Individual time-stepping in cosmological simulations: A challenge for strong scaling and domain decomposition algorithms

Matthieu Schaller

Leiden Observatory, Netherlands

with

Stefan Arridge, Josh Borrow, Alexei Borissov (DiRAC), Richard Bower, Aidan Chalk (Hartree Centre), Peter Draper, Pascal Elahi (Perth), Pedro Gonnet (Google), Loic Hausamman (EPFL), Yves Revaz (EPFL), Bert Vandenbroucke (St. Andrews), James Willis

This work started as a collaboration between two departments at Durham University (UK):

- The Institute for Computational Cosmology,
- The School of Engineering and Computing Sciences,

with contributions from the astronomy groups at the university of St-Andrews (UK), Perth (Australia), Leiden (Netherlands), Lausanne (Switzerland) as well as the DiRAC and CSCS software teams.

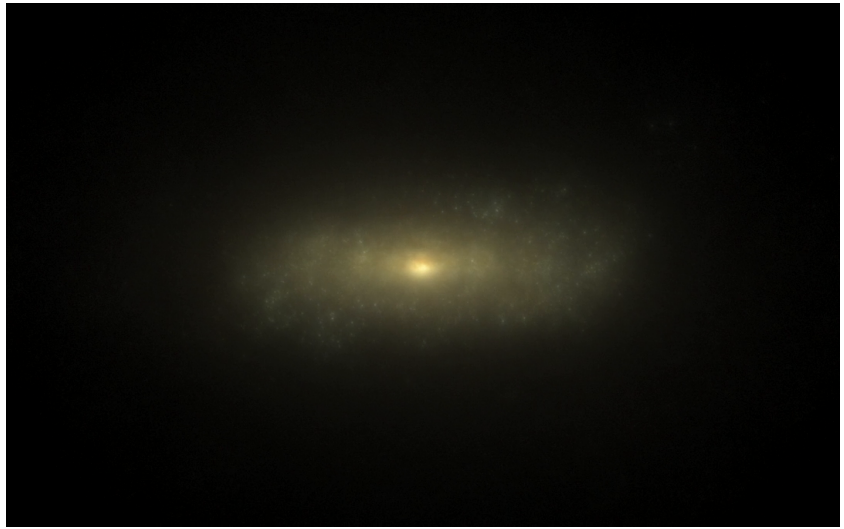
This research is partly funded by an Intel IPCC since January 2015.

Our software is part of the benchmarking challenge for new DiRAC (UK) systems.

What we do and how we do it

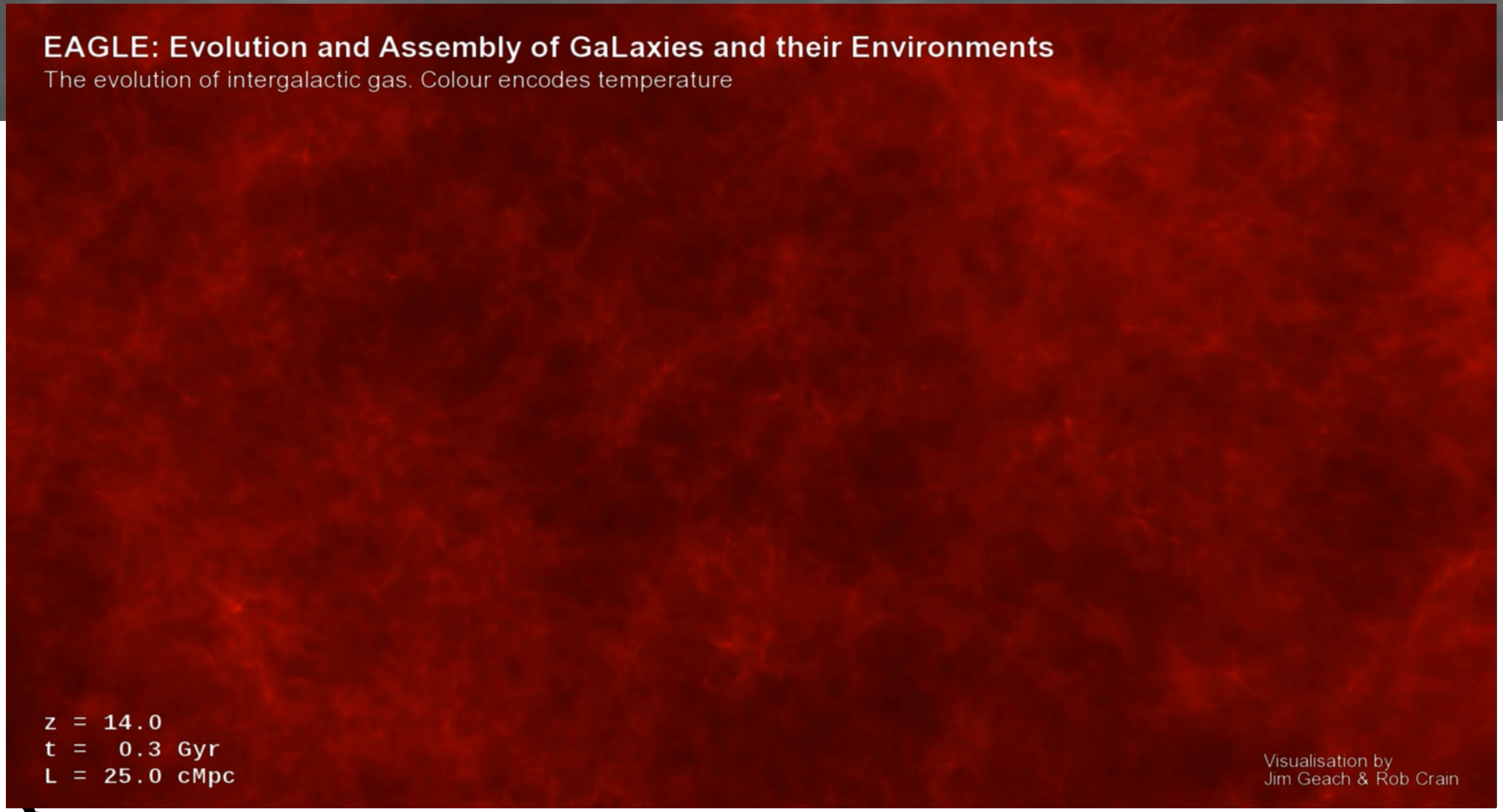
- Cosmological simulations of the formation of the Universe and galaxy formation.

- EAGLE project: 48 days of computing on 4096 cores. >500 TBytes of data products (post-processed data is public!). Most cited astronomy paper of 2015.



Trayford+2015, MNRAS

http://www.eaglesim.org/


EAGLE: Evolution and Assembly of GaLaxies and their Environments
The evolution of intergalactic gas. Colour encodes temperature
z = 14.0
t = 0.3 Gyr
L = 25.0 cMpc
Visualisation by
Jim Geach & Rob Crain

What we do and how we do it

- Solve coupled equations of gravity and hydrodynamics using particle-based methods (SPH, FVPM, …).
- Consider the interaction between gas and stars/black holes as part of a large and complex *subgrid* model with free parameters to be calibrated.
- Evolve multiple matter species at the same time.
- Dynamic range of >6 orders of magnitude in “particle size”.
- Typically have >2M time-steps.

