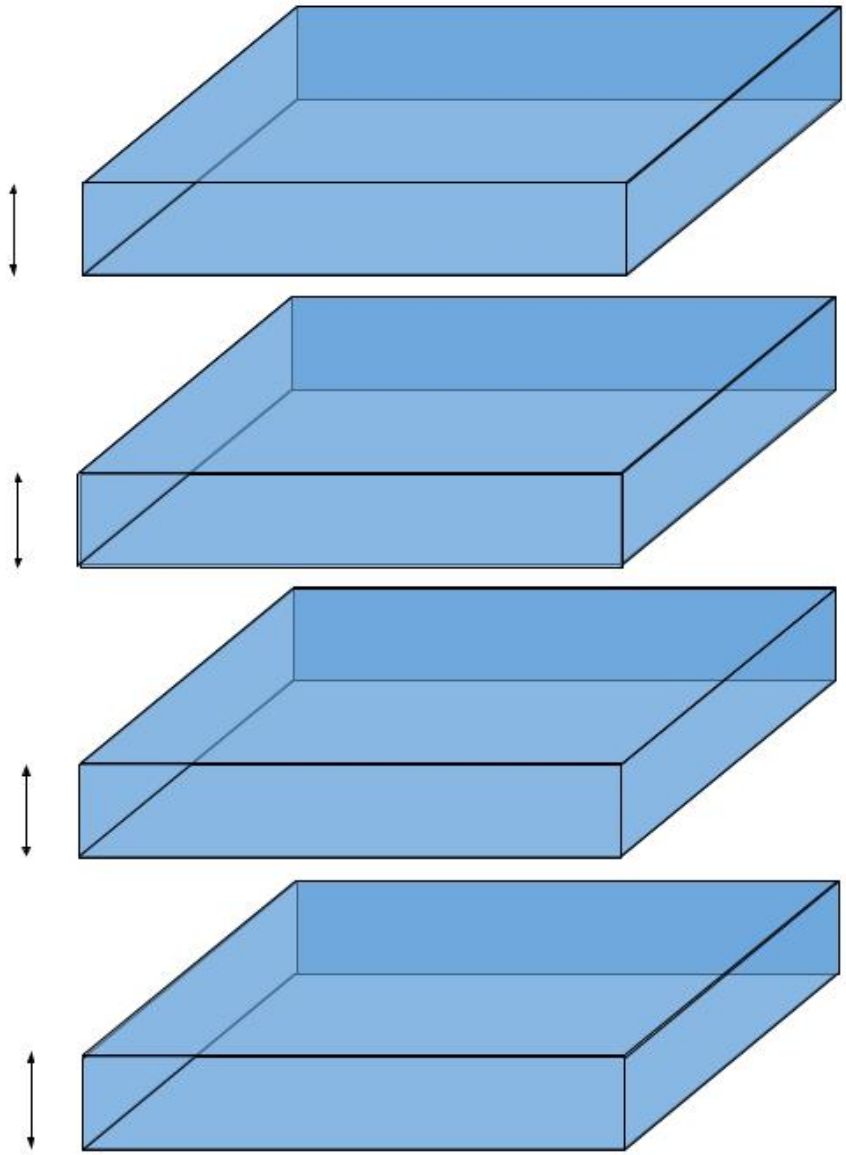



U0

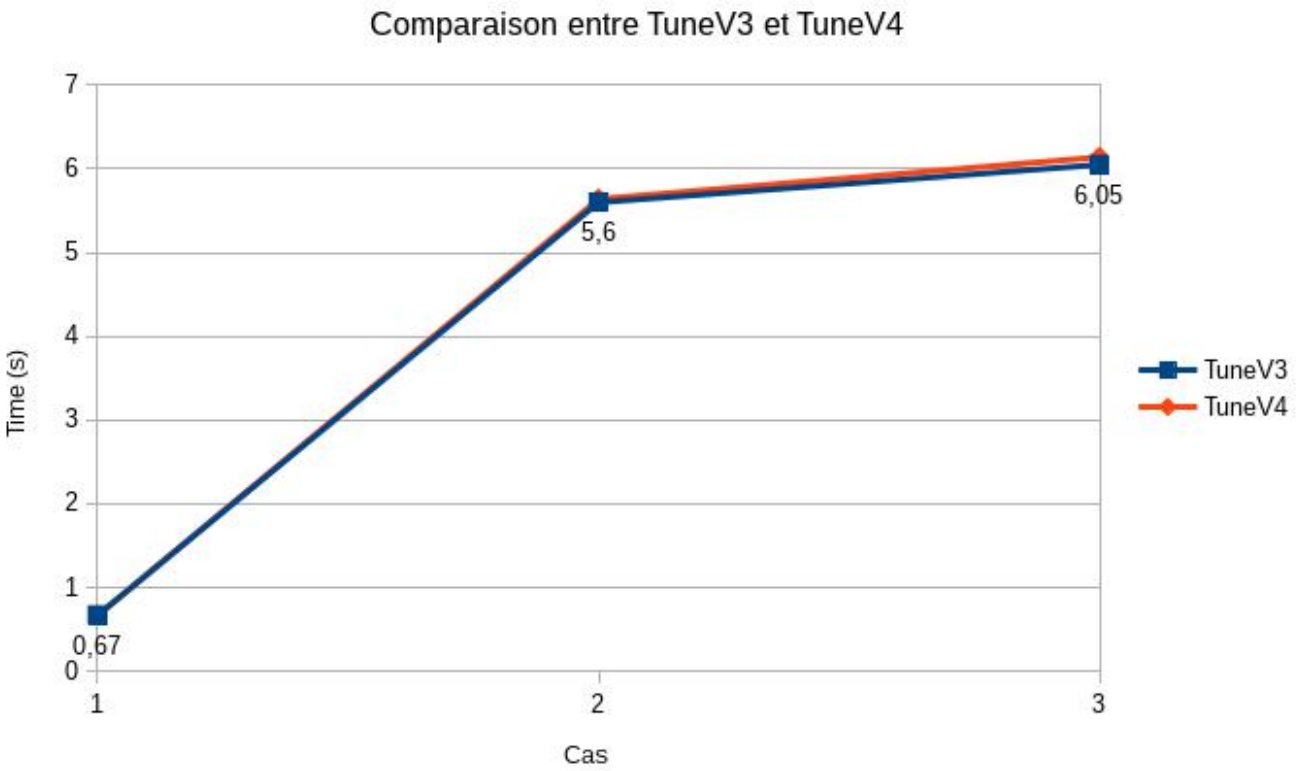






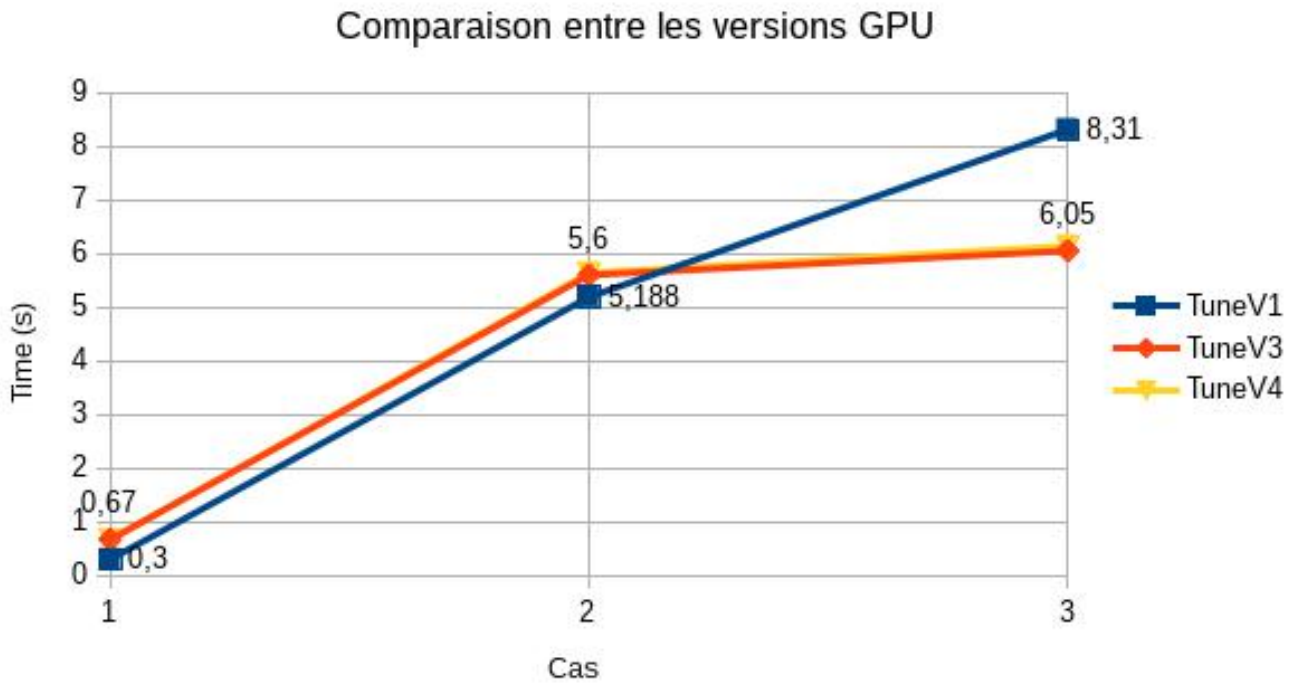


Version 4
Résultats

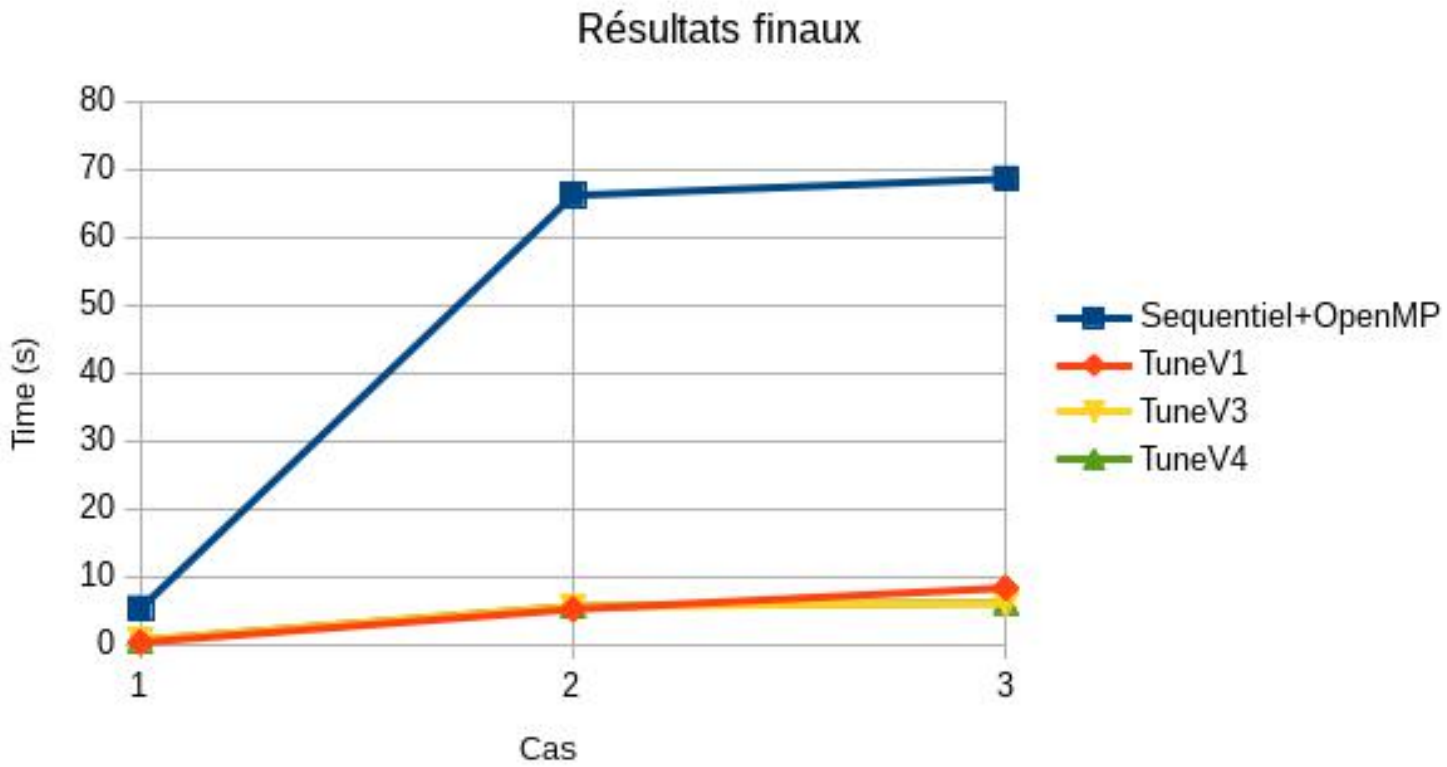


Pour nbsnap=1,5,15 (avec 100 et 200 itérations), les temps d'éxecution sur le cas 3 diffèrent de quelques centièmes entre la tuneV3 et la tuneV4

Résultats


Comparaison entre les versions GPU
Time (s)
Cas
0
1
2
3
4
5
6
7
8
9
1
2
3
0,67
0,3
5,6
5,188
8,31
6,05
TuneV1
TuneV3
TuneV4

Résultats



Accélérations (tuneV3/VSeq):
Cas 1 : 14.8 | Cas 2 : 15.9 | Cas 3 : 15.3

Conclusion

- Gains significatifs obtenus avec l'utilisation du GPU
- Améliorations possibles: Utilisation de plusieurs noeuds GPU, d'OpenCL sur le CPU,de MPI/CUDA,...