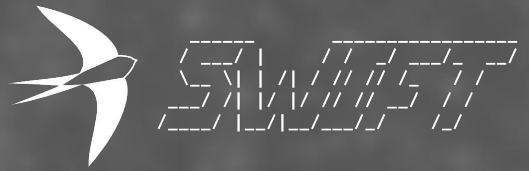
The scalability challenge of extreme dynamic range problems

Josh Borrow, Matthieu Schaller + SWIFT team

Leiden Observatory, Netherlands



SWIFT 101

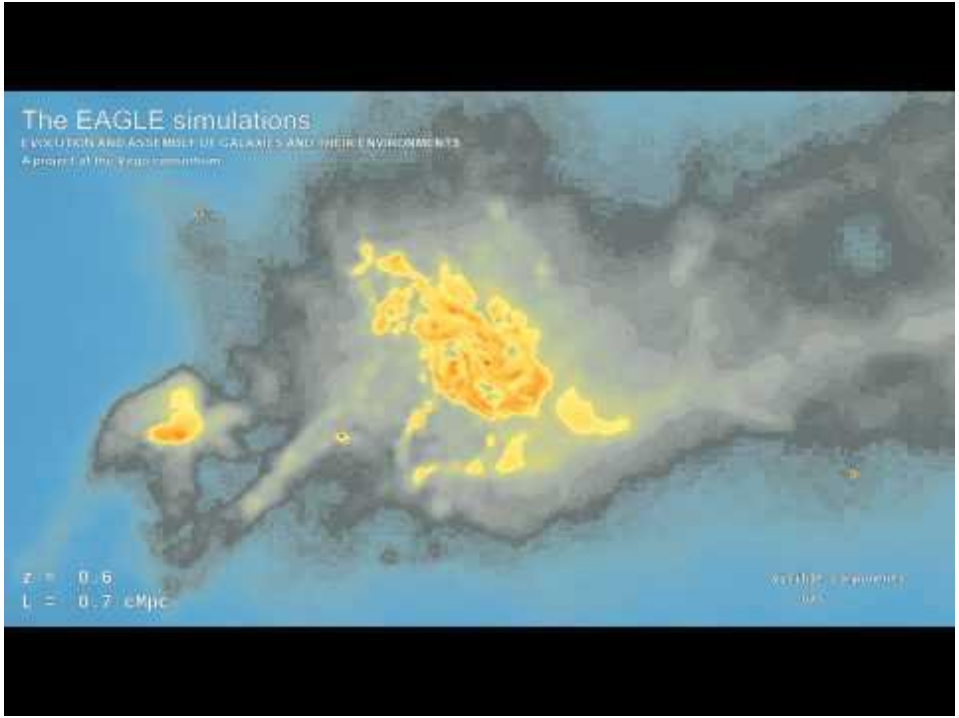
- Open-source particle-based hydro + gravity code (C99) designed for cosmology and galaxy formation.
- In brief:
  - Task-based parallelism inside the nodes.
  - Asynchronous MPI in-between.
  - Domain decomposition of the task graph not of the data.
  - Vector instructions for the core routines.
  - Neighbour search using recursive grid and pseudo-Verlet list.
  - Runs up to $>10^{10}$ particles performed.

SPHERIC Trondheim
arXiv:1606.02738

Context: Cosmological simulations

EAGLE project:

- SPH (no Riemann solver)
- Coupled to gravity
- 3×10^6 time-steps
- 3.5×10^9 particles
- 48 days on 4000 cores
- Most cited astro paper in 2015



Our target cosmological simulation

Setup:

- 200×10^9 particles.
- 3×10^6 time-steps.

System:

- 100’000 cores

Typical code:

- 10^{-4} s / update / core

Our target cosmological simulation

Setup:

- 200×10^9 particles.
- 3×10^6 time-steps.

System:

