Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

The OpenGADGET3 code for cosmological simulations

- A technical update -

Milena Valentini

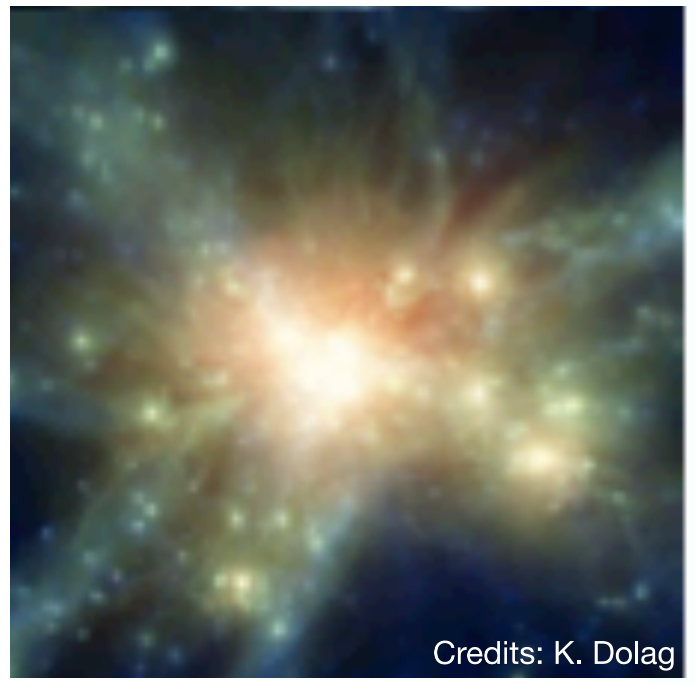
UNIVERSITÀ DEGLI STUDI DI TRIESTE
Dipartimento di Fisica
Dipartimento d'Eccellenza 2023-2027

Spoke 3 WP 1 and 2 monthly meeting, May 13th, 2025

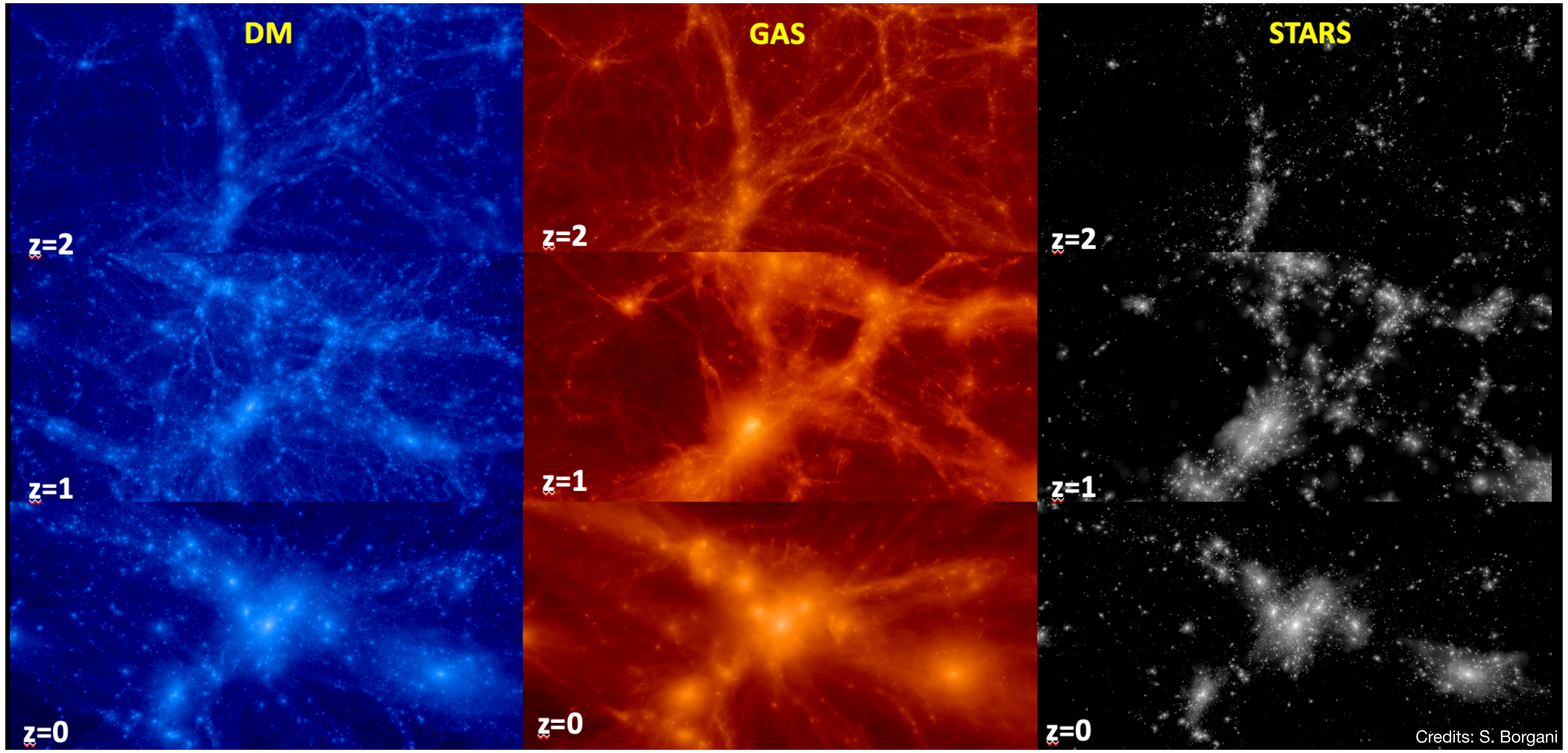
The *Open GADGET3* code: a state-of-the-art code for HPC

Scientific rationale

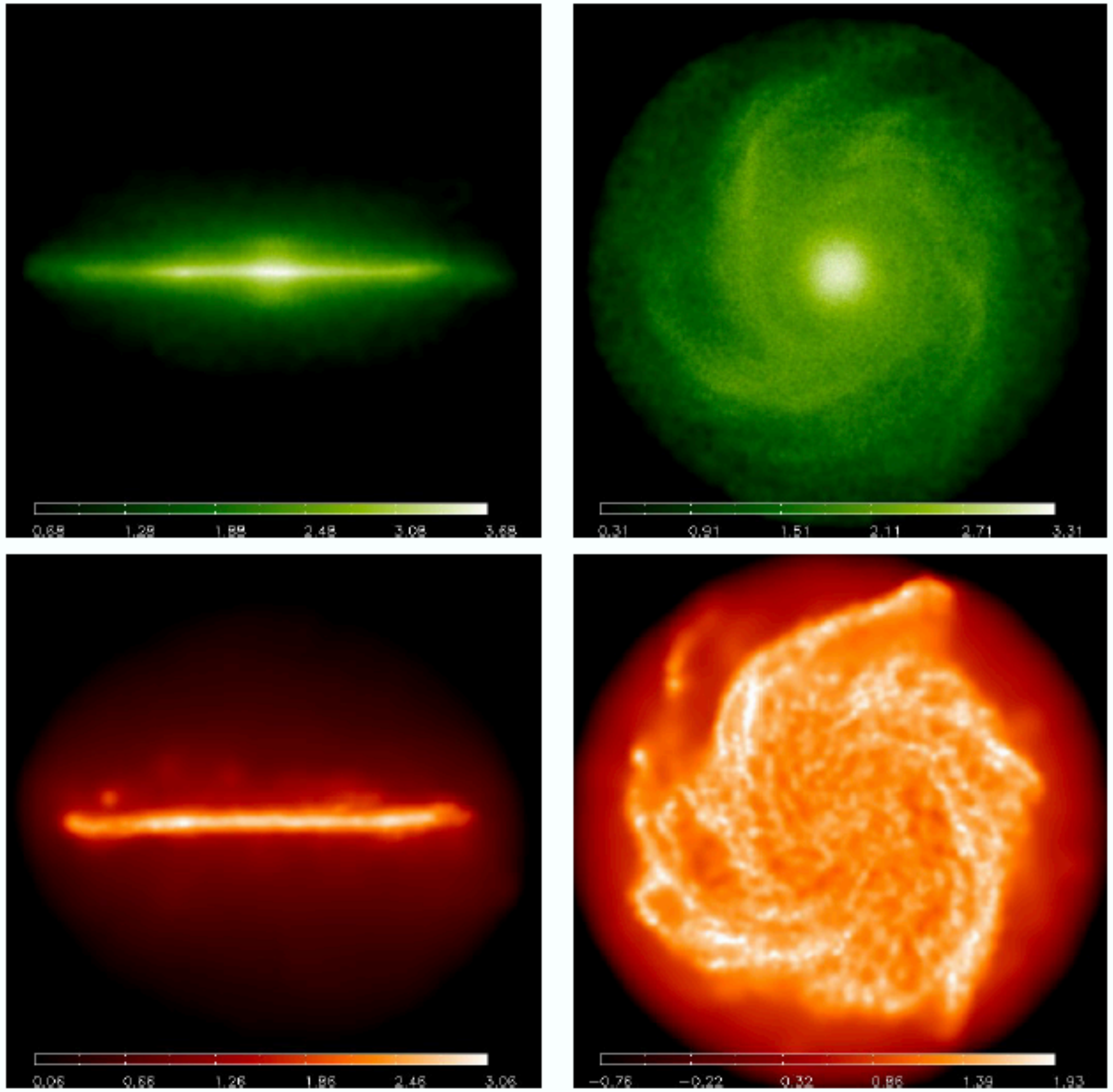
- Numerical cosmology
- Structure formation and evolution



