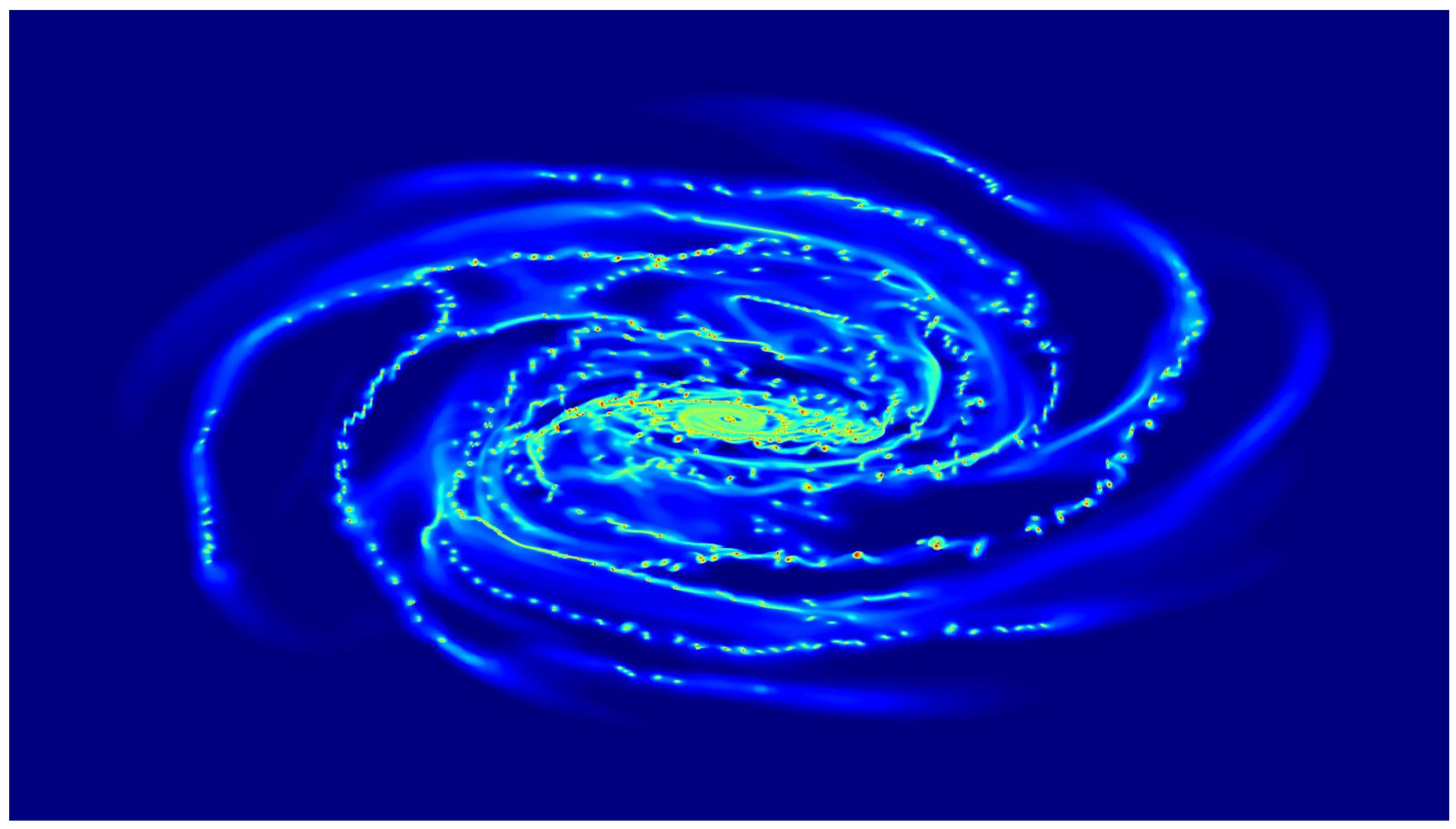
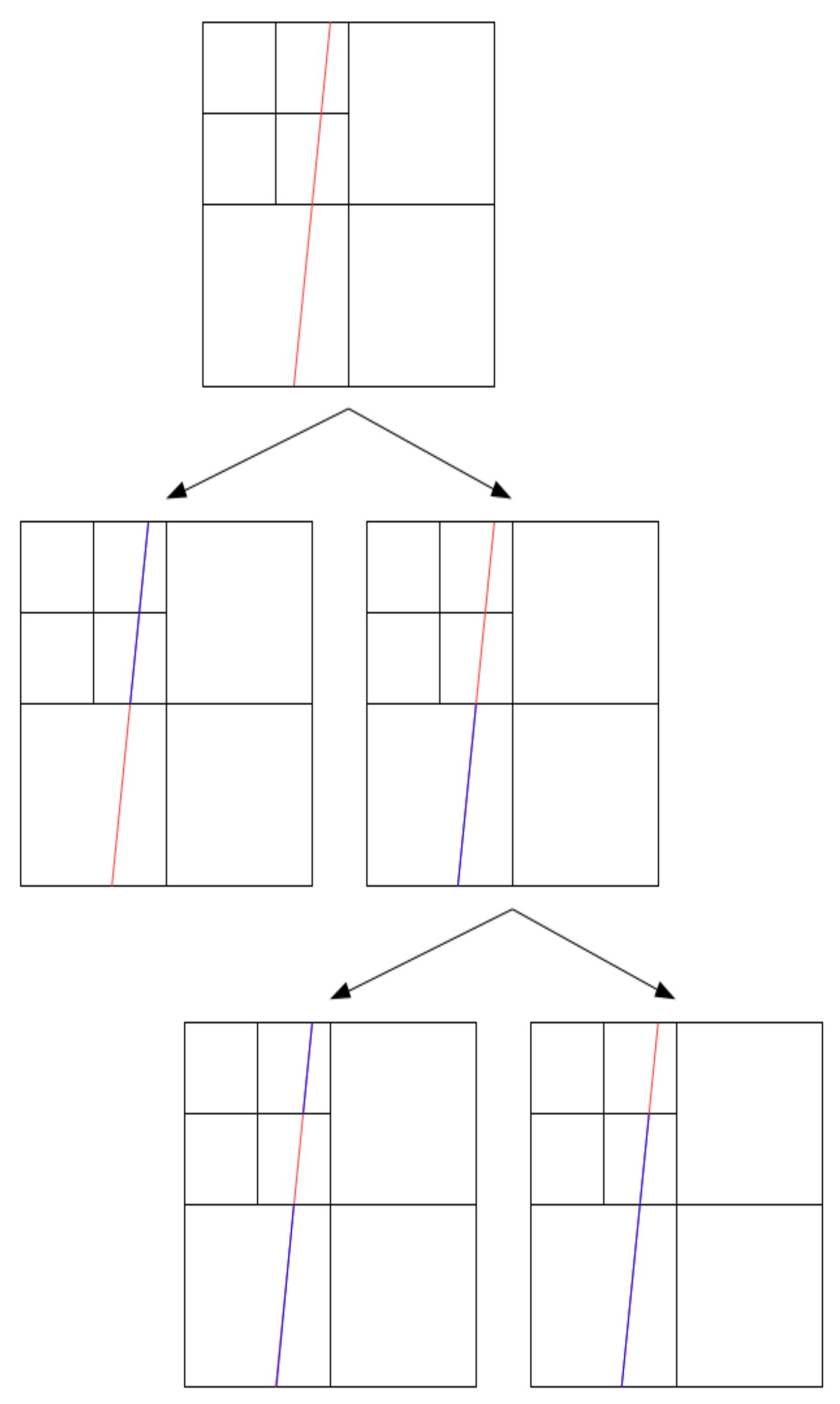
# Visualization of Octree adaptive mesh refinement (AMR) in astrophysical simulations