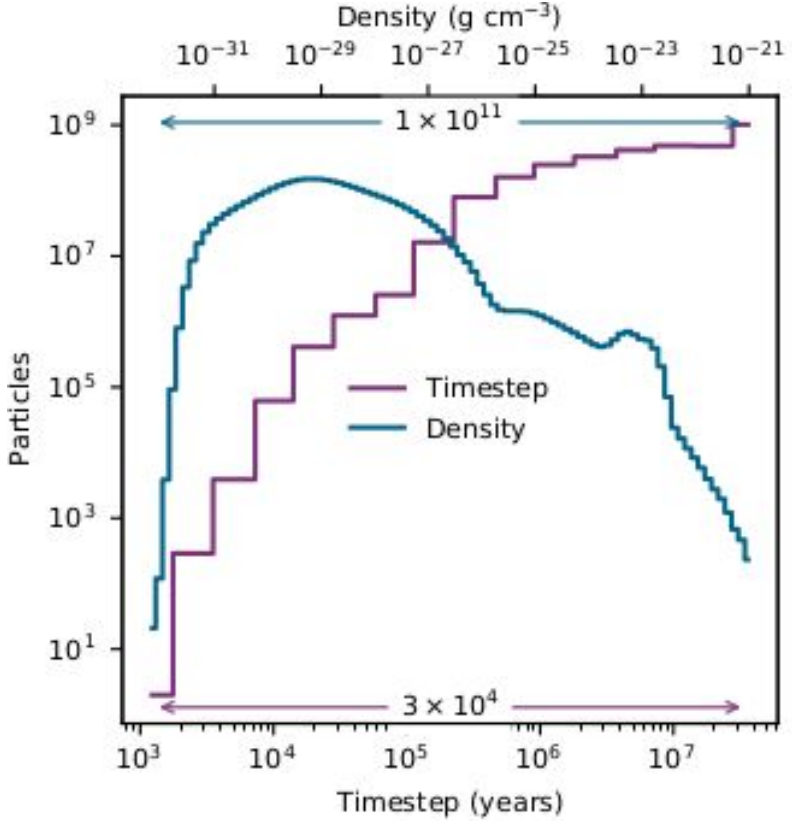
- 100’000 cores

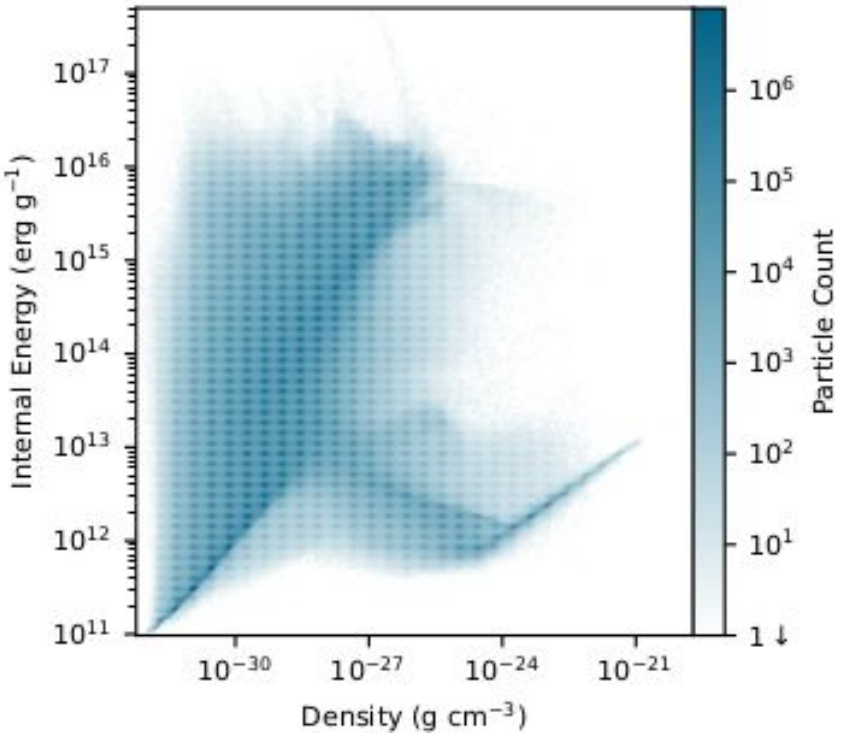
Typical code:

- 10^{-4} s / update / core

19 years of wall-clock time !!

A very large dynamic range

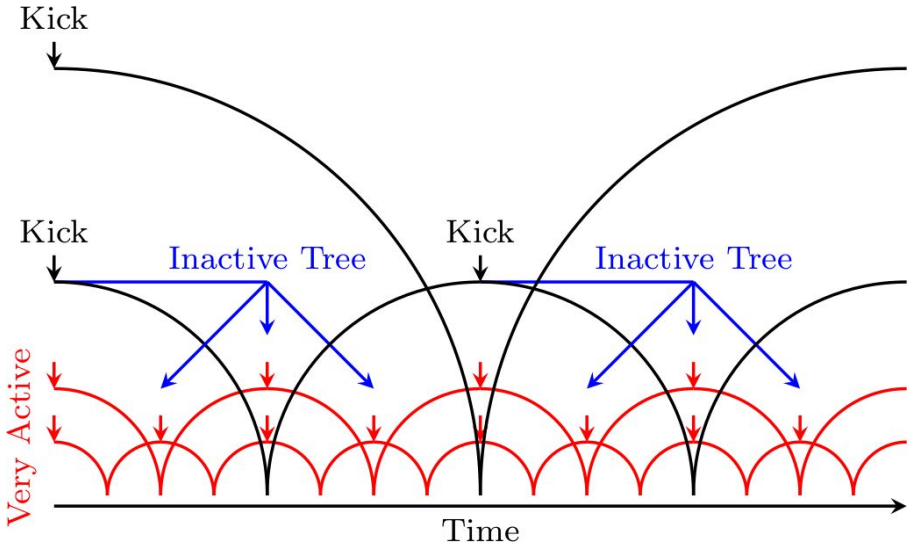




Time integration operator splitting

Classic leap-frog:

$K(\Delta t/2) \times D(\Delta t) \times K(\Delta/2)$



Splitting the “drift”:

$K(\Delta t/2) \times D(\Delta t/2^n) \cdots D(\Delta t/2^n) \times K(\Delta/2)$

Time integration operator splitting

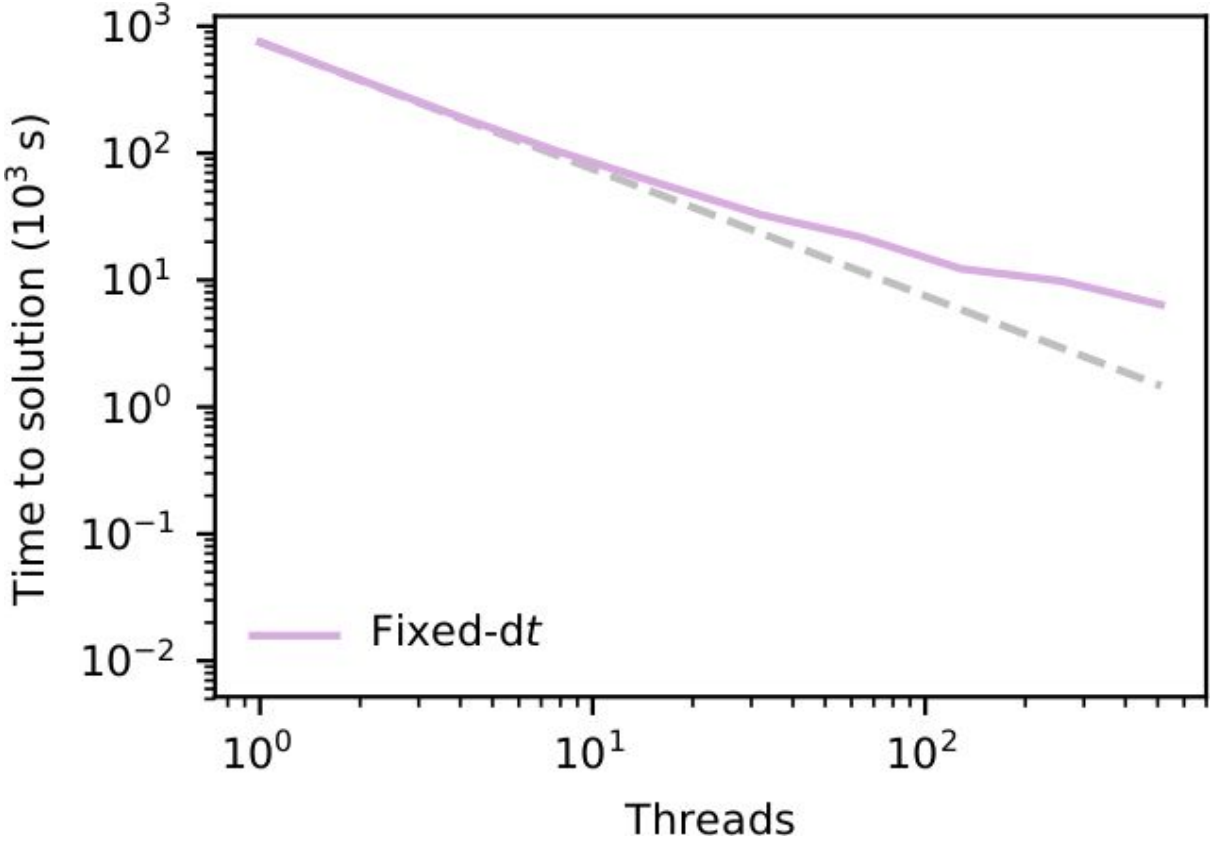
Algorithm:

- Compute “kick” (accelerations) only for the particles in the current time bin.

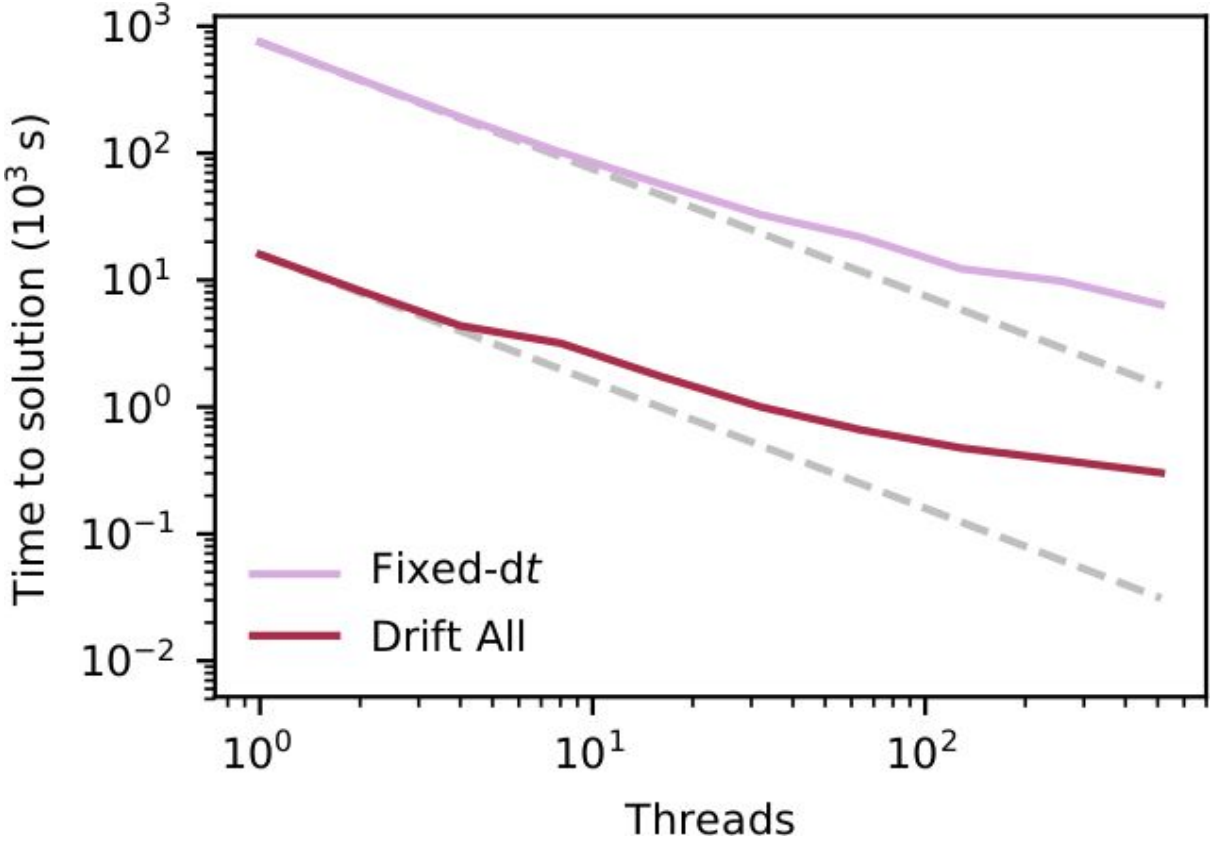
- Apply the “drift” (move) to all the particles.

Nothing new. Applied everywhere especially in gravity codes.