Credits: K. Dolag



Credits: S. Borgani



Technical Objectives, Methodologies and Solutions

USM LMU
Osservatorio Astronomico di Trieste – Istituto Nazionale di Astrofisica
Università degli Studi di Trieste

The OpenGadget3 code

- **TreePM+SPH code**
- **Highly optimised code:** MPI parallelised + OpenMP
- **Two hydro solvers:** improved SPH formalism or MFM
- **Two sub-grid models** (Muppi, and one based on Springel&Hernquist 2003)
- **Several modules for sub-resolution physics:** star formation, stellar feedback, BH accretion and feedback, chemical enrichment, dust evolution, magnetic fields, cosmic rays
- **Runs on CPUs and GPUs**

MUPPI sub-resolution model

star formation
- description of a multi-phase ISM with H_2-based star formation
- thermal, kinetic, and low-metallicity stellar feedback
- improved cooling table interpolation
- stellar evolution and chemical enrichment

BH
- angular-momentum-dependent gas accretion, dynamical friction, spin evolution
- isotropic, thermal AGN feedback + mechanical AGN feedback

dust
- formation and evolution of dust, and dust-assisted cooling

Main tasks within the WP 2 of Spoke 3

Develop Open-GADGET further:
- **including additional physics modules**
- **enhancing code modularity and readability**
- **improving code performance**

Core teams in Trieste and Munich

The *Open* *GADGET3* code: a state-of-the-art code for HPC

USM LMU

The OpenGadget3 code

- **TreePM+SPH code**
- **Highly optimised code:** MPI parallelised + OpenMP
- **Two hydro solvers:** improved SPH formalism or MFM
- **Two sub-grid models** (Muppi, and one based on Springel&Hernquist 2003)
- **Several modules for sub-resolution physics:** star formation, stellar feedback, BH accretion and feedback, chemical enrichment, dust evolution, magnetic fields, cosmic rays
- **Runs on CPUs and GPUs**

Core team in Trieste: S. Borgani, L. Tornatore, G. Murante, M. Valentini, T. Castro, P. Monaco, G. Taffoni, A. Damiano, G. Granato, D. Goz, P. Barai, M. Gitton-R., A. Saro, M. Viel

and collaboration in Munich led by K. Dolag

Main tasks within the WP 2 of Spoke 3

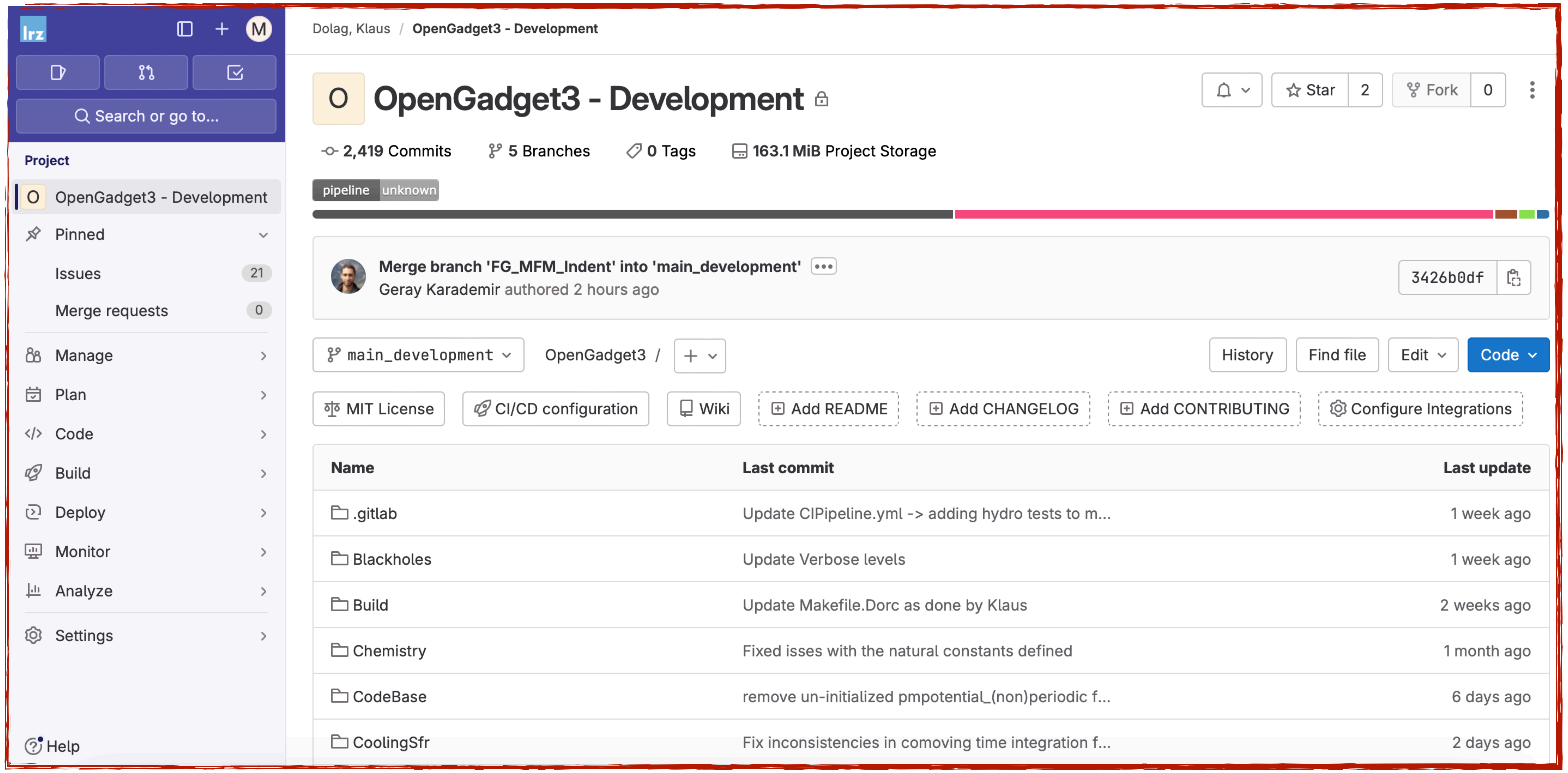
Develop Open-GADGET further:

- including additional physics modules
- enhancing code modularity and readability
- improving code performance

Core teams in Trieste and Munich

The *OpenGADGET3* code: a state-of-the-art code for HPC

- Our code is on GitLab
- We defined a more accurate working strategy
- Quite large (> 30 people from different institutes) user community



- **Re-structuring** of the code **(modularity)**
- **Cleaning** the code **and documenting** its status

