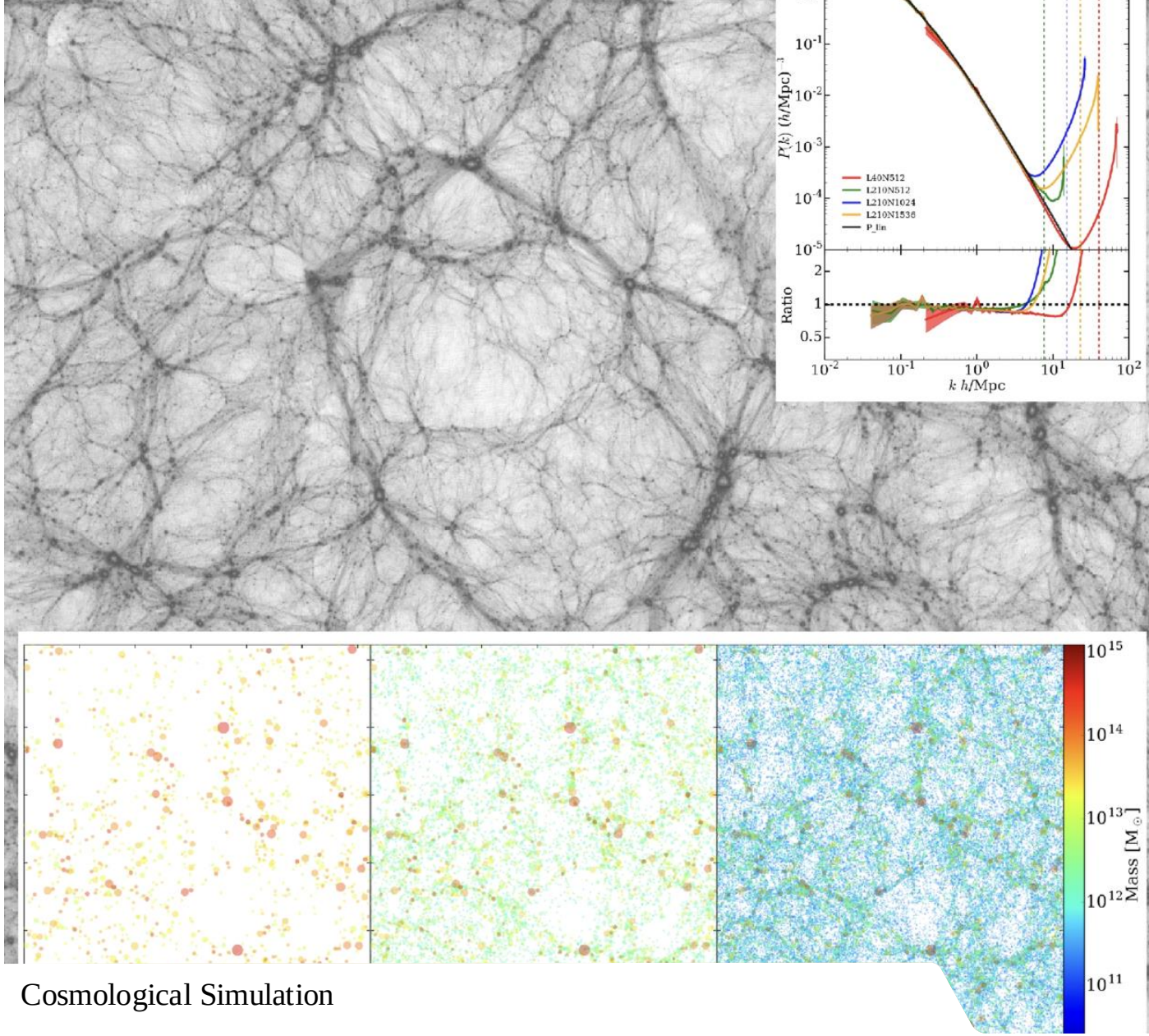
VELOCIraptor: A MPI+OpenMP data analysis tool for cosmological simulations

Dr. Pascal Jahan Elahi
Pawsey Supercomputing Research Centre
Perth, WA
EuroMPI 2024