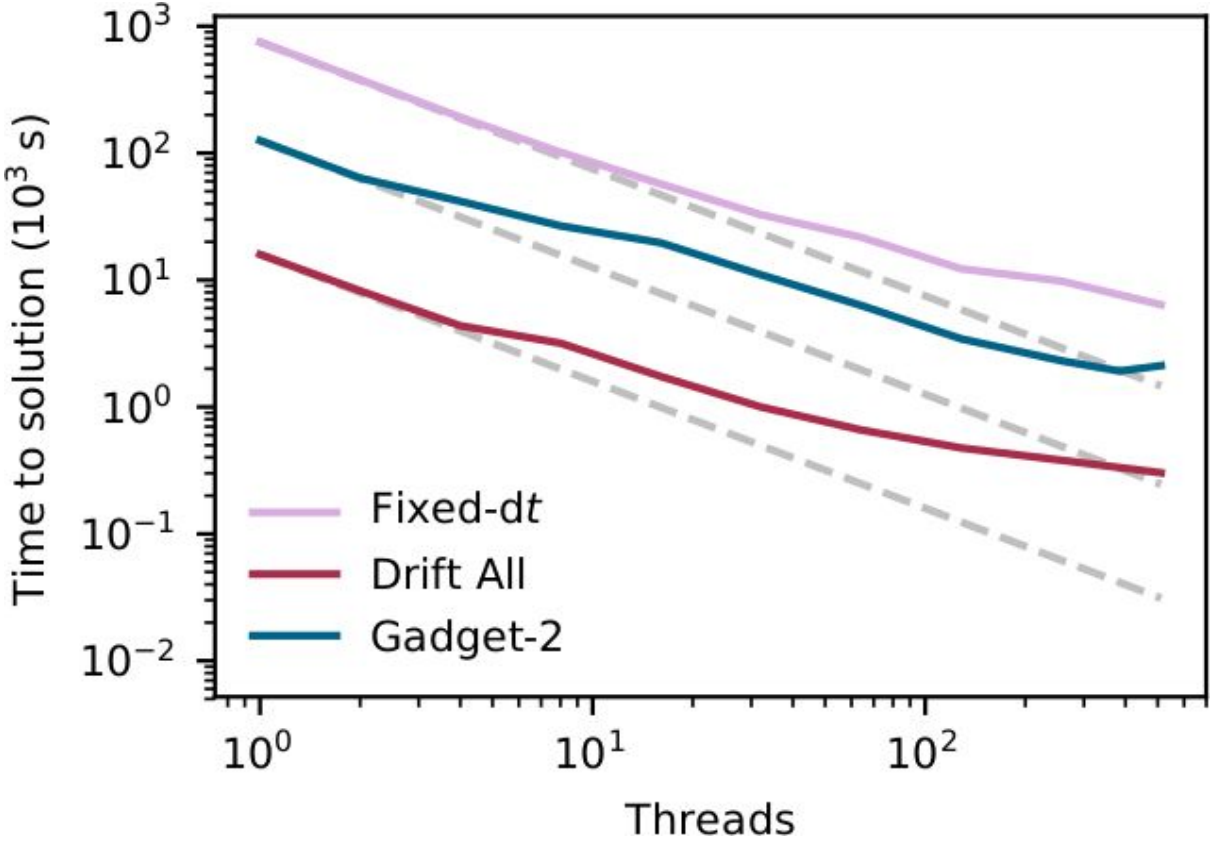
Efficiency


Time to solution (10^3 s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
Fixed-dt
10^0
10^1
10^2
Threads

Efficiency


Time to solution (10^3 s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
Fixed-dt
Drift All
10^0
10^1
10^2
Threads

Efficiency


Time to solution (10^3 s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^0
10^1
10^2
Threads
Fixed-dt
Drift All
Gadget-2

Going beyond

The canonical algorithm drifts **all** the particles to the current point in time.

Do we need that?

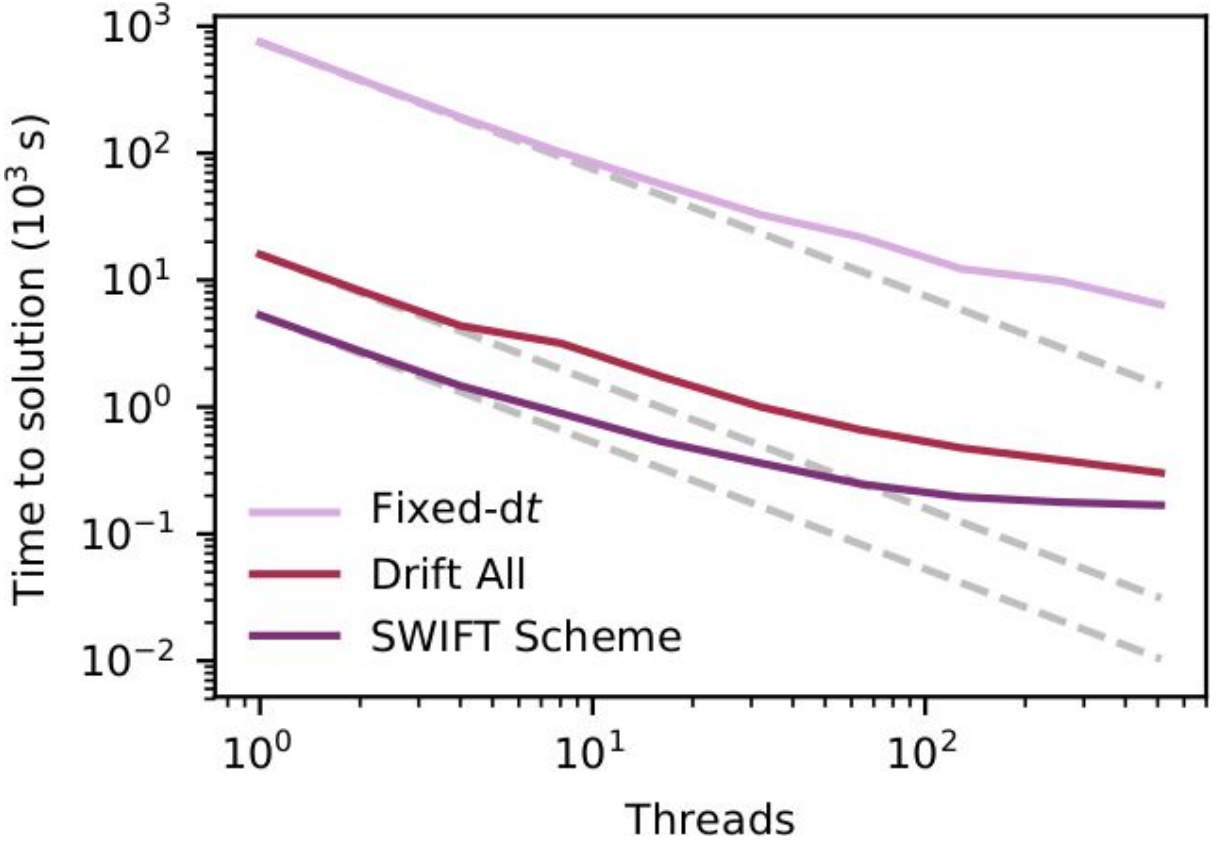
Going beyond

The canonical algorithm drifts **all** the particles to the current point in time.

