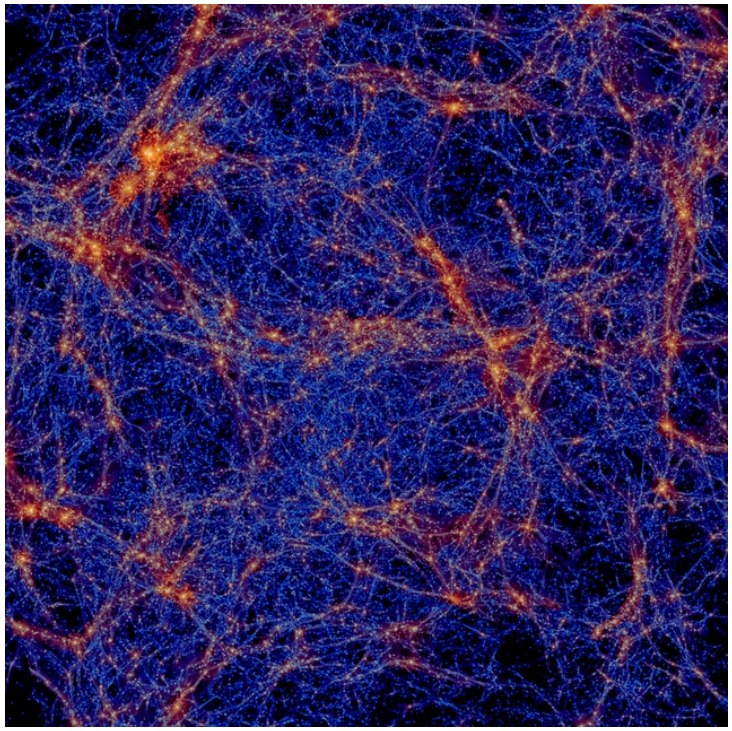
Rendu volumique complexe avec Python et OpenCL