Outline

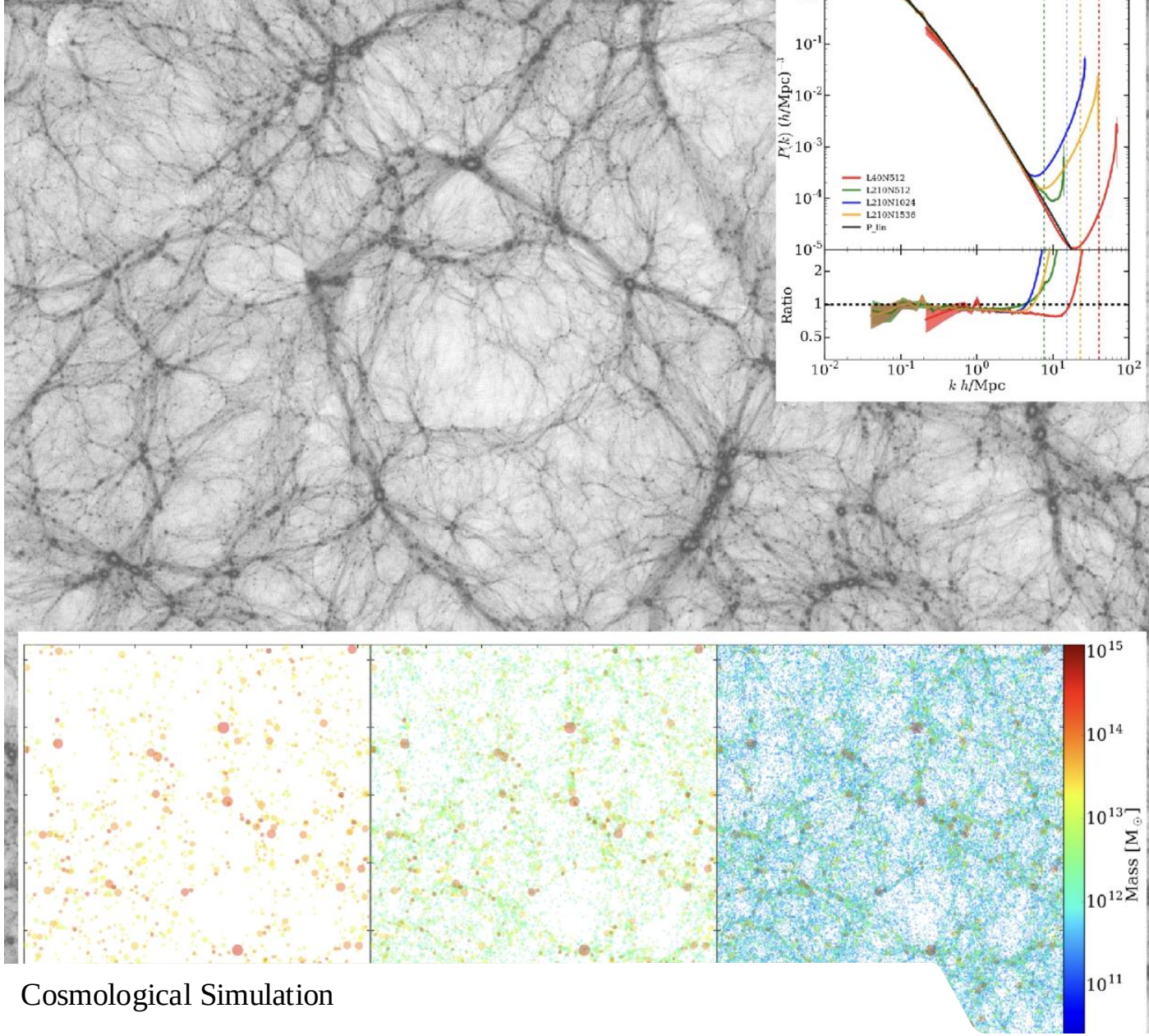
- Introduction to Numerical Cosmology & Structure Finding
- VELOCIraptor
- MPI-raptors
- ~~OpenMP-raptors~~