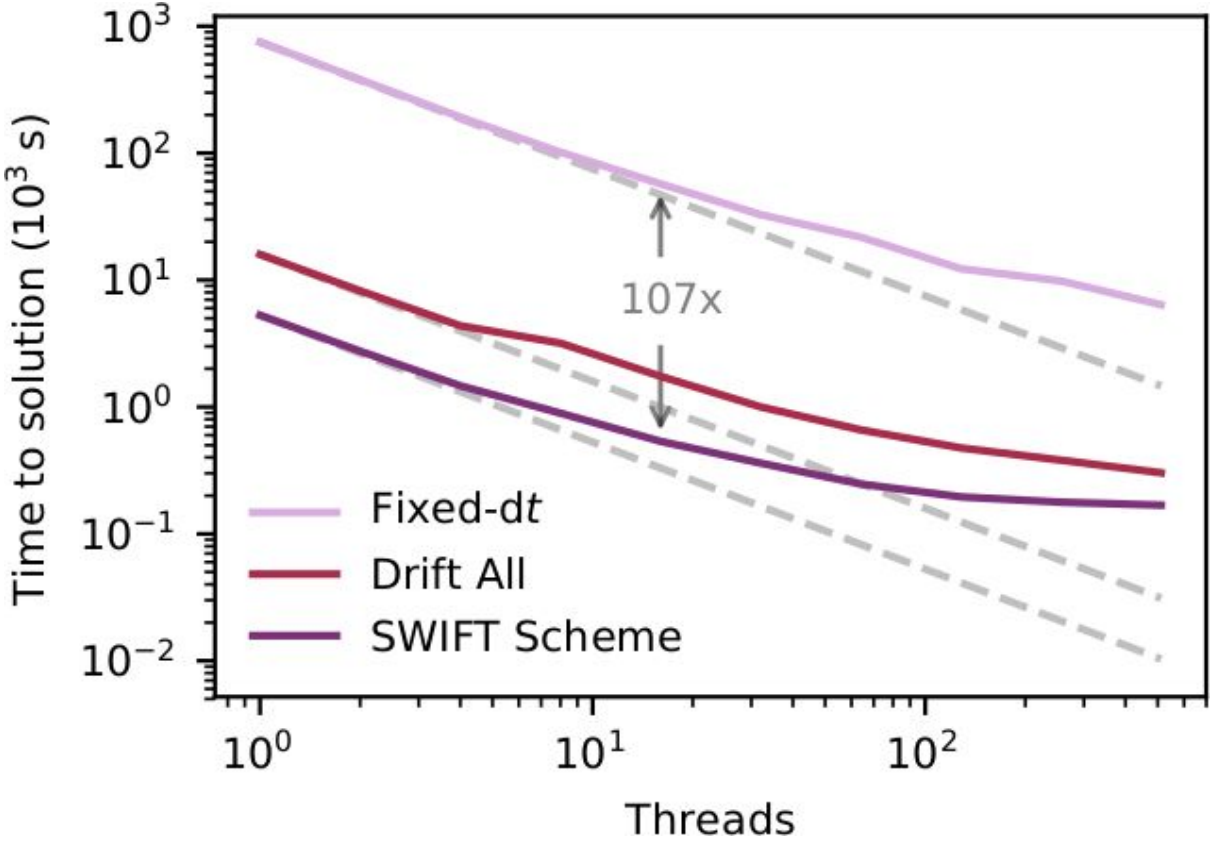
Do we need that? No! Only the particles that are neighbours of an active particle need to be moved forward.

-> Tree-walk “activating” the tasks in parts of the domain that need to. Followed by the actual calculation.

Efficiency


Time to solution (10^3 s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
Fixed-dt
Drift All
SWIFT Scheme
10^0
10^1
10^2
Threads

Efficiency


Time to solution (10^3 s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
107x
Fixed-dt
Drift All
SWIFT Scheme
10^0
10^1
10^2
Threads