Labadens Marc, Chapon Damien, Pomarède Daniel, and Teyssier Romain
Centre d'Etudes de Saclay, 91191 Gif-sur-Yvette Contact : marc.labadens@cea.fr

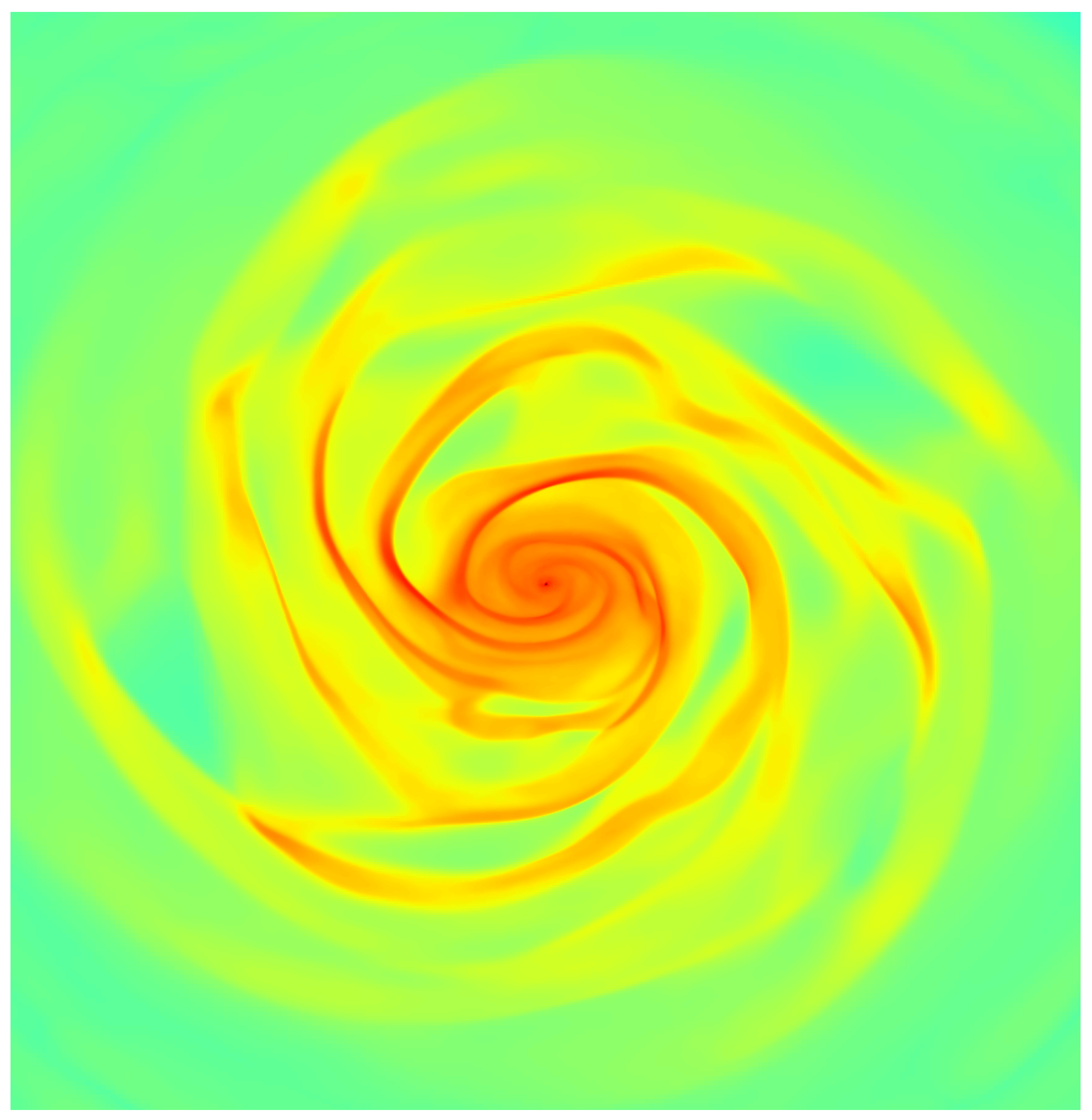
## The RAMSES code [1]

The RAMSES code was developed in the astrophysical service of the "Commissariat à l'énergie atomique et aux énergies alternatives". This code simulate the behavior of self gravitating fluid, and is widely used for galaxy formation studies. It uses Fortran 90 and the MPI (Message Passing Interface) library to run in parallel on supercomputers. This code works with octree AMR (Adaptive Mesh Refinement) to allow a better numerical precision while keeping a reasonably low computing cost.





