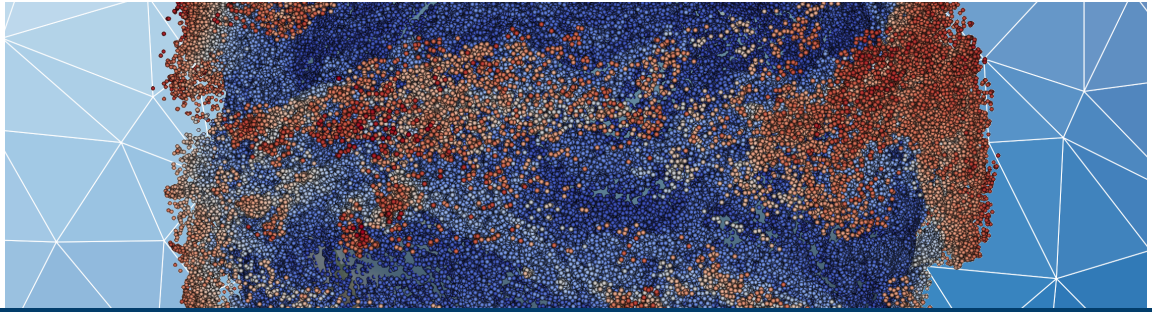
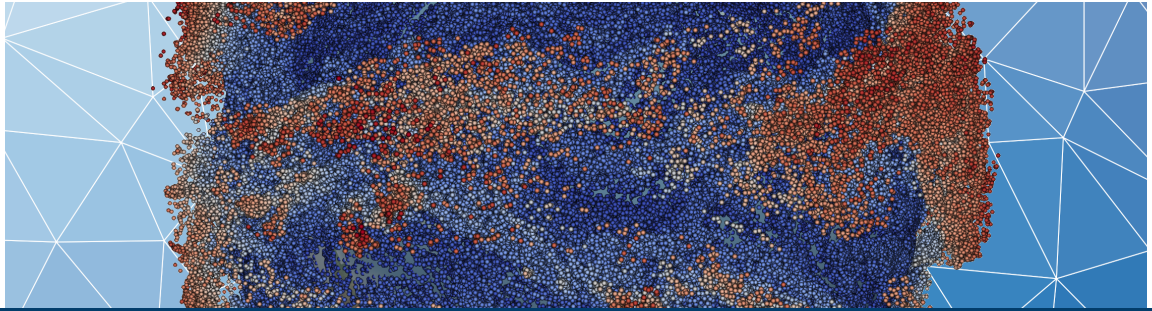


DOMAIN DECOMPOSITION IN MPTRAC

Performance evaluation and scaling tests

September 30, 2025 | Marvin Henke | Jülich Supercomputing Centre - GSP
Supervisors: Dr. Lars Hoffmann, Jan Clemens



Part I: Introduction

SIMULATION OF ATMOSPHERIC PROCESSES

Example cases

Volcanic Eruption Plume Tracking

Application: Simulations of volcanic sulfur dioxide emissions → aircraft safety
Reference: Hoffmann, L. et al. (2016). *J. Geophys. Res. Atmos.*, **121**, 4651–4673.
DOI: 10.1002/2015JD023749

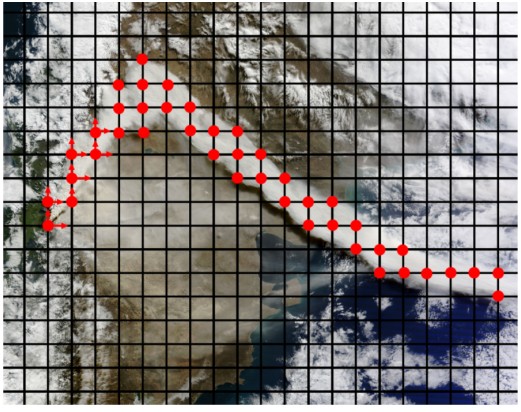
Source: Joshua Stevens/NASA Earth Observatory/AFP

MPTRAC

Massive-Parallel Trajectory Calculations

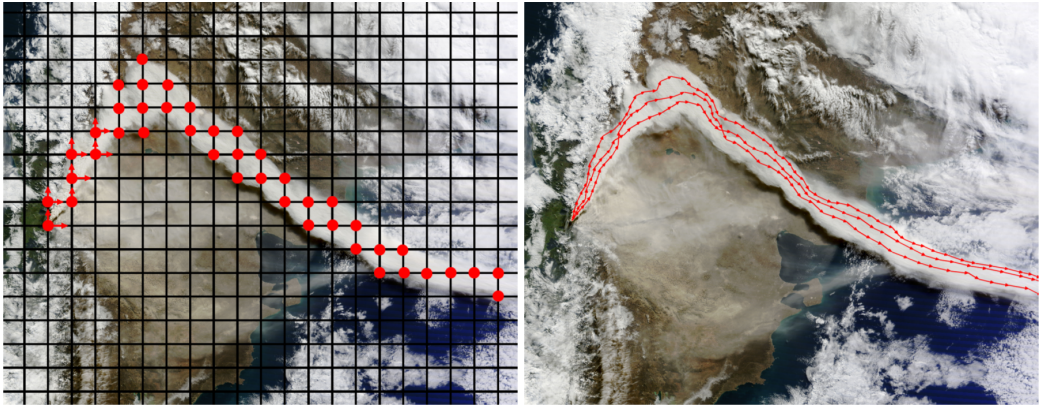
- **Lagrangian** transport model
- Simulates air parcel **advection**, **diffusion**, deposition & chemistry
- Handles processing of meteorological data
- Supports diverse output and visualization (including Python, ParaView, Gnuplot).
- Efficient use of HPC and GPU systems.
- **Open-source** and community support.