Hydrodynamics scheme: The problem to solve

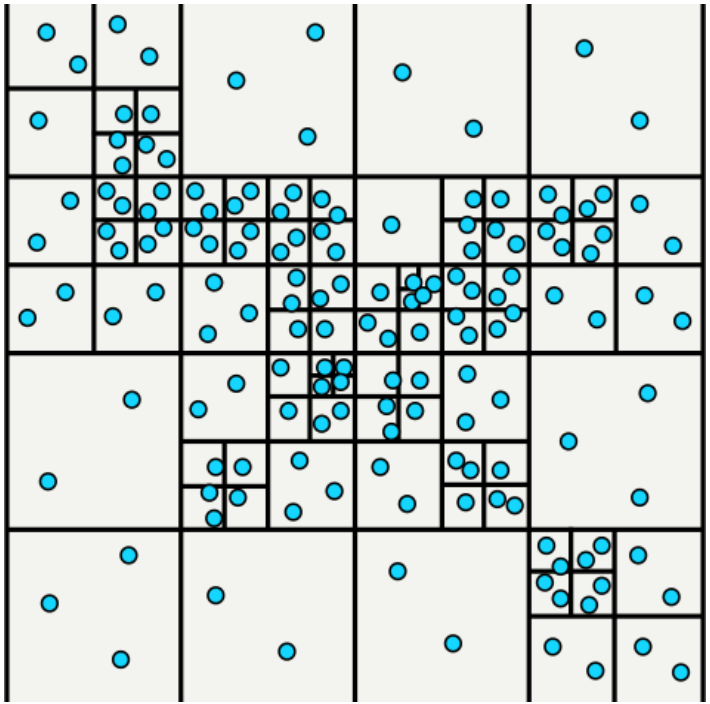
For a set of N (>10^{10}) particles, we want to exchange hydrodynamical forces between all neighbouring particles within a given (time and space variable) search radius. Large density imbalances develop over time.

Challenges:

- Particles are unstructured in space, large density variations.
- Particles will move and the neighbour list of each particle evolves over time.
- Interaction between two particles is computationally cheap (low FLOP/byte).
- Individual time-steps for each particle.

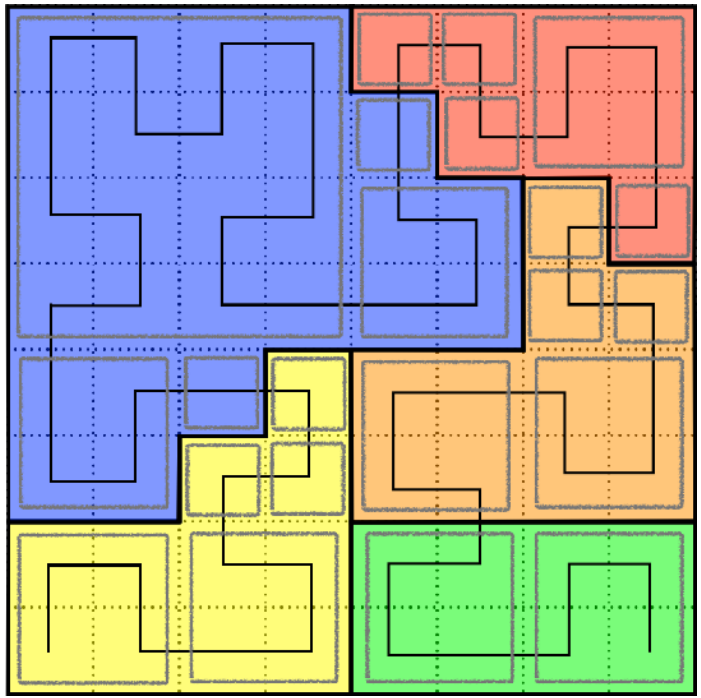
SPH scheme: The traditional method

- “Industry standard” code is *Gadget* (Springel+2005).
- MPI-only code.
- Neighbour search based on oct-tree.
- Domain decomposition based on a space-filling curve.



Domain Decomposition

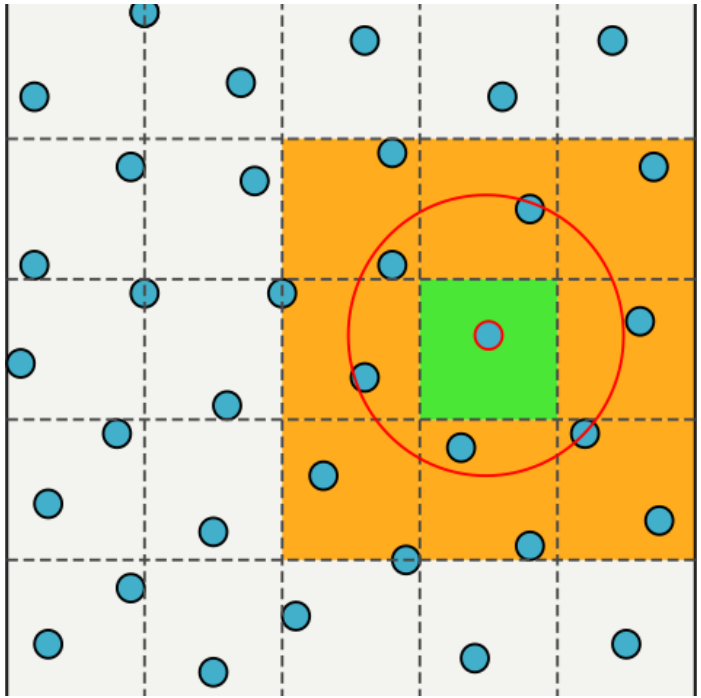
1. Sort the data along a space-filling curve.
2. Split the curve such that each node gets the same number of particles.
3. Do this at the top-level or close to the top.



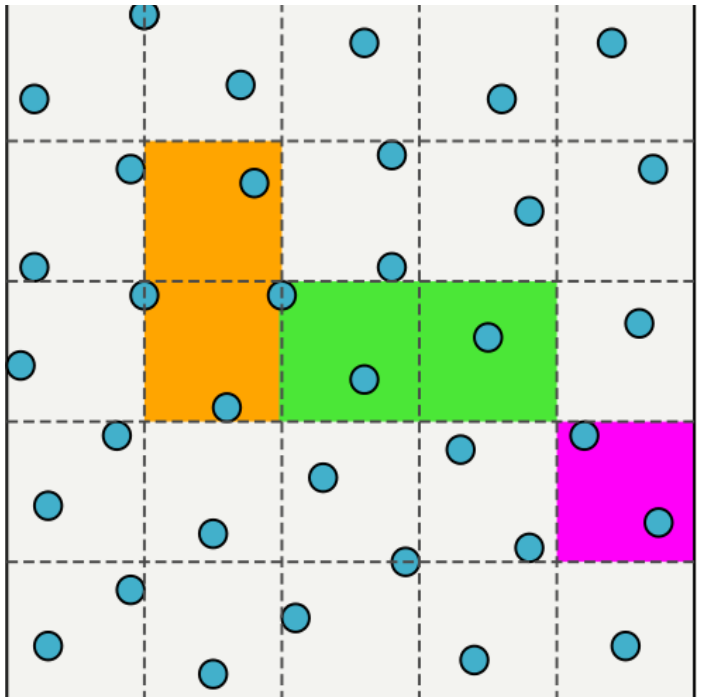
SWIFT basics

- Use an AMR-grid-based Verlet-list neighbour search.
- Use Task-based parallelism on the node.
- Use asynchronous MPI communications in the task framework.
- Use SIMD intrinsics for the core of the calculation.

