Numerical Simulation Methods in Geophysics, Part 13: Summary

1. MGPY+MGIN

thomas.guenther@geophysik.tu-freiberg.de

TU BERGAKADEMIE FREIBERG
TUBAF
Die Ressourcenuniversität.
Seit 1765.

Recap

- learned basics of FD, FE and FV methods
- solved different PDE types:
  - elliptic Poisson (potential): heat, flow, DC, gravity, mag)
  - parabolic (diffusion): heat transfer, time-domain EM
  - hyperbolic (waves): seismics, GPR (just 1D)
  - elliptic Helmholtz: frequency-domain EM (or heat)
- script in development on tubaf-em.github.io/Numerical_Simulation

The methods

1. The Finite Difference (FD) method
   - elliptic: Poisson equation in 1D, look into 2D/3D
   - parabolic: diffusion equation in 1D, time-stepping
   - hyperbolic: acoustic wave equation in 1D
2. The Finite Element (FE) method
   - Poisson equation in 1D & 2D
   - complex Helmholtz equation in 2D for EM problems
   - solving EM problems and computational aspects
3. The Finite volume method: just looking

The methods

The Finite Difference method

approximates the partial derivatives by difference quotients (beware Δx and Δa)

The Finite Element method

approximates the solution through base functions in integrative sense

The Finite Volume method

approximates the solution by piecewise constant values and ensures conservation law by fluxes

Take-aways

1. You always get a numerical solution but it can be crap.
2. FE is the most flexible and wide-spread method.
3. FD & FV simple to implement, FV keeps continuity (transport!).
4. FD parameter weighting in 2D/3D problems $\Rightarrow$ inaccuracies
5. Spatial refinement or higher polynomial order improve accuracy, particularly at source positions with strong curvature.
6. Sometimes the boundaries need to be set really far away.
7. Secondary field computation partly overcomes the latter two.
8. Implicit or mixed time-stepping more accurate and stable than explicit

Next-generation FE: Oskar

- built upon pyGIMLi v2.0

Simplest modelling example

```
from oskar ScalarSpace, VectorSpace, laplace, div, grad
s = ScalarSpace(mesh, name='T', p=2)
T = solve(-laplace(s) == 1, bc={'Dirichlet':{'*':0}})
```

Step by step

```
PDE = lambda u: -div(alpha*grad(u))
A, rhs = (PDE(s) == f).weakForm.assemble()
```

Post-processing

```
j = a * grad(u)
```

What’s left to mention?

Accuracy

1. Verification methods
2. Error estimation and goal-oriented refinement

Discontinuous Galerkin method

typical for hyperbolic problems

weak form of wave equation with fluxes (FV)

$$M\partial_t q(t) - A^T q(t) = -F(a, q(t))$$

$$\Rightarrow \partial_t q = \mathbf{M}^{-1}(A^T q(t) - F(a, q(t)))$$

locally for each element & communication through fluxes (like in FV)

Spectral element method

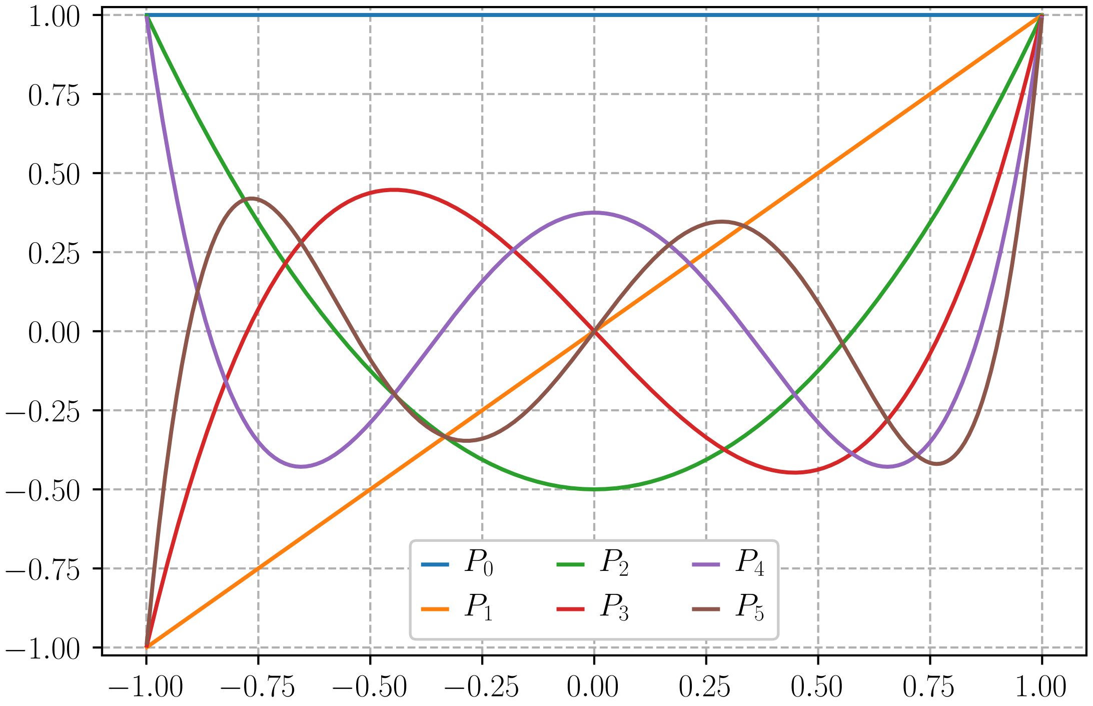
typically used for global wave phenomena

$$u = \sum u_i \phi_i(\mathcal{P}r)$$

ϕ Lagrangian polynoms

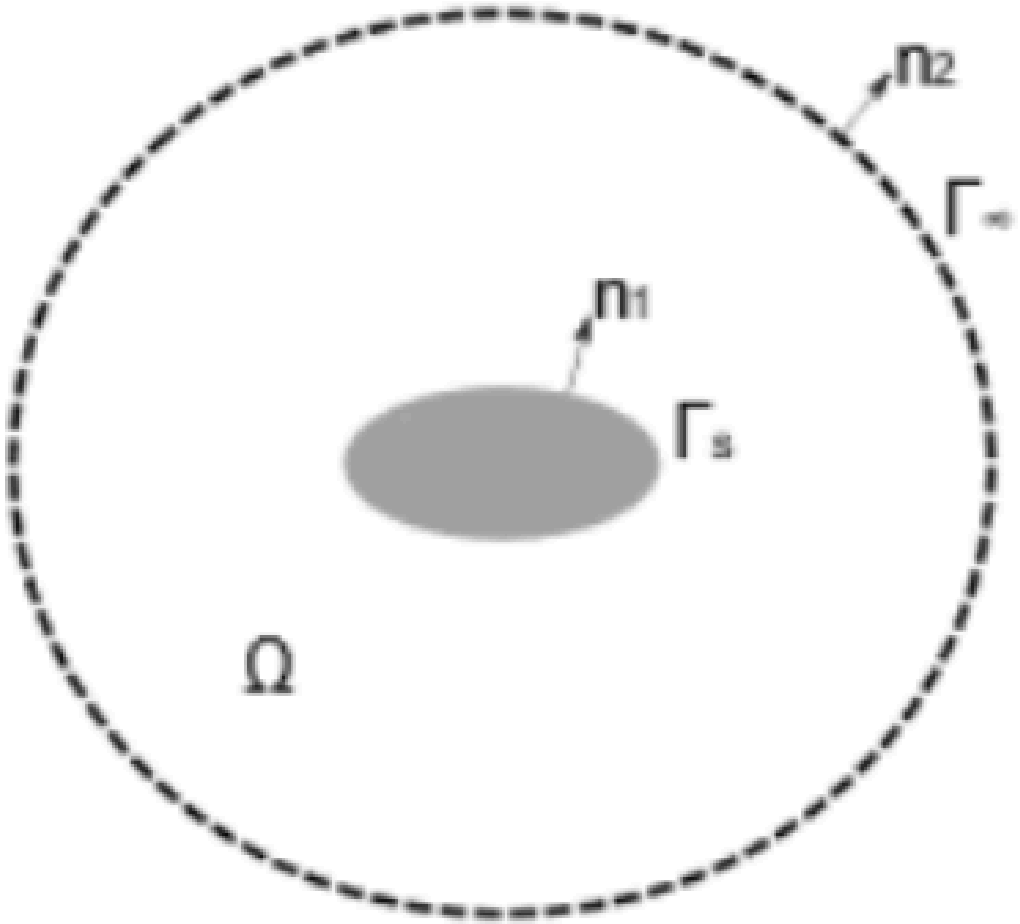
$$l_i^N = \prod_k^N \frac{\xi - \xi_k}{\xi_i - \xi_k}$$