Cosmological Simulation

Cosmological Simulations

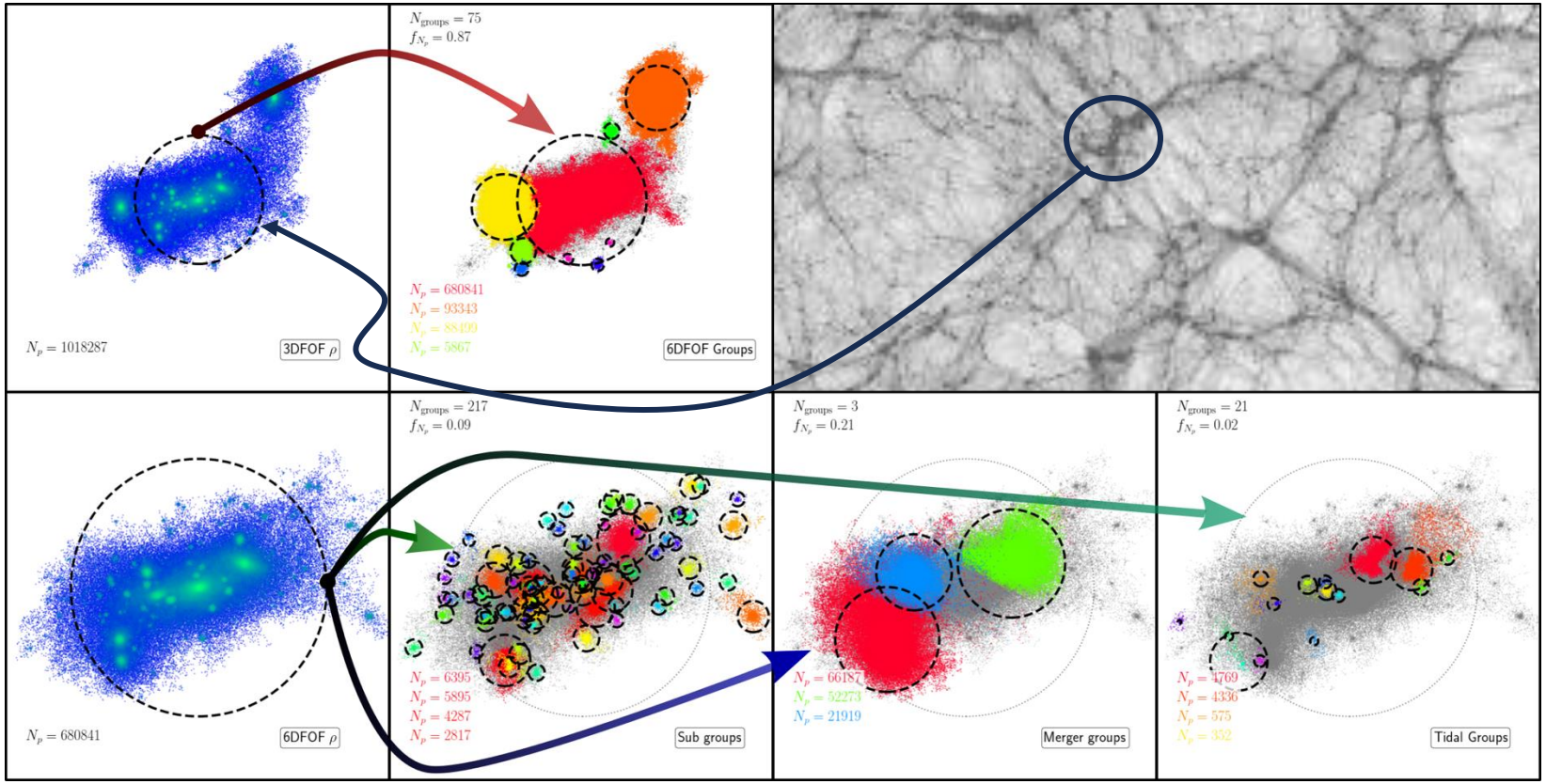
- Common approach to understanding the large-scale universe and the objects that form within it, like galaxies, is to run high-resolution, large-volume cosmological simulations.
  - Hundreds of billions of elements, 10s to 100 Setonix nodes. Necessitates MPI
- Critical step in extracting information from sophisticated, multi-billion particle simulations is the non-trivial identification of cosmic structures, like dark matter halos and synthetic galaxies.
  - Need to load data rapidly to quickly analyse 100s of snapshots. Needs 10s to 100 Setonix nodes. Necessitates MPI.