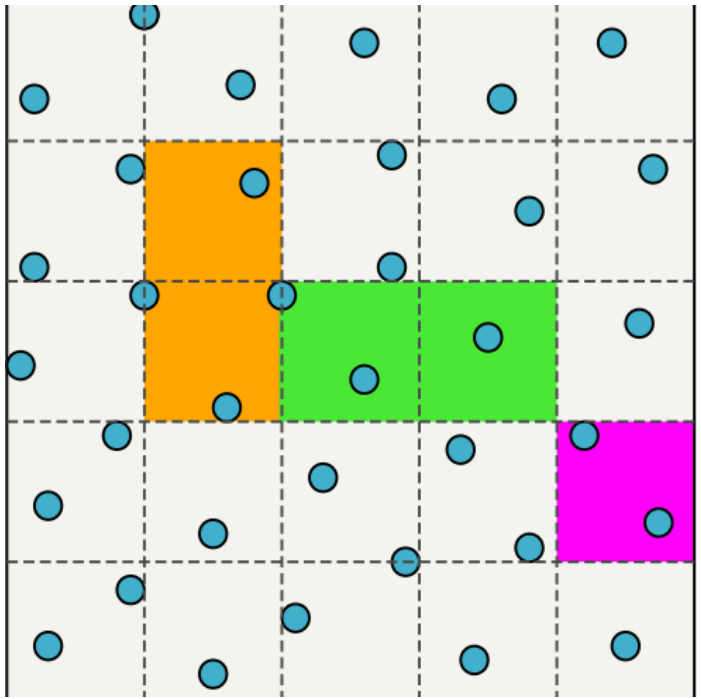
AMR Verlet-list neighbour search



- Target ~500 particles per cell via adaptive mesh refinement.
- Cell size naturally matches particle neighbour search radius.
- Particles only interact with particles in the same cell or any direct neighbouring cell.



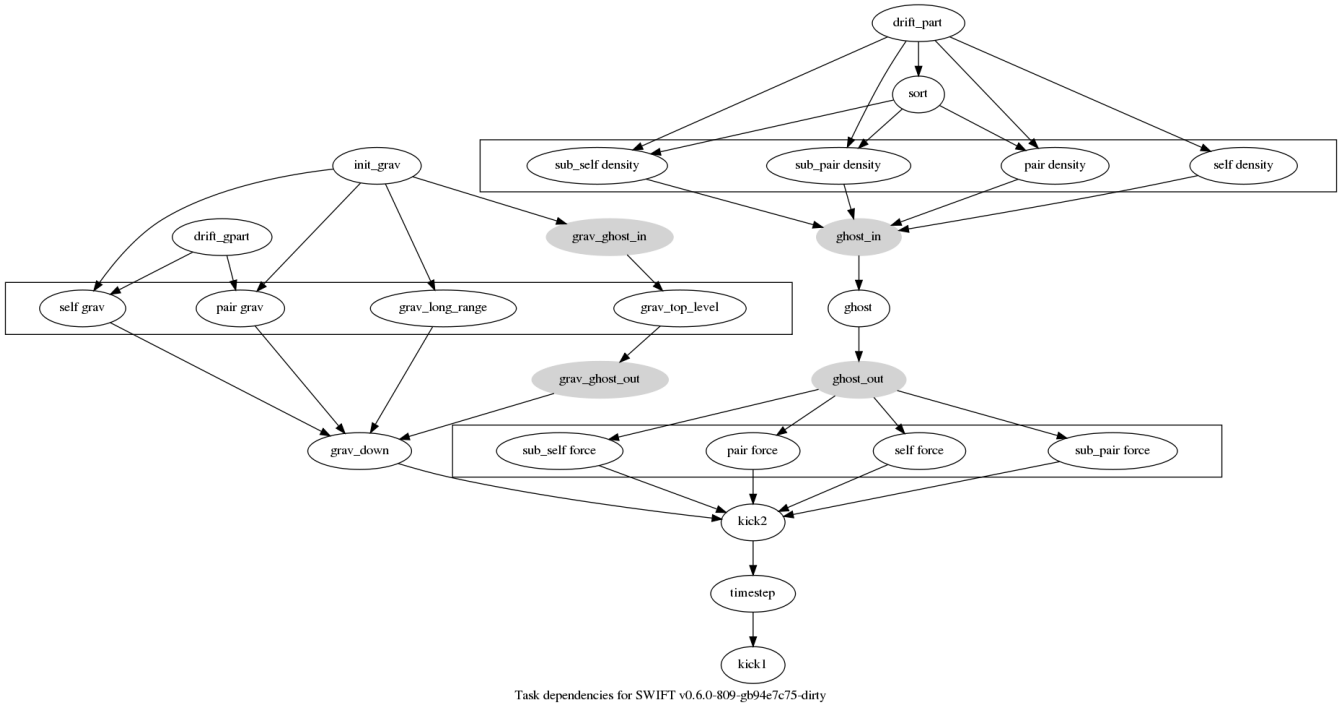
- Cells pairs do not need to be processed in any pre-defined order.
- Only need to make sure two threads do not work on the same cell.
- Cell pairs can have vastly different work-loads.



- Cells pairs do not need to be processed in any pre-defined order.
- Only need to make sure two threads do not work on the same cell.
- Cell pairs can have vastly different work-loads.