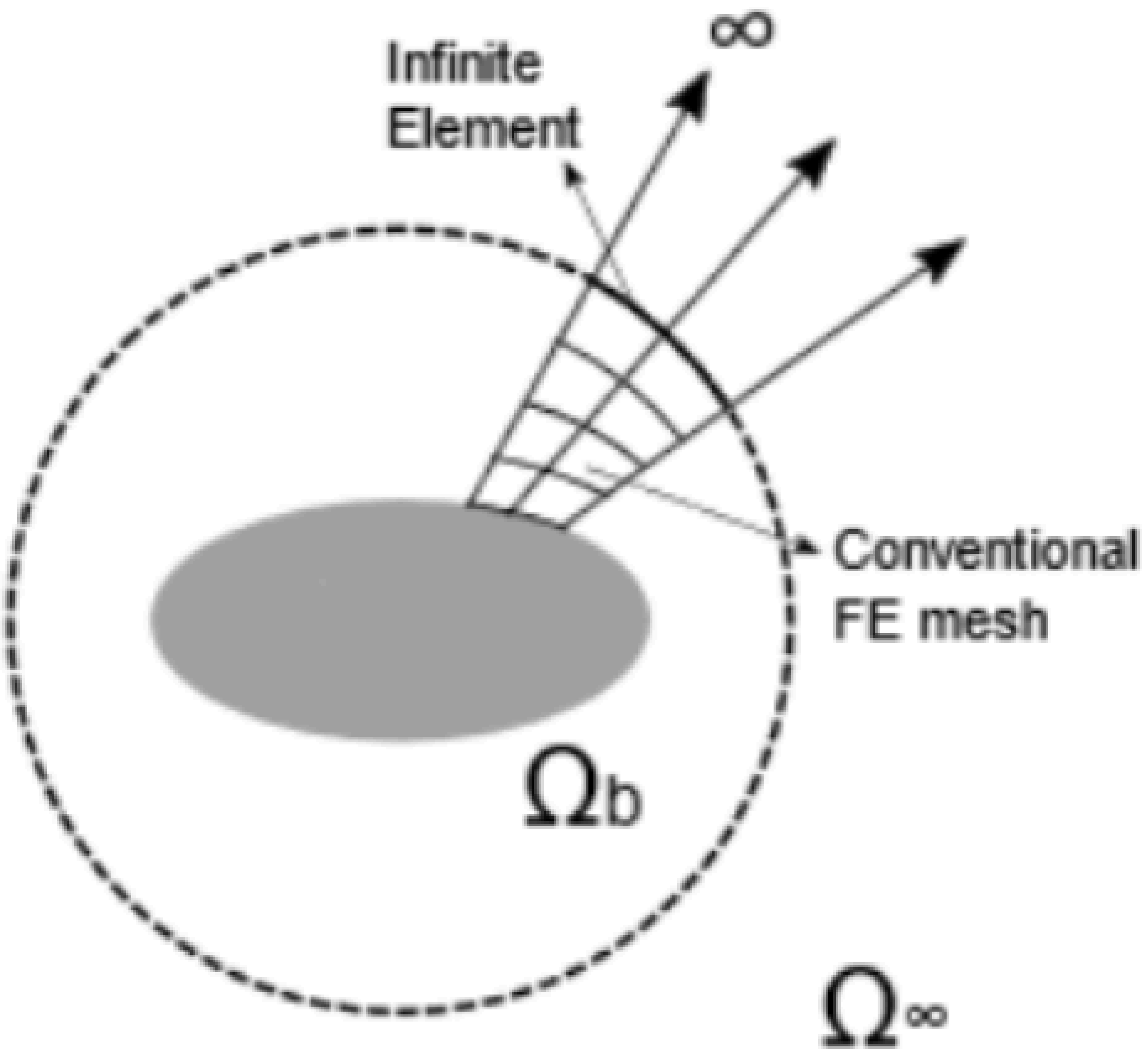
or Chebychev polynoms



First six Lagrangian polynomials

Infinite Elements





Meshless modelling

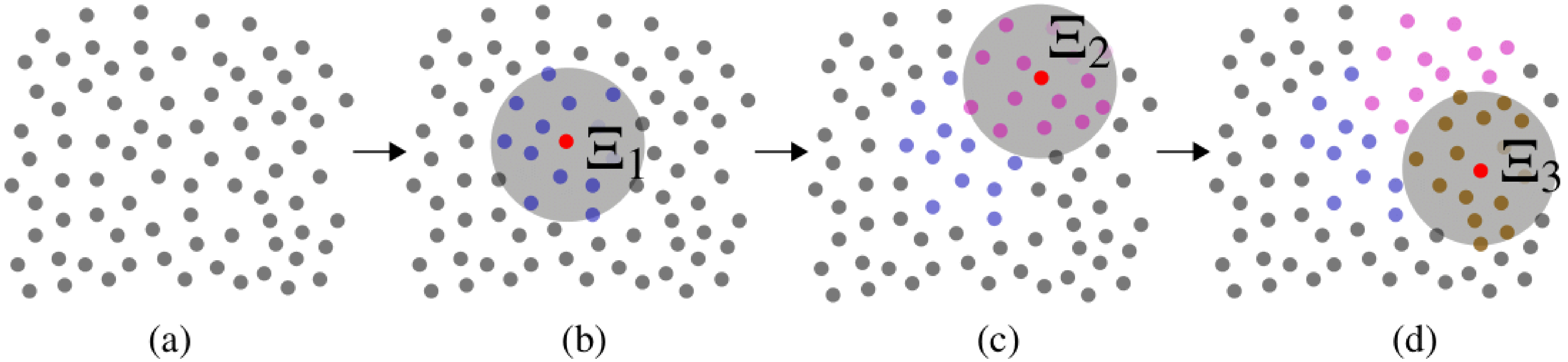


Figure 3.5: Overview of meshless computational scheme. From a set of points (a) local subsets (stencils) Ξ_i $i = \{1, 2, 3\}$ *are selected as displayed in (b) to (c) to construct a meshless approximation.*

Meshless divergence operator (Wittke, 2017)