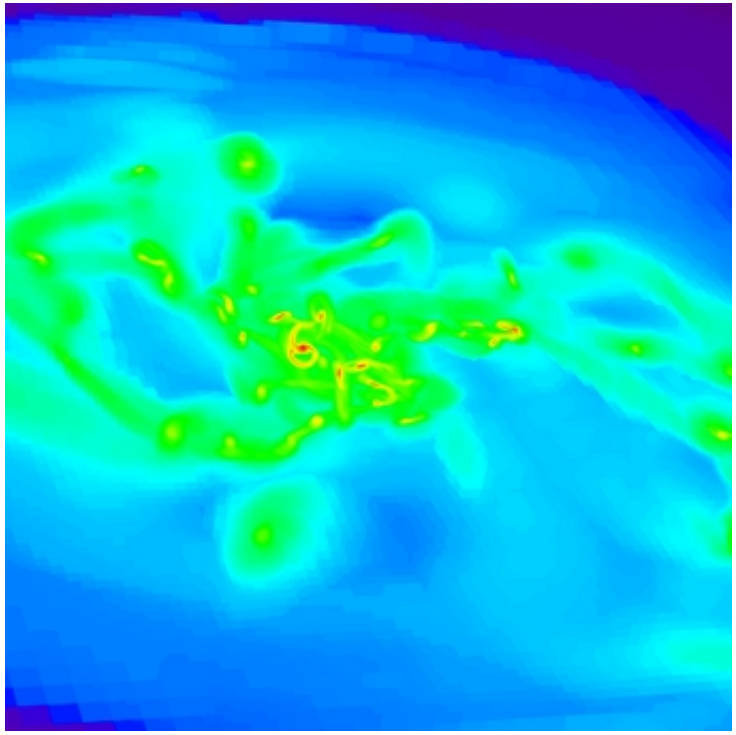
Marc LABADENS

CEA - IRFU - SEDI - LILAS

Le Projet COAST

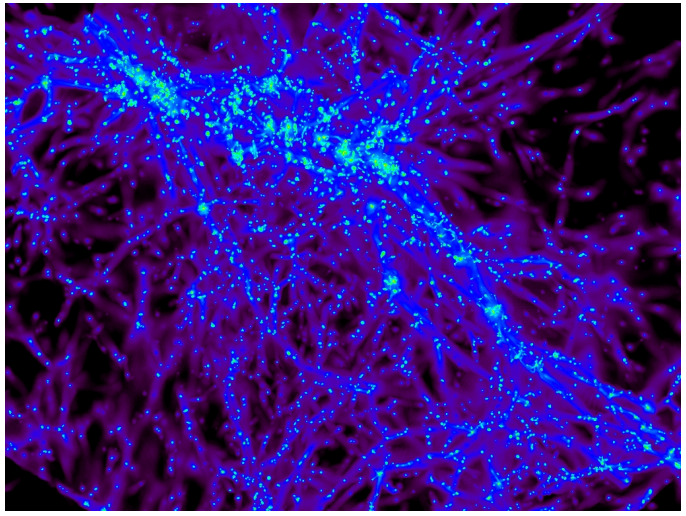
- "COmputational ASTrophysics" - Saclay

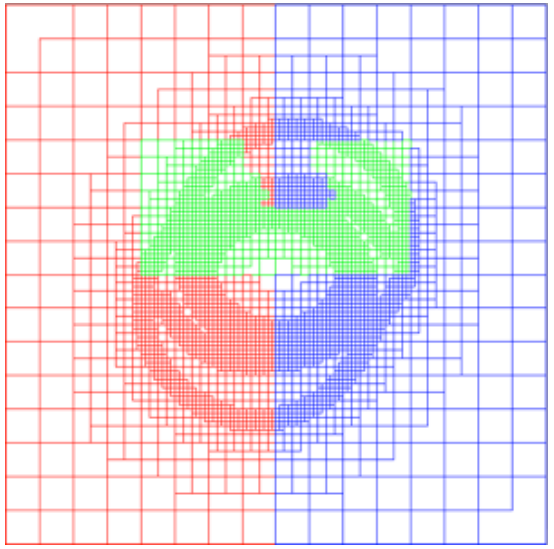




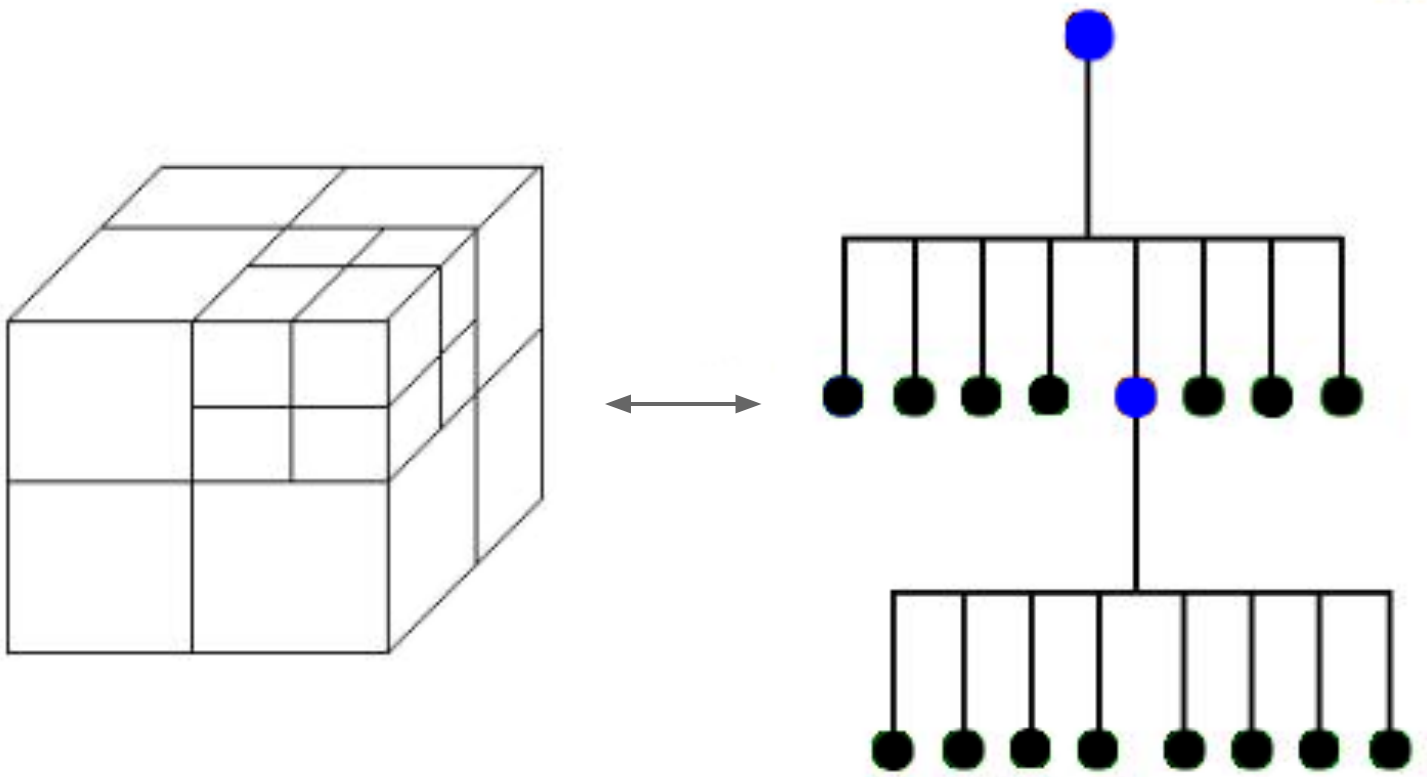
Le Code Ramses

- Résout les équations d'hydro sur une grille
- Octree "Adaptive Mesh Refinement" (AMR)
- Massivement parallèle avec la bibliothèque MPI (Message Passing Interface)