Parallel efficiency

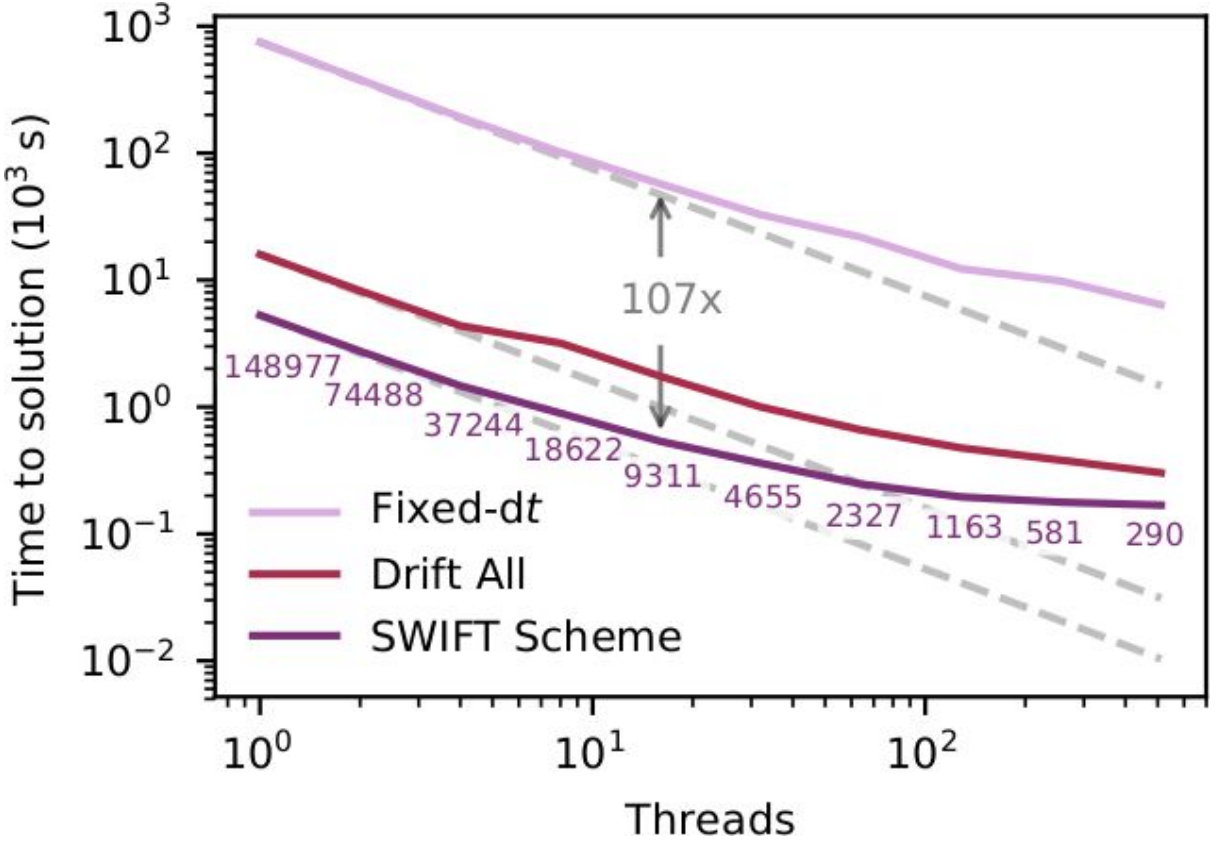
Parallel efficiency has dropped.

-> You computer scientist friend won’t be happy.

But time-to-solution has decreased a lot.

-> Your scientist friend will be happy.

Not enough stuff to do

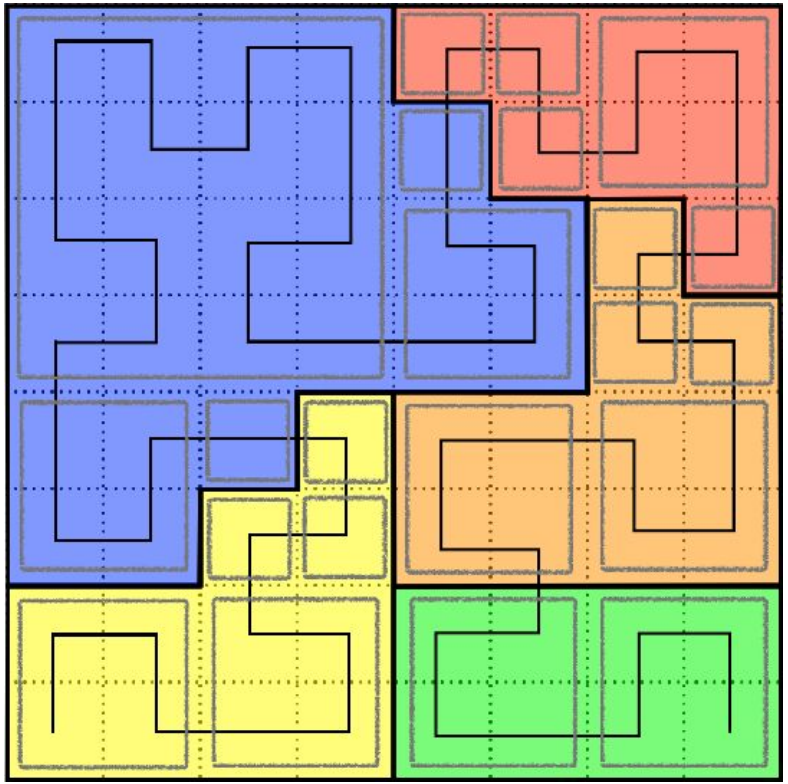


How do we load-balance this efficiently?

Smallest time-step problem

- Clearly impossible to distribute N particles effectively on M nodes if N < M.
- Solution is then to make these steps as cheap as possible and cut *all* overheads.
- The main one is MPI. Let’s cut that.

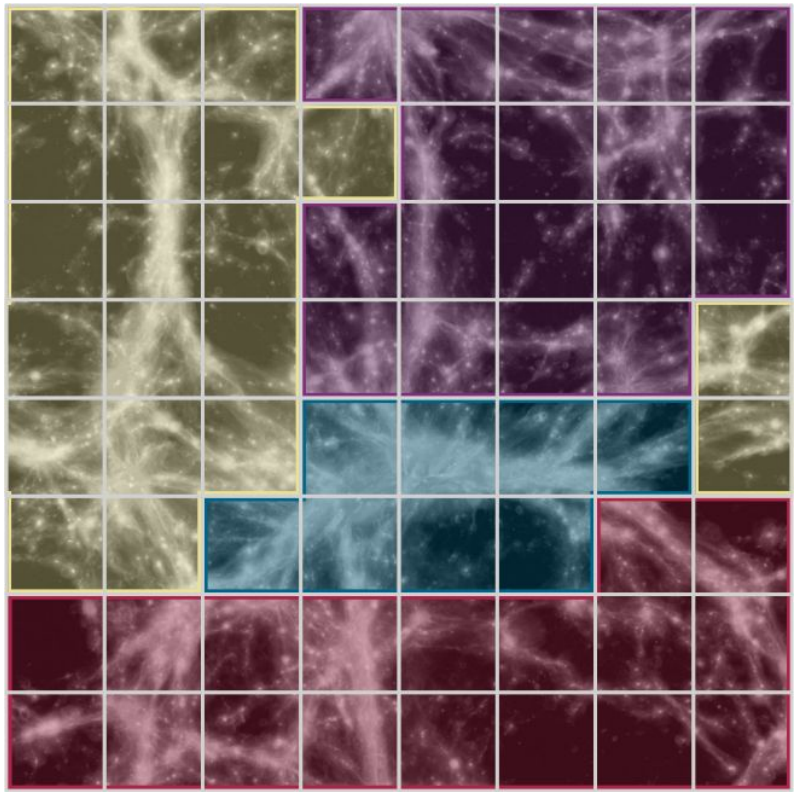
Classic domain decomposition



Space-filling curve:

- Very common.
- Balances the data (mostly).
- Makes sure the average run time is low.