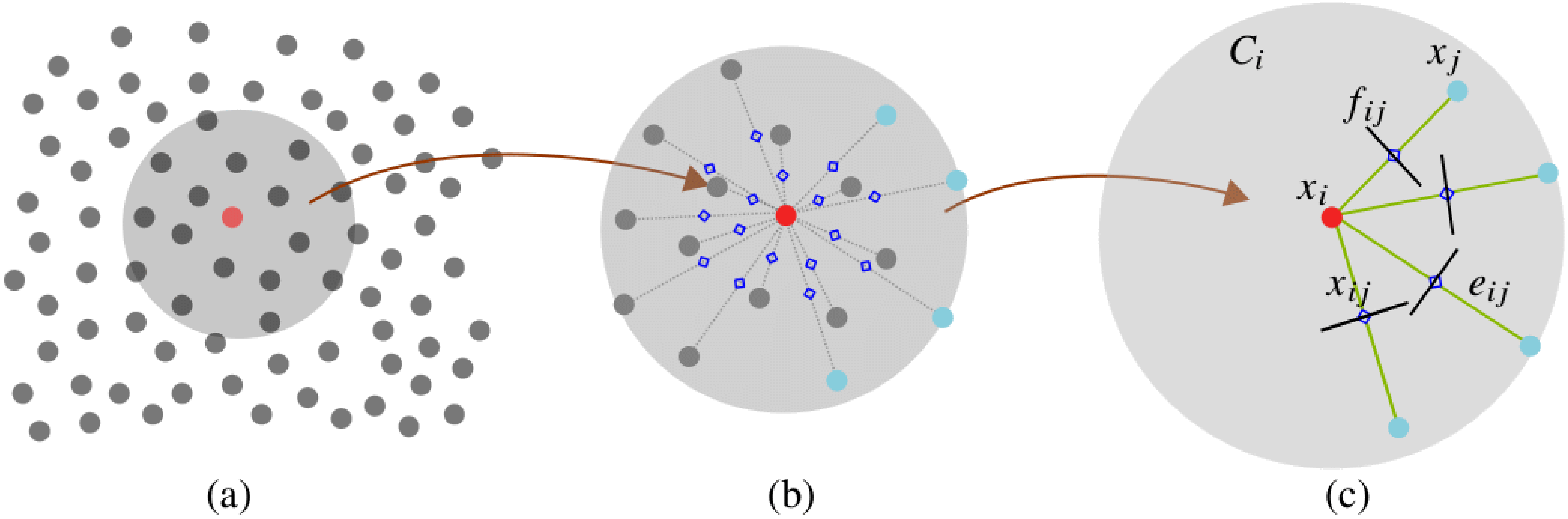
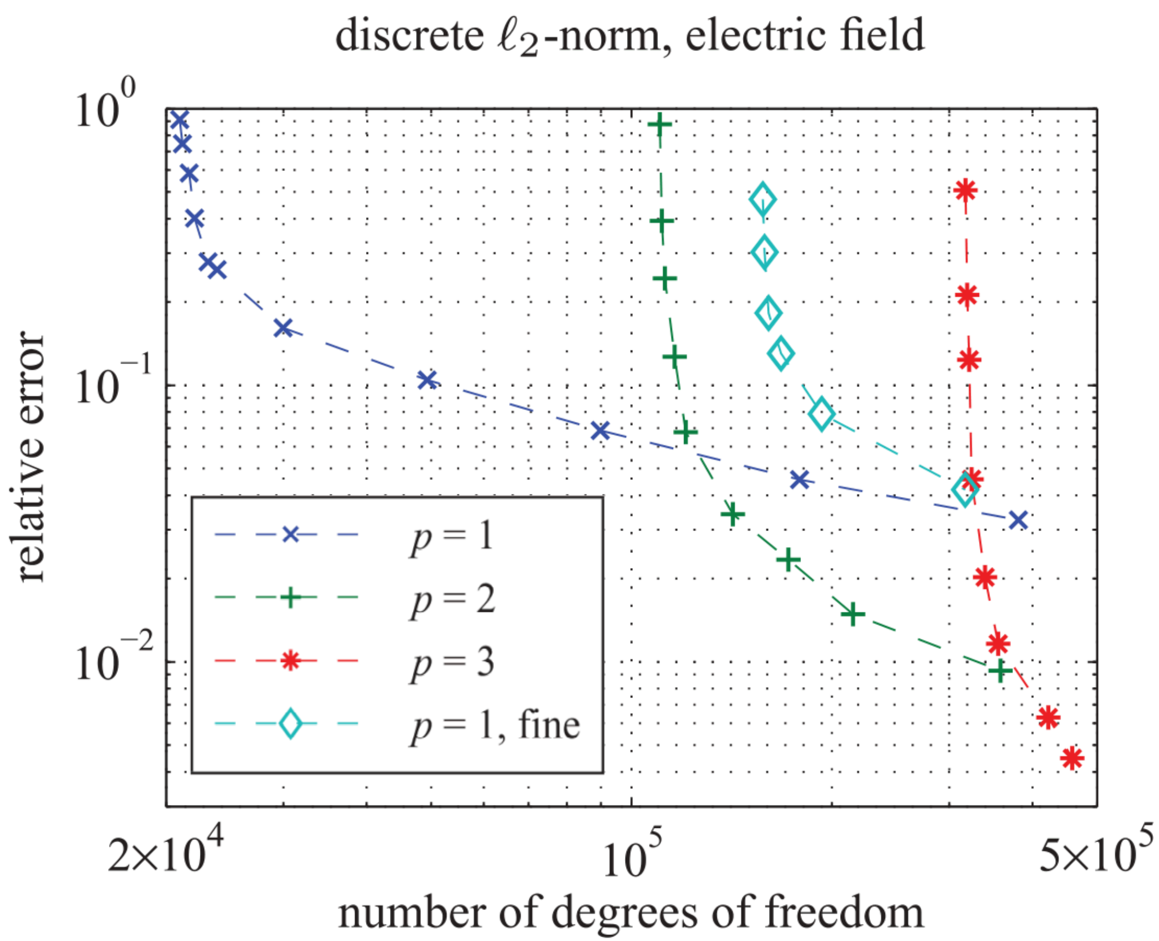


Figure 3.6: Stencil-wise construction of a local primal-dual grid complex. (a) Selecting enough points to form a stencil in a predefined neighbourhood. (b) Definition of midpoints (blue diamonds) from the central point to all other points. (c) Definition of the primal edges e_{ij} and dual faces f_{ij} inside a dual cell C_i from the definition of midpoints x_{ij}.

The method of exact solutions (MES)

- find domain or condition for which an analytical solution exists
- often for Dirac type sources ⇒ solution is a **Green’s function**
- integrate over Greens function for extended sources
- convergency tests (refine spatially and polynomially)



MES example from Oskar

The method of manufactured solutions (MMS)

- use any function or distribution for solution $\mathbf{u}$
- create right-hand-side vector $\mathbf{f}$ using FEM
- solve for $\mathbf{u}$ and compare with

MMS example from Oskar

Error estimation and mesh refinement

- get idea of accuracy of the solution
- refinement of cells with high error (e.g. large gradients)
- comparison between successive refinement solutions

Error estimation (residual-based)

Poisson problem $-\nabla^2 = f$ with bilinear form $a(u, v) = \int \boldsymbol{\nabla} u \boldsymbol{\nabla} v \mathrm{d}\Omega$

finite-dimensional function space V_h: $a(u_h, v_h) = l(v_h)$

estimate error e_h in bilinear form $a(e_h, v) = a(u, v) - a(u_h, v)$

residual $R = f + \nabla^2 u_h$ leads to $a(e_h, v) = \sum_c \int_{\Omega_c} R v \mathrm{d}\Omega_c$

Error estimation (recovery-based)

gradients across element boundaries tend to be discontinuous

compare original (unsmoothed) gradient of the solution with improved

$$(E_h)^2 = \int |M(u_h) - \boldsymbol{\nabla} u_h|^2 \mathrm{d}\Omega$$

M obtained by smoothing over patch of elements around each element

Goal-oriented mesh refinement

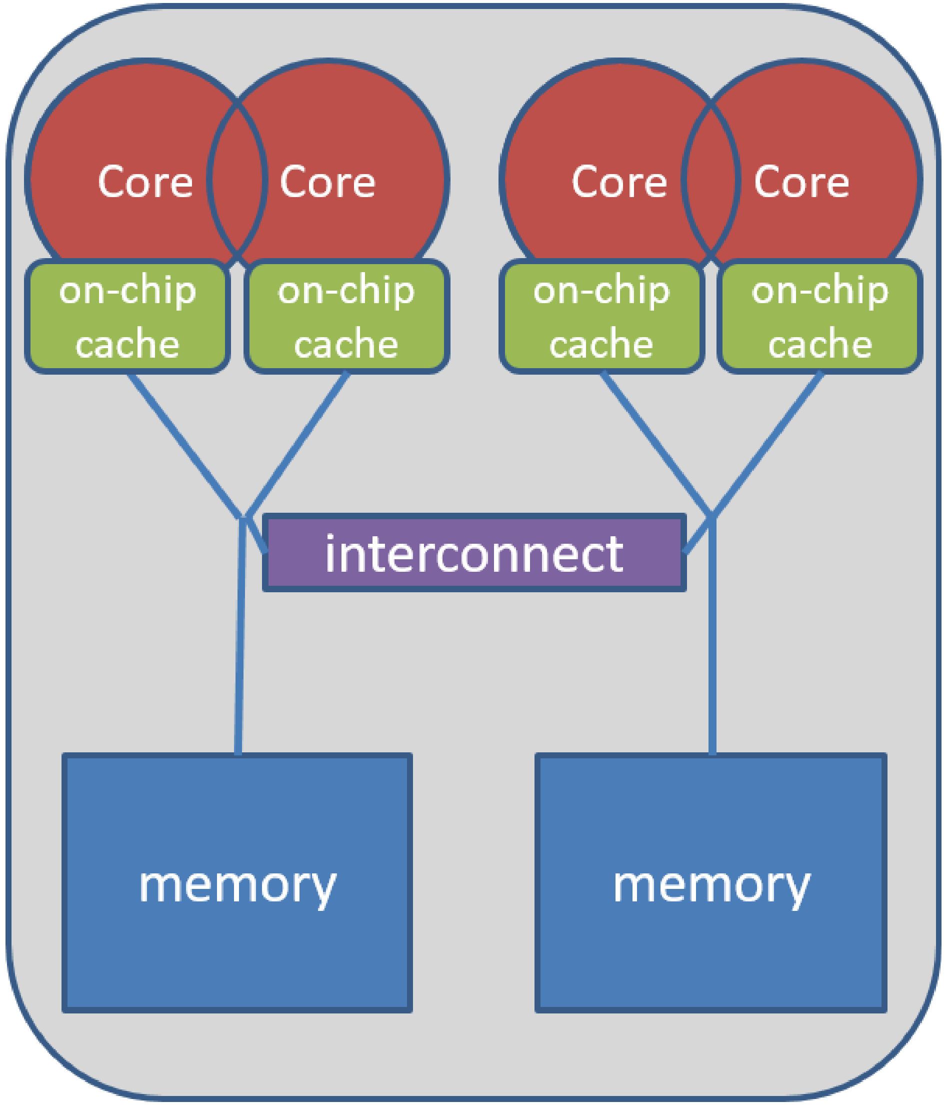
primal and dual (adjoint) problem (with receiver as hypothetical source)