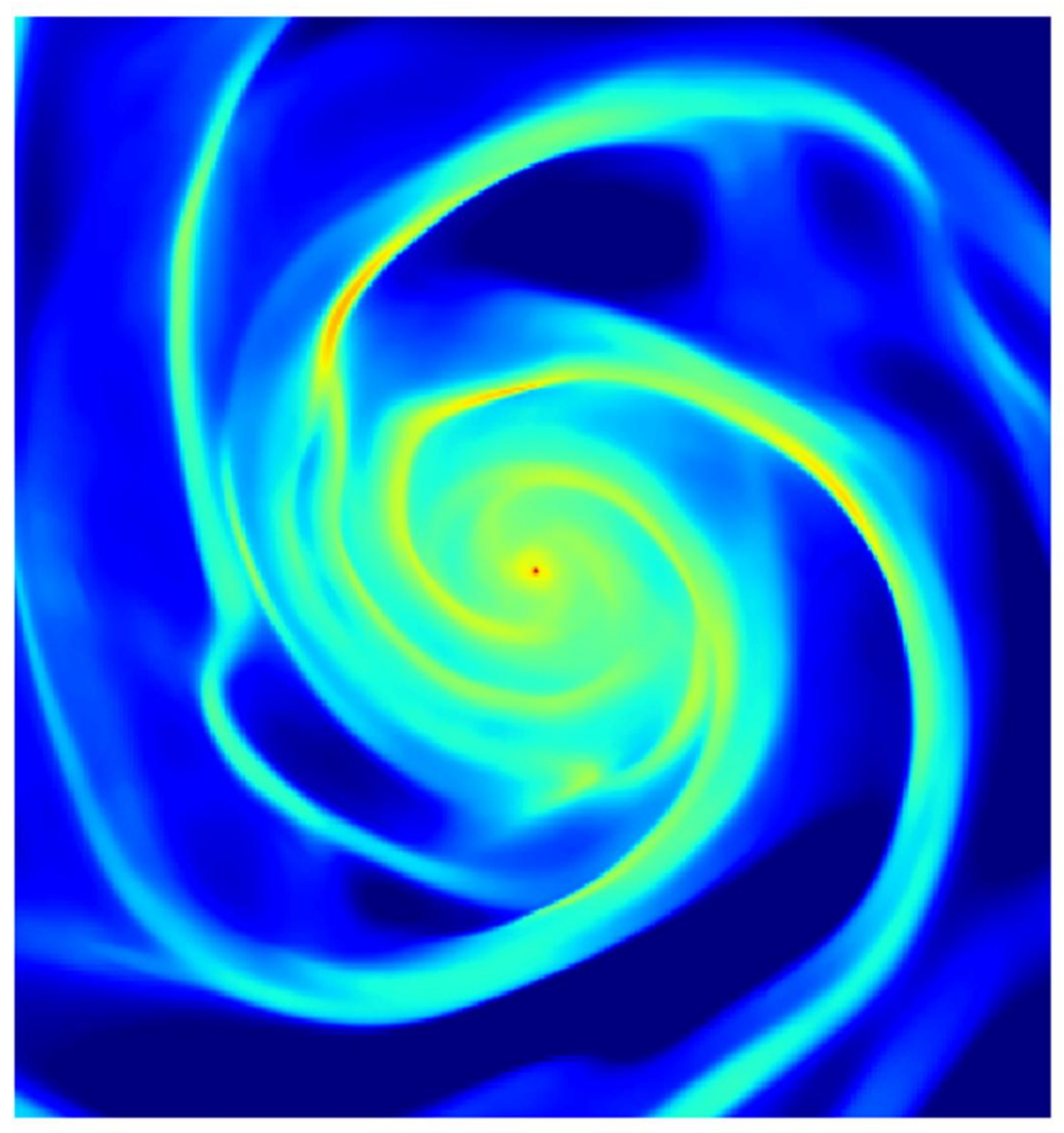
Tree based ray tracing approach

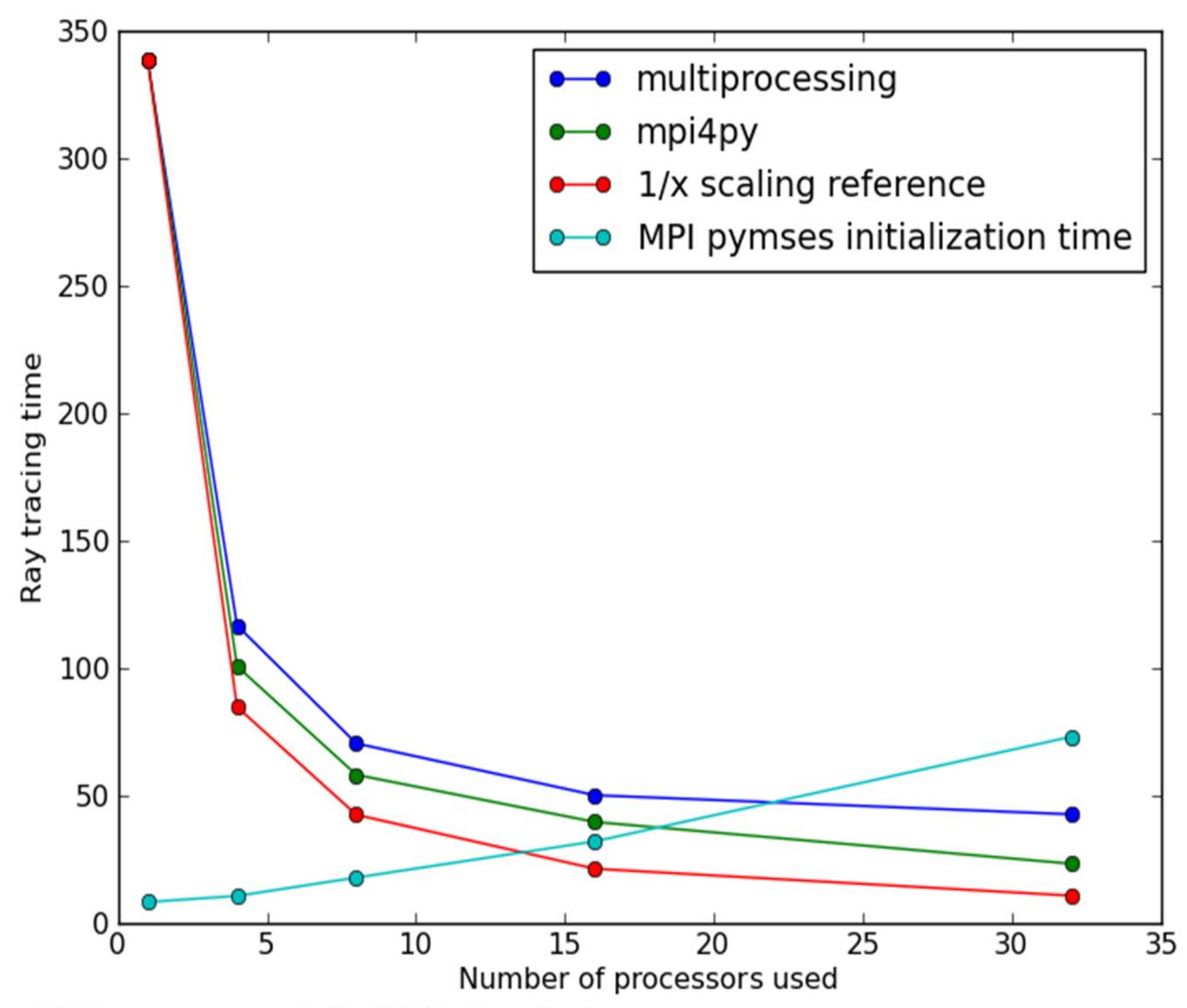


Simple high resolution ray tracing aligned with the simulation axis

## The PYMSES code [2]

This code is a python module written specially to work with RAMSES output data files. It allows an easy access to data, dealing with the domain decomposition and memory management problems. It allows the user to use powerful python libraries like Numpy/Scipy [3], Matplotlib, PIL, HDF5/PyTables... It also does three-dimensional volume rendering with specific algorithm optimized to work on RAMSES distributed data.