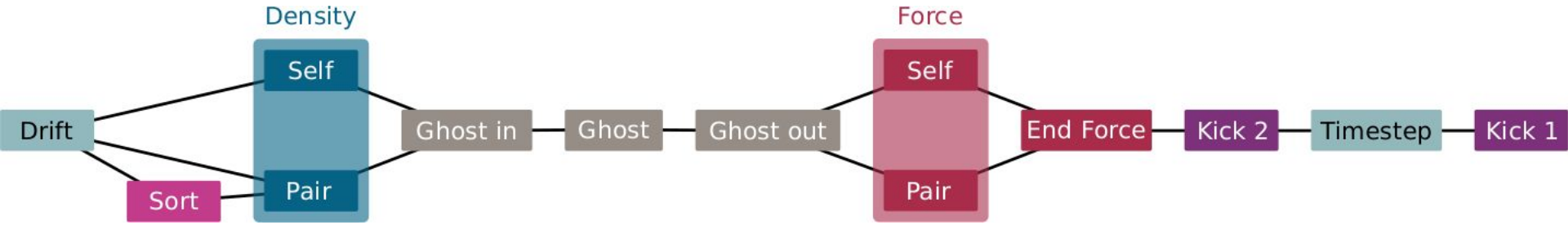
Domain Decomposition - visually



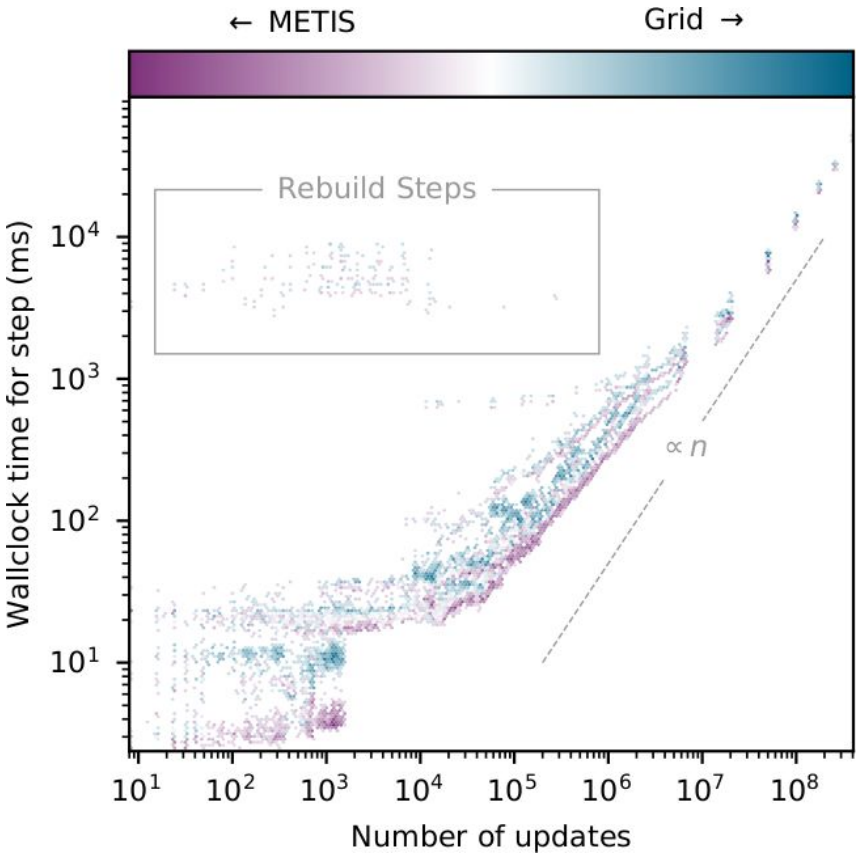
- Each MPI rank gets a number of cells.
- How do we distribute these cells among nodes efficiently?

SWIFT strategy

- SWIFT uses task-based parallelism.
- Let’s decompose the task-graph.
- Give **heavy weights** to the communication tasks.
- Ask graph decomposition library to solve the problem.