inner product of solutions

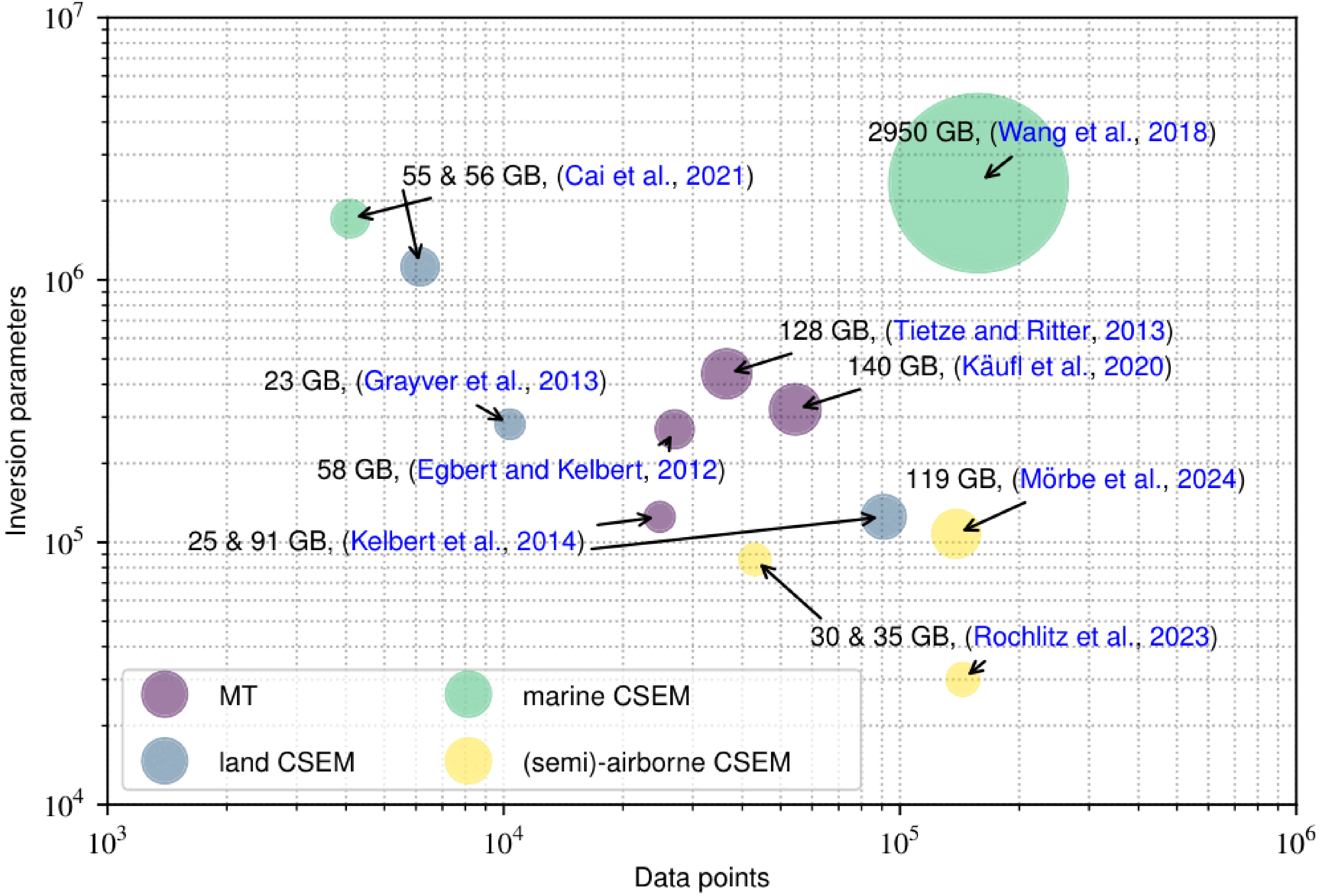
$$\Phi_{lmn} = \int_{c_n} \mathbf{F}^l \mathbf{F}^m \mathrm{d}\Omega_n$$

Performance in Parallel

- Desktop computers (1 CPU with many cores)
- fat node (shared memory) architecture
- cluster (distributed) architectures:
  - NUMA (non-uniform memory access)