Cosmological Simulation

VELOCIraptor

- A phase-space [multi-dimensional] (sub)halo finder capable of identifying dark matter halos and galaxies
- Open source https://github.com/pelahi/VELOCIraptor-STF.git & documentation http://velociraptor-stf.readthedocs.io/en/latest
- MPI + OpenMP, C++17, CMAKE build system,, Parallel IO via HDF5 (ADIOS in works)



Halo and Substructure Identification from Simulation

VELOCIraptor

Configuration

- Read configuration & init deployment setup (MPI, OpenMP)

Input

- Read input file domain setup MPI decomposition
- Fully read data in parallel
- Can have different # of tasks reading input file(s)

Field Search

- Each MPI process searches for field halos (parent clusters) locally and across neighbouring MPI domains.
- MPI process can locally search with OpenMP
- Localise all field halos to single MPI domain, adjust MPI decomposition

Substructure

- MPI localised field halos searched for (sub)substructures in all (sub)structures.
- OpenMP enabled

Analysis

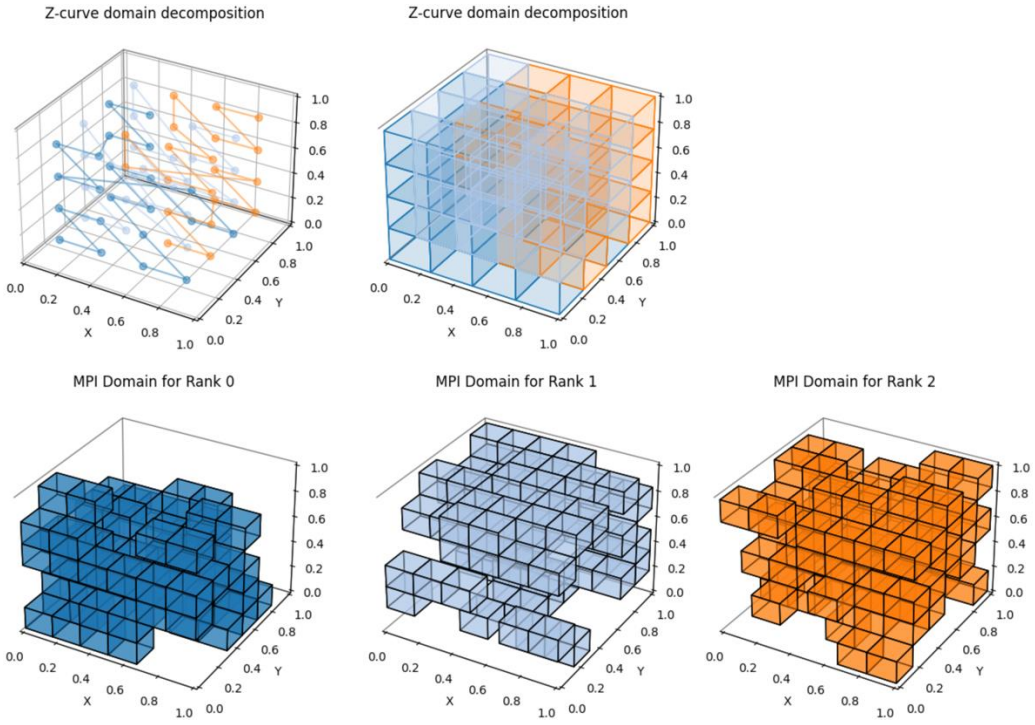
- Calculate properties for all objects.
- OpenMP enabled

Output

- Output fields containing object properties and particle associations in parallel.
- Can gather MPI process to write to single file or many.