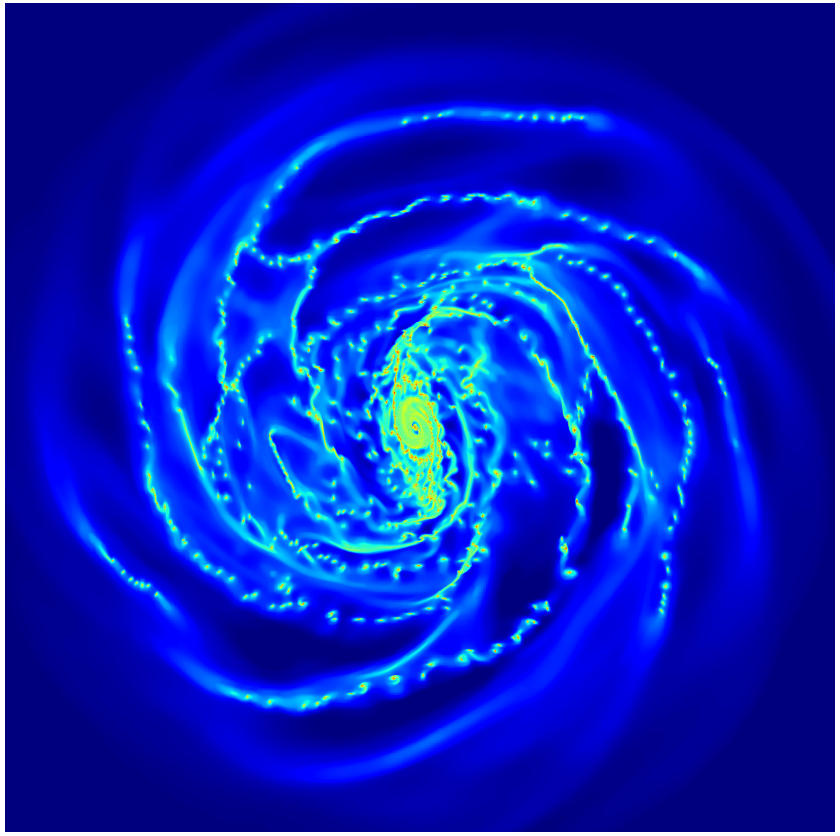




Grille de type Octree AMR



Simulation de type Voie Lactée



6080 processeurs sur "Curie", octree poussé au niveau 22

Logiciels de visualisation disponibles?

- Les logiciels libres disponibles ne permettent pas de charger la structure de donnée en Octree de RAMSES
- Pour un octree au delà du niveau 10 (1024^3~1 milliards de mailles), projeter ces données dans une grille cartésienne devient trop lourd

-> C'est ce qui nous a mené à développer nos propres solutions

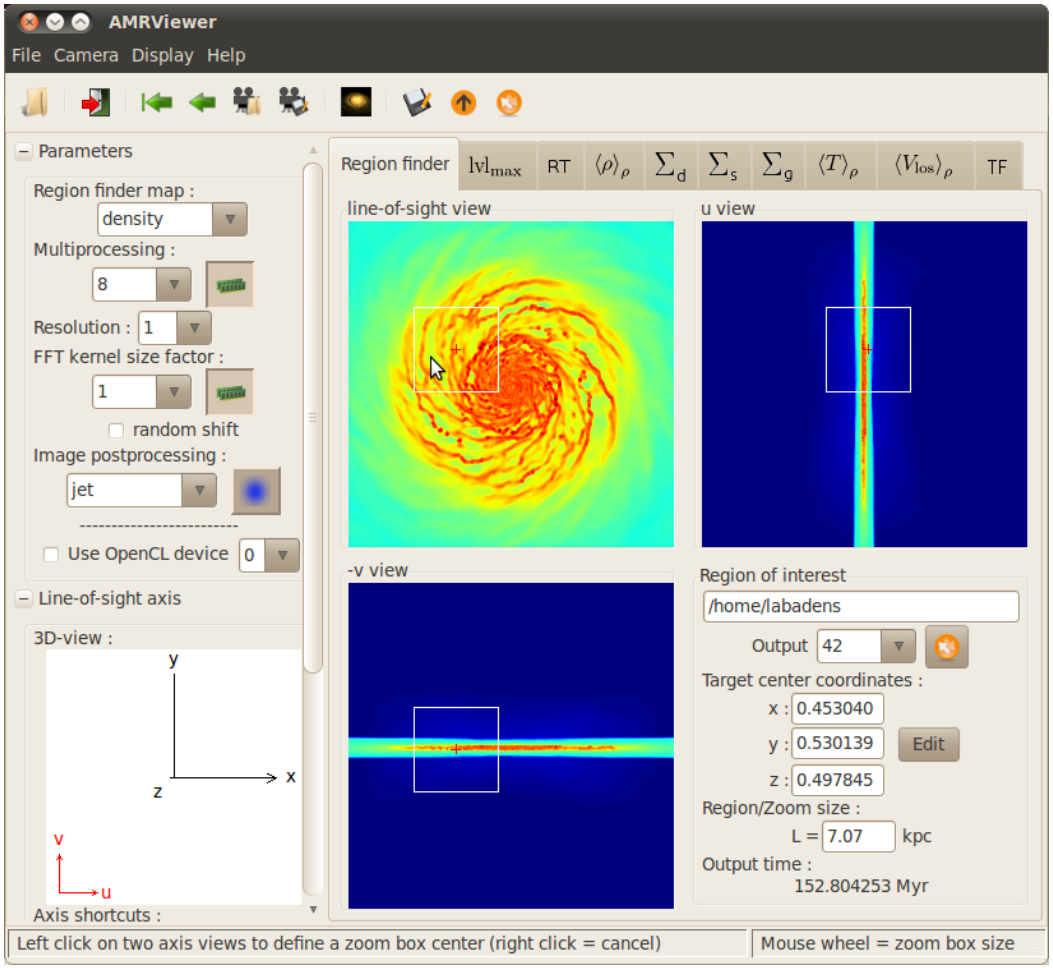
Cadre de travail retenu :