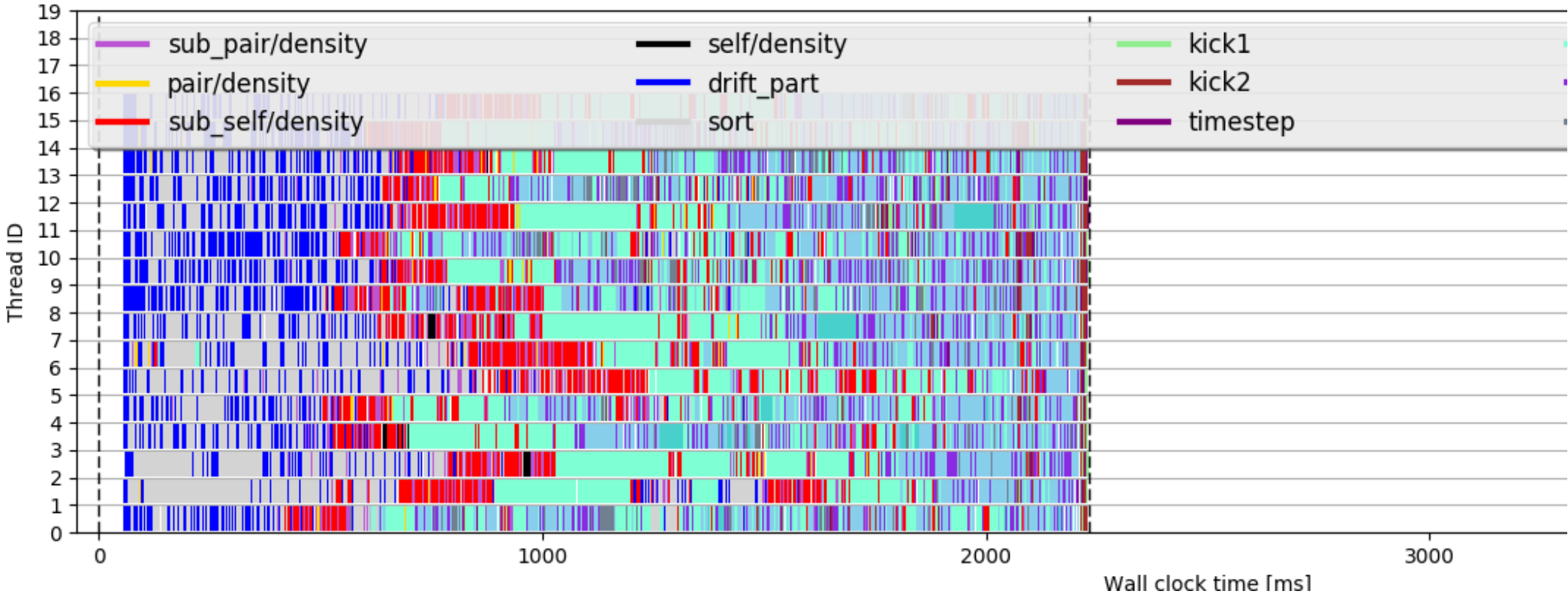
→ Need runtime dynamic scheduling

Task-based parallelism for SPH



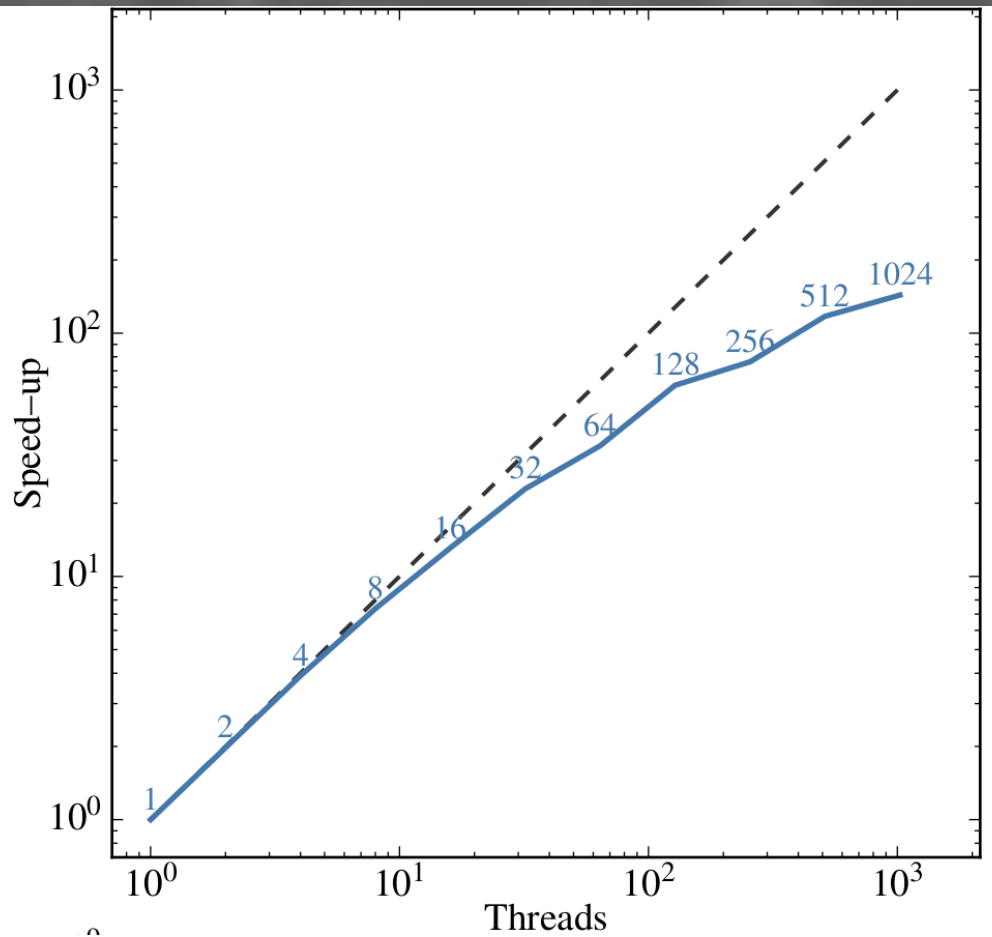
- Task-graph describing the science.
- Arrows depict dependencies.

Task-based parallelism in action

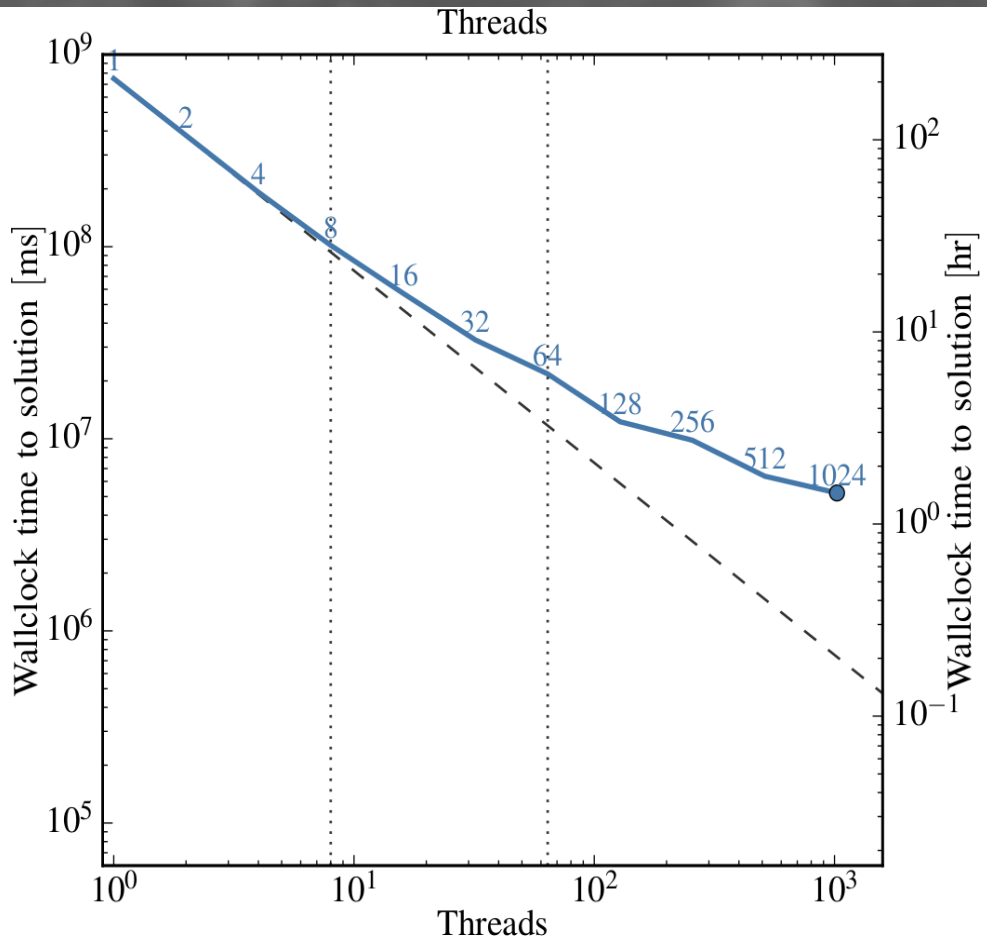


Task-graph for one time-step. Colours correspond to different task types.
Almost perfect load-balance achieved on 16 cores.