A diagnostic

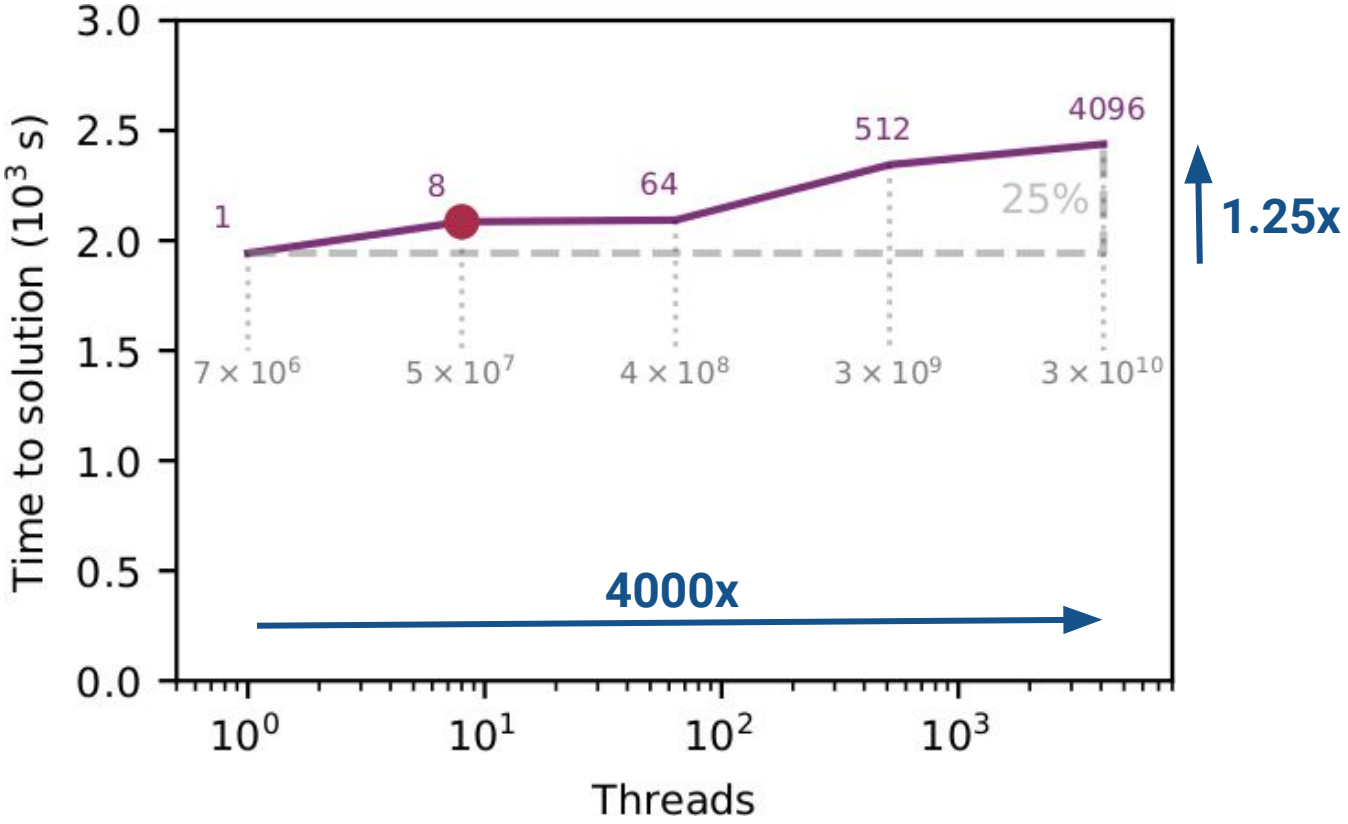


400×10^6 particles

4 nodes

16 cores/node

Weak-scaling results



Conclusions

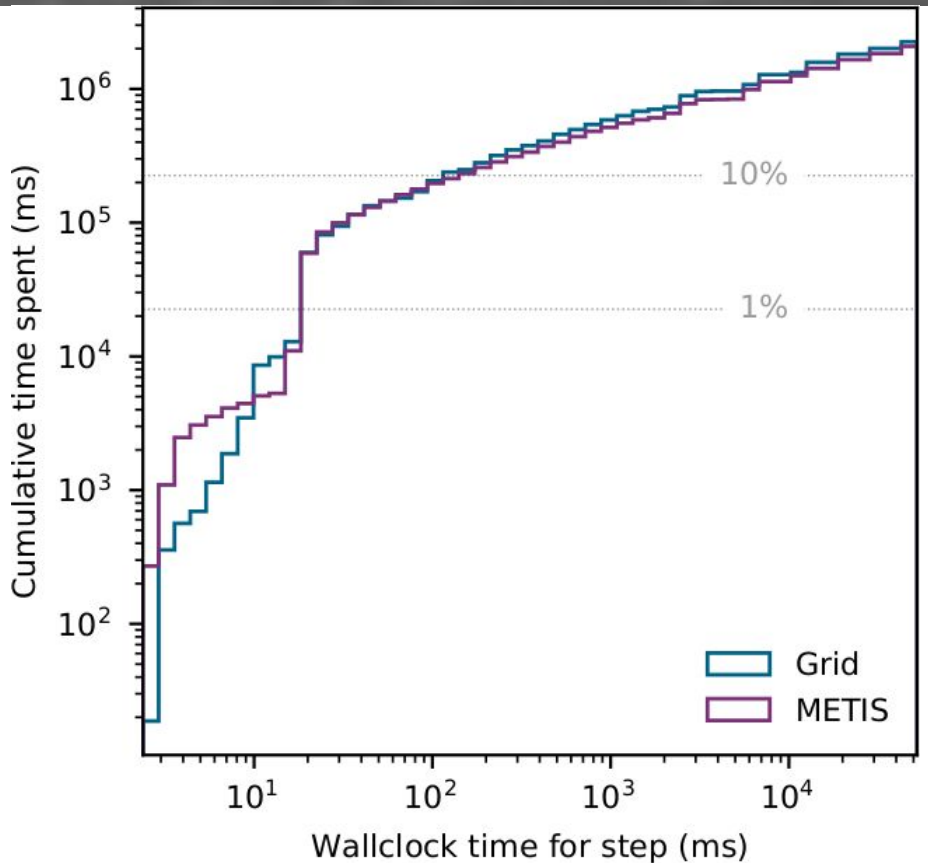
- Local time-stepping is crucial to extract performance.
- Scaling is *then* not the relevant metric.
  Time-to-solution is.
- Load-balancing must:
  - Decompose the *work* not the *data*.
  - Avoid MPI on the smallest steps.
  - Task-based parallelism offers ideal framework for that.

```
 _____      _________________
/ ___/ |    / /  _/ ___/_  __/
\__ \| |/| / // // /_   / /
___/ /|  |/ // // __/  / /
/____/ |__/|__/___/_/  /_/
```

SPH With Interleaved Fine-grained Tasking

www.swiftsim.com

@SwiftSimulation


Cumulative time spent (ms)
10^6
10^5
10^4
10^3
10^2
10%
1%
Grid
METIS
10^1
10^2
10^3
10^4
Wallclock time for step (ms)