- Python
- Numpy
- Cython - C code

  -> PyMSES = Python modules for RAMSES

- Le goulot d'étranglement des performances reste l'étape de lecture des données
- Il est vraiment préférable de ne pas avoir à déplacer les données

Interface graphique PyMSES



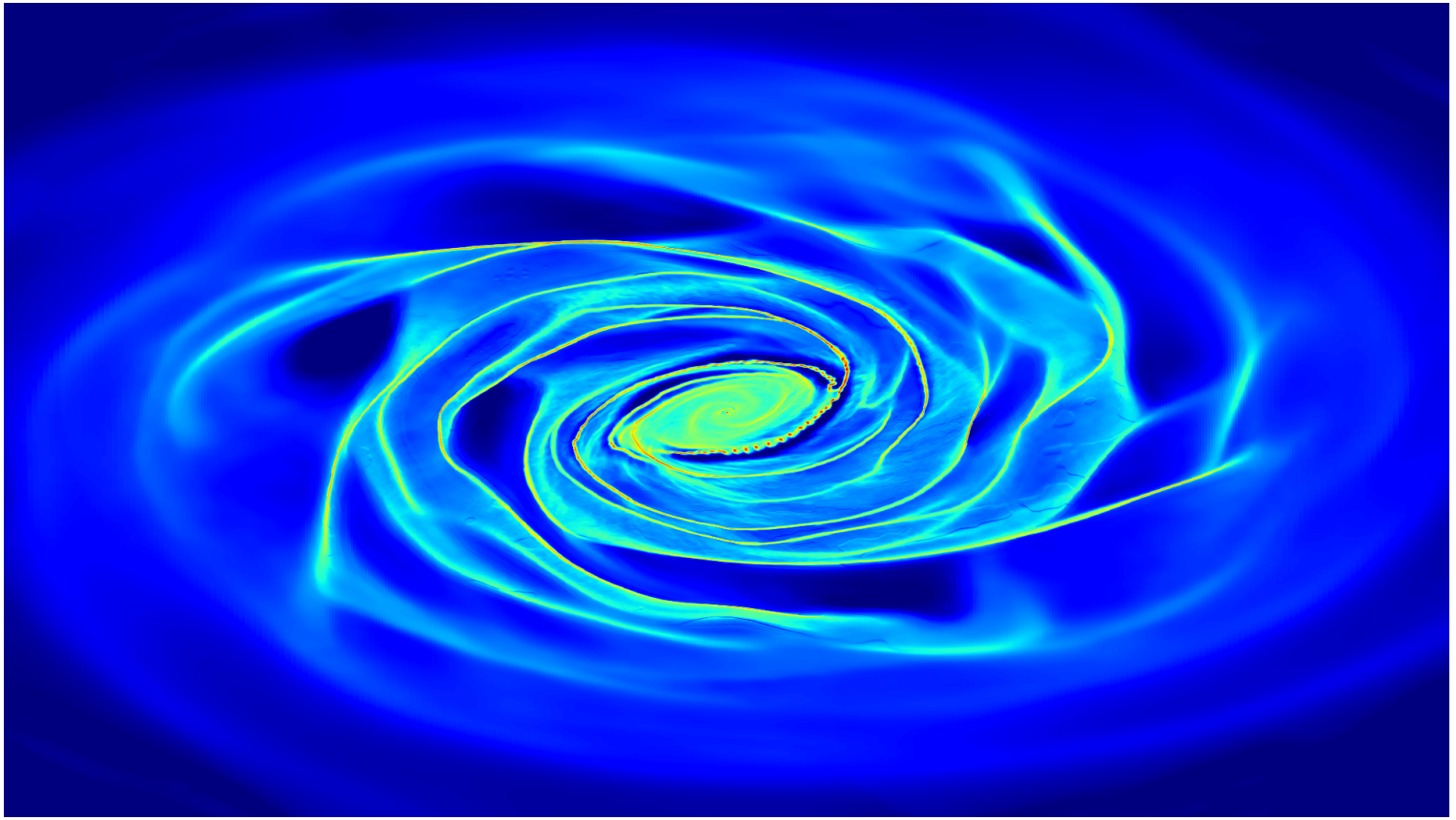
Techniques de visualisation

Volume rendering

Technique du lancer de rayon

- Pour chaque pixel de l'image, un rayon est lancé au travers du volume simulé
- On trouve les intersections entre le rayon et le maillage -> somme des contributions de chaque maille traversée

Technique du lancer de rayon



"Volume ray casting"

Avantages :

- Meilleure qualité d'image

Inconvénients:

- Temps de calcul important : pas encore interactif sur un volume de donnée conséquent
- Sans interpolation, les mailles grossières se détachent

Traversée "Top-down" de l'Octree

- Commence à la racine, et progresse vers les feuilles de l'arbre
- Première implémentation adaptée aux projections de maximum d'intensité

Traversée "Bottom-up" de l'Octree

- Travaille seulement sur les feuilles
- Utilisation d'un tableau additionel des voisins
- Exécution un peu plus rapide sur nos données