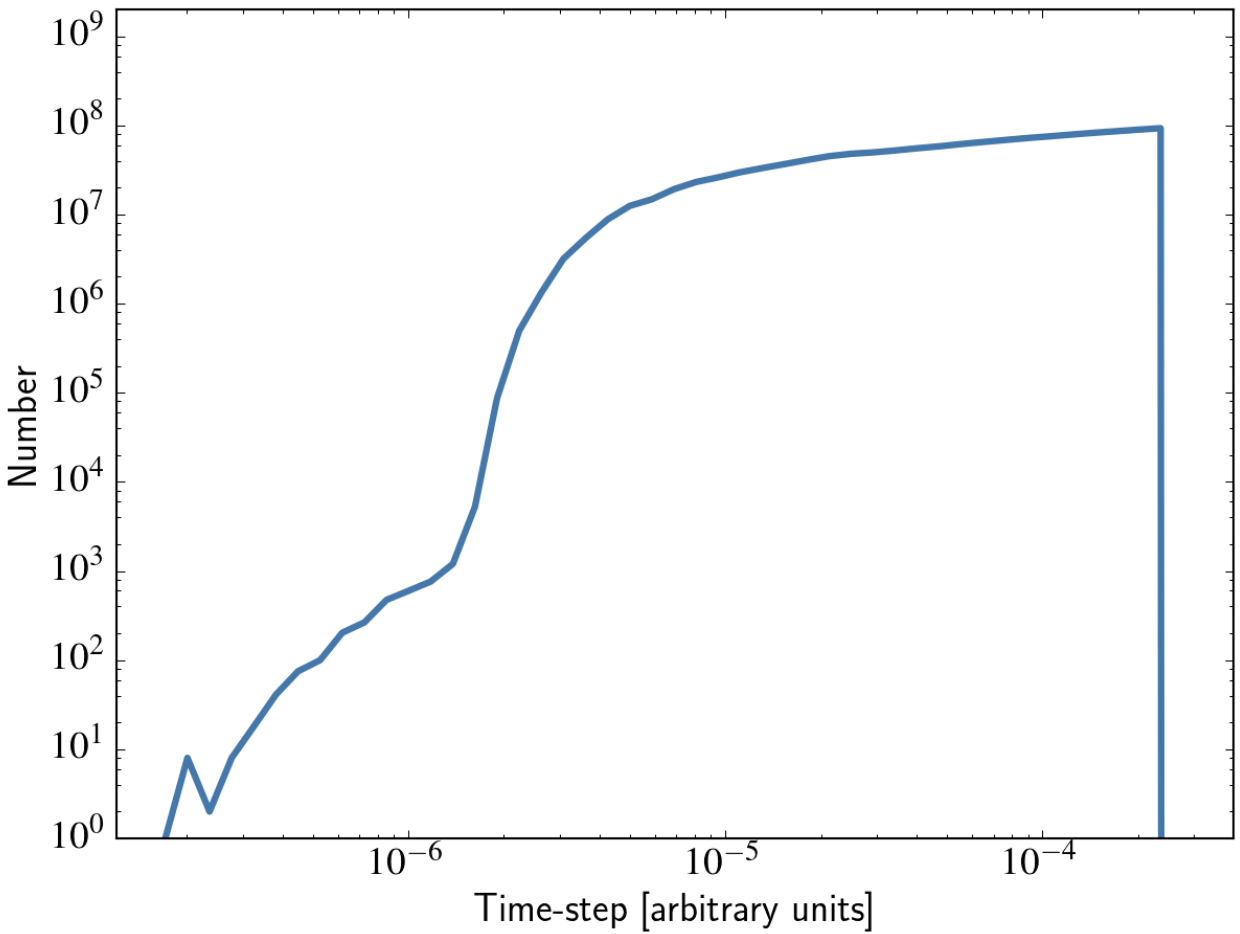
Scaling with a global time-step



Time-to-solution with a global time-step



Time-step hierarchy



Local time-stepping

Our particle methods for hydro-dynamics have a linear cost.

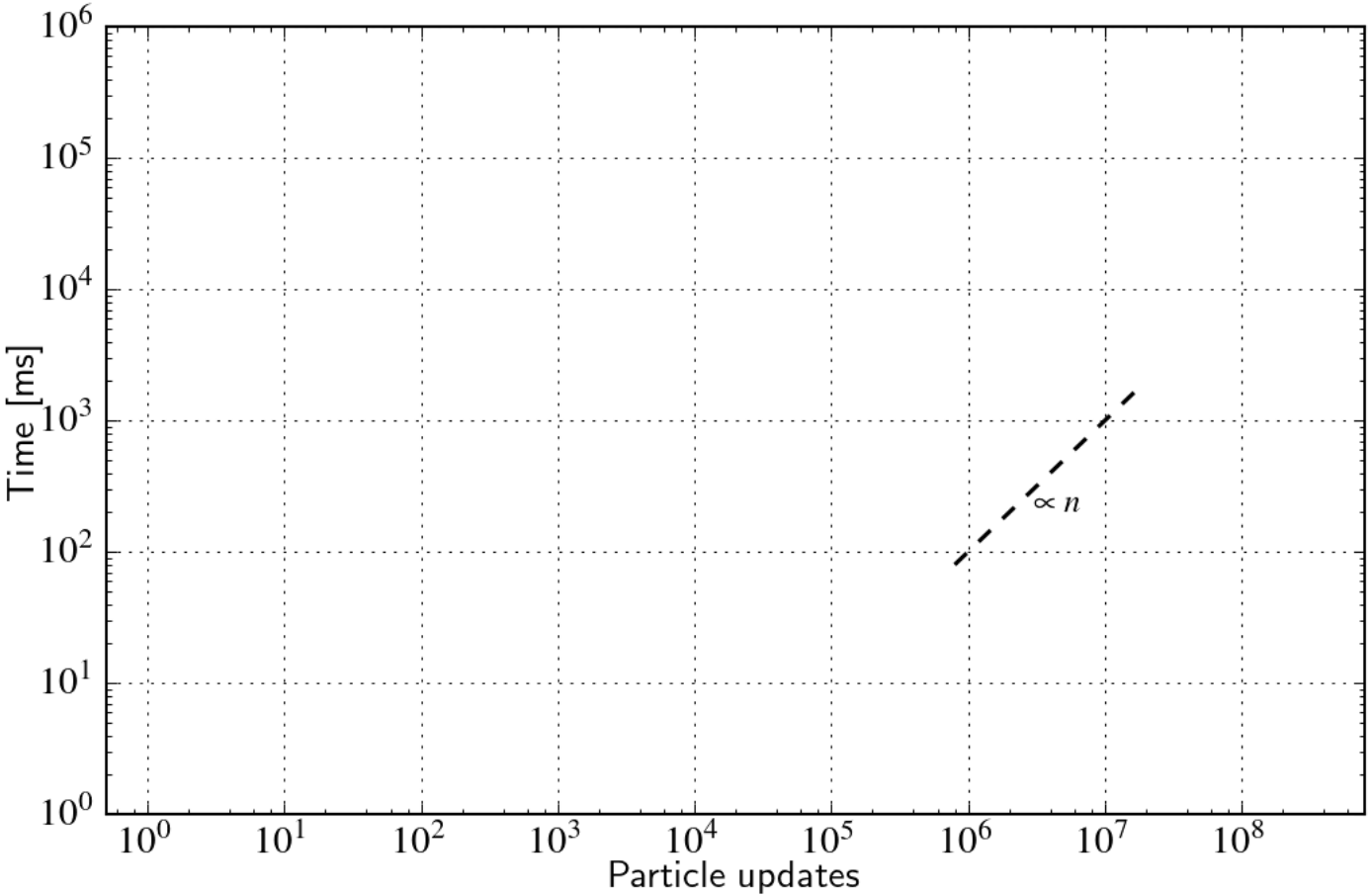
→ By only updating the particles on a short time-step we can gain orders of magnitude in run time.