EULERIAN VERSUS LAGRANGIAN MODELING



(a) Eulerian

EULERIAN VERSUS LAGRANGIAN MODELING



(a) Eulerian (b) Lagrangian

Source: Hoffmann et al. (2025)

ADVECTION

Theoretical Parallelism in MPTRAC

- **Core Advection Equation (Particle Trajectory)**:

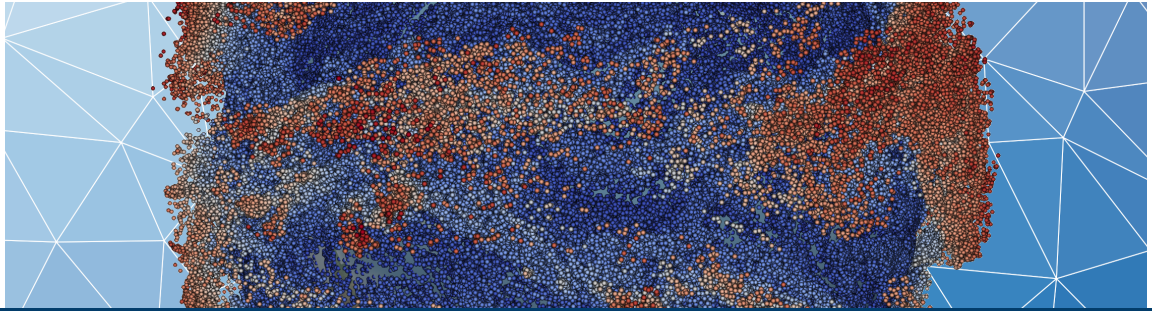
$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{v}(\boldsymbol{x}, t)$$

- **Local Dependency / Particle Independence**
- **Inherent Parallelizability**:
  - Workload would scale **linearly** with particle count

METEOROLOGICAL DATA

Requirements in MPTRAC

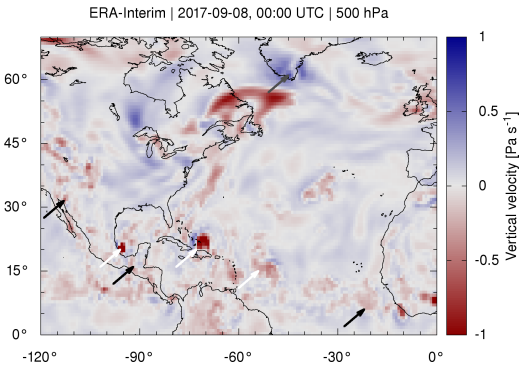
- **Required meteorological fields**:
  - 3D Wind Field (zonal, meridional, vertical wind components)
  - Pressure (as a vertical coordinate)
  - Temperature (for density/pressure calculations)
- Example: **ERA5** reanalysis data at 0.25° resolution (hourly)
- Meteorological data is **precomputed** on a fixed 3D grid (e.g., latitude $\times$ longitude $\times$ pressure/height levels)
- **Data Access Pattern**:
  - Repeated **loading (Δt_l)** of grid data for current and next timestep
  - Repeated **sampling ($\Delta t_\mathrm{s} \leq \Delta t_\mathrm{l}$)** during advection



Part II: Problem, Method, and Results

RESOLVING SMALL SCALE TURBULENCE

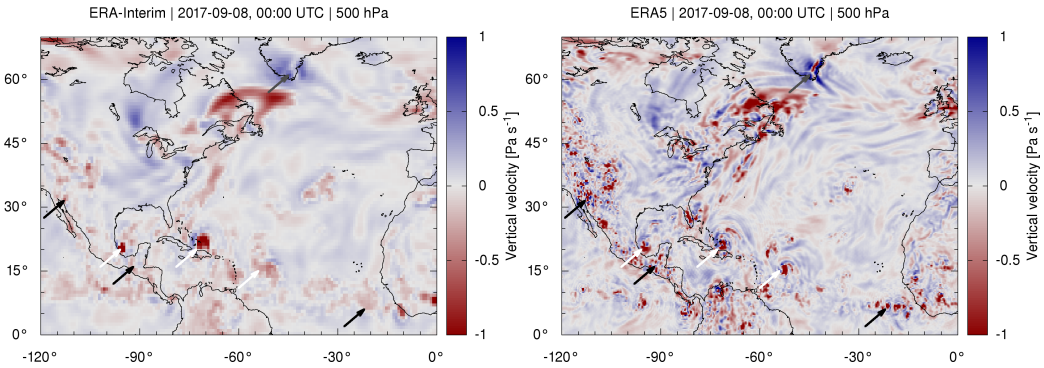
using domain decomposition



(a) ERA-Interim (resolution: 79 km, 60 height levels)

RESOLVING SMALL SCALE TURBULENCE

using domain decomposition



(a) ERA-Interim (resolution: 79 km, 60 height levels)

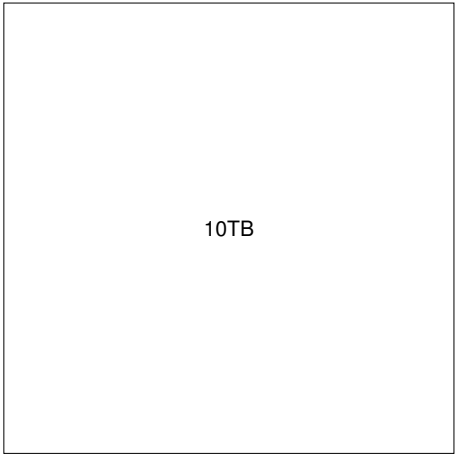
(b) ERA5 (resolution: 31km, 137 height levels)

Source: Hoffmann et al. (2019)

RESOLVING SMALL SCALE TURBULENCE

using domain decomposition

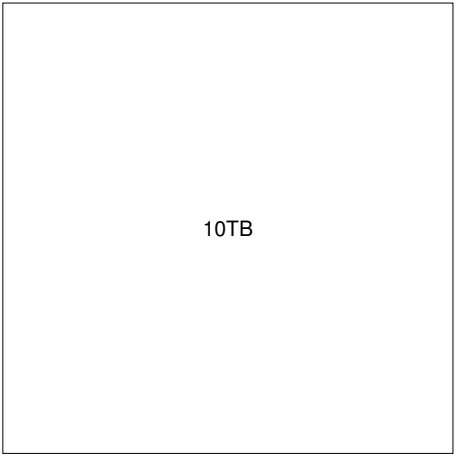
- Target resolution: 1 km $\rightarrow$ $\sim$ 10 TB Filesize