Parallélisme du code

avec python

Python multiprocessing

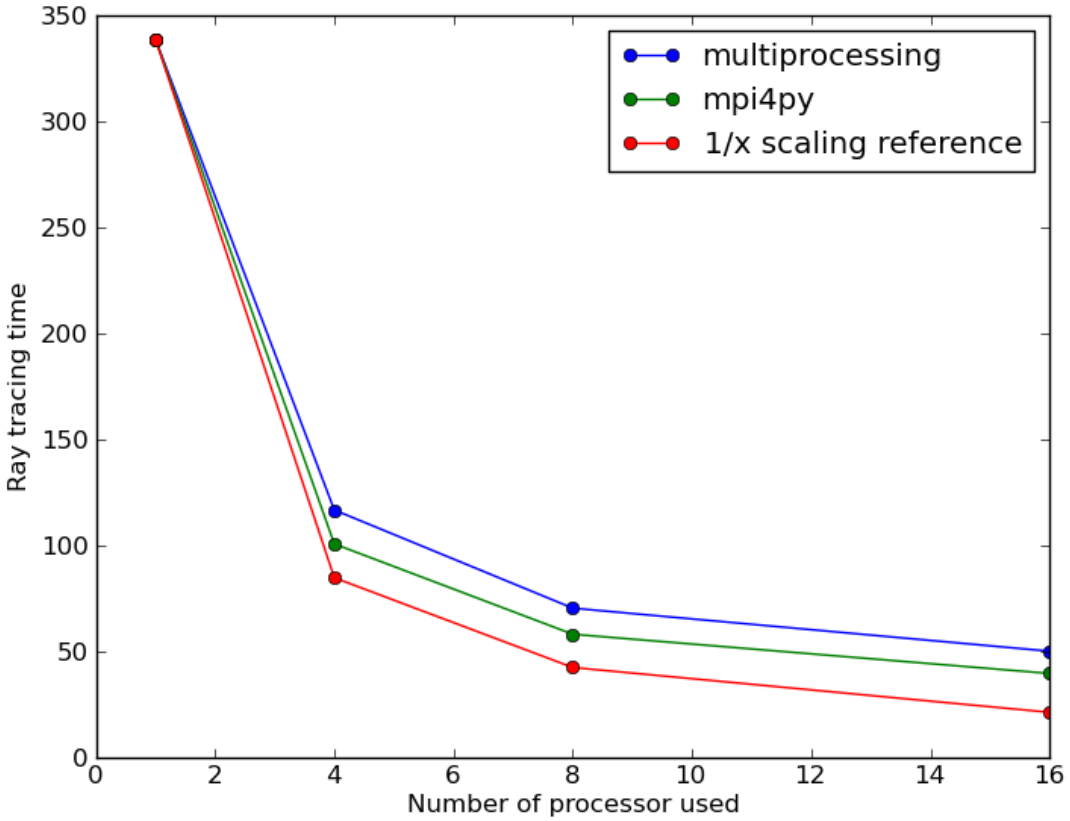
Avantages :

- Fait partie de la bibliothèque standard de Python, facile à implémenter

Inconvénients:

- Pas de mémoire partagée sur les tableaux Numpy (équilibrage plus difficile)
- Limité (nombre de processeurs/RAM) à un noeud de calcul seulement

