A Parallel Multigrid Matrix-Free Solver using Schwarz Smoothers

Daniel Arndt
Pablo Lucero Julius Witte
Patrick Esser Guido Kanschat

Heidelberg University

PDE Software Frameworks 2016

Mathematics Research Centre, Warwick

04.-08. July 2016

IWR
Interdisciplinary Center for Scientific Computing

Table of Contents

Problem

A typical finite element problem reads in weak form

$$a(u_h, v_h) = \langle f, v_h \rangle \qquad \forall v_h \in V_h$$

Solving Large Problems is restricted by

- Memory limitations
  - $\Rightarrow$ Matrix-Free methods
- Time limitations (Scalability)
  - $\Rightarrow$ Parallelization
  - $\Rightarrow$ Geometric Multigrid

Matrix-Free

Iterative solvers only require a matrix-vector product.

$\Rightarrow$ Idea:

- Storing the matrix not required
- "Assemble" the matrix-vector product directly, when solving.[1]

Matrix-free algorithm:

- v=0
- loop over cells
  1. Extract local vector values on cell: $u_K = P_K u$
  2. Apply operation locally on cell: $v_K = A_K u_K$ (without forming A_K)
  3. Sum results into the global solution vector: $v = v + P_K^T v_K$

[1] Martin Kronbichler and Katharina Kormann. "A generic interface for parallel cell-based finite element operator application". In: *Computers & Fluids* 63 (2012), pp. 135 –147. ISSN: 0045-7930

Matrix-Free

Matrix-Based

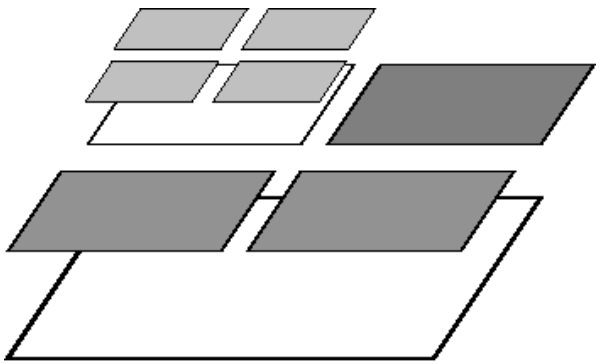
1. Build matrix $A^K_{i,j} = \int_K a(\phi_j, \phi_i)\, dx$
2. Solve using iterative method (CG) $\rightarrow$ compute $v_i = (Au)_i$.

Matrix-Free

Compute matrix-vector product directly

$$v_i = (Au)_i = \sum_K \int_K a(u, \phi_i)\, dx$$

Parallel Adaptive Multigrid



- Only method known that is $\mathcal{O}(N)$
- Geometric multigrid on adaptively refined meshes with local smoothing[2]
- `deal.II`[3] implements massively parallel geometric multigrid on both continuous and discontinuous elements.

[1] Bärbel Janssen and Guido Kanschat. "Adaptive Multilevel Methods with Local Smoothing for H^1- and H^{curl}-Conforming High Order Finite Element Methods". In: *SIAM Journal on Scientific Computing* 33.4 (2011), pp. 2095–2114

[2] W. Bangerth et al. "The `deal.II` Library, Version 8.4". In: *Journal of Numerical Mathematics* 24 (2016)

Parallel Adaptive Multigrid - V-Cycle

1. Pre-smoothing:

$$u^{(k+1)} = u^{(k)} - R_l^{-1}(A_l u^{(k)} - f_l), \qquad 0 \leq k < m_{pre}$$

2. Coarse grid correction:

$$\begin{aligned} f_{l-1} &= \Pi_{l-1}^{T}(f_l - A_l u^{(m_{pre})}) \\ v^{(k+1)} &= MG_{l-1}(v^{(k)}, f_{l-1}), \qquad 0 \leq k < m_{coarse} \\ w^{(0)} &= u^{(m_{pre})} + v^{(m_{coarse})} \end{aligned}$$

3. Post-smoothing:

$$w^{(k+1)} = w^{(k)} - R_l^{-1}(A_l w^{(k)} - f_l), \qquad 0 \leq k < m_{post}$$

4. Assign: $MG(u^{(0)}, f_l) = w^{(m_{post})}$

Coarse grid solver $MG_0(u(0), f) = A_0^{-1} f_0$

Parallel Adaptive Multigrid - V-Cycle

1. Pre-smoothing:

$$u^{(k+1)} = u^{(k)} - R_l^{-1}(A_l u^{(k)} - f_l), \qquad 0 \leq k < m_{pre}$$

2. Coarse grid correction:

$$\begin{aligned} f_{l-1} &= \Pi_{l-1}^{T}(f_l - A_l u^{(m_{pre})}) \\ v^{(k+1)} &= MG_{l-1}(v^{(k)}, f_{l-1}), \qquad 0 \leq k < m_{coarse} \\ w^{(0)} &= u^{(m_{pre})} + v^{(m_{coarse})} \end{aligned}$$