RESOLVING SMALL SCALE TURBULENCE

using domain decomposition

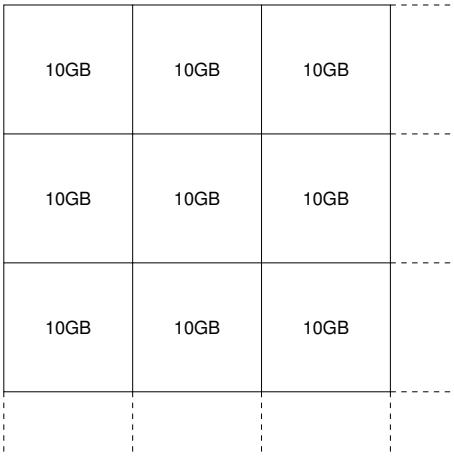
- Target resolution: 1 km $\rightarrow$ $\sim$ 10 TB Filesize
- GPUs only have 40 GB



RESOLVING SMALL SCALE TURBULENCE

using domain decomposition

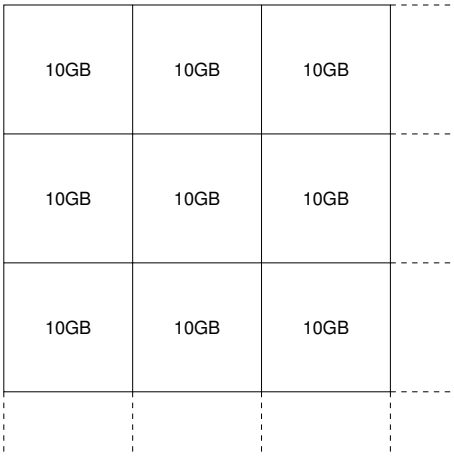
- Target resolution: 1 km $\rightarrow$ $\sim$ 10 TB Filesize
- GPUs only have 40 GB
- Domain decomposition (32×32)



RESOLVING SMALL SCALE TURBULENCE

using domain decomposition

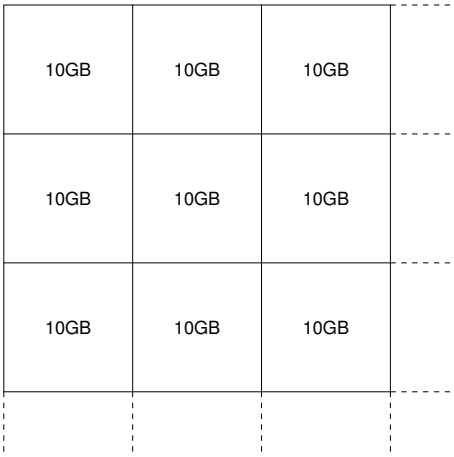
- Target resolution: 1 km $\rightarrow$ $\sim$ 10 TB Filesize
- GPUs only have 40 GB
- Domain decomposition (32×32)
  - $\rightarrow$ each subdomain only has to load $\sim \frac{1}{1024}$ of the full domain
  - $\sim$ 10 GB per GPU



RESOLVING SMALL SCALE TURBULENCE

using domain decomposition

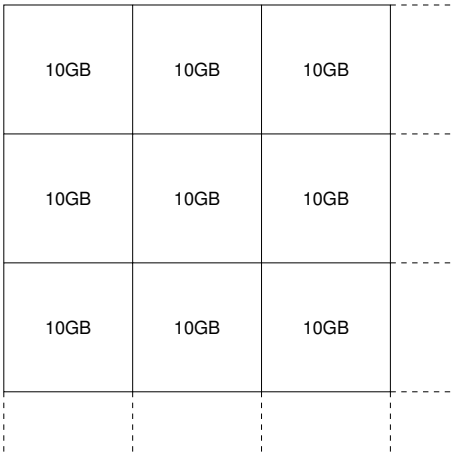
- Target resolution: 1 km $\rightarrow$ $\sim$ 10 TB Filesize
- GPUs only have 40 GB
- Domain decomposition (32×32)
  - $\rightarrow$ each subdomain only has to load $\sim \frac{1}{1024}$ of the full domain
  - $\sim$ 10 GB per GPU
- All particles are assigned to a subdomain



RESOLVING SMALL SCALE TURBULENCE

using domain decomposition

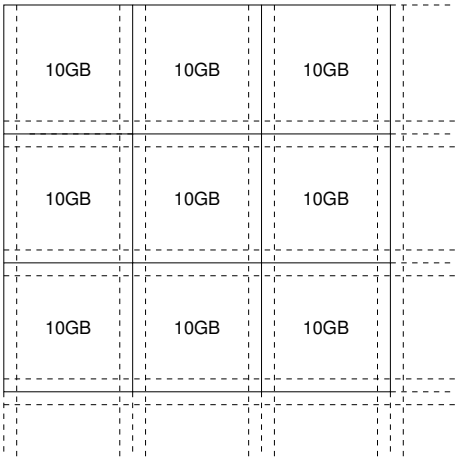
- Target resolution: 1 km $\rightarrow$ $\sim$ 10 TB Filesize
- GPUs only have 40 GB
- Domain decomposition (32×32)
  - $\rightarrow$ each subdomain only has to load $\sim \frac{1}{1024}$ of the full domain
  - $\sim$ 10 GB per GPU
- All particles are assigned to a subdomain
- $n \approx \frac{N}{D}$
  - N : total number of particles
  - D : number of subdomains
  - n : particles per subdomain