(not a new idea. Been done since the 80s in astrophysics)

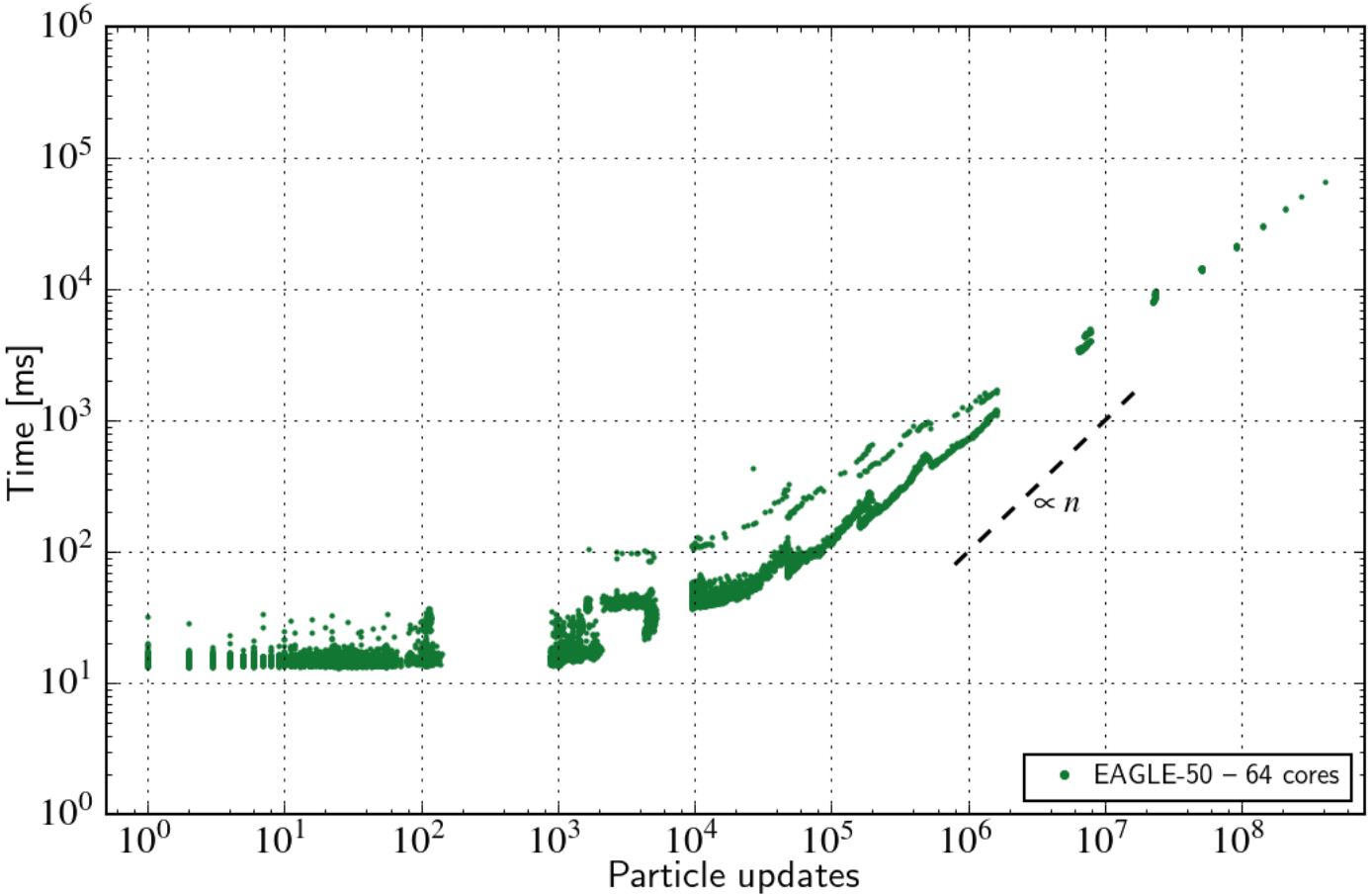
→ You also kill your scaling.

The question is then how to load-balance and parallelize this.
How do you update ~10 particles efficiently on 1000 nodes?

The challenge visually



The challenge visually



A hand-written solution

- Avoid MPI for small updates.
- The particles with the smallest time-steps should be at the centre of their domains and not require any “halo” particles.
- Small time-steps are at the centre of galaxies.

- Identify galaxies → Grow domains organically around them
  → Get efficient code.

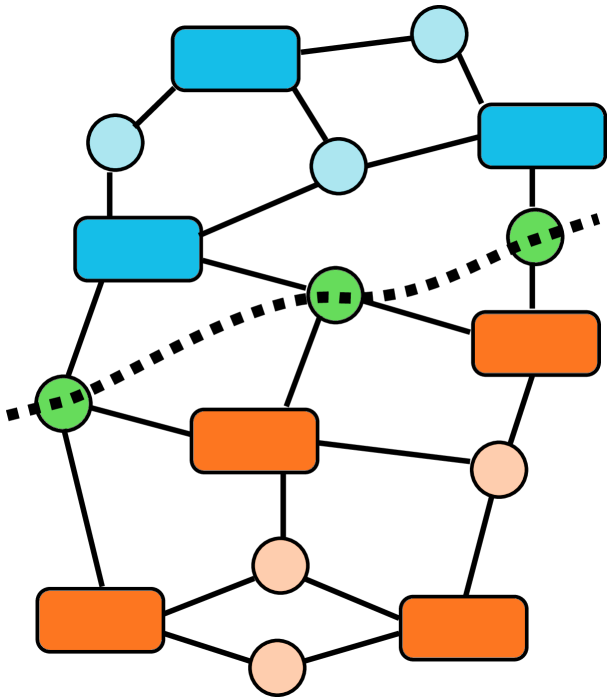
A hand-written solution

- Avoid MPI for small updates.
- The particles with the smallest time-steps should be at the centre of their domains and not require any “halo” particles.
- Small time-steps are at the centre of galaxies.

- Identify galaxies → Grow domains organically around them
  → Get efficient code.

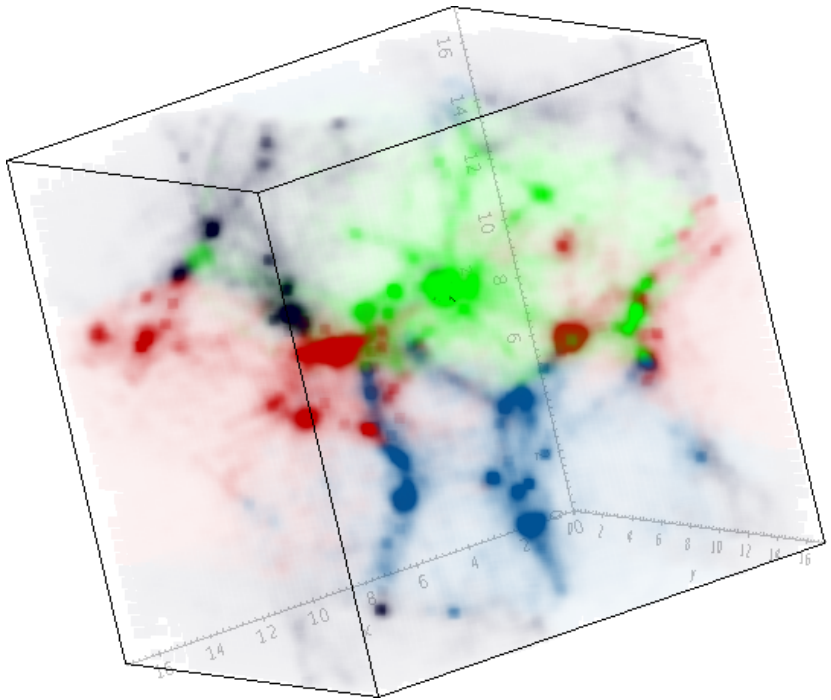
Drawbacks: Not generic. Identifying galaxies is not trivial.