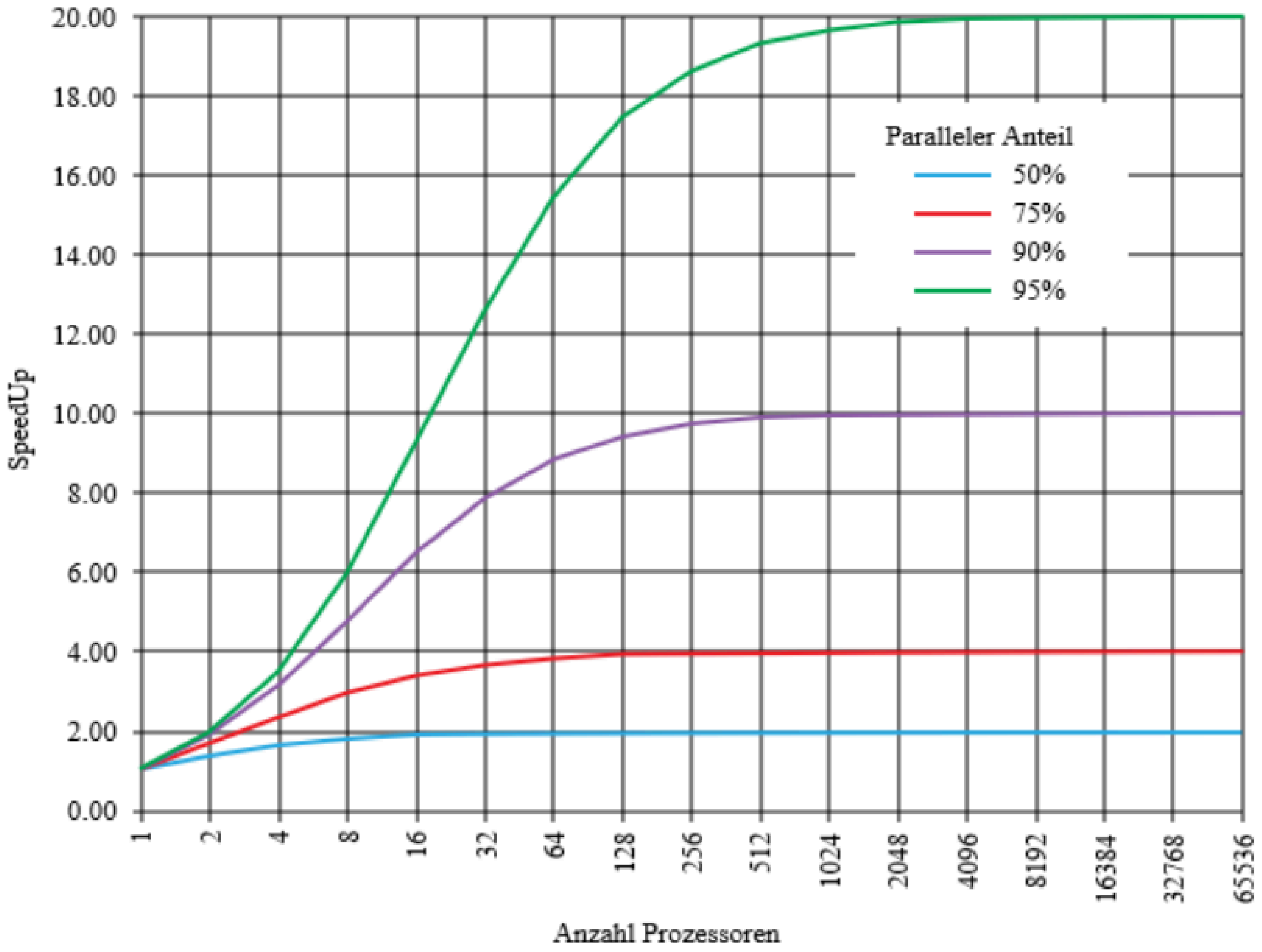


Computational size

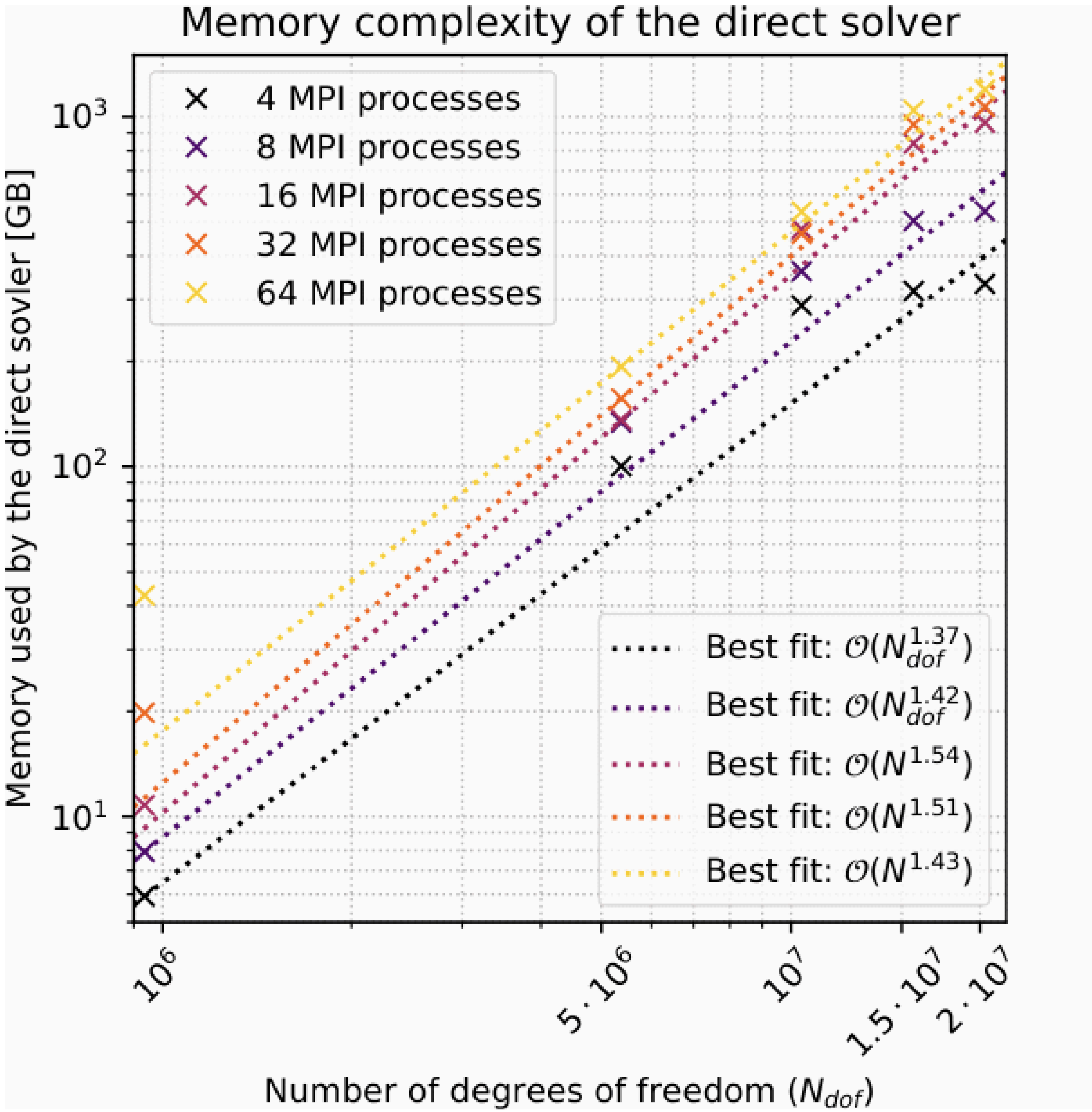


Amdahl’s law

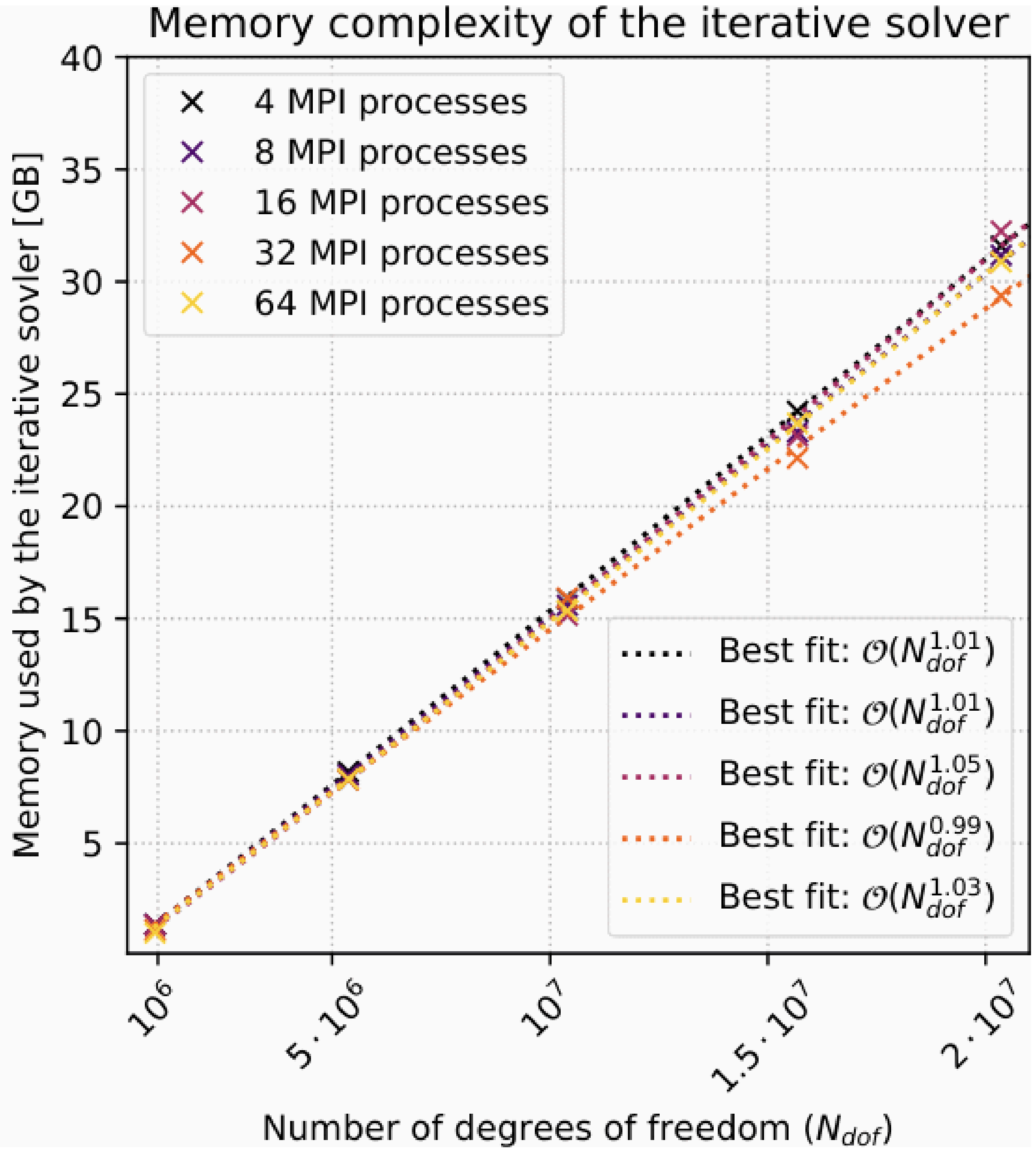
Distribute runtime into serial and parallel part $t = t_s + t_p/N(+t_c(N))$