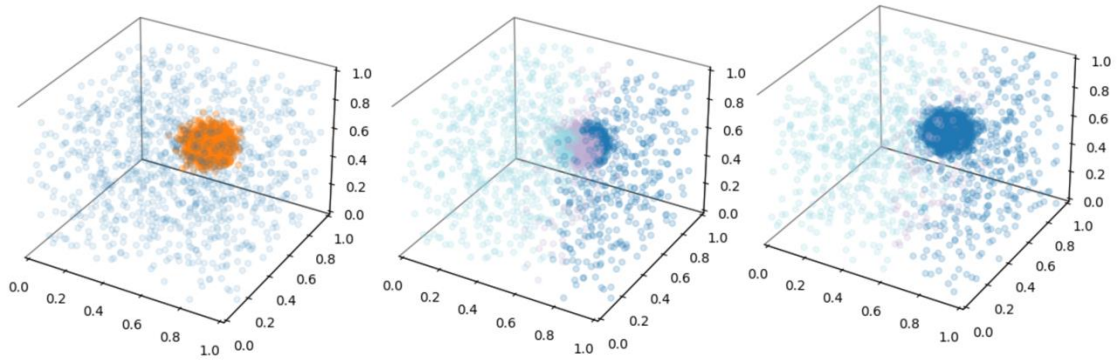
MPI Input & Decomposition

- Input can single file or split between many files
- Number of MPI no relation to number of input files
- Parallel read where number of read tasks need not be MPI_COMM_WORLD.
  - Read ranks are spread across, maximising node spread of reading ranks.
  - Can have tasks just waiting for input.
- Communication dominated by asynchronous point-to-point
- To ensure MPI not flooded, communication sent in chunks.
- Input data highly clustered, load balancing is critical. Not just about memory want to reducing amount of point-to-point communication.
- Use a mesh placed across the input domain and then assigns cells in the mesh to a given MPI rank using a space-filling Z-curve.



MPI Search & Communication

- MPI will local search is particles for those that meet linking criteria. Iterate across MPI domains till no new links are found.
- Now localise clusters to MPI domain with fewest particles.
  - Each rank does not know a priori what other ranks contain desired particles and what other ranks require particles from it.
  - MPI rank able to determine if local particle's search window intersects the volume of a cell that belongs to other MPI ranks using a fast binary tree



Example particle distribution with single cluster (left)
Resulting initial 3 Rank MPI decomposition (middle): Volume fraction = [0.56, 0.1, 04], Particle fraction = [0.36, 0.35, 0.28]. Particles coloured by rank.
Final MPI decomposition after linking where single object now on Rank = 2. Mesh now has 19 cells shared between at least 2 ranks.

- Communication dominated by asynchronous point-to-point.
- To reduce wait times and congestion use communication work queue containing all communicating pairs

```
//NProcs = total number of Process in MPI_COMM_WORLD
std::vector<tuple<int, int>> MPIGenerateCommPairs(unsigned long long
    *send_info)
{
    std::vector<std::tuple<int, int>> commpair;
    for(auto task1 = 0; task1 < NProcs; task1++)
    {
        for(auto task2 = task1+1; task2 < NProcs; task2++)
        {
            if (send_info[task1 * NProcs + task2] == 0 && send_info[task2
                * NProcs + task1] == 0) continue;
            commpair.push_back(make_tuple(task1, task2));
        }
    }
    // ensure rank = 0 doesn't dominate communication pairs by
        randomizing
    unsigned seed = 4322;
    std::shuffle(commpair.begin(), commpair.end(),
        std::default_random_engine(seed));
    return commpair;
}
```

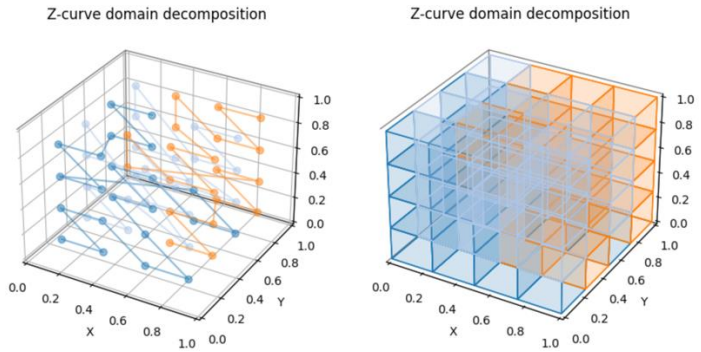
MPI IO

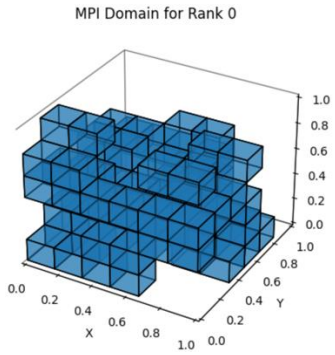
- Use HDF5 library to produce parallel IO file(s).
- Includes a test to check HDF5 Parallel IO performance. Expanding this to determine performance
- Output is aggregated using MPI communicators, one for each file.
  - MPI ranks writing to same file are close in rank, thereby reducing internode communication when coordinating file writes.