Lancer de rayon + multiprocessing

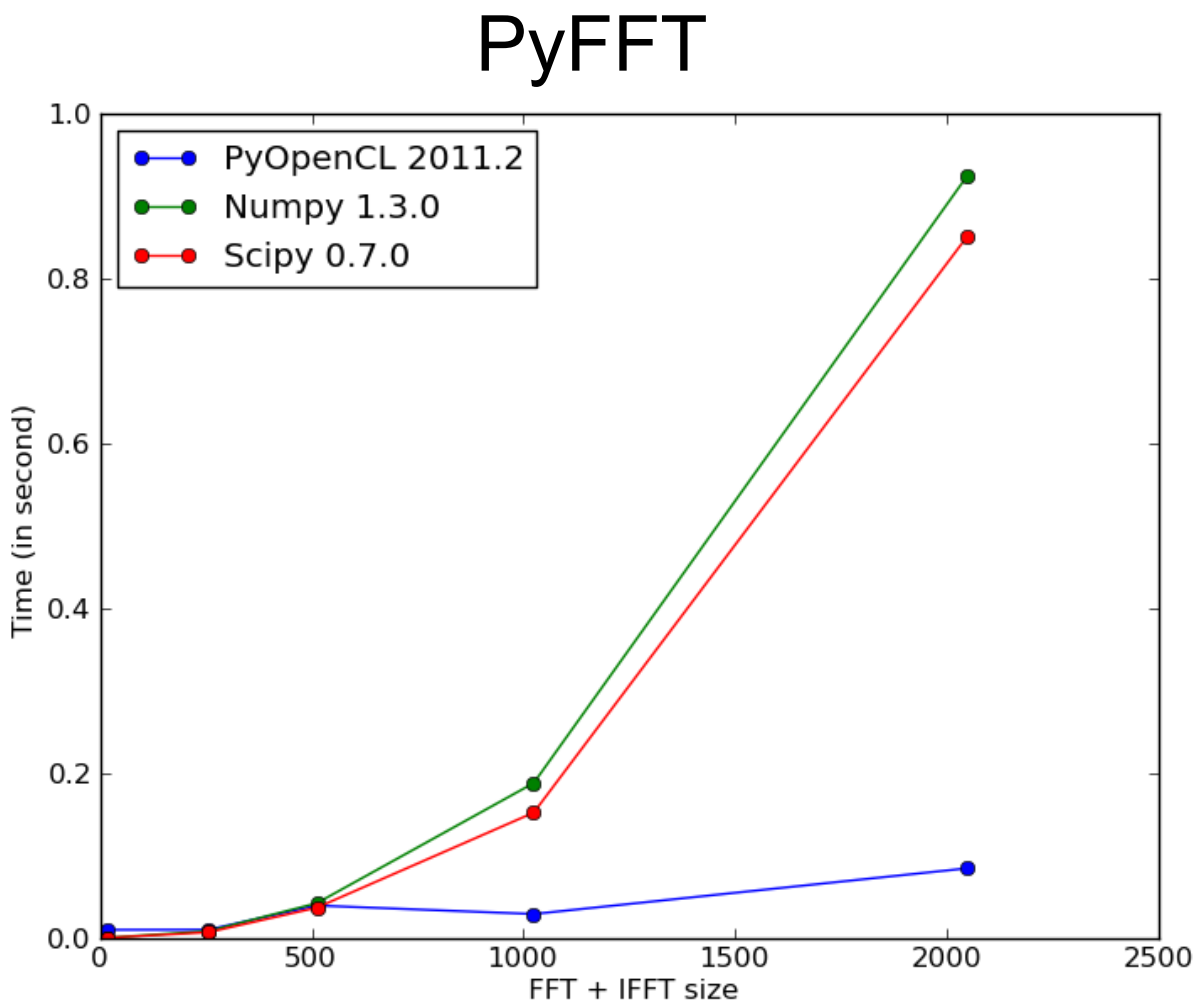


- Lecture des fichiers en parallèle, "sort last rendering"

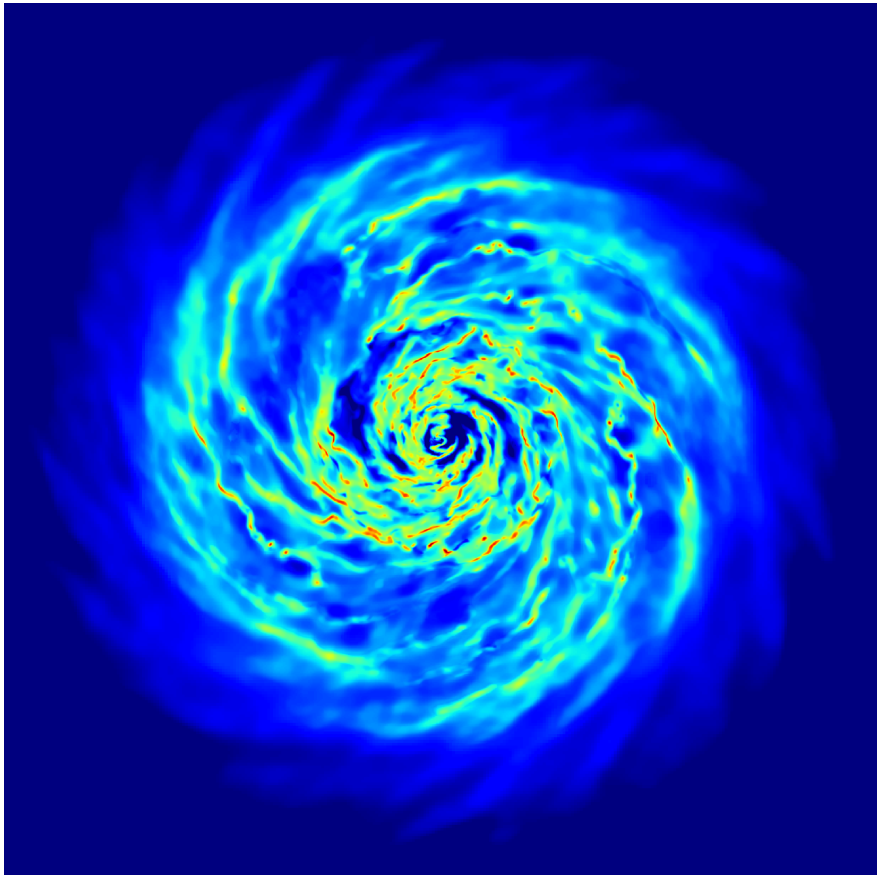
PyOpenCL

- Standard ouvert de programmation parallèle
- Tests d'exécution possibles à la fois sur carte graphique et sur CPU multi-coeurs.
- Tests fait avec un GPU Nvidia Quadro FX 1800 et un CPU Intel Xeon @ 2.67GHz