Perfomance analysis - Memory

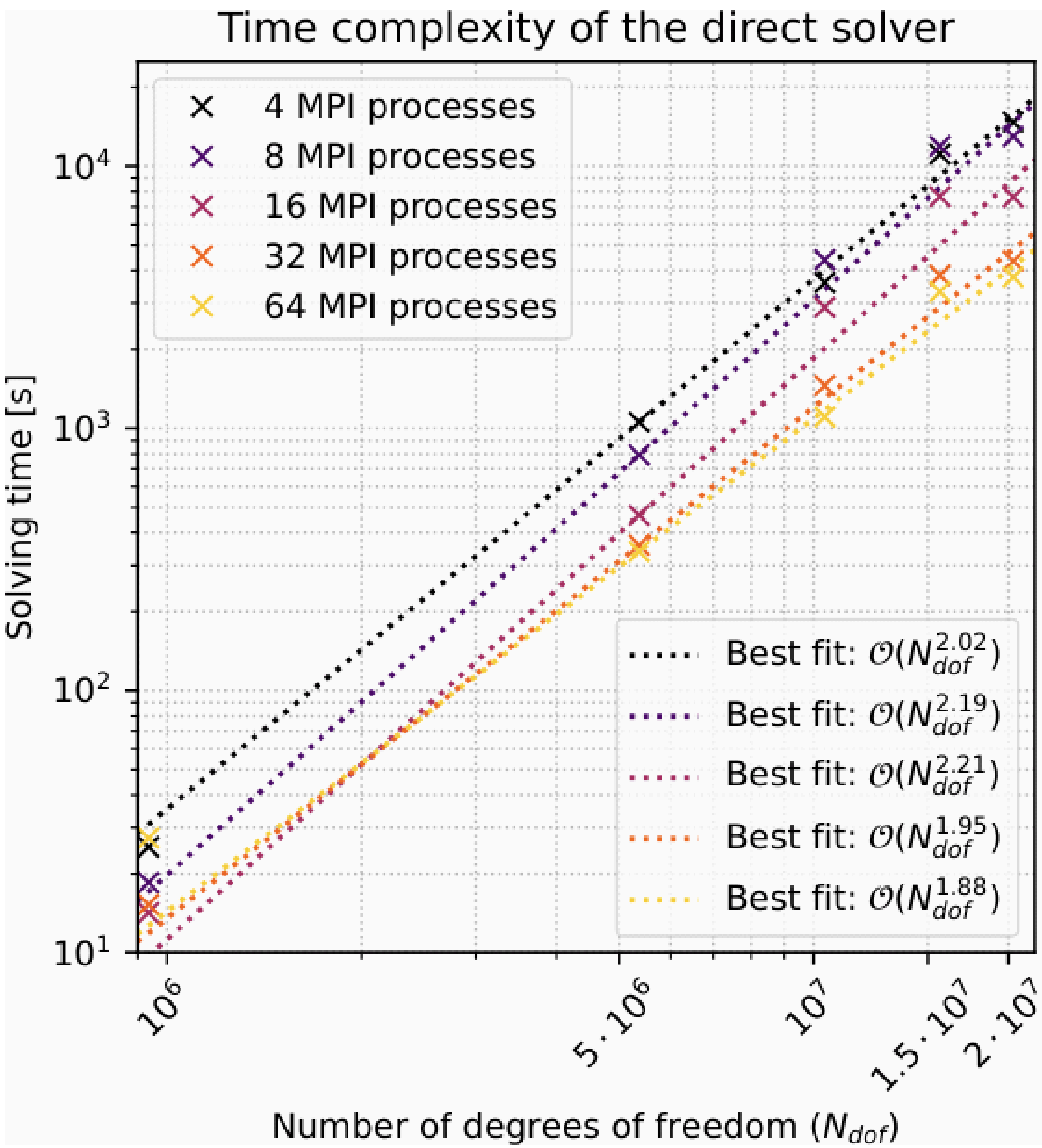


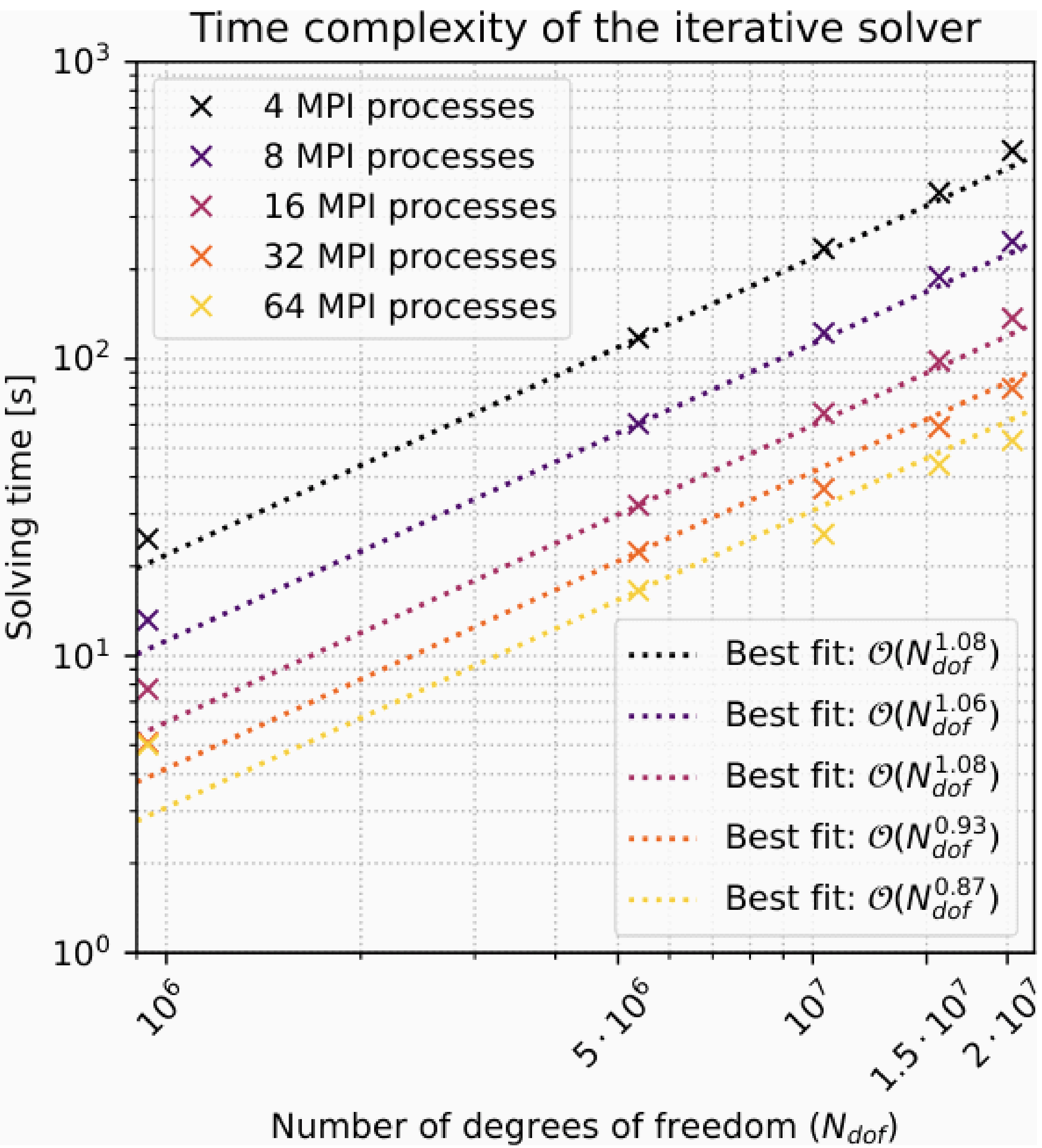
Memory complexity of the direct solver



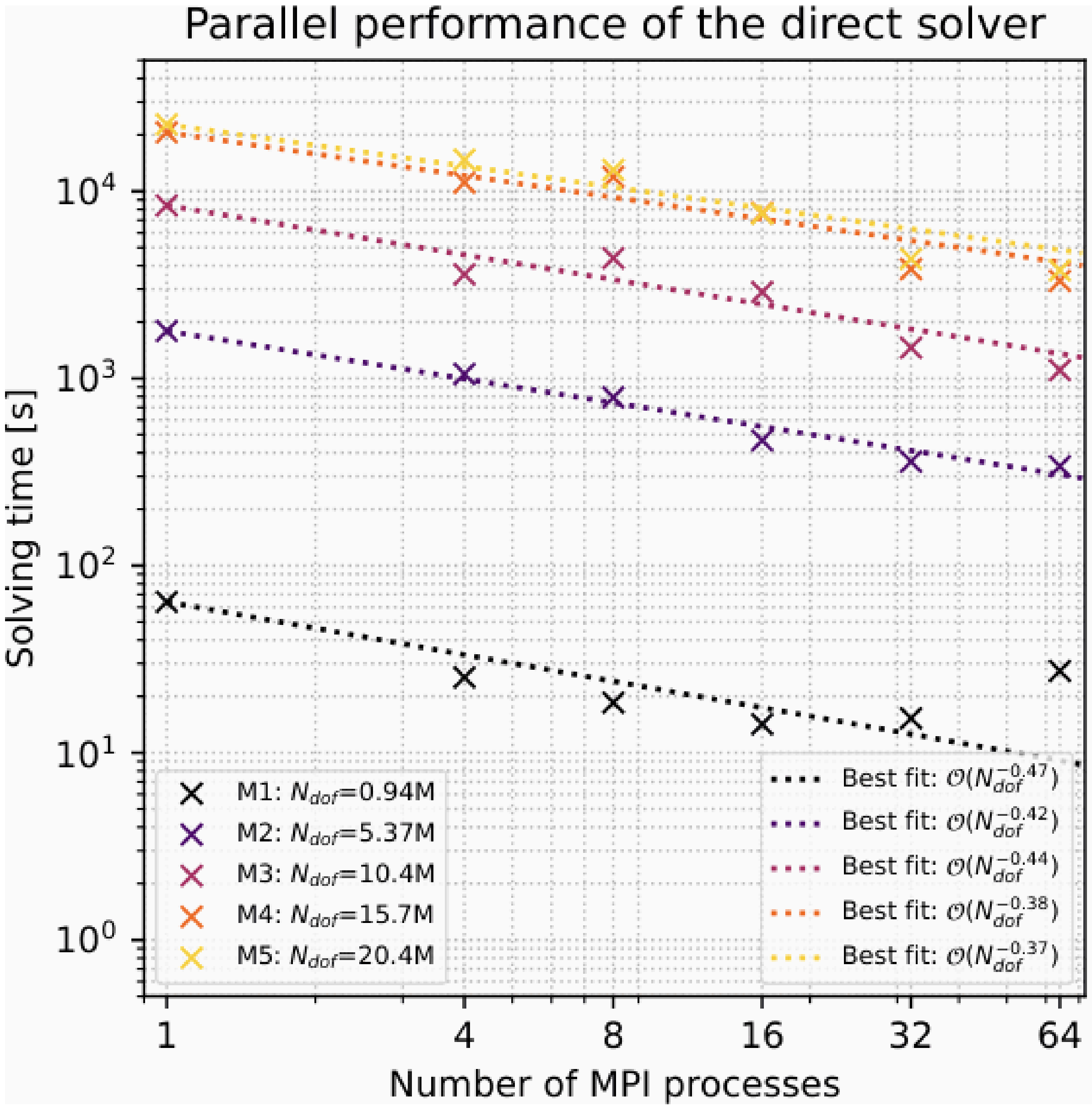
Memory complexity of the iterative solver