A Graph-based strategy



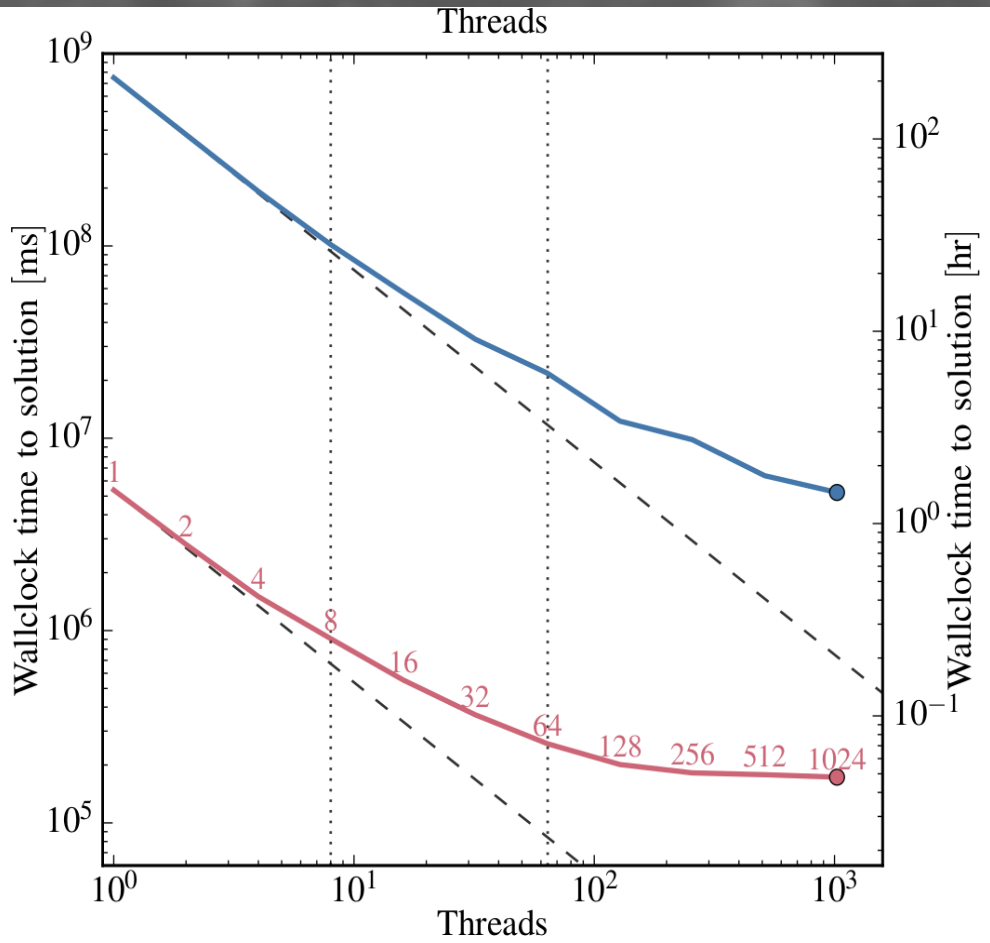
- For each task, we compute the amount of work (=runtime) required.
- We build a graph where the data are nodes and tasks are hyper-edges.
- METIS is used to split the graph such that the work (not the data!) is balanced.
- Extra cost added for communication tasks to minimise them.

What does it look like?



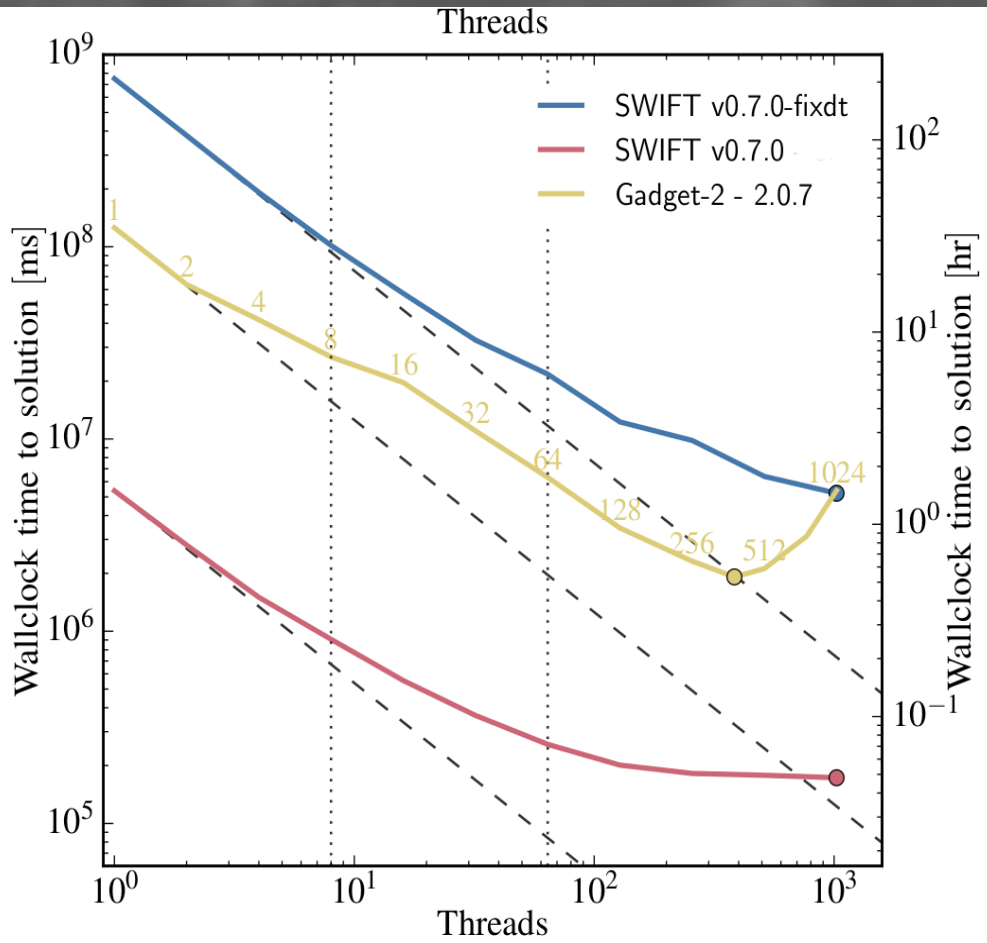
- No regular grid pattern.
- No space-filling curve pattern.
- Good (work) load-balancing by construction.
- The most dense regions are at the centre of their respective domains.