RESOLVING SMALL SCALE TURBULENCE

using domain decomposition

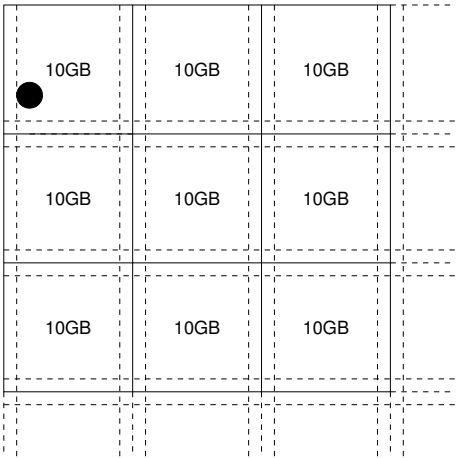
- Target resolution: 1 km $\rightarrow$ $\sim$ 10 TB Filesize
- GPUs only have 40 GB
- Domain decomposition (32×32)
  - $\rightarrow$ each subdomain only has to load $\sim \frac{1}{1024}$ of the full domain
  - $\sim$ 10 GB per GPU
- All particles are assigned to a subdomain
- $n \approx \frac{N}{D}$
  - N : total number of particles
  - D : number of subdomains
  - n : particles per subdomain
- Halos for particles crossing between subdomains



RESOLVING SMALL SCALE TURBULENCE

using domain decomposition

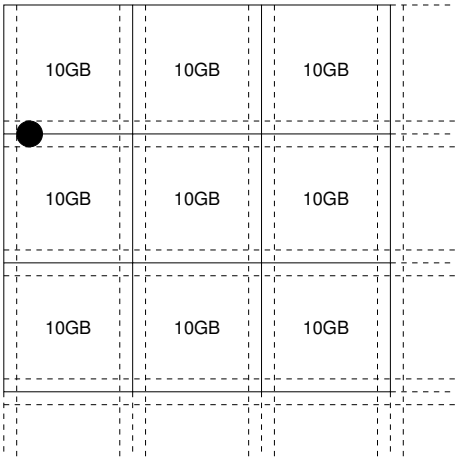
- Target resolution: 1 km $\rightarrow$ $\sim$ 10 TB Filesize
- GPUs only have 40 GB
- Domain decomposition (32×32)
  - $\rightarrow$ each subdomain only has to load $\sim \frac{1}{1024}$ of the full domain
  - $\sim$ 10 GB per GPU
- All particles are assigned to a subdomain
- $n \approx \frac{N}{D}$
  - N : total number of particles
  - D : number of subdomains
  - n : particles per subdomain
- Halos for particles crossing between subdomains



RESOLVING SMALL SCALE TURBULENCE

using domain decomposition

- Target resolution: 1 km $\rightarrow$ $\sim$ 10 TB Filesize
- GPUs only have 40 GB
- Domain decomposition (32×32)
  - $\rightarrow$ each subdomain only has to load $\sim \frac{1}{1024}$ of the full domain
  - $\sim$ 10 GB per GPU
- All particles are assigned to a subdomain
- $n \approx \frac{N}{D}$
  - N : total number of particles
  - D : number of subdomains
  - n : particles per subdomain
- Halos for particles crossing between subdomains