Perfomance analysis - Time

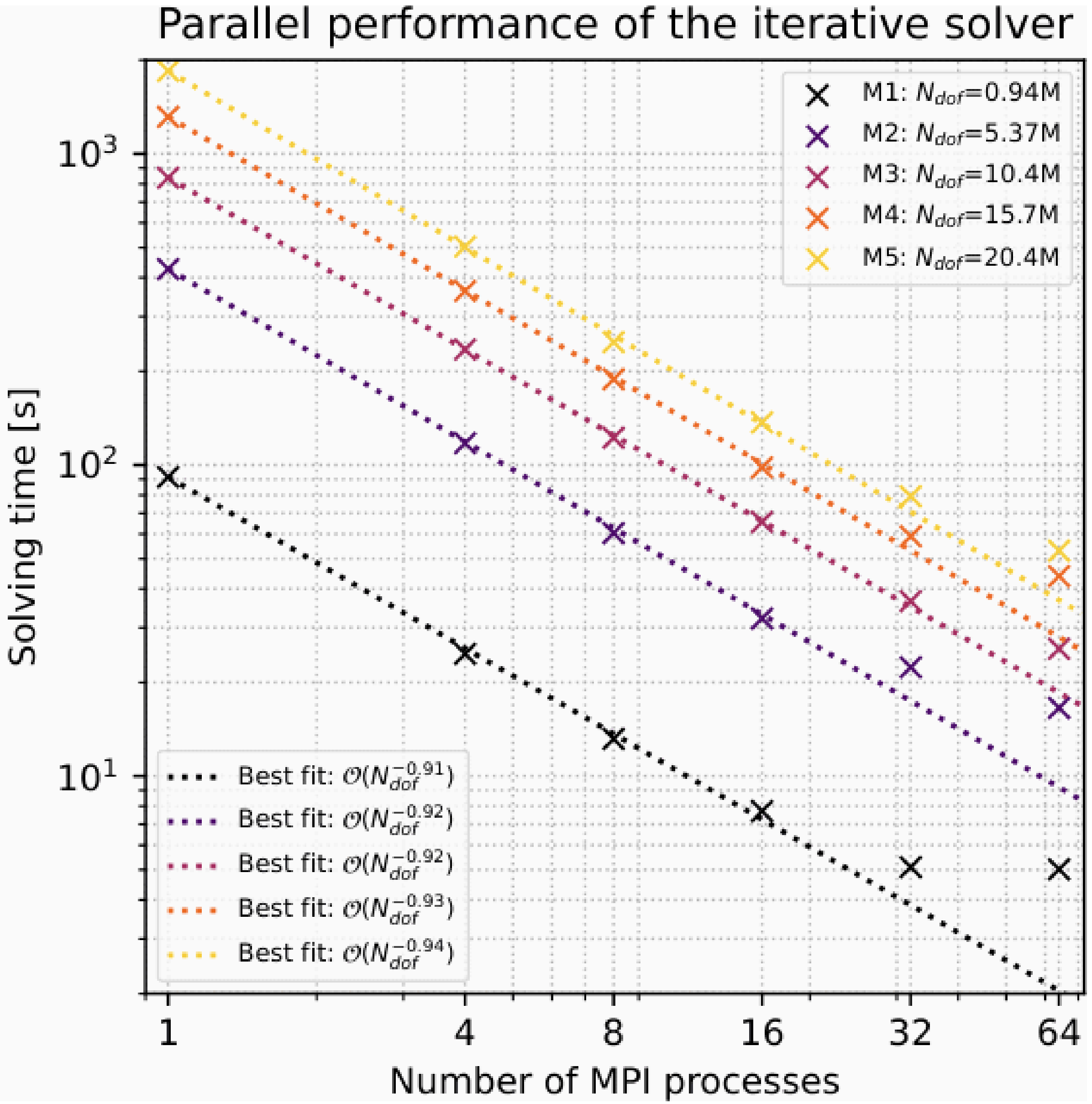




Perfomance analysis - MPI scaling

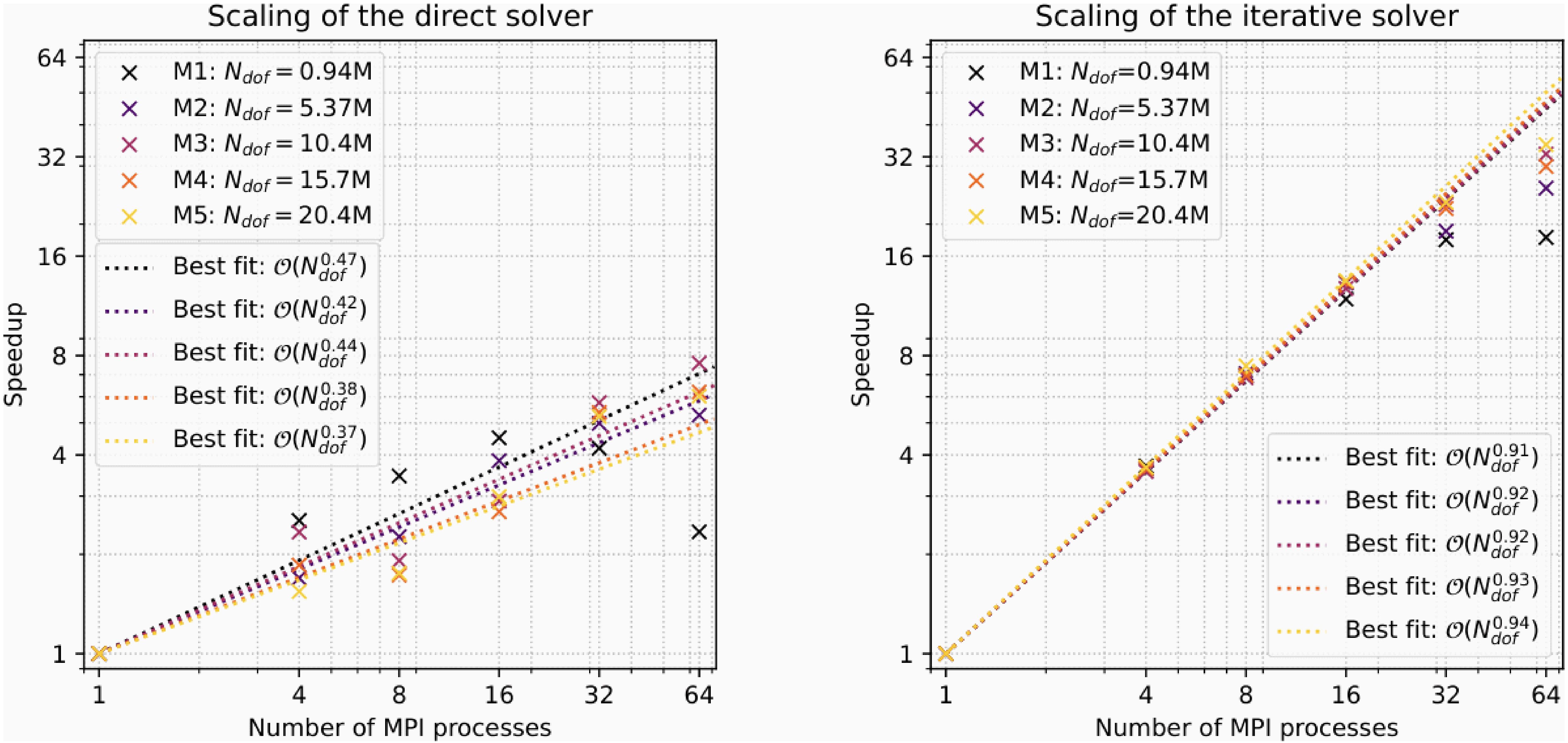


Parallel performance of the direct solver

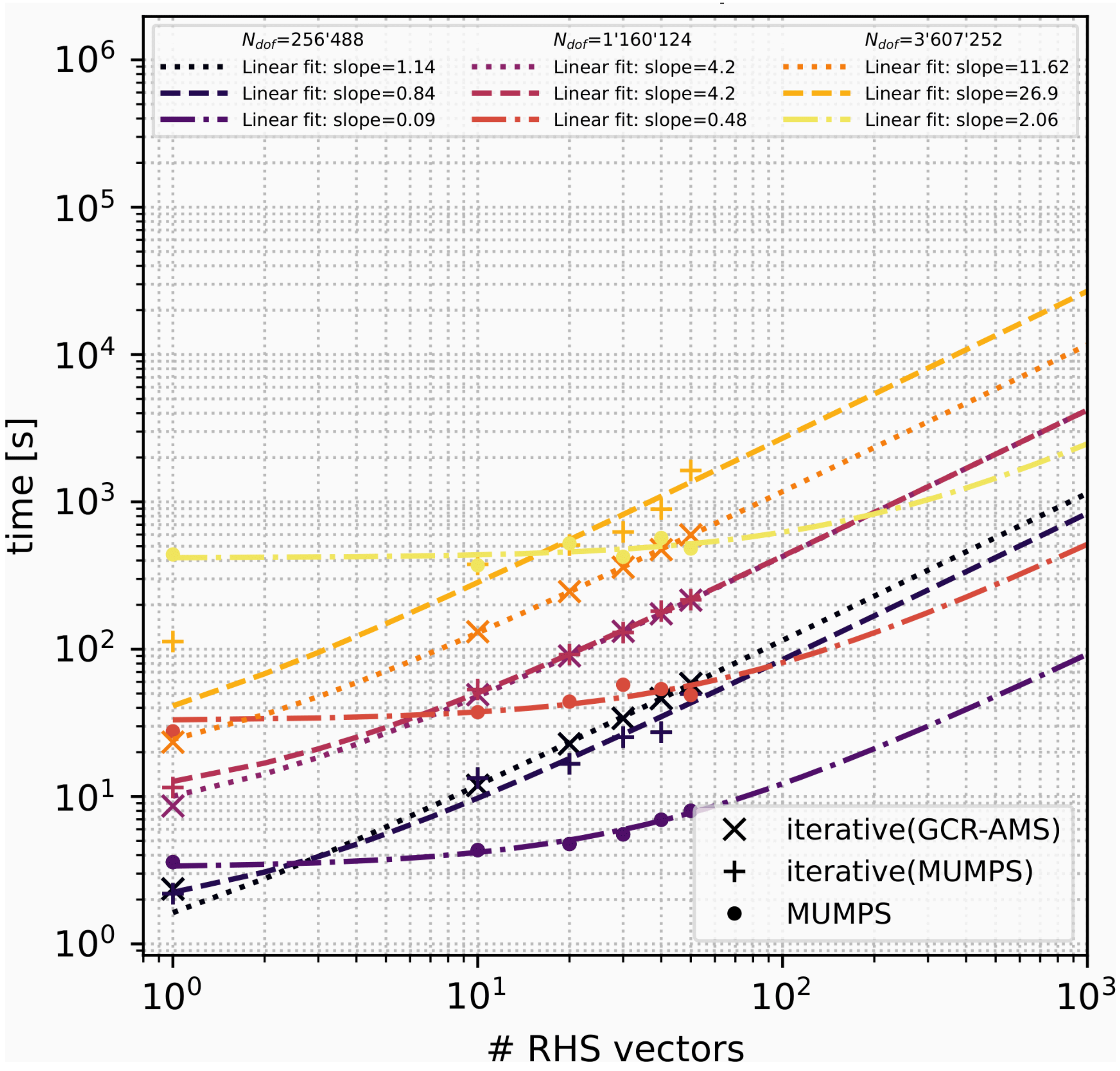


Parallel performance of the iterative solver

Perfomance analysis - MPI scaling



Multiple right-hand sides


N_{dof}=256'488
Linear fit: slope=1.14
Linear fit: slope=0.84
Linear fit: slope=0.09
N_{dof}=1'160'124
Linear fit: slope=4.2
Linear fit: slope=4.2
Linear fit: slope=0.48
N_{dof}=3'607'252
Linear fit: slope=11.62
Linear fit: slope=26.9
Linear fit: slope=2.06
time [s]
10^6
10^5
10^4
10^3
10^2
10^1
10^0
× iterative(GCR-AMS)
+ iterative(MUMPS)
● MUMPS
10^0
10^1
10^2
10^3
# RHS vectors