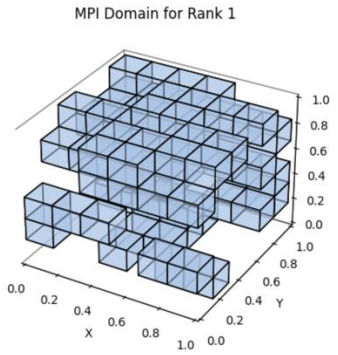
Reading: Example of ranks reading single input file. Here, nodes have 9 cores, 3 nodes, and total number of MPI ranks reading is 5

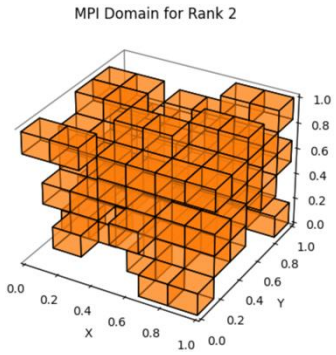
Writing: Example MPI ranks colour coded according to file they write to. Here, nodes have 9 cores, 3 nodes, and total number of MPI ranks per file is maximum of 4.

MPI Future Work





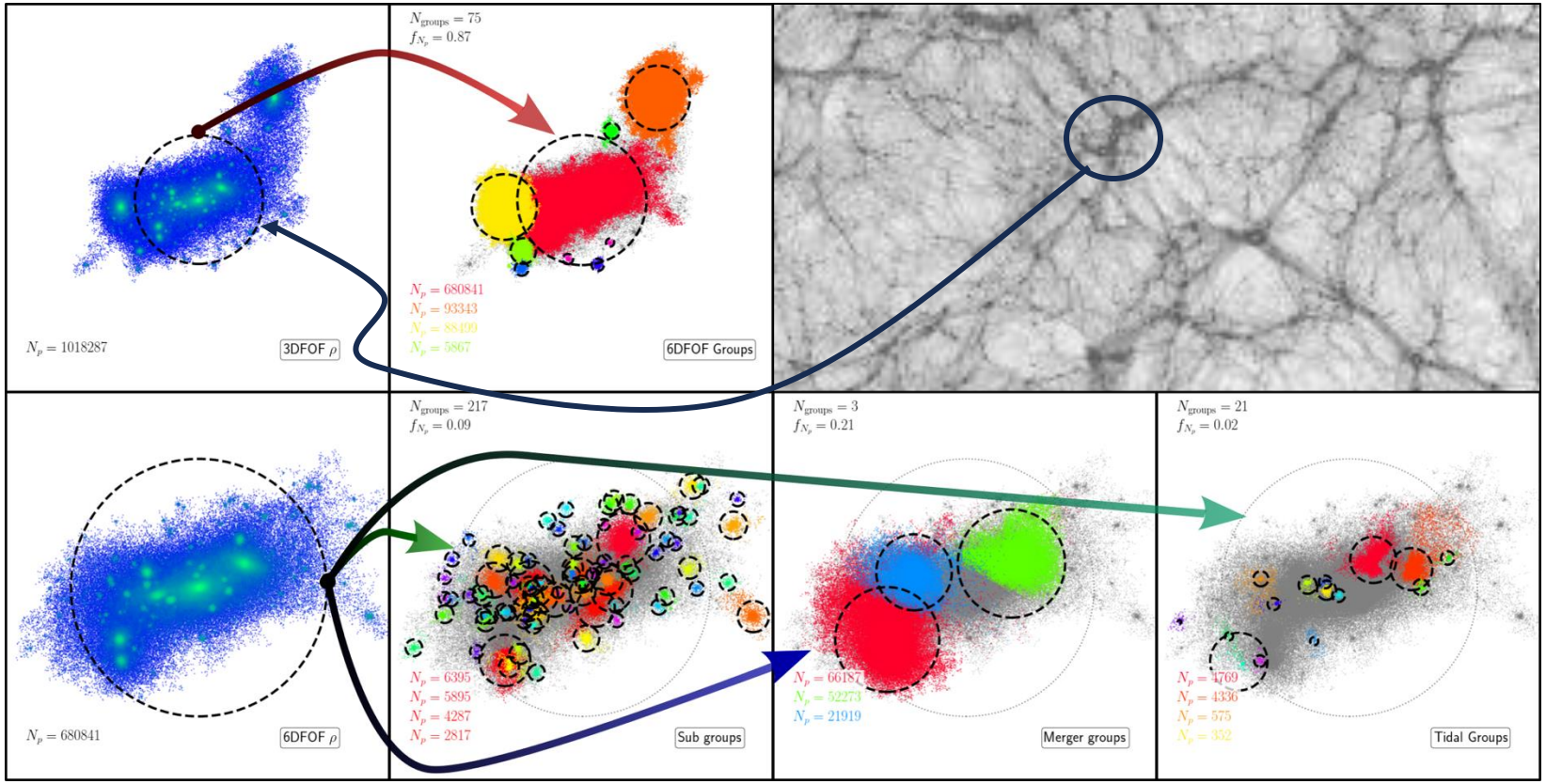




- Compute and IO decomposition does make some assumptions about process thread affinity and topology but this need not be correct.
- Further progress to generate rank ordering based on core affinity, numa regions and node placement (using `numactl`, `lscpu`, `hostname`, etc to gather initial setup)