MPTRAC TESTCASE

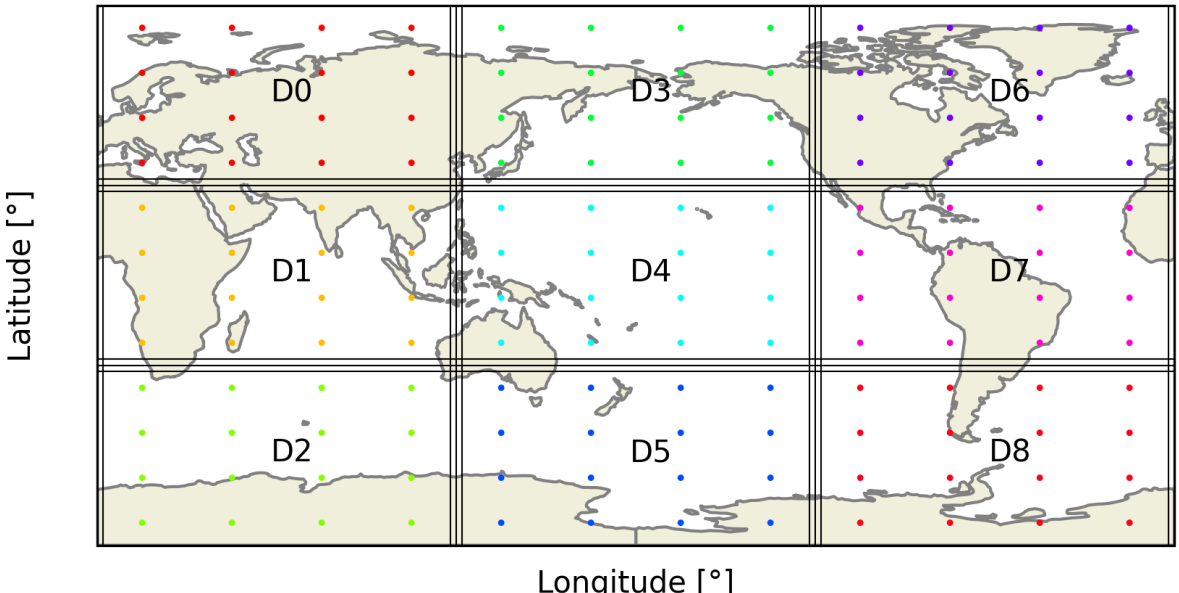
for the domain decomposition

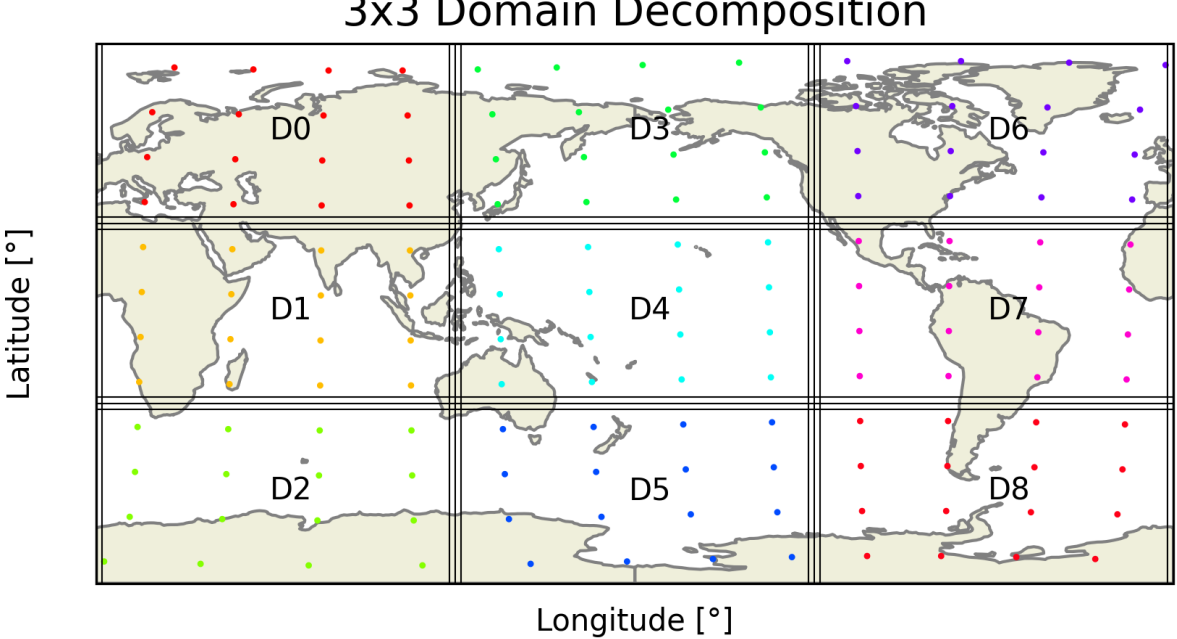
Purpose

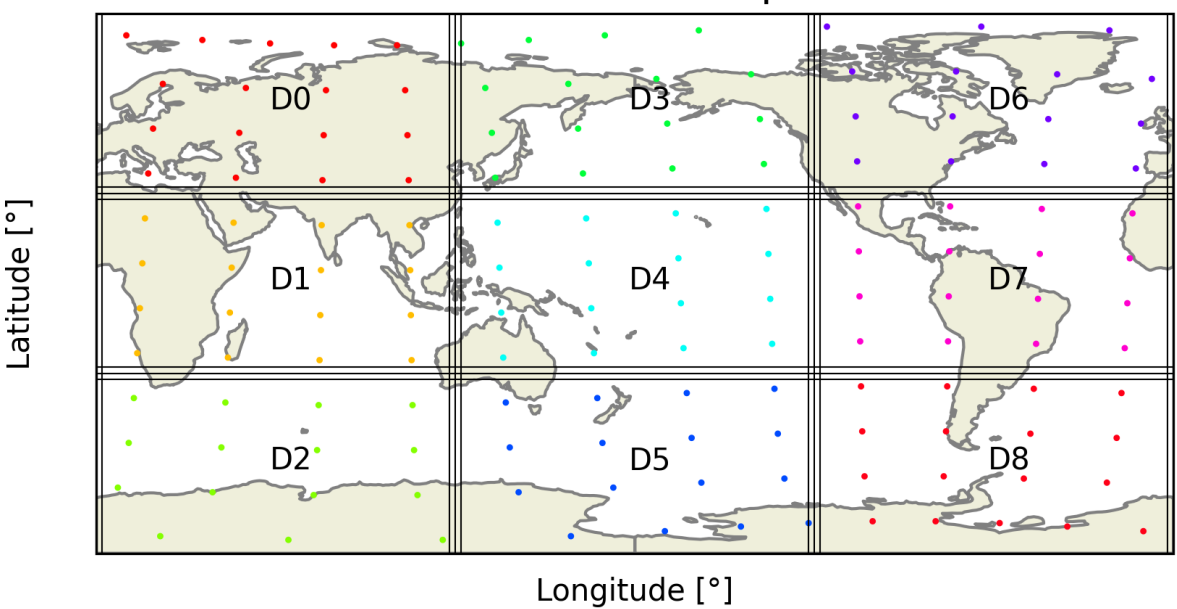
- Testing the domain decomposition code
- Can be used as a basis for scaling tests