Ongoing: Performance profiling and benchmarking

Compared scaling (CPU) of pre-Spoke3 code VS current version on LUMI

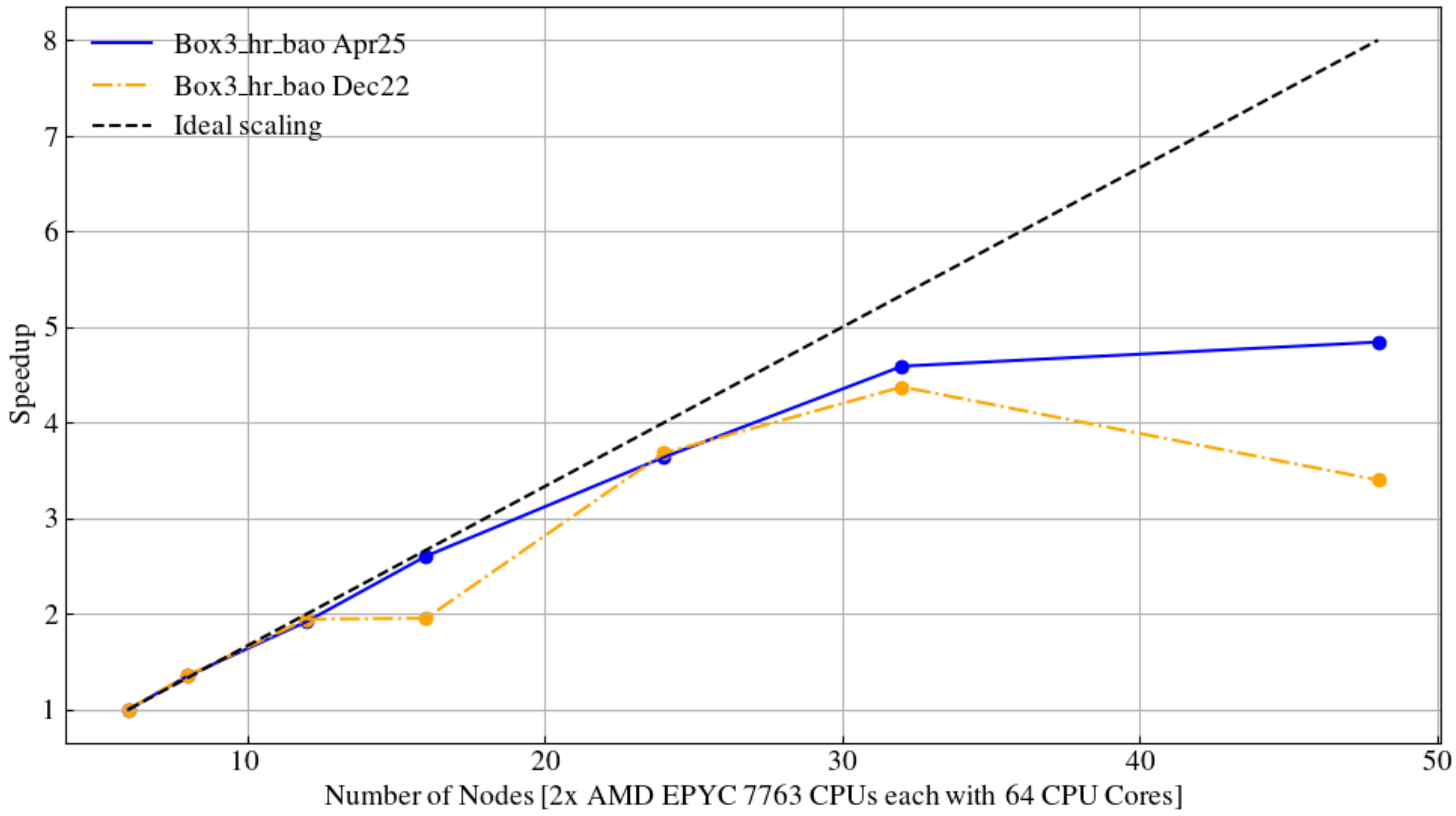
Full-physics run, starting from the Magneticum ICs (http://www.magneticum.org/simulations.html)

Similar scaling properties with slightly better results by the new version

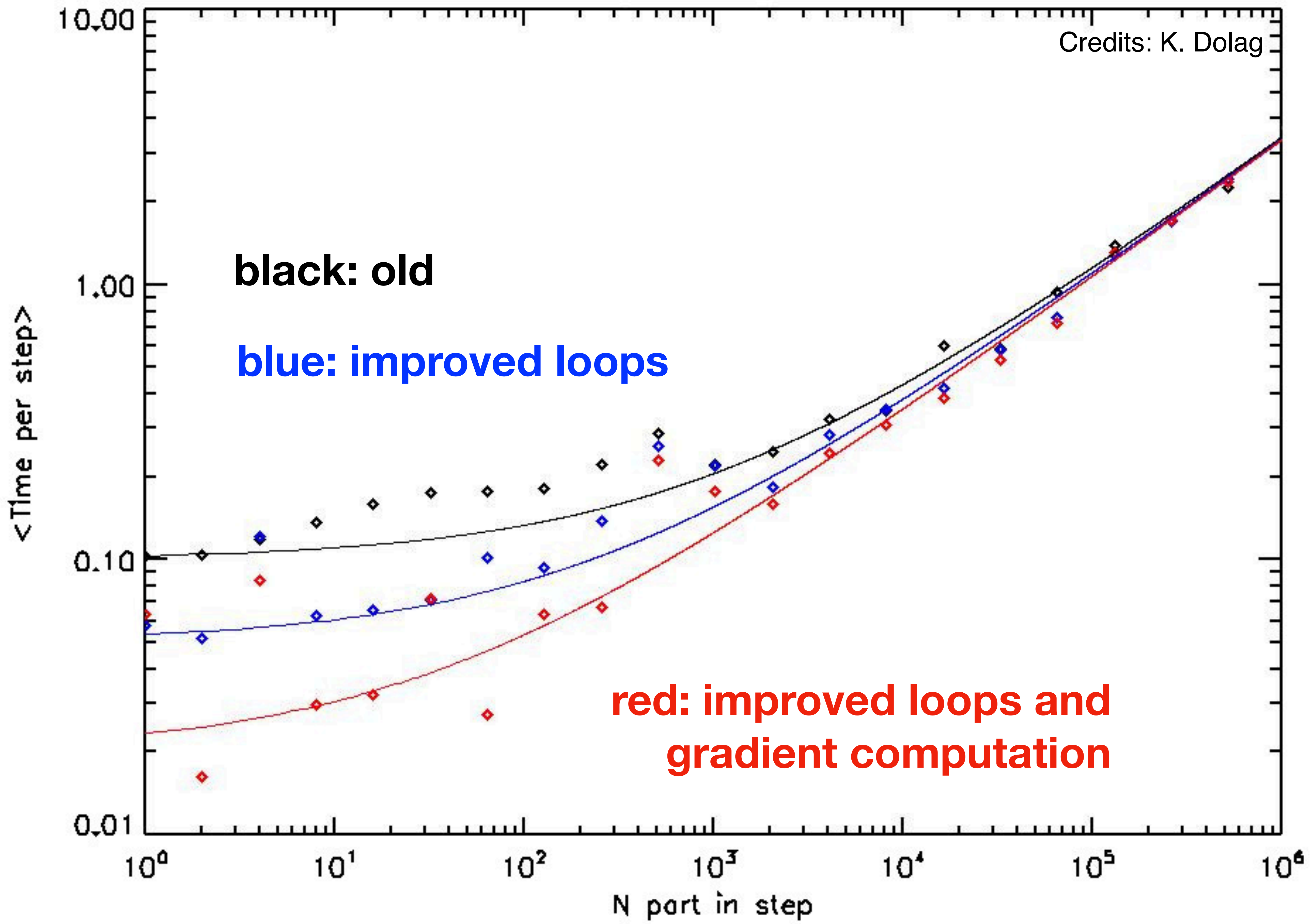
Currently trying to redo this test with an evolved simulation (reading in ~10 yr old file is a challenge)

Strong scaling on LUMI

Credits: G. Karademir and L. Tornatore



Ongoing: Performance profiling and benchmarking



Comparison of the required time per time step at different numbers of particles in each time bin.

CPU optimization

Loop restructuring leads to a 2x performance in timesteps with a small # of particles (blue VS black curves)

Updates on the gradient computation and more precise memory allocation further increase the performance (red VS blue)

In total, these improvements speed up the calculation of the smallest time bins by up a factor of ~5 (red VS black).

Ongoing: Assessing scalability, targeting performance issues

1. GPU scalability

OpenGadget has most of the modules running on GPUs (thanks to A. Ragagnin, L. Tornatore et al.).