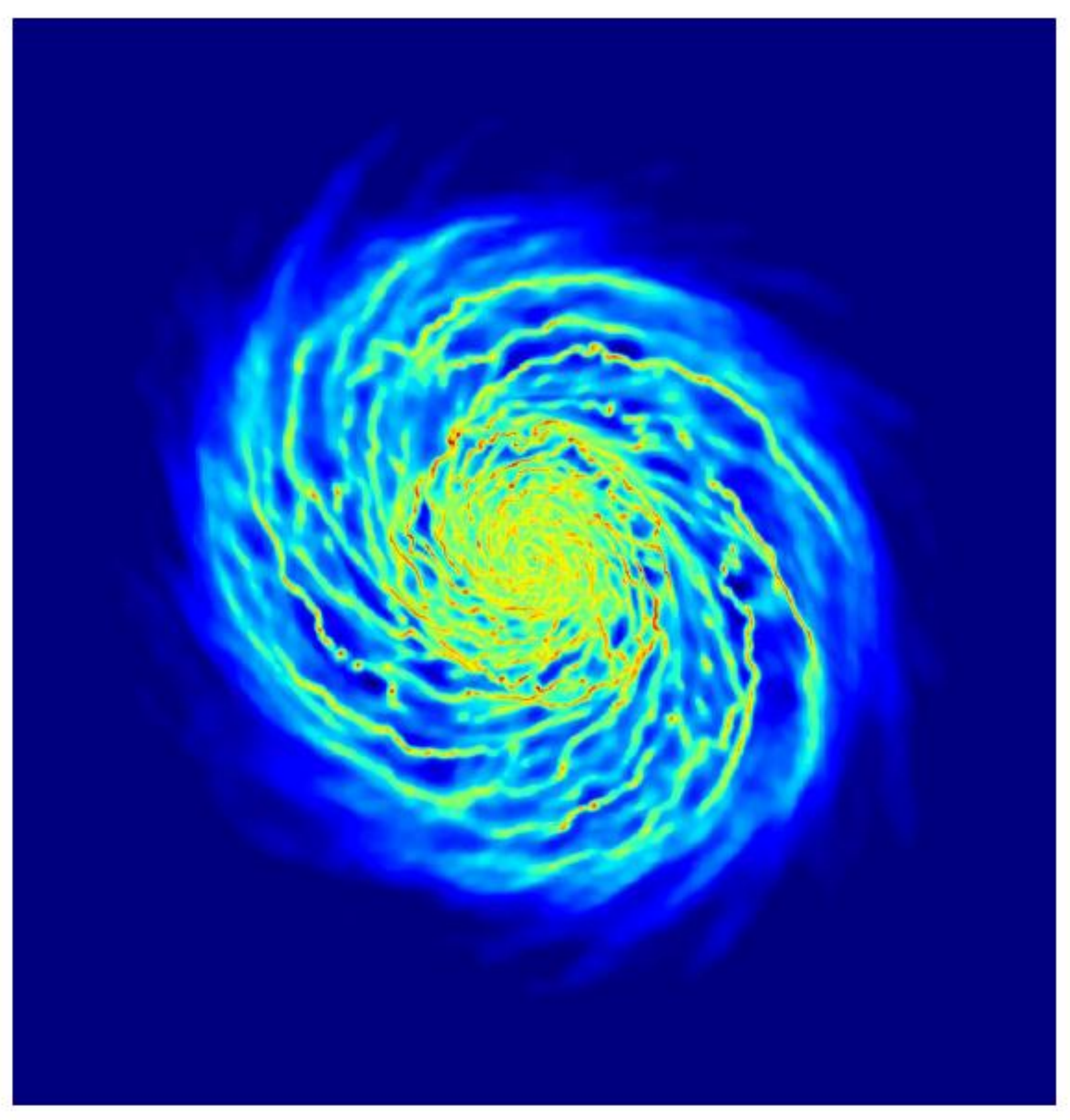


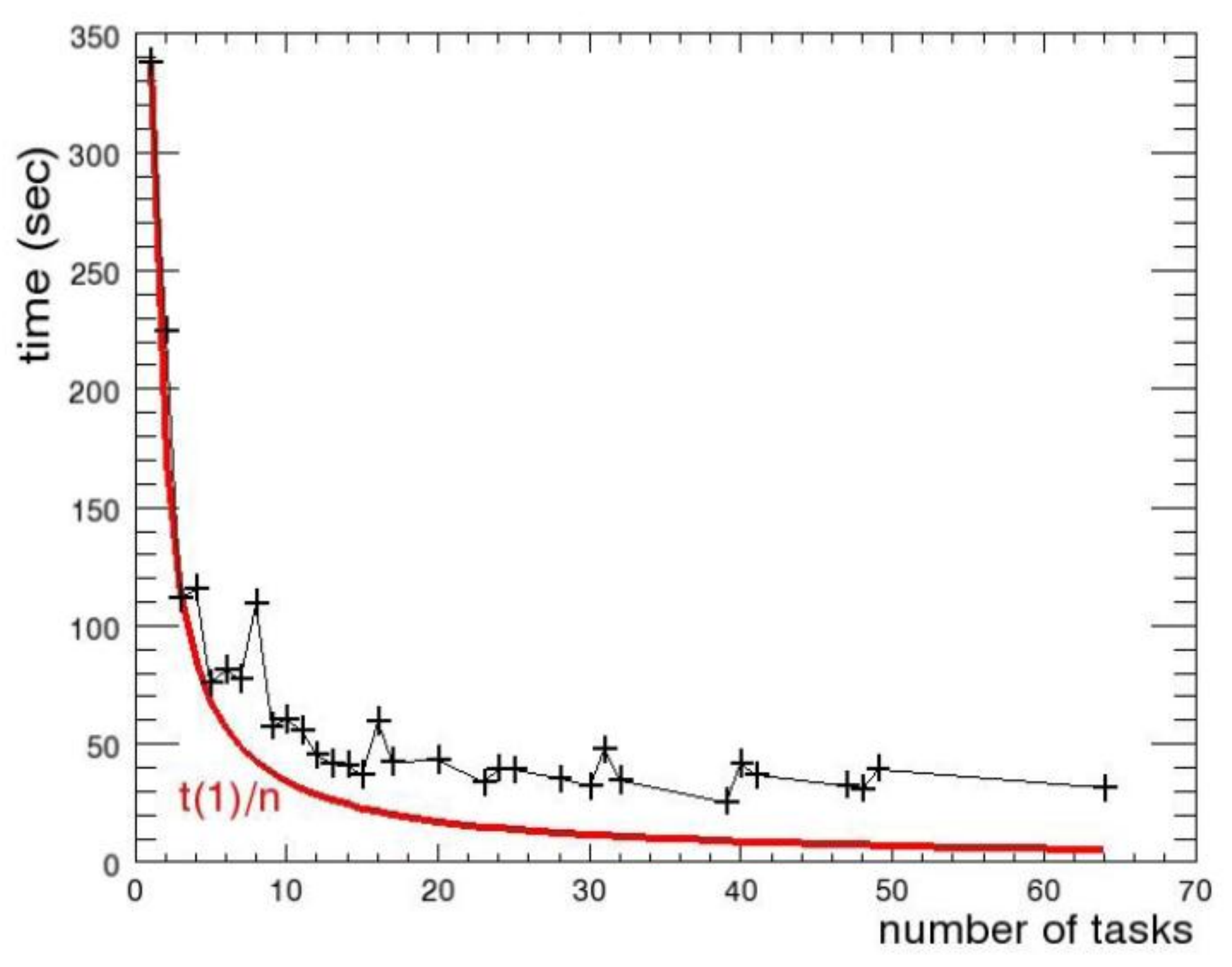


MPI versus multiprocessing benchmark on a 6080 cpu cores RAMSES simulation

## PYMSES parallel execution :

Two main solutions have been successfully investigated to turn the initial sequential PYMSES volume rendering code into a parallel code. First we used the multiprocessing module [5], which allows to make the most of our multi-core processing machines. It is part of the python standard library (since python 2.6), and works with parallel execution of sub process. Second we used the MPI library [6], which is useful to use supercomputers with distributed memory.





MPI ray tracing scaling benchmark on a 64 cpu cores RAMSES simulation

## Load balancing strategy :

We chose to reuse RAMSES load balancing trough parallel data file reading. Indeed, RAMSES data are written in as many file as there were cpus used in the RAMSES simulation, with an optimized distribution using Hilbert key domain decomposition.