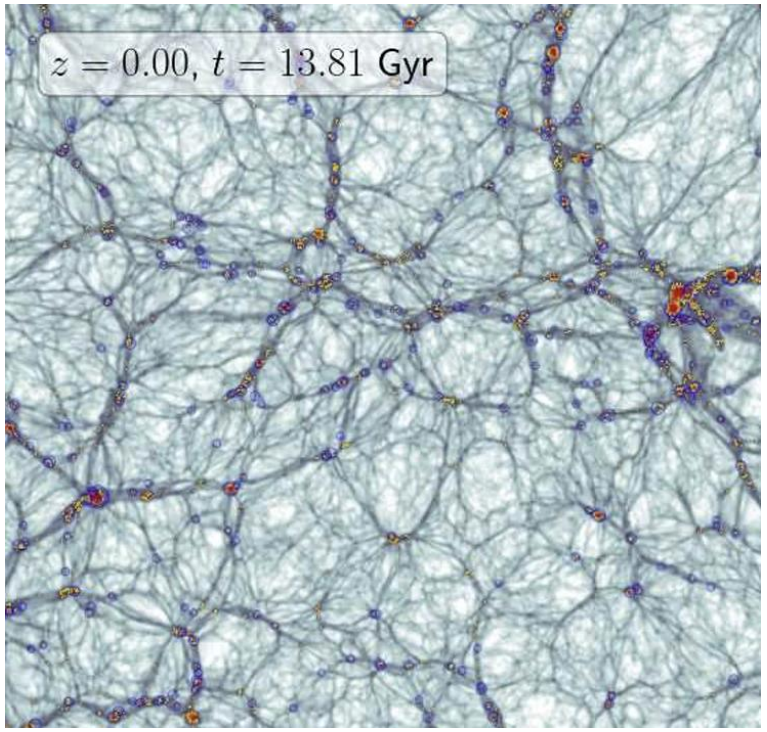
Summary

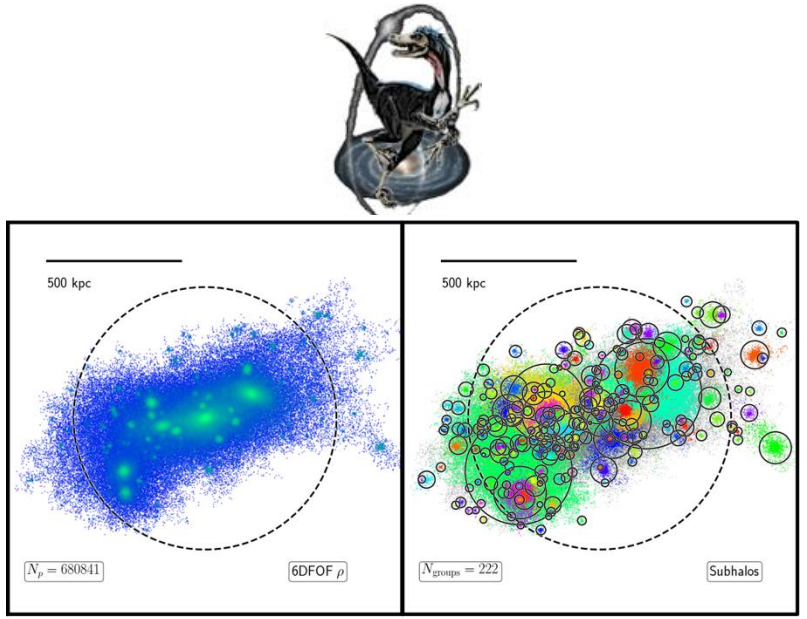
- MPI + OpenMP, C++17, CMAKE build system, Parallel IO via HDF5 capable of running on thousands of cores.
- Adding better OpenMP task parallels, GPU parallelism, MPI decomposition tied to numa topology, ADIOS IO for improved parallel IO and streaming.

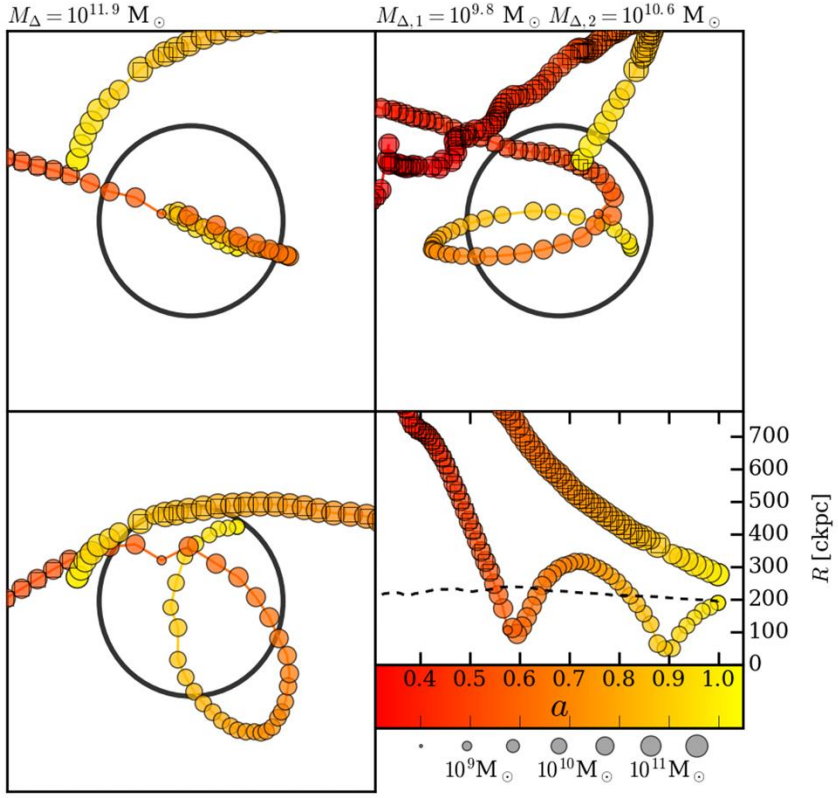


Halo and Substructure Identification from Simulation

Questions?


$z = 0.00,\ t = 13.81$ Gyr


500 kpc
$N_p = 680841$
6DFOF ρ
500 kpc
$N_{\rm groups} = 222$
Subhalos


$M_\Delta = 10^{11.9}$ M$_\odot$
$M_{\Delta,1} = 10^{9.8}$ M$_\odot$ $M_{\Delta,2} = 10^{10.6}$ M$_\odot$
700
600
500
400
300
200
100
0
R [ckpc]
0.4 0.5 0.6 0.7 0.8 0.9 1.0
a
10^9M$_\odot$ 10^{10}M$_\odot$ 10^{11}M$_\odot$