We are assessing in detail the scalability of this implementation in order to highlight the blocking factors, mitigate their impact or turn to new strategies with greater parallelism

2. Performance issues

Detailed profiling with the assistance of POP and SPACE Centers of Excellence

Coordinator of the work: L. Tornatore

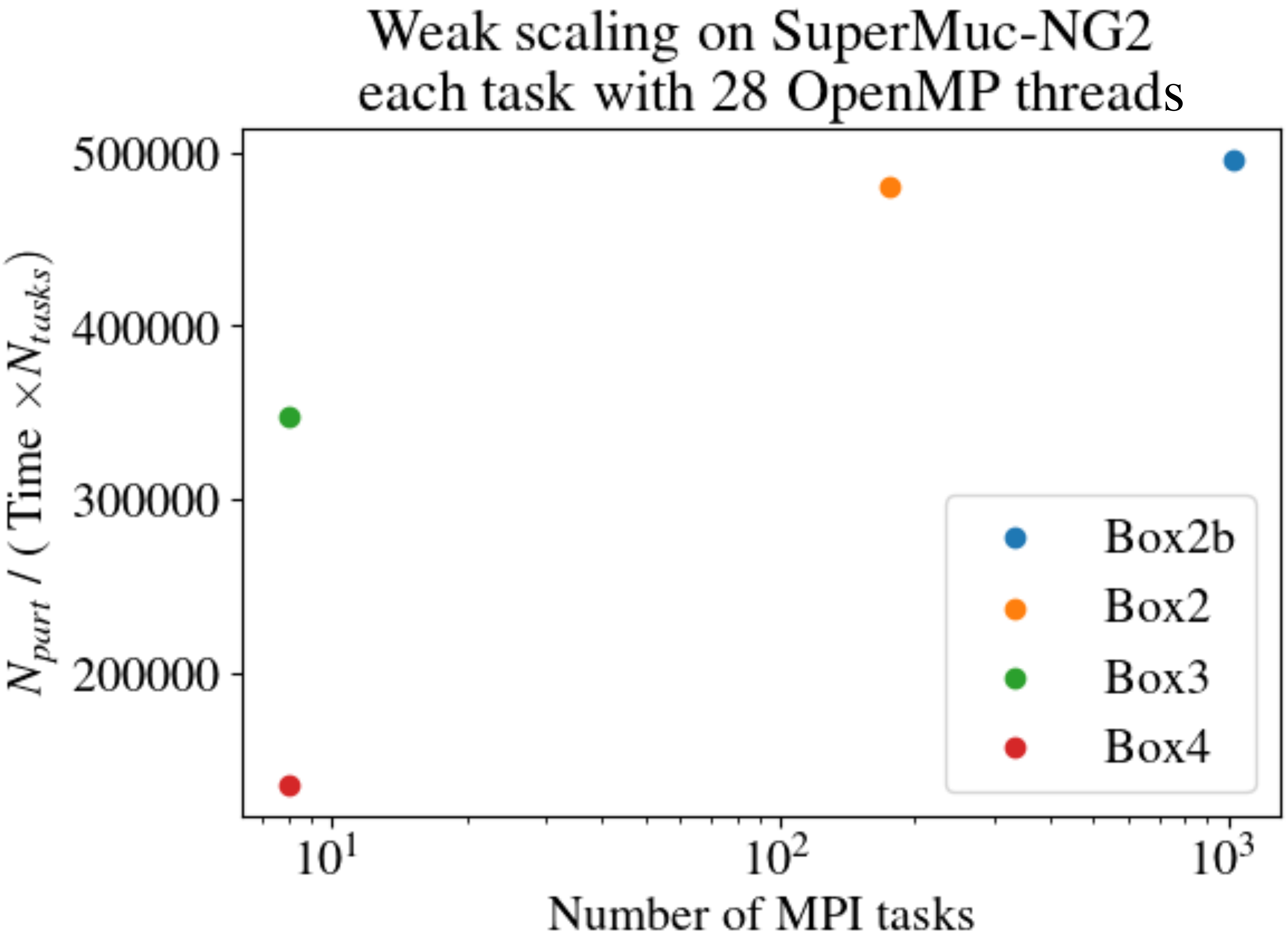
also in collaboration with: exact lab, SPACE CENTRE OF EXCELLENCE FOR HPC ASTROPHYSICAL APPLICATIONS and CINECA

Ongoing: Assessing scalability, targeting performance issues

GPU offloading status

OpenACC current status:
— gravity is working reasonably well
— find_hsml module is ported as well
— hydro module is under active development

+ OpenMP:
— successful offloading of gravity module based on current OpenMP implementation
— it provides similar (~3x) speed-up as the OpenACC implementation

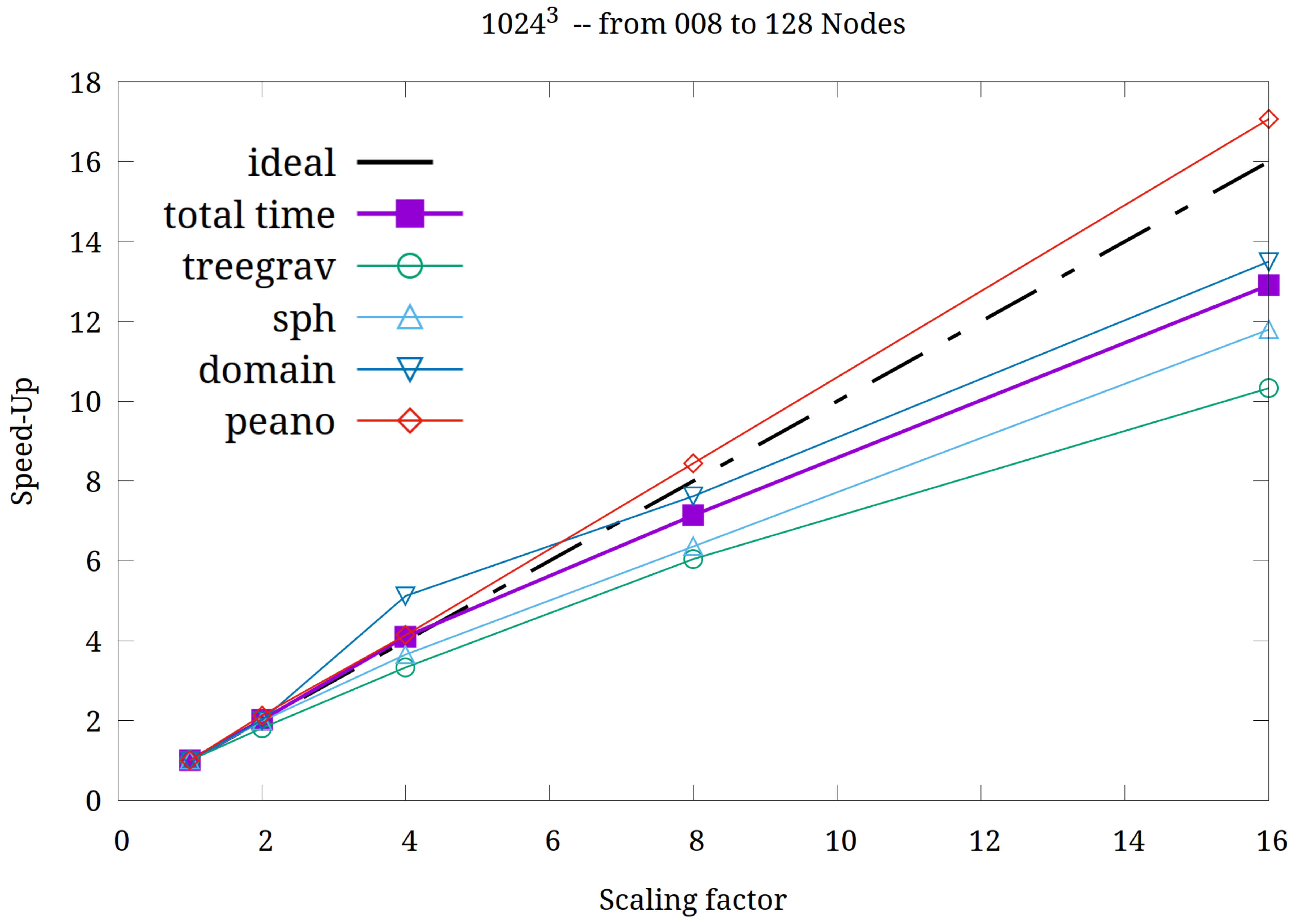


(See box sizes at http://www.magneticum.org/simulations.html)