3. Post-smoothing:

$$w^{(k+1)} = w^{(k)} - R_l^{-1}(A_l w^{(k)} - f_l), \qquad 0 \leq k < m_{post}$$

4. Assign: $MG(u^{(0)}, f_l) = w^{(m_{post})}$

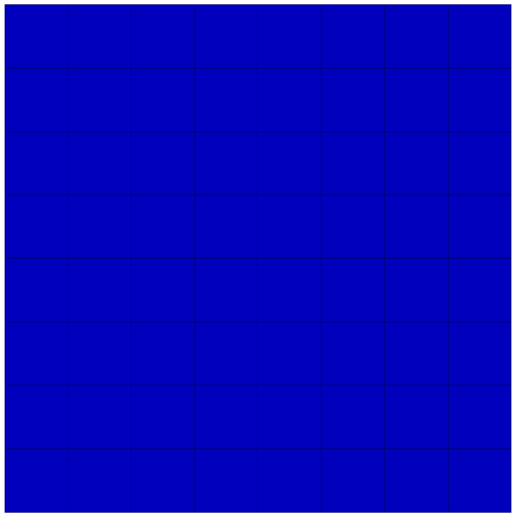
Coarse grid solver $MG_0(u(0), f) = A_0^{-1} f_0$

Additive Schwarz Smoother

Hermann Amandus Schwarz

- Take the local structure of the problem into account
- Use local problems for preconditioning

$$R_l = \sum_{K \in \mathcal{T}_l} P_K A_K^{-1}$$



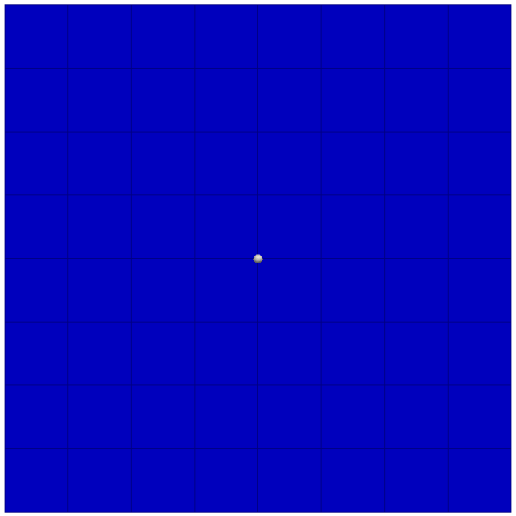
Additive Schwarz Smoother

Hermann Amandus Schwarz

- Take the local structure of the problem into account
- Use local problems for preconditioning

$$R_l = \sum_{K \in \mathcal{T}_l} P_K A_K^{-1}$$

- Point patches $\Rightarrow$ Jacobi



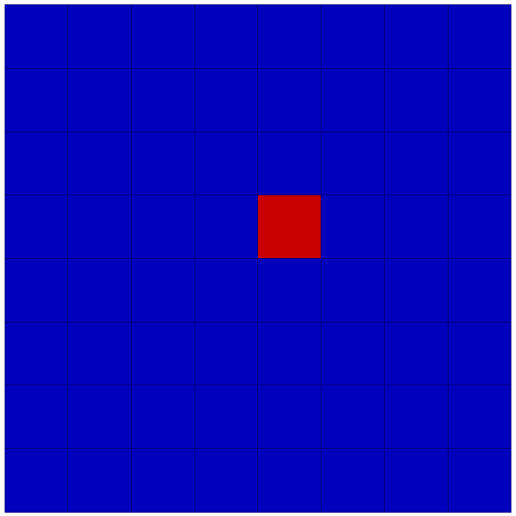
Additive Schwarz Smoother

Hermann Amandus Schwarz

- Take the local structure of the problem into account
- Use local problems for preconditioning