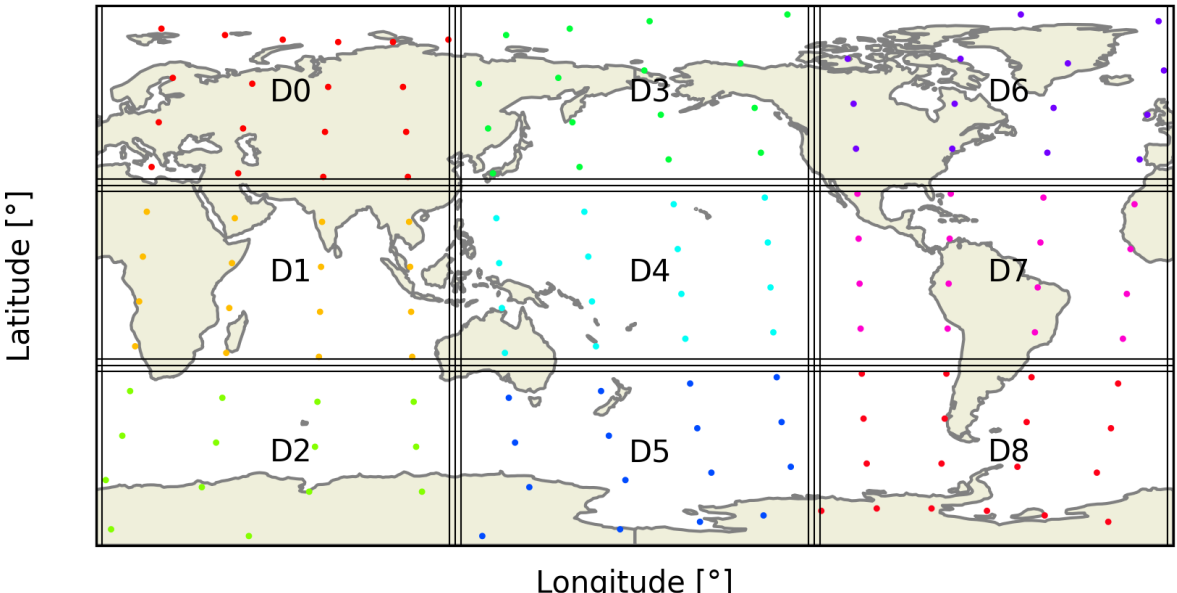
Functionality

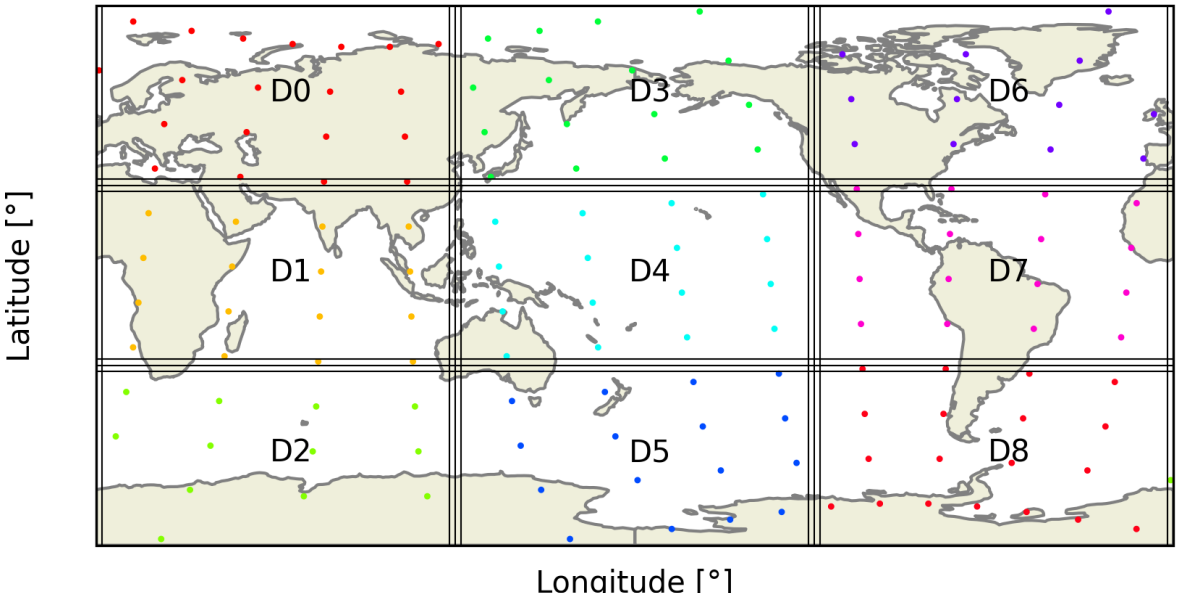
- generate **atmospheric** data (air parcels) for all domains (homogeneous distribution across the globe)
- generate **wind field** (static body rotation around a given axis)
- run MPTRAC with domain decomposition
- work well on **small machines**

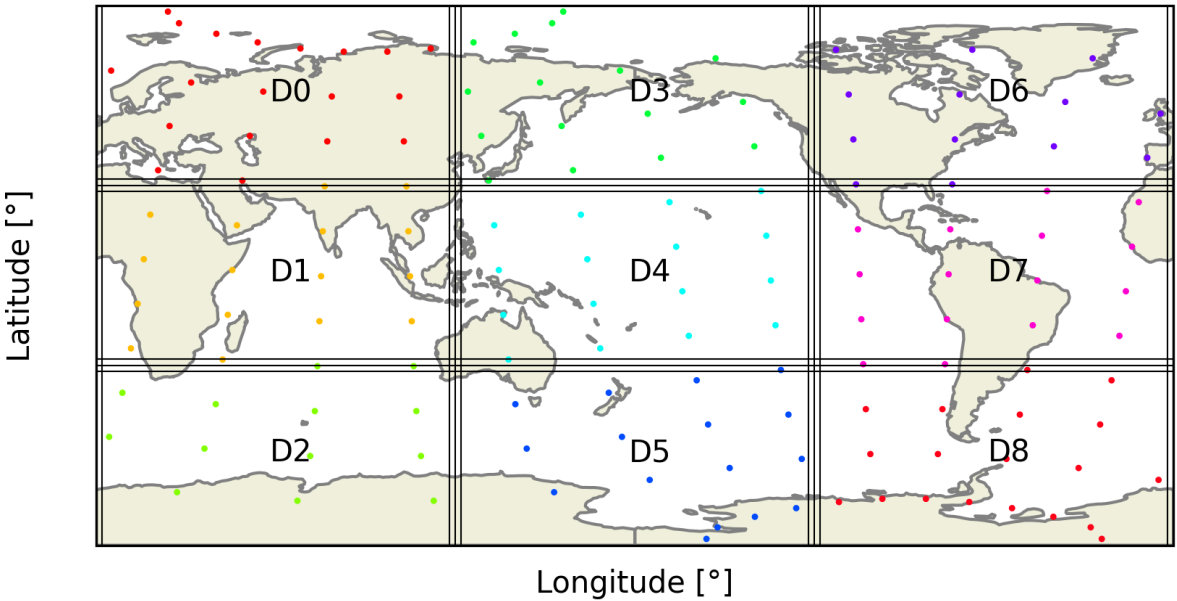

3x3 Domain Decomposition
D0
D3
D6
D1
D4
D7
D2
D5
D8
Latitude [°]
Longitude [°]