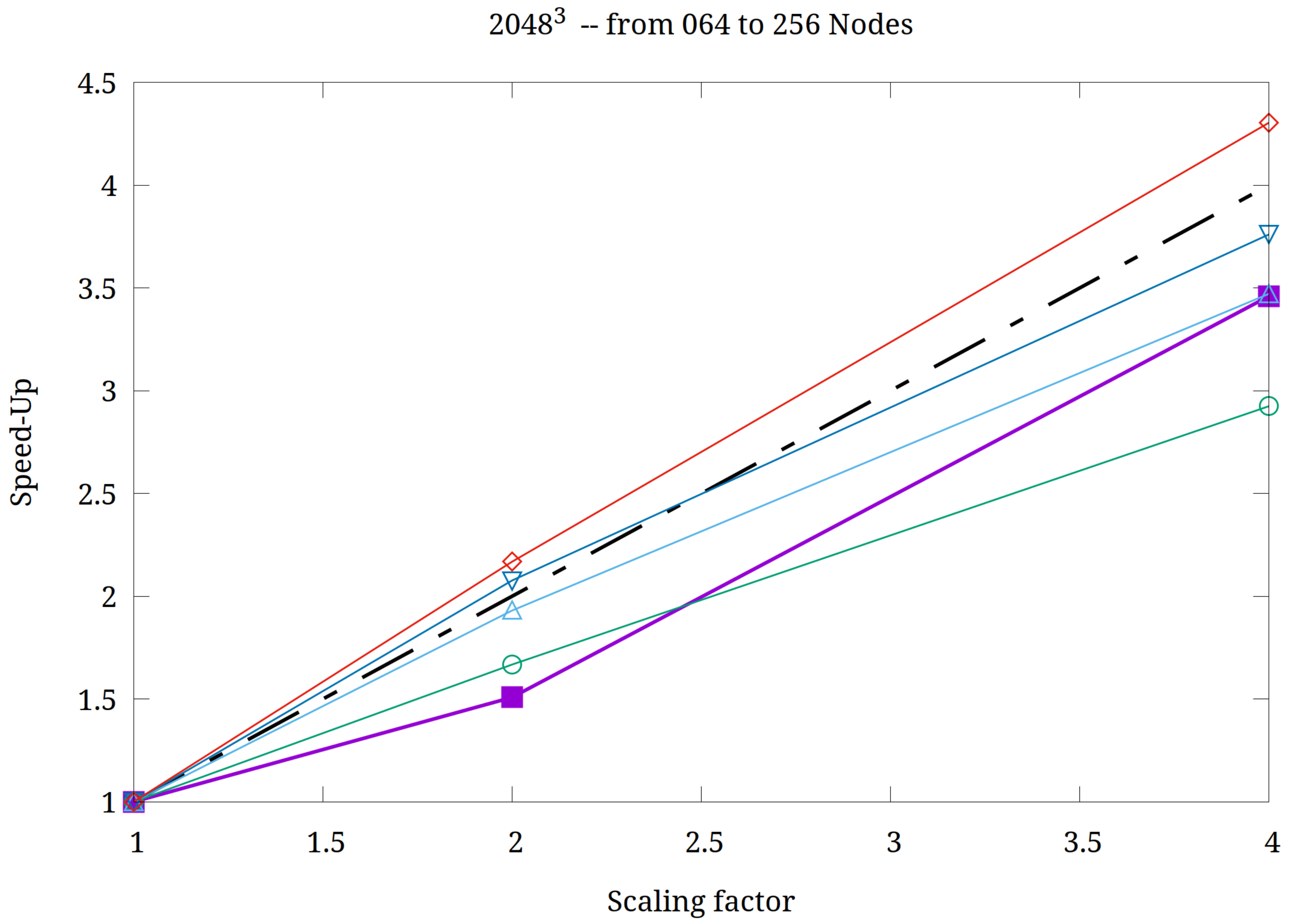
Ongoing: Assessing scalability, targeting performance issues

1) GPU scalability: Strong scaling speed-up

2×1024^3, 120 Mpc, up to **512 GPUs**



2×2048^3, 240 Mpc, up to **1024 GPUs**



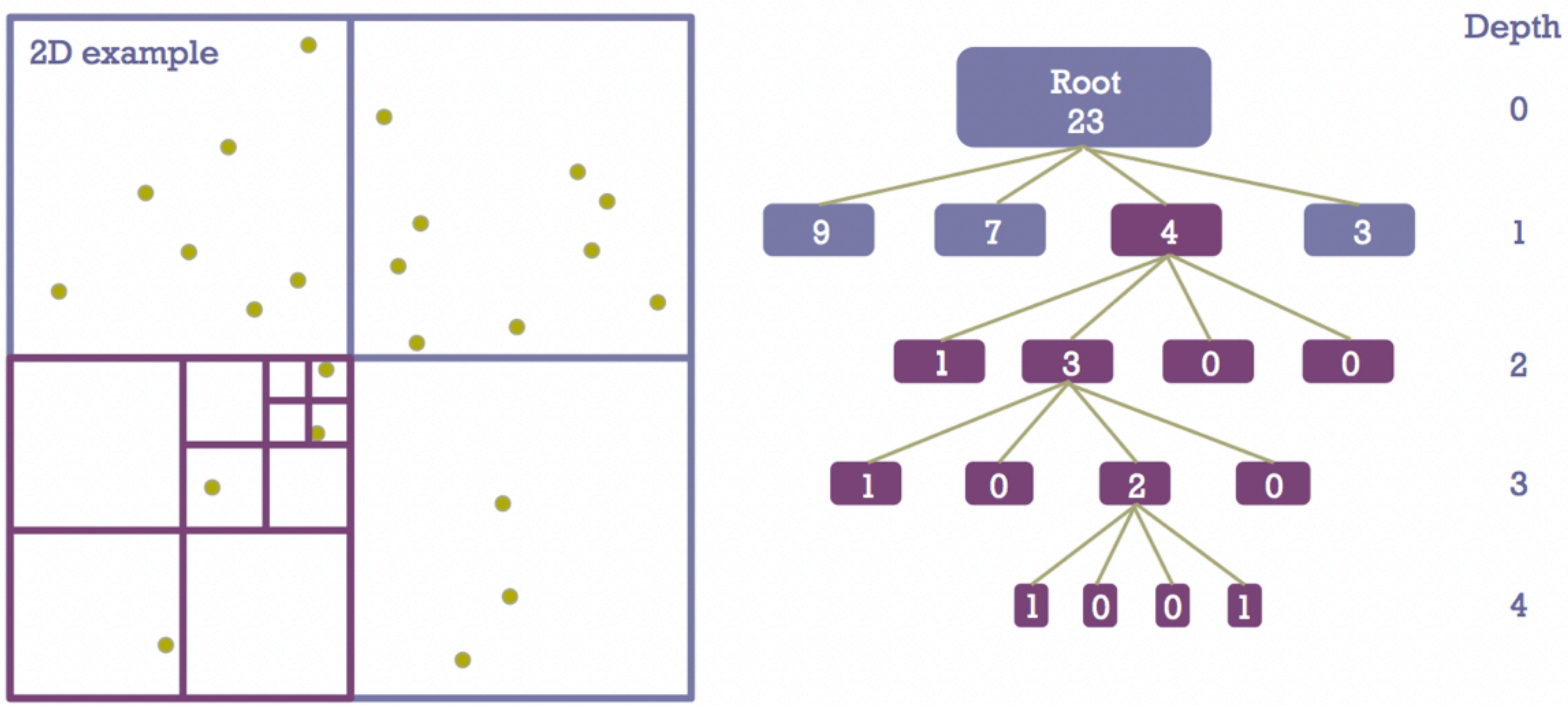
Running a suite of tests, we are assessing in detail the scalability, from 4 nodes up to the entire Leonardo

Ongoing: Assessing scalability, targeting performance issues

GPU scalability: more in detail

The gravity tree has some performance issues:

- Tree Walk → Barnes&Hut is not GPU-friendly



- **Tree algorithm** (e.g. Barnes & Hut, 1986) —> hierarchical multipole method
- Key idea: **arrange particles in groups**, according to their distance from the considered particle
- Particle **grouping** is **by means of a tree structure**.
- The tree consists in a recursive **slicing of the computational domain into** sub-domains (**nodes**), until a sub-domain which contains only one particle or none is reached.
- The force addition from each group of particles is supplied by its **multipole expansion**.
- An **opening angle** (encoding the algorithm precision) decides whether the force contribution through the multipole expansion can be computed or if we have to **continue walking along the tree** until nodes small and distant enough to provide accurate force contribution through multipole expansions are reached.