$$R_l = \sum_{K \in \mathcal{T}_l} P_K A_K^{-1}$$

- Cell patches $\Rightarrow$ BlockJacobi



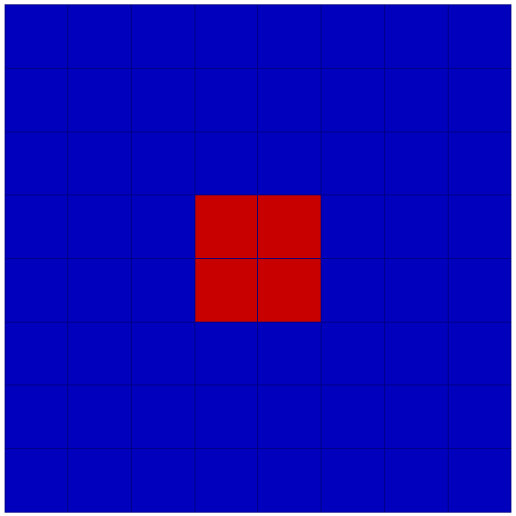
Additive Schwarz Smoother

Hermann Amandus Schwarz

- Take the local structure of the problem into account
- Use local problems for preconditioning

$$R_l = \sum_{K \in \mathcal{T}_l} P_K A_K^{-1}$$

- Vertex patches $\Rightarrow$ BlockJacobi



Dictionary

Often not all the local inverses are different or at least don't differ a lot.

- Idea: Store only local inverses that differ significantly
- Hope: Comparable results to BlockJacobi smoothing.

$$\frac{\|A_i^{-1}A_j - Id\|}{n^2} < \text{tol} \qquad \Rightarrow \qquad A_j^{-1} \approx A_i^{-1}.$$

Parallelization

Use parallel linear algebra structures provided by

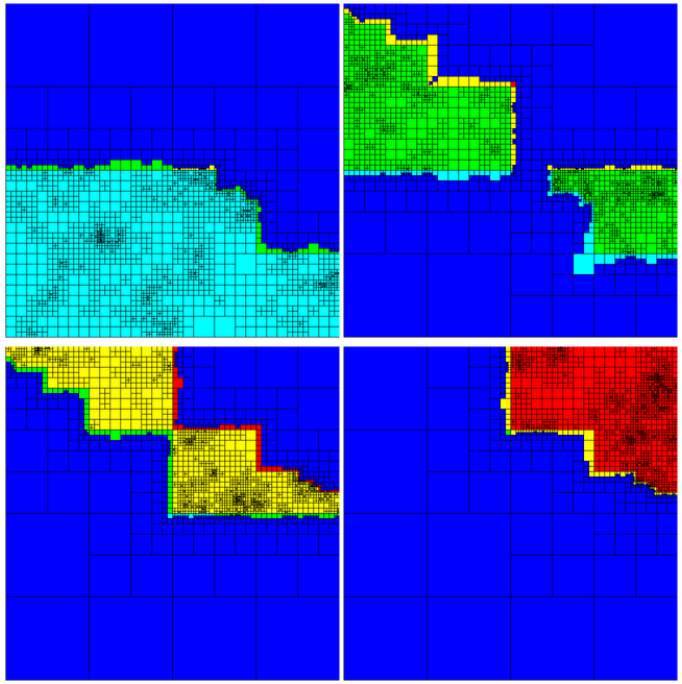
- PETSc or Trilinos
- `deal.II` directly (`parallel::distributed::Vector`)

MPI

- Partitioning of the domain and stored data (`p4est`[4])

Threads

- Solving the local problems
- Computing the local residuals in parallel



[4] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. "`p4est`: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees". In: *SIAM Journal on Scientific Computing* 33.3 (2011), pp. 1103–1133. DOI: 10.1137/100791634

Usage - Code example - Cell

```
template <int dim,bool same_diagonal>
void MatrixIntegrator<dim, same_diagonal>::cell(...) const
{
  const auto n_blocks = dinfo.block_info->local().size();
  std::vector<std::vector<std::vector<double> > > coeffs;
  //setup coeffs
  [...]

  for (unsigned int b=0; b<n_blocks; ++b )
  {
    const auto &fev = info.fe_values(dinfo.block_info->base_element(b));
    const auto n_quads = fev.n_quadrature_points;
    const auto n_components = fev.get_fe().n_components();
    coeffs.resize(n_components);
    auto &M = dinfo.matrix(b*n_blocks + b).matrix;
    LocalIntegrators::Diffusion::cell_matrix<dim>(M,fev,coeffs);
  }
}
```