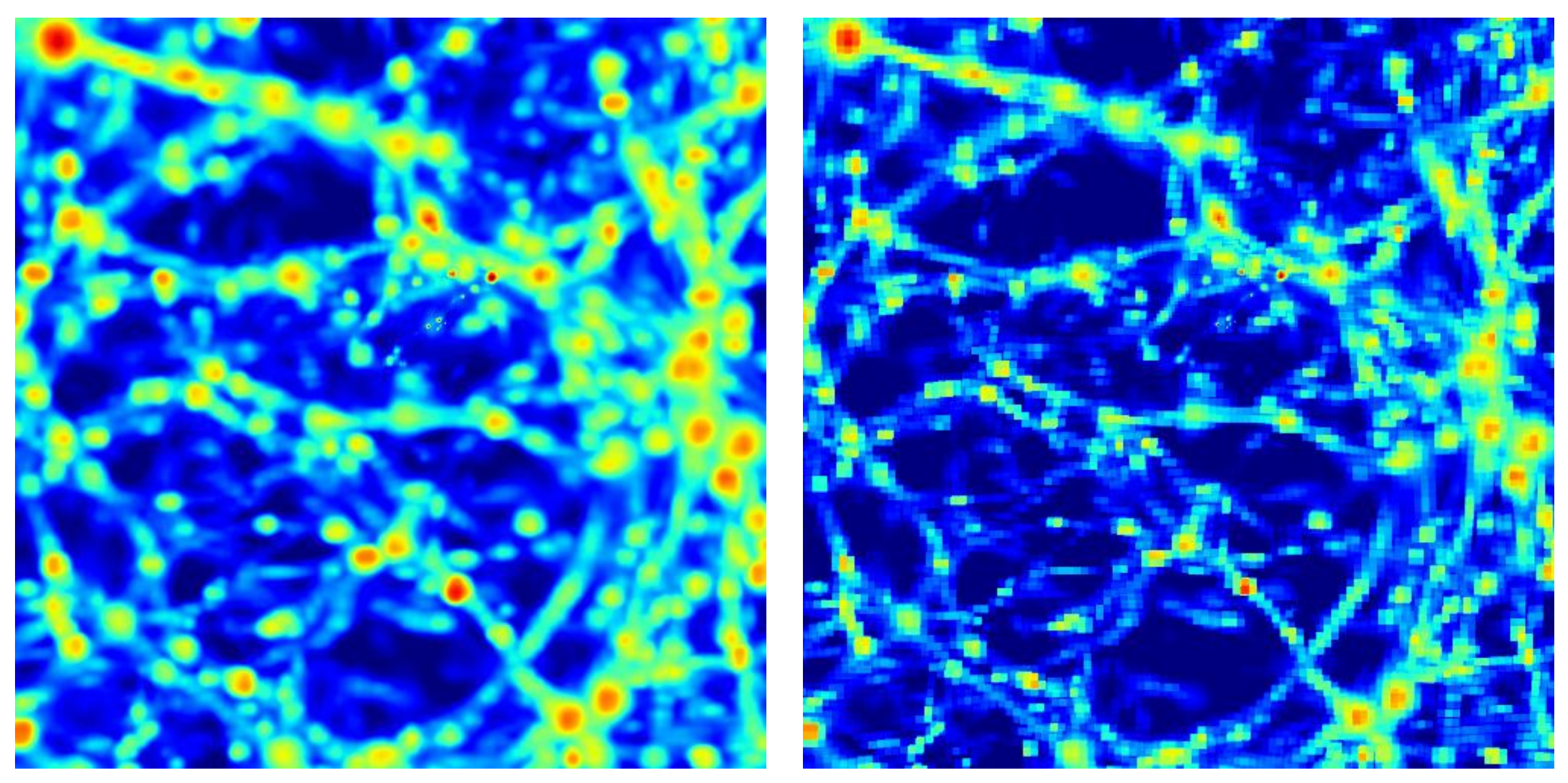
However, the load balancing was not yet optimal, this is why we tried to use an even better load balancing though a quick pre visualization to get a better idea of the processing time cost for each file.

The multiprocessing library enable a simple load balancing solution between different process: We only need to start more process than there are processors on the machine. The load balancing problem is then simply solved by the operating system of the machine. One limitation is the system ability to create and use a great number of threads.

## Volume rendering techniques:

The **ray tracing** is computed directly on the octree structure. We find the coordinates of the intersections between the ray and the grid. Then we go down the tree until we find every leaf cells. Eventually we sum values of the grid cells with a weight corresponding to the ray length through the cell multiplied by the cell value.