Ongoing: Assessing scalability, targeting performance issues

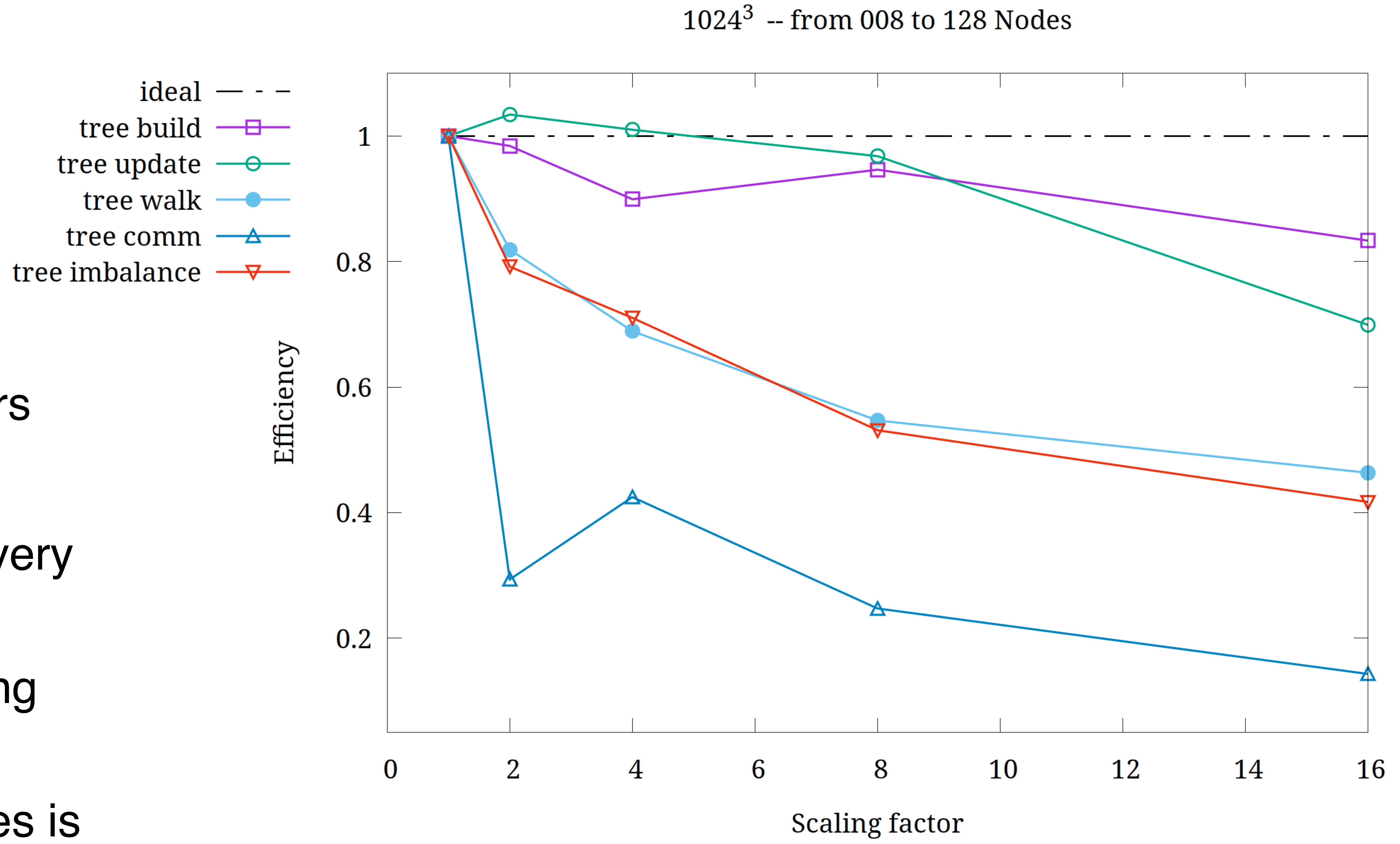
GPU scalability: more in detail

The gravity tree has some performance issues:

- Tree Walk → Barnes&Hut is not GPU-friendly
- Communication

The current GPU offloading of Barnes&Hut in OG3 suffers from three main issues:

- **Thread divergence**: because the walk is unique per every particle
- Non-coalesced **memory access**: as there is no mapping between particles in memory and in 3D space
- **Memory and computation inefficiency**: opening nodes is if-based and the nodes are sparse in memory



Ongoing: Assessing scalability, targeting performance issues

GPU scalability: more in detail

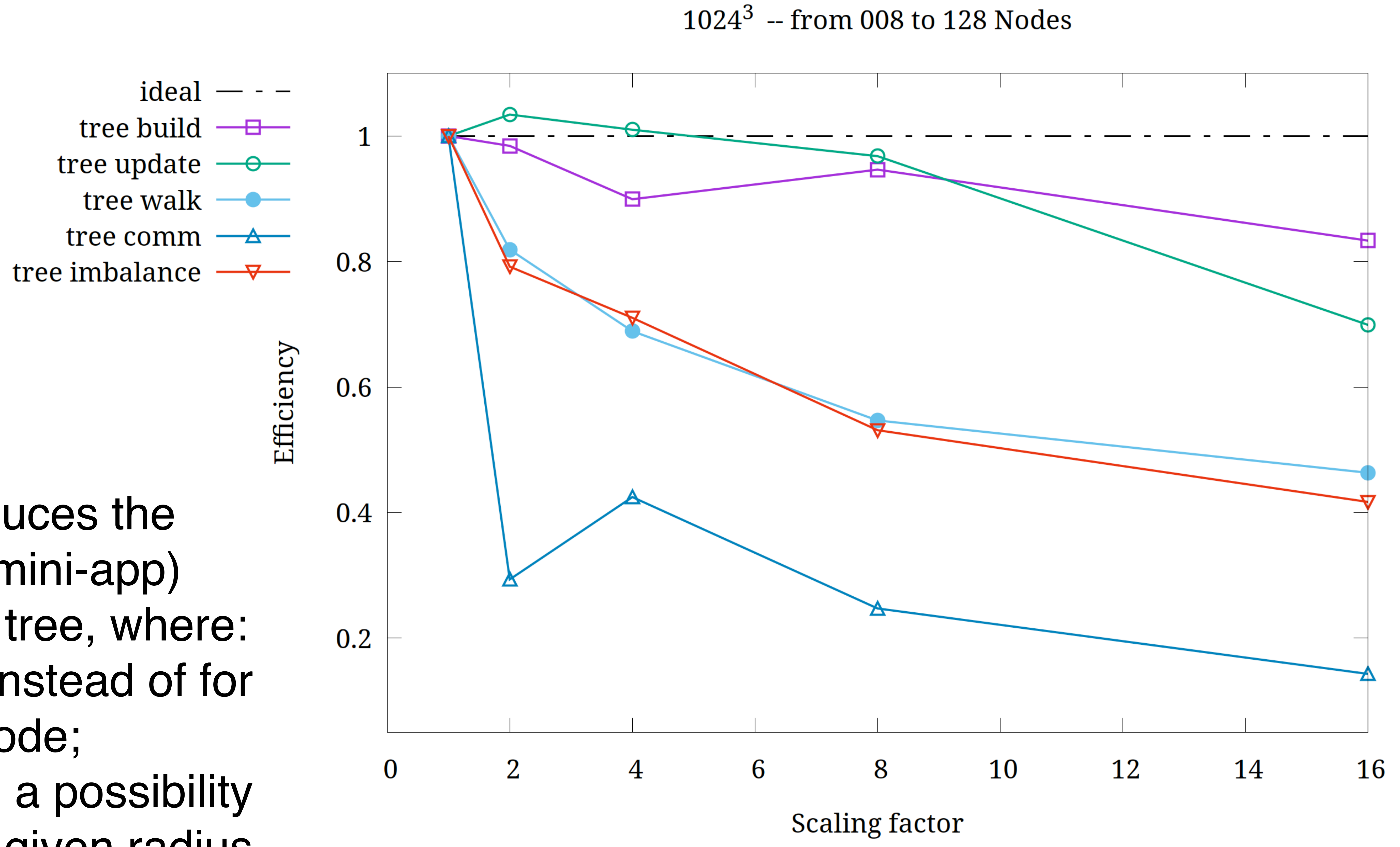
The gravity tree has some performance issues:

- Tree Walk → Barnes&Hut is not GPU-friendly
- Communication

We are working on a different implementation:

We have extracted a kernel of the code which reproduces the conditions under which gravity is computed in OG3 (mini-app) and which will feature the new implementation of the tree, where:

1. the walk is done for a bunch of particles all together instead of for every single particle, by grouping particles per tree node;
2. the Barnes and Hut scheme is not adopted anymore: a possibility is to opt for a direct computation of the force within a given radius, to avoid to check whether nodes have to be opened and the tree walked further.



Ongoing: The new GPU offloading

Two main strategies:

1. **building** a tree with more than one particle per leaf, and adopt as for the Barnes&Hut walk the center of mass of the leaf to which the particle belongs.
2. Introducing a partitioning of particles such as particles belonging to the same “boxleaf” are also in the same memory segment.

Results:

— assigning each leaf to a different OpenACC instruction makes threads within the same directives follow the same Barnes&Hut walk (reduced thread-divergence),

— memory access are on data that are close in memory (coalesced memory access).

Ongoing: The new GPU offloading

Two main strategies:

1. building a tree with more than one particle per leaf, and adopt as for the Barnes&Hut walk the center of mass of the leaf to which the particle belongs.
2. Introducing a partitioning of particles such as particles belonging to the same "boxleaf" are also in the same memory segment.

More in detail on the local tree construction:
NOT geometric anymore, but completely based on the Peano-Hilbert space-filling curve, in particular:

- each node of the tree corresponds to a "cube" of the Peano-Hilbert curve;
- nodes indexing is done in Peano-Hilbert order;
- particles are assigned to leaves according to their Peano-Hilbert keys (i.e. particles belonging to the same leaf are stored contiguously in memory).

The new GPU offloading (in short):

1. refactoring of the Barnes&Hut algorithm towards an enhanced GPU effectiveness;
2. a new tree construction, branchless and extremely GPU-friendly.