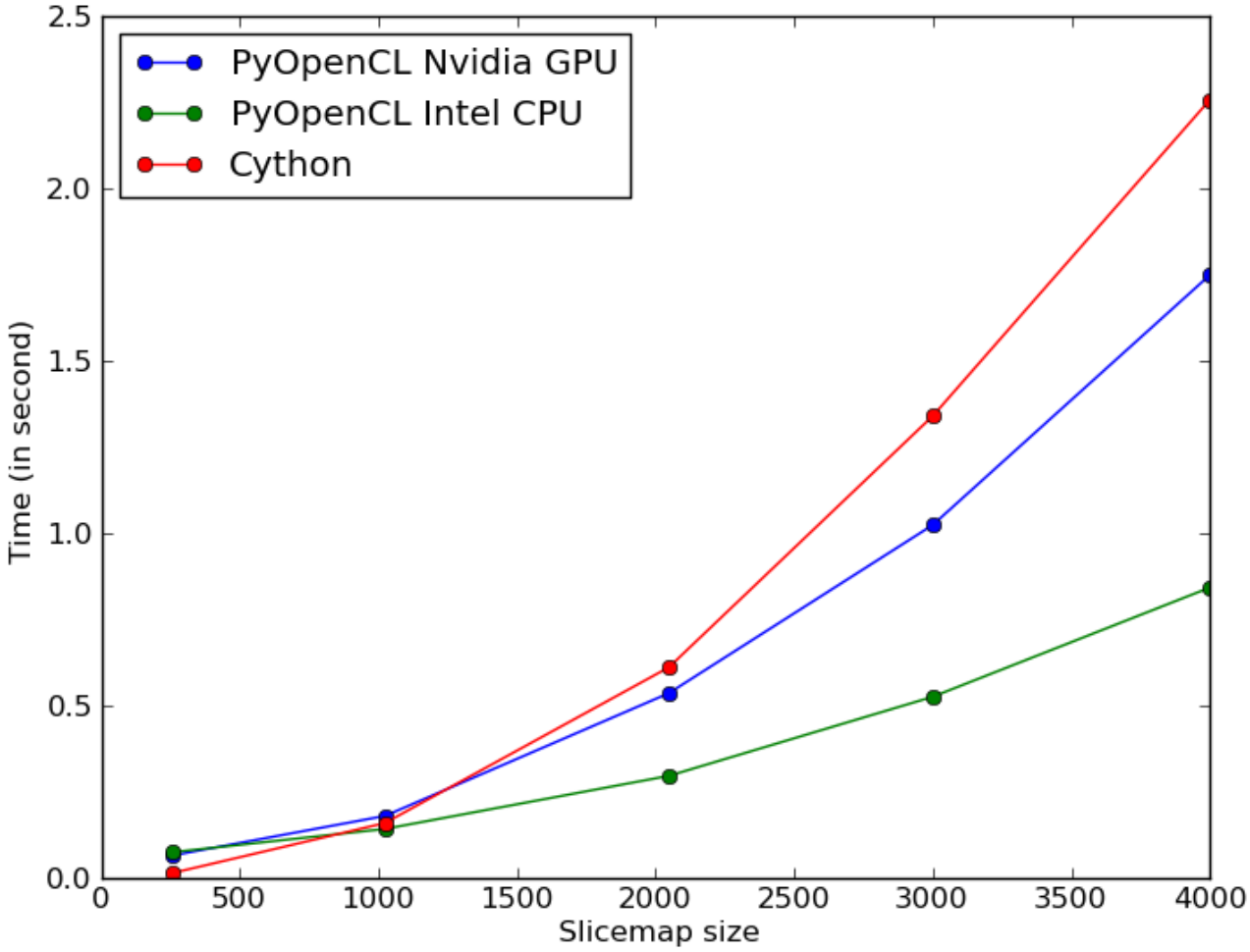
PyOpenCL - PyFFT - Quadro FX 1800



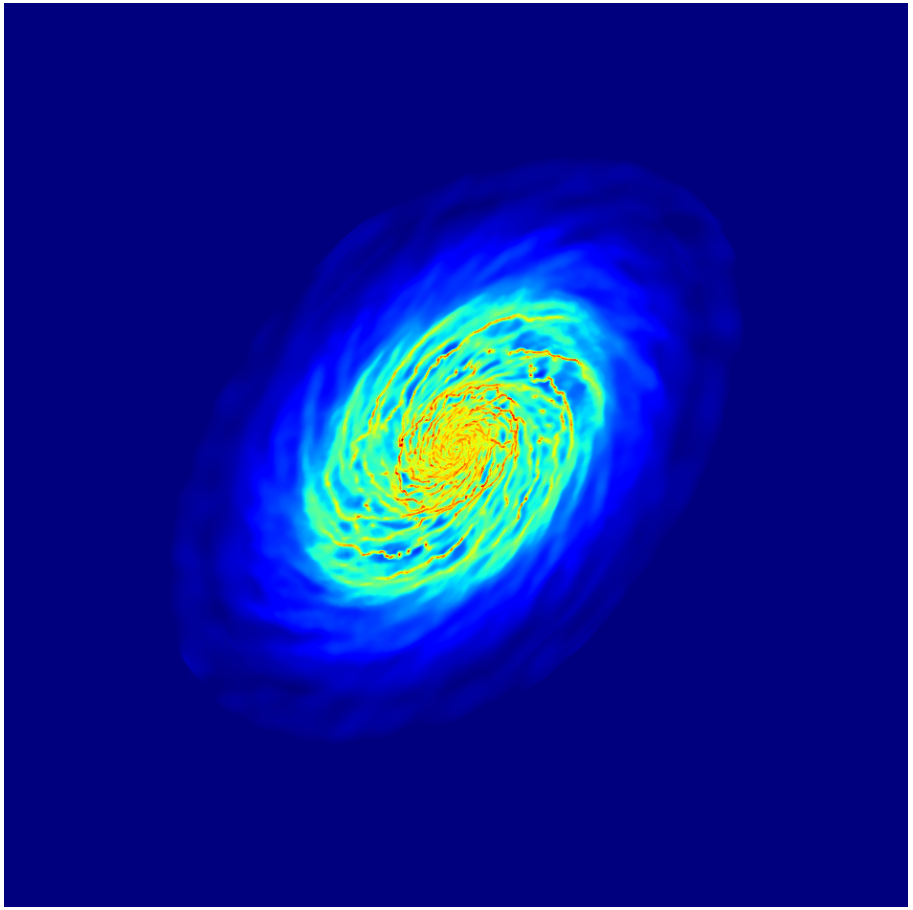
PyOpenCL - PyMSES Slicemap



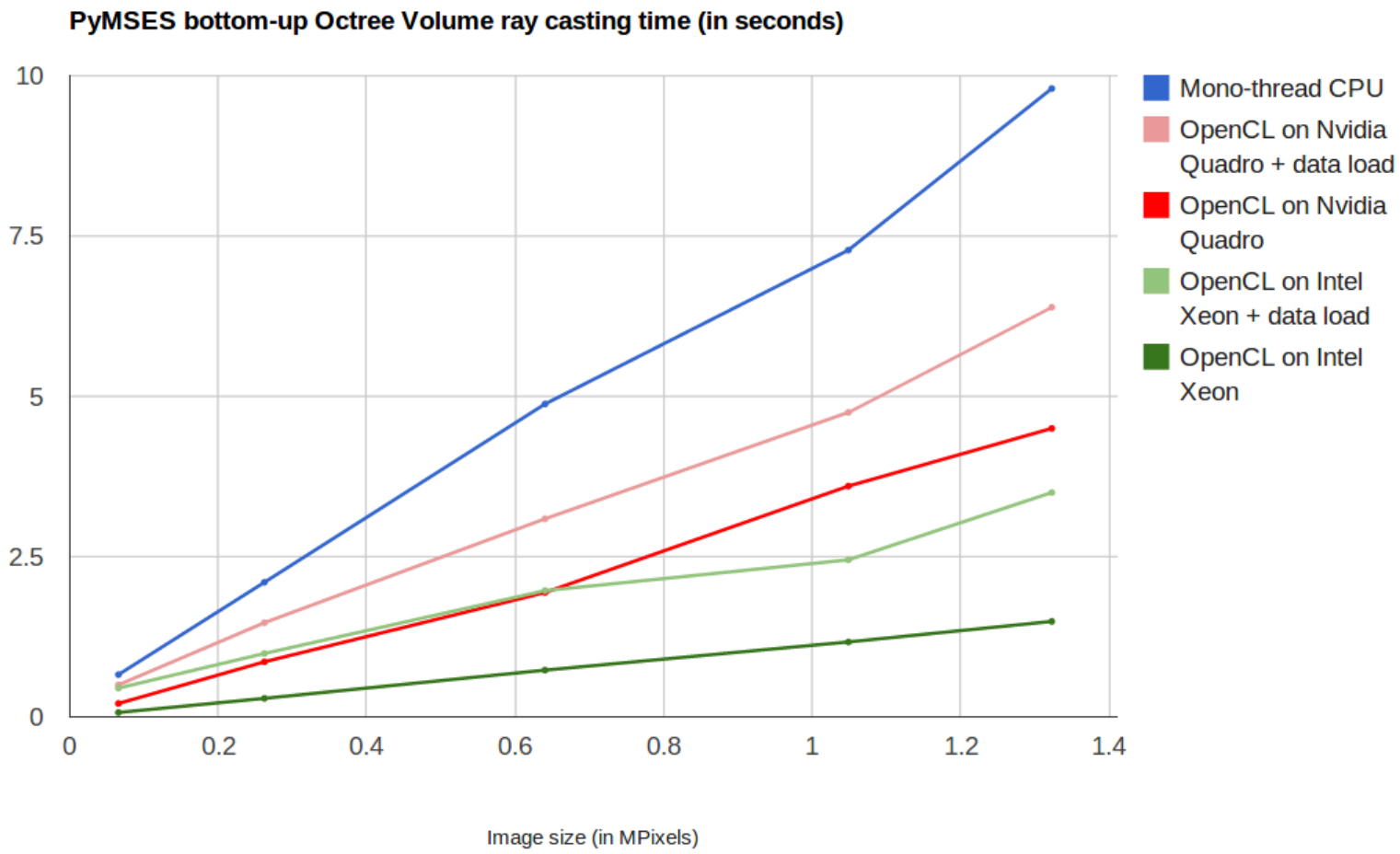
PyOpenCL PyMSES Slicemap



PyOpenCL "sort first rendering"



PyOpenCL "sort first rendering"



PyOpenCL

- Pas de difficulté d'implémentation à partir d'un code en C
- Il faut installer OpenCL pour chaque accélérateur
- Exécution + rapide sur le CPU Intel que sur le GPU dans notre cas !

Questions ?