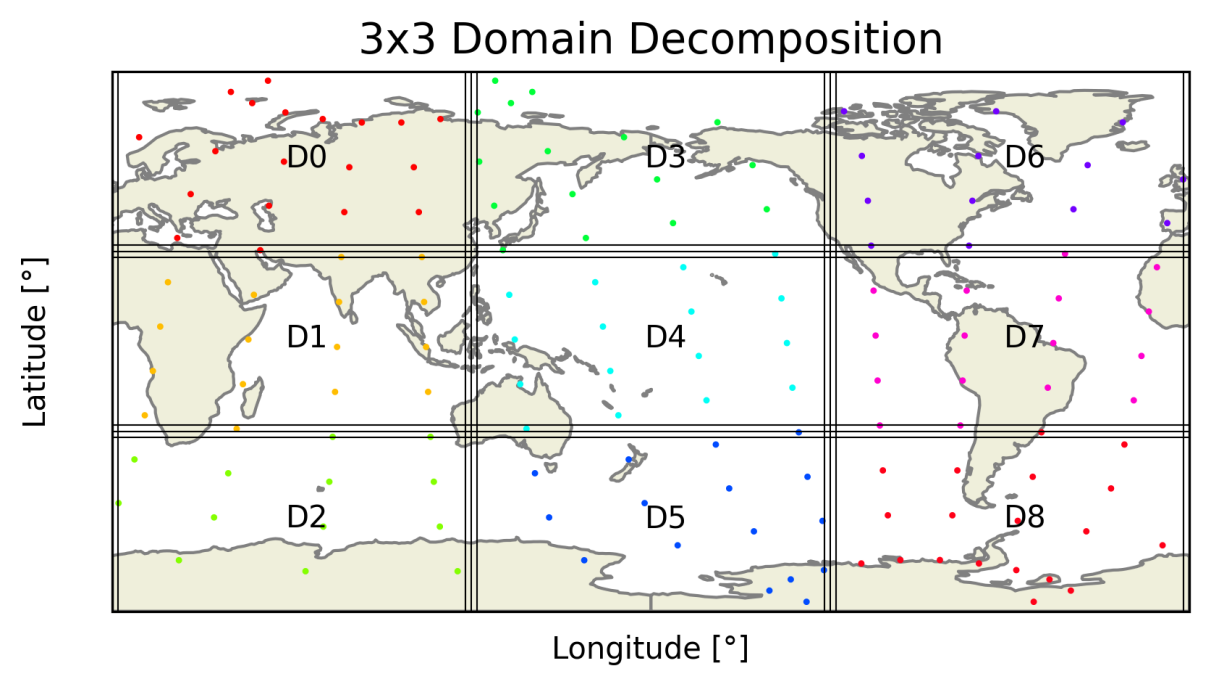

3x3 Domain Decomposition
D0
D3
D6
D1
D4
D7
D2
D5
D8
Latitude [°]
Longitude [°]


3x3 Domain Decomposition
D0
D3
D6
D1
D4
D7
D2
D5
D8
Latitude [°]
Longitude [°]


3x3 Domain Decomposition
D0
D3
D6
D1
D4
D7
D2
D5
D8
Latitude [°]
Longitude [°]

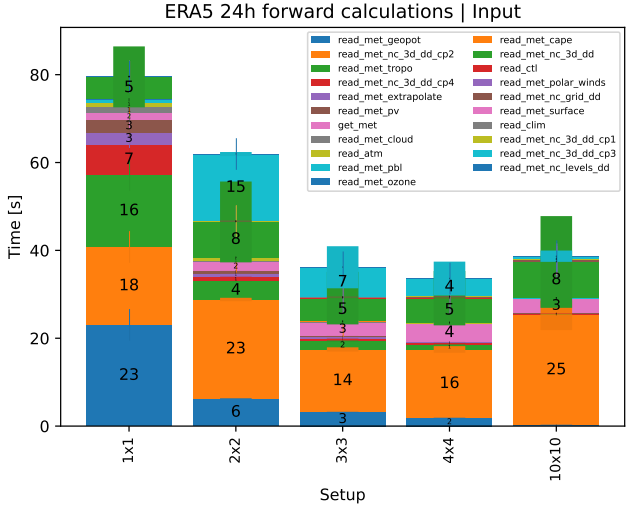

3x3 Domain Decomposition
D0
D3
D6
D1
D4
D7
D2
D5
D8
Latitude [°]
Longitude [°]


3x3 Domain Decomposition
D0
D3
D6
D1
D4
D7
D2
D5
D8
Latitude [°]
Longitude [°]


3x3 Domain Decomposition
D0
D3
D6
D1
D4
D7
D2
D5
D8
Latitude [°]
Longitude [°]

MPTRAC I/O BENCHMARK

for the domain decomposition



JÜLICH Forschungszentrum

MINIMAL REPRODUCIBLE EXAMPLE

NetCDF benchmark

Scaling

NetCDF read calls do not scale well. For parallel reading with multiple nodes on large files a linear increase in the total I/O speed is expected (neglecting Overheads).

MRE

- Isolate the problem in a separate code
- Implement the same domain decomposition as in MPTRAC
- Only NetCDF I/O operations
- Run on the same meteorological files
- Read the files successively
- Read all variables

```
188     for (int f = 0; f < nfiles; f++) {
189         // Open each netCDF file in parallel mode
190         retval = nc_open_par(file_list[f], NC_NOWRITE, MPI_COMM_WORLD, MPI_INFO_NULL, &ncid);
191 >       if (retval != NC_NOERR) { ···
195
196         // Set parallel access mode for all data variables (skip dimension variables)
197 >       for (int varid = 0; varid < nvars+dimvars; varid++) {···
206
207         double file_start = get_time_sec();
208         for (int varid = 0; varid < nvars+dimvars; varid++) {
209             if (is_dimvar[varid]) continue;
210             // Read the subdomain for this variable
211             start[lat_idx] = base_lat0;
212             start[lon_idx] = base_lon0;
213             count[lat_idx] = base_lat1-base_lat0+1;
214             count[lon_idx] = base_lon1-base_lon0+1;
215             retval = nc_get_vara_float(ncid, varid, start, count, buffer);
216 >           if (retval != NC_NOERR) {···
220             buffer[0] *= 3.4;
221             // Read periodic halo if applicable
222             if (halo > 0 && has_periodic_halo) {
223                 start[lon_idx] = periodic_halo_lon_start;
224                 count[lon_idx] = halo;
225                 retval = nc_get_vara_float(ncid, varid, start, count, buffer);
226 >               if (retval != NC_NOERR) {···
230                 buffer[0] *= 3.4;
231             }
232         }
233         nc_close(ncid);
234         MPI_Barrier(MPI_COMM_WORLD);
235         double file_end = get_time_sec();
236         file_times[f] = file_end - file_start;
237     }
```