Ongoing: Assessing scalability, targeting performance issues

	2048(512x4)[1]	4096(1024x4)[2]	8192(2048x4)[3]
Global efficiency	78.23	73.34	49.75
-- Parallel efficiency	78.23	73.74	55.54
-- Load balance	85.89	83.68	71.74
-- Communication efficiency	91.09	88.11	77.41
-- Computation scalability	100.00	99.47	89.59
-- IPC scalability	100.00	100.87	102.63
-- Instruction scalability	100.00	98.69	86.81
-- Frequency scalability	100.00	99.91	100.56

Percentage(%): 0–100

	2048(512x4)[1]	4096(1024x4)[2]	8192(2048x4)[3]
-- Hybrid Parallel efficiency	78.23	73.74	55.54
-- MPI Parallel efficiency	80.33	76.67	68.64
-- MPI Load balance	88.14	86.91	88.59
-- MPI Communication efficiency	91.14	88.22	77.48
-- Serialization efficiency			
-- Transfer efficiency			
-- OpenMP Parallel efficiency	97.39	96.17	80.91
-- OpenMP Load Balance	97.45	96.29	80.98
-- OpenMP Communication efficiency	99.94	99.88	99.91

Percentage(%): 0–100

2) Performance issues: vectorization

With the assistance of the POP CoE, and within the SPACE CoE, we are profiling in details the code's behaviour.

The results are summarized in tables, as sketched in the figure on the left
(here the example is for the gravity-tree; rows are different metrics, columns refer to the total number of threads)

from which some key indicators can be collected

Number of processes	2048	4096	8192
Elapsed time (sec)	47.714394	25.446344	18.755917
Efficiency	1.0	0.937549	0.635991
Speedup	1.0	1.875098	2.543965
Average IPC	0.961925	0.970340	0.987195
Average frequency (GHz)	3.190112	3.187294	3.207830

Ongoing: Assessing scalability, targeting performance issues

	2048(512x4)[1]	4096(1024x4)[2]	8192(2048x4)[3]
Global efficiency	78.23	73.34	49.75
-- Parallel efficiency	78.23	73.74	55.54
-- Load balance	85.89	83.68	71.74
-- Communication efficiency	91.09	88.11	77.41
-- Computation scalability	100.00	99.47	89.59
-- IPC scalability	100.00	100.87	102.63
-- Instruction scalability	100.00	98.69	86.81
-- Frequency scalability	100.00	99.91	100.56

	2048(512x4)[1]	4096(1024x4)[2]	8192(2048x4)[3]
-- Hybrid Parallel efficiency	78.23	73.74	55.54
-- MPI Parallel efficiency	80.33	76.67	68.64
-- MPI Load balance	88.14	86.91	88.59
-- MPI Communication efficiency	91.14	88.22	77.48
-- Serialization efficiency			
-- Transfer efficiency			
-- OpenMP Parallel efficiency	97.39	96.17	80.91
-- OpenMP Load Balance	97.45	96.29	80.98
-- OpenMP Communication efficiency	99.94	99.88	99.91

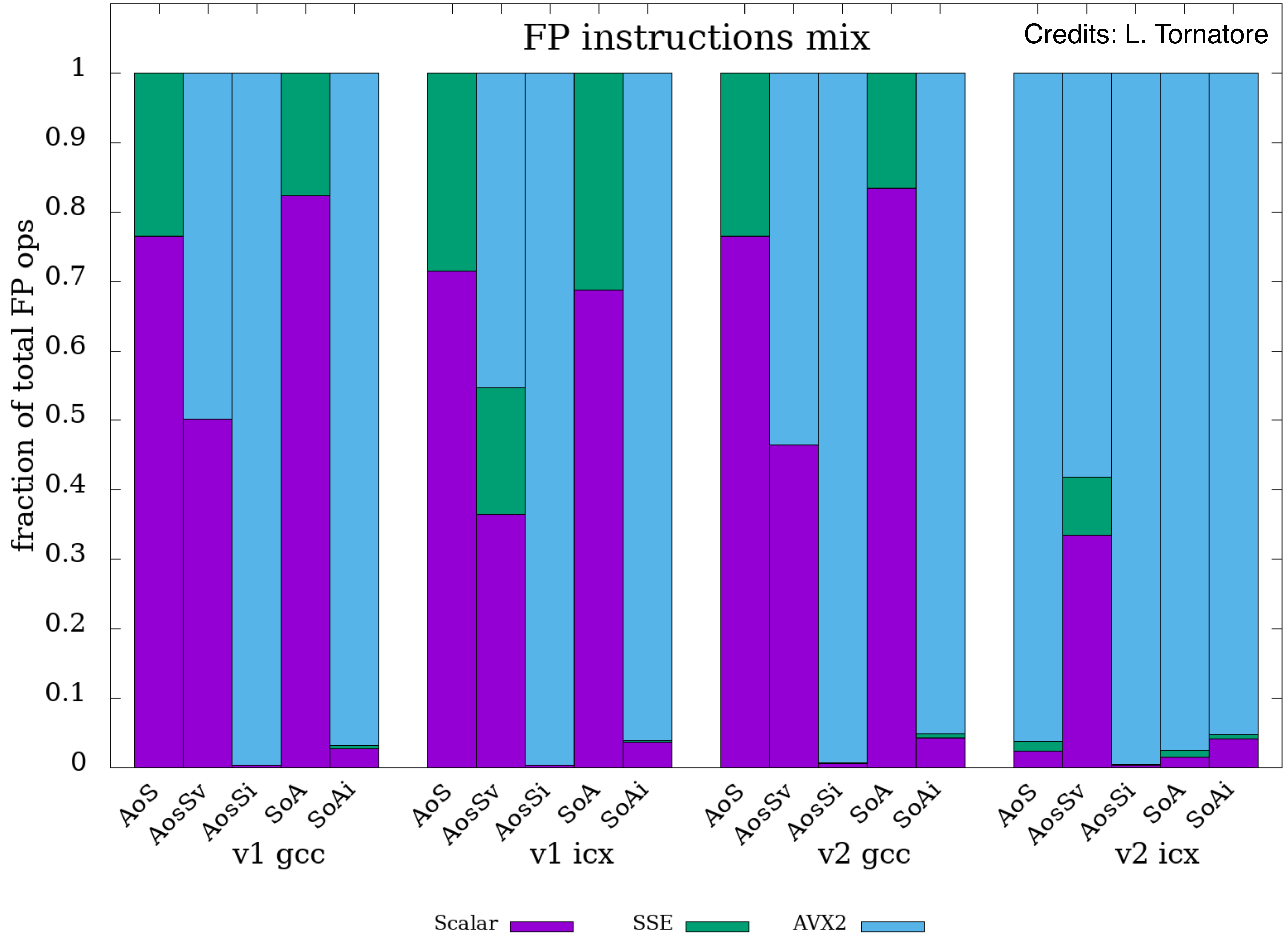
2) Performance issues: vectorization

The low IPC (Instructions Per Cycle), although constant with decreasing workload, indicates that the computational efficiency is not high.
Further inspection returned that in particular the **vectorization ratio is very small** (~10%) and limited to 128bits registers

→ the main target is to re-formulate the data structures that now consists in Arrays of (large)Structures

Number of processes	2048	4096	8192
Elapsed time (sec)	47.714394	25.446344	18.755917
Efficiency	1.0	0.937549	0.635991
Speedup	1.0	1.875098	2.543965
Average IPC	0.961925	0.970340	0.987195
Average frequency (GHz)	3.190112	3.187294	3.207830

Ongoing: Assessing scalability, targeting performance issues



Vectorization ratio achieved on average (= fraction of vector floating point (FP) instructions issued to the total number of FP instructions) under different assumptions.