Time-to-solution with a local time-step



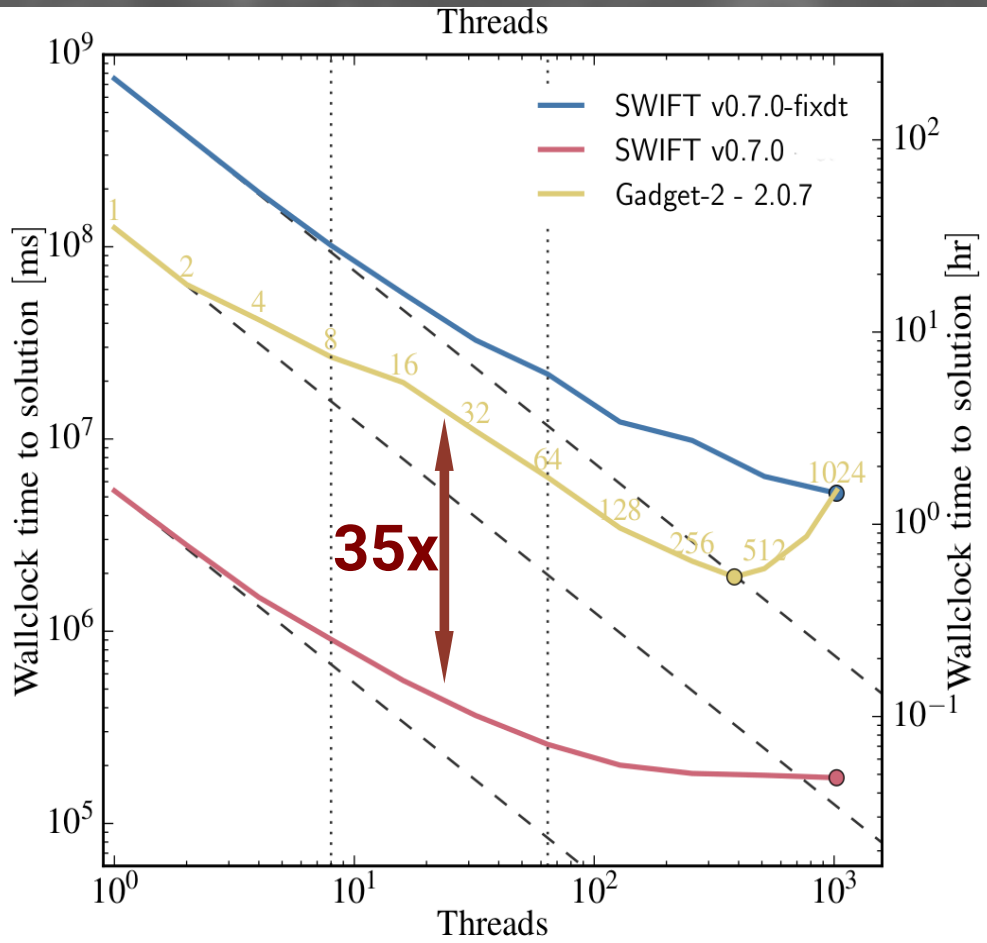
Time-to-solution with a local time-step

- Realistic problem
- Same accuracy.
- Same hardware.
- Same compiler.
- Same solution.



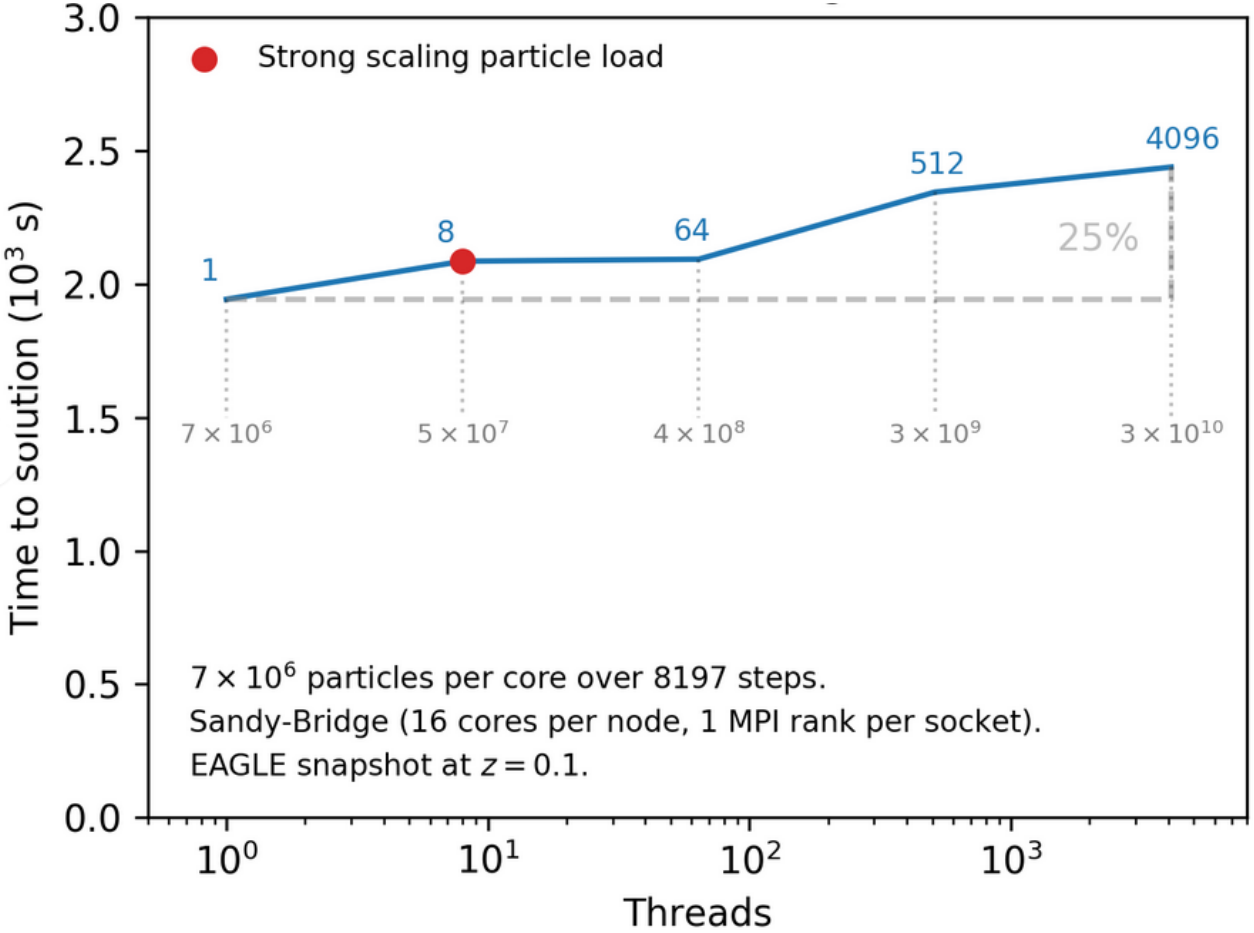
Time-to-solution with a local time-step

- Realistic problem
- Same accuracy.
- Same hardware.
- Same compiler.
- Same solution.



Half way to exa-scale!

Sustained performance in weak-scaling



Conclusions

- New algorithms can lead to significant speed-ups over conventional methods. Implicit methods vs. explicit local time-step based solvers.

  >30x over Gadget → “half way from peta-scale to exa-scale via algorithms”

- Task-based parallelism as a viable model for actual scientific applications.

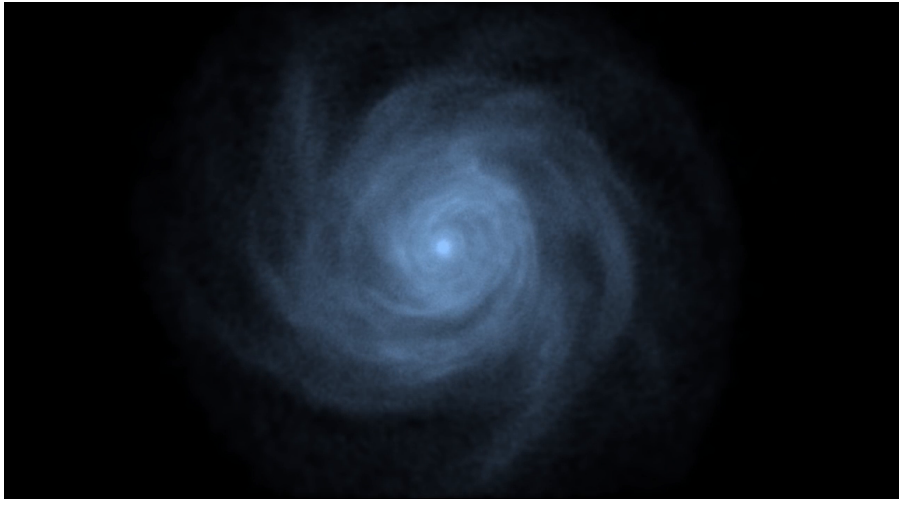
  → Not just a research concept.

- Pure scaling is not necessarily the best metric to judge an application or hardware.

  → “time-to-solution” or “time-to-science” should be considered.

@SwiftSimulation

www.swiftsim.com



Hausammann, Revaz, Schaller