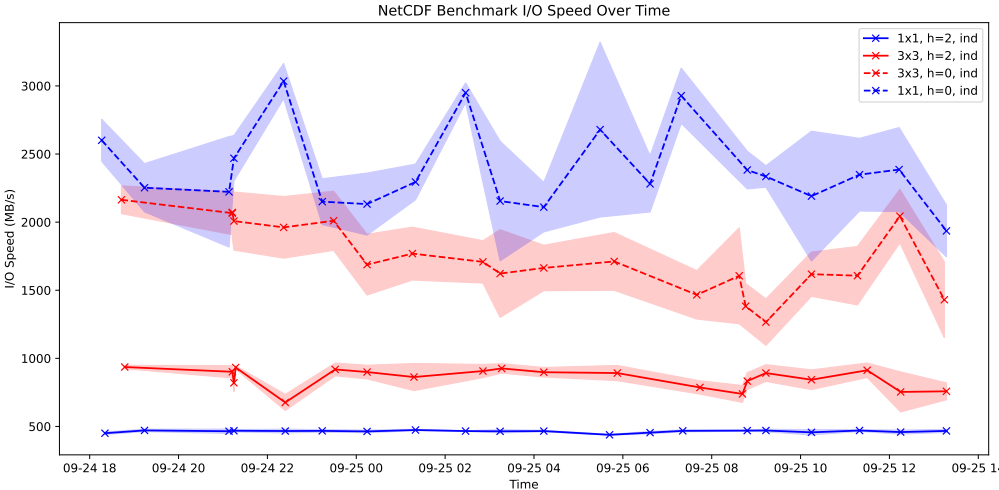
NETCDF BENCHMARK RESULTS

comparison with MPTRAC benchmarks

- MPTRAC I/O Speed for 25 files with 1.318 GB each:
  - 1x1 $\rightarrow 0.41 \frac{\text{GB}}{\text{s}}$
  - 3x3 $\rightarrow 0.89 \frac{\text{GB}}{\text{s}}$
- NetCDF benchmark I/O Speed for the same files:
  - 1x1 $\rightarrow 0.46 \frac{\text{GB}}{\text{s}}$
  - 3x3 $\rightarrow 0.85 \frac{\text{GB}}{\text{s}}$
- good overall agreement
- for 9 Nodes we only have an I/O speed increase of a factor of ~ 2
- what is the problem?

NETCDF BENCHMARK SERIES

with and without halo reading

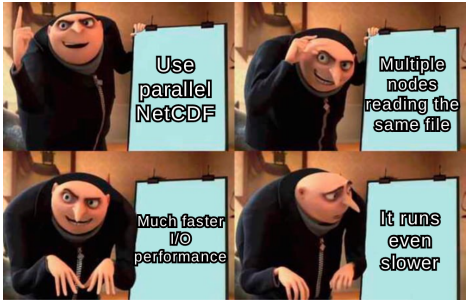


JÜLICH Forschungszentrum

NETCDF BENCHMARK

conclusions

- NetCDF I/O is **slower** than expected and **doesn't scale**
- Using no halos when reading increases the base I/O speed
- Files might not be large enough (but benchmarks for 5 GB files show still no scaling)
- parallel NetCDF seems to be **unsuitable** for the Reading of the meteorological files for MPTRAC



MPTRAC SCALING

without I/O

Default

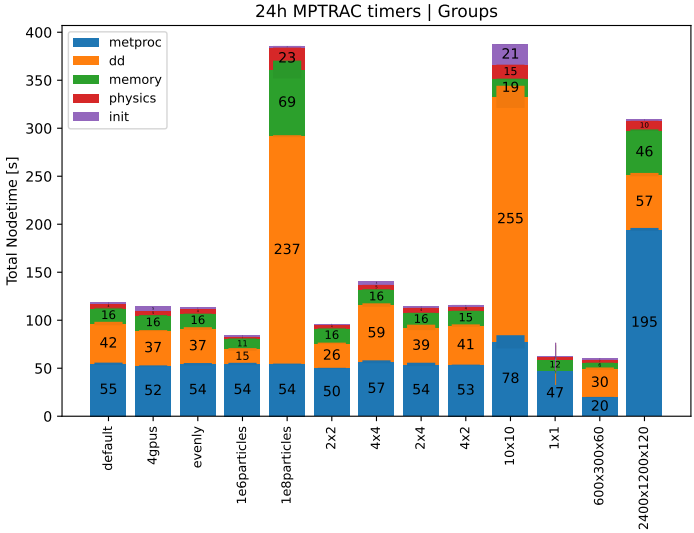
- 1 GPU per node
- unevenly distributed
- 1e7 particles
- 3x3 domains
- grid: 1200x600x120, dt_mod: 180 s

- **Vary** the parameters from the default
- For each case we compute the total nodetime as the **sum of the timers** over the nodes. For a perfectly parallel code this would be a constant.

MPTRAC SCALING RESULTS

without I/O

- 10×10 subdomains are ~ 15 times faster than 1×1
- physics calculations scale very well
- Communication overhead limits scalability
- processing of the meteo data does not scale



OUTLOOK

Scaling requirements

- high resolution of the meteorological data ($\sim 40000 \times 20000 \times 137$) $\Rightarrow$ 10 TB filesize
- for each subdomain:
  - grid: $\sim 1100 \times 550 \times 137$
  - 10^6 particles
- 36×36 subdomains
- 1296 GPUs needed
- 324 Nodes needed (5 % of Jupiter)

Issues

Currently the domain decomposition has problems at the poles, where all subdomains neighbor each other, which results in errors for ≥ 3 subdomains at a pole

SUMMARY

Conclusions

- Domain decomposition is working as the testcase shows
- Biggest issue is the NetCDF file reading
  - no scaling for parallel reading with multiple nodes
  - overall bad performance even in the minimal reproducible example
- Scaling of the domain decomposition not perfect

Further work

- Do simulations with $1100 \times 550 \times 137$ gridpoints per subdomain and 10^6 particles for different number of subdomains (1×1 / 2×2 / 3×3 / 4×4 / 6×6 / 10×10)
- Fix issues at the poles
- Further benchmark the I/O NetCDF performance using the MRE or explore different I/O systems