2) Performance issues: vectorization

We have tested the effect of different data layout on the achievable vectorization in a loop that reproduces the N-Body pattern, assuming that:

- A fraction of particle is active
- Every active particle interacts with its neighbours
- Neighbours are not close in memory

We experimented AoS, AosS and SoA with some carefully crafted loops to

- enhance auto-vectorization by the compiler (AoS, SoA)
- test compilers vector extensions (AosSv)
- explicitly use vector intrinsics (AosSi, SoAi)

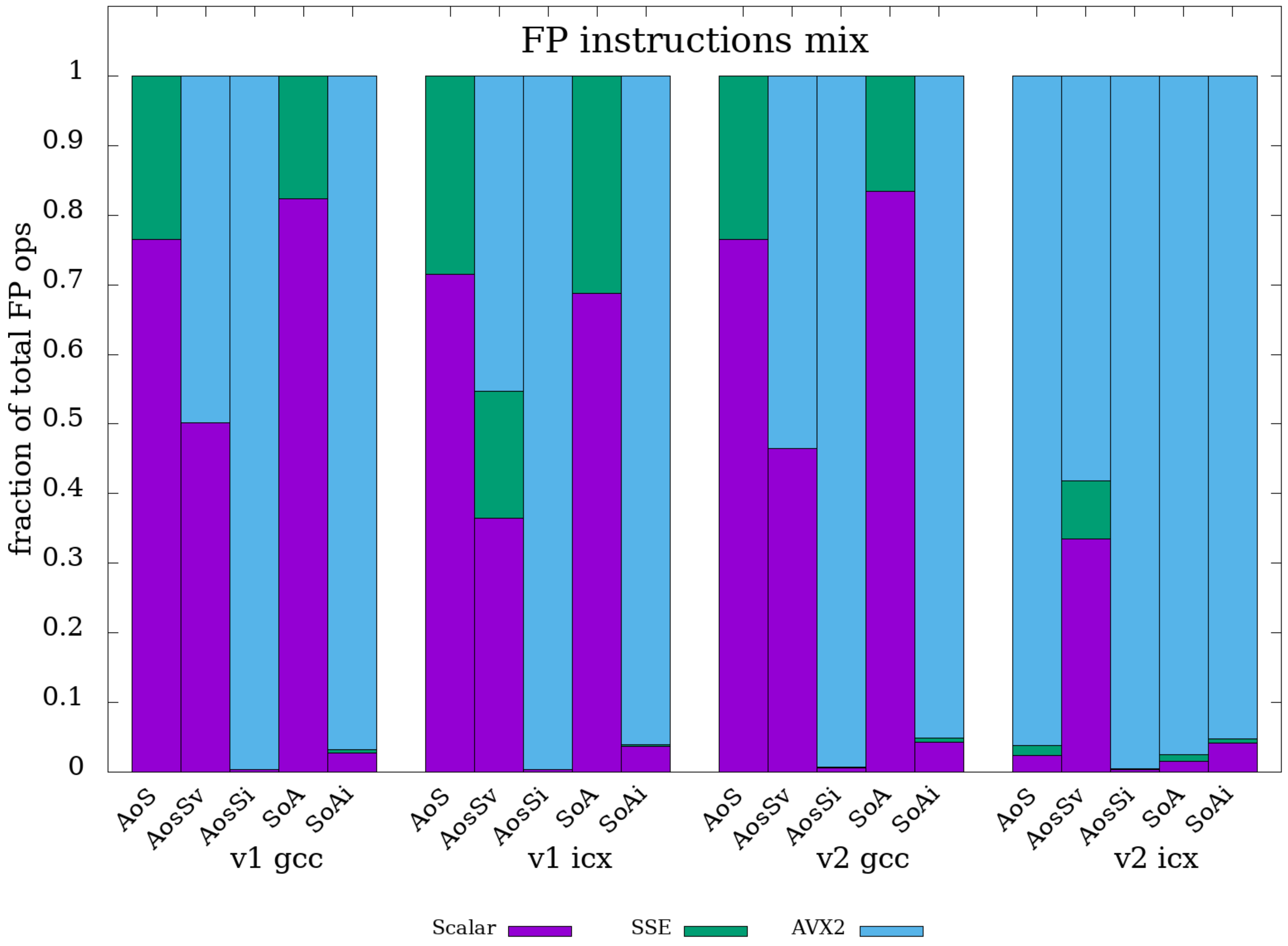
Also, we have tested the effect of enhancing the memory contiguity (**v1** VS **v2**) on different compilers (**gnu** VS **intel**)

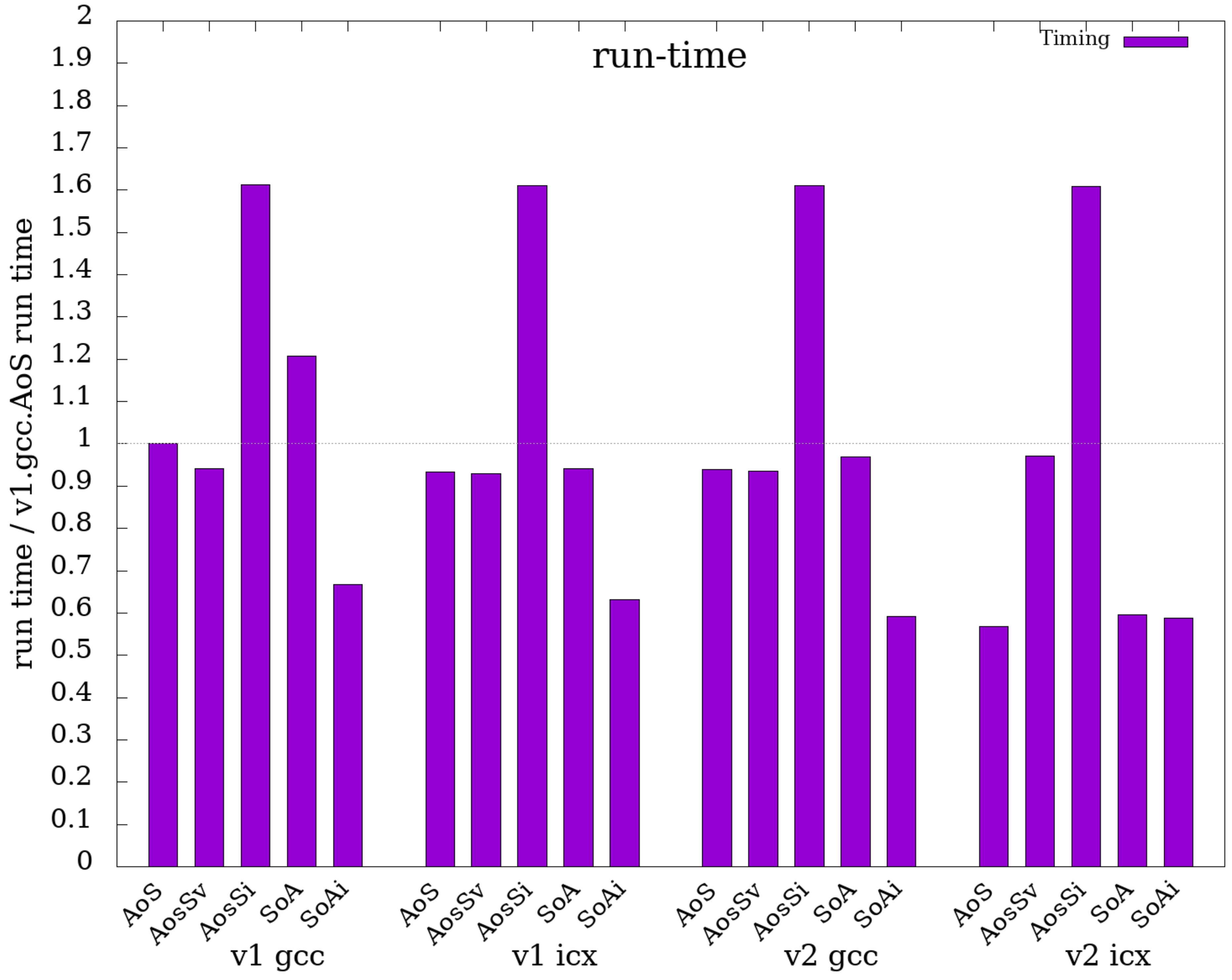
Cons of vector instructions: every instruction requires more CPU cycles, the CPU frequency is generally decreased for an intense vector burst

Ongoing: Assessing scalability, targeting performance issues

2) Performance issues: vectorization

Credits: L. Tornatore





Results from LEONARDO DCGP, obtained by measuring performance counters via PAPI

1. A large vectorization fraction with the wrong data layout is not an advantage (e.g. AosSv) because a larger # of instructions is issued and the cpu frequency is decreased

2. Smaller structures offer ~10% of gain in terms of run-time (e.g. AosSv)

3. Memory contiguity seems to be the most promising trick (go from v1 to v2), especially if the compiler is good in spotting opportunities (see icx vs gcc in v2.AoS)

Next Steps and Expected Results

Ongoing:

- As for now, the new tree is built on CPUs and then moved to GPUs, where the Barnes&Hut walk is performed. The used algorithm is mostly recursive, however we are working on a non-recursive version for tree construction on GPUs.

- Validation of the new gravity solver is ongoing by comparison with current OG3 implementation.

- Additional modification to build a Tree which is suited for GPUs (similar to the Cornerstone octree, by Keller+ 2023). Here, particles are subdivided in boxes: particles in the same box interact via direct sum, particles on different boxes interact using the Barnes&Hut algorithm over the Cornerstone tree.

- Working on topology awareness: capability of the code to explore the NUMA topology of a machine.

So far, results in line with timescale, milestones and KPIs identified.