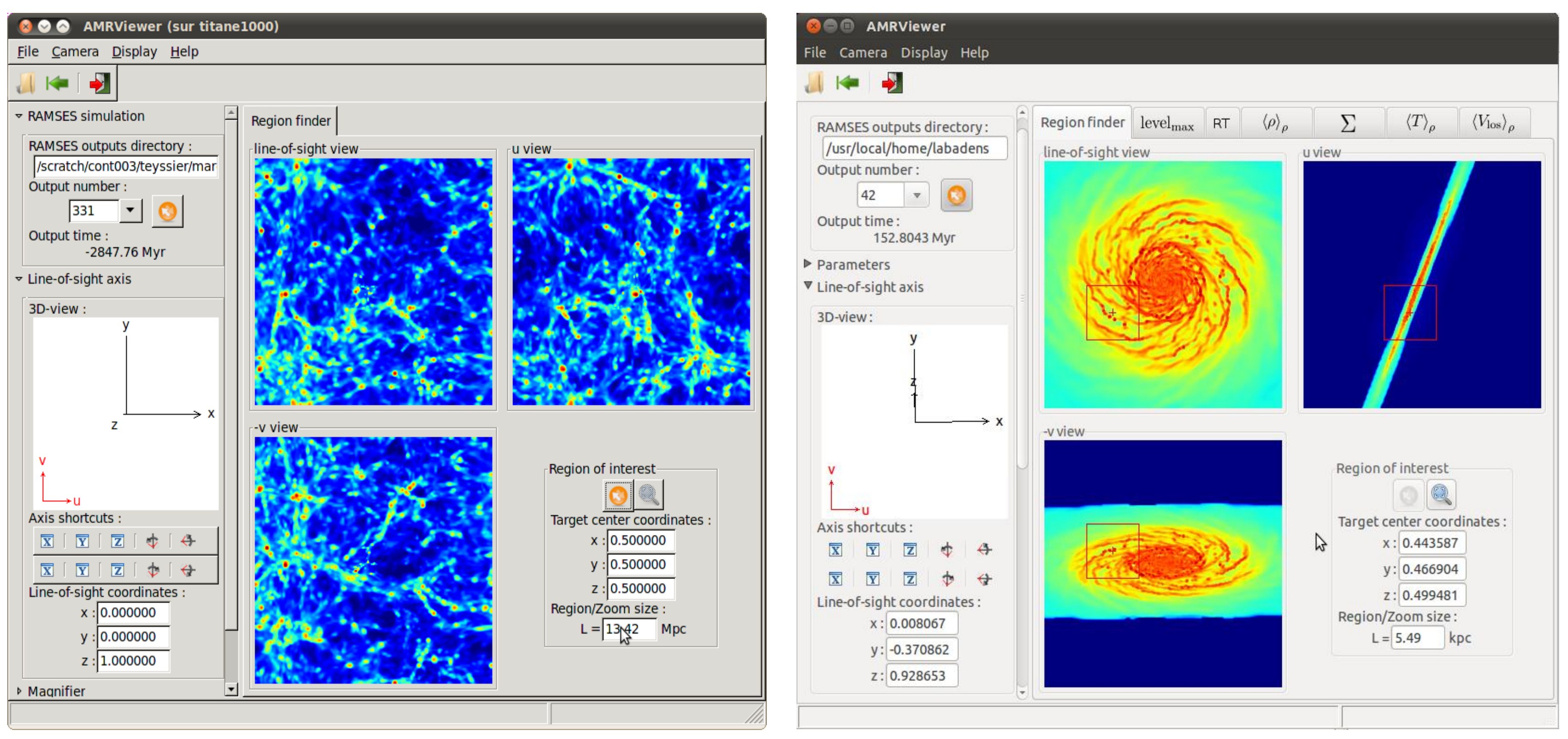
An other useful volume rendering technique implemented in PYMSES is the **splatting** technique. AMR cells are first converted into points with an additional size information depending on the AMR cell size. Points are then projected on the resulting image to render using optimized Numpy libraries. We get Gaussian three-dimensional spherical points by convolution with Gaussian kernel using Numpy Fast Fourier Transform [4]. The resulting image is close to a ray tracing image with an adaptive Gaussian blur. We can also speed up this process with parallel projection and FFT process. This technique is useful to get a quick three-dimensional view of a large dataset.



Comparison between splatting and ray tracing

## PYMSES memory management :

To avoid memory problems due to large size of RAMSES output, PYMSES handle data files sequentially. PYMSES needs more memory when it runs the code in parallel, as each process loads different datafiles simultaneously. On a supercomputer, the MPI PYMSES ray tracing code load indeed data in parallel, to make the most of the large distributed memory system available.



RAMSES outputs directory : /scratch/cont003/teyssier/mar
Output number : 331
Output time : -2847.76 Myr
Line-of-sight coordinates : x : 0.000000 y : 0.000000 z : 1.000000
Target center coordinates : x : 0.500000 y : 0.500000 z : 0.500000
Region/Zoom size : L = 13.42 Mpc

RAMSES outputs directory : /usr/local/home/labadens
Output number : 42
Output time : 152.8043 Myr
Line-of-sight coordinates : x : 0.008067 y : -0.370862 z : 0.928653
Target center coordinates : x : 0.443587 y : 0.466904 z : 0.499481
Region/Zoom size : L = 5.49 kpc

The amrviewer PYMSES GUI

## References :

[1] R. Teyssier. The ramses code. *http://web.me.com/romain.teyssier/Site/RAMSES.html*, 2002-2011.
[2] Marc LABADENS Thomas GUILLET, Damien CHAPON. Pymses. *http://irfu.cea.fr/Projets/PYMSES/intro.html*, 2011.
[3] Peterson P. Jones R., Oliphant T. scipy. *http://scipy.org*, 2011.
[4] D. Chapon. *PhD Thesis*. PhD thesis, Université Paris 7, 2011.
[5] R. Oudkerk. The Python multiprocessing module., 2006-2008.
[6] L. Dalcin. Mpi for python. *http://mpi4py.scipy.org/*, 2010-2011.