Usage - Code example - Face

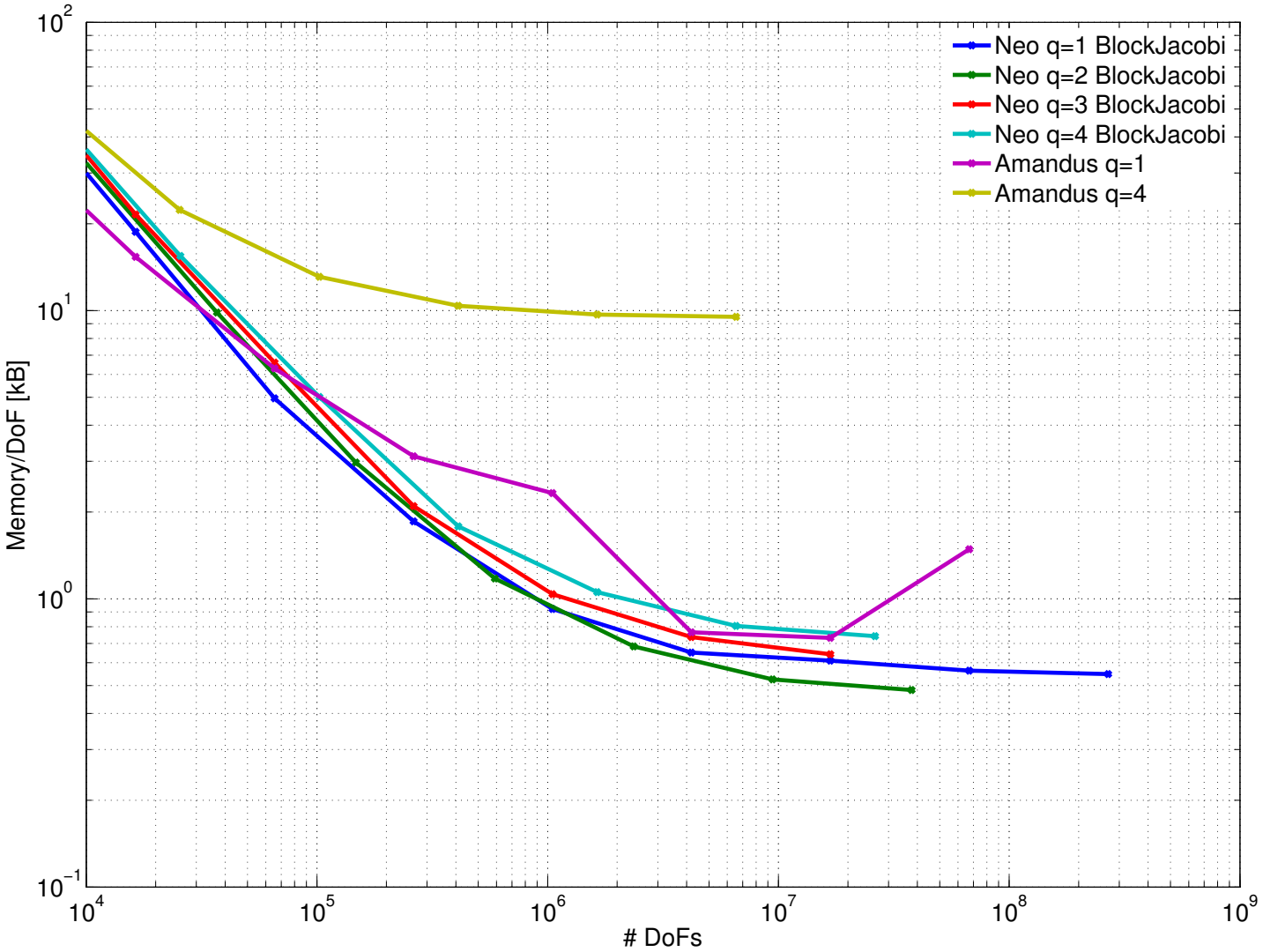
```
template <int dim, bool same_diagonal>
void MatrixIntegrator<dim,same_diagonal>::face(...) const
{
  std::vector<std::vector<std::vector<double> > > coeffs;
  for (unsigned int b=0; b<n_blocks; ++b )
  {
    auto &fev1 = info1.fe_values(dinfo1.block_info->base_element(b));
    auto &fev2 = info2.fe_values(dinfo2.block_info->base_element(b));

    auto &RM11 = dinfo1.matrix(b*n_blocks + b,false).matrix;
    auto &RM12 = dinfo2.matrix(b*n_blocks + b,true).matrix;
    auto &RM21 = dinfo1.matrix(b*n_blocks + b,true).matrix;
    auto &RM22 = dinfo2.matrix(b*n_blocks + b,false).matrix;
    //setup coeffs
    [...]
    LocalIntegrators::Diffusion::ip_matrix<dim>
    (RM11,RM12,RM21,RM22,fev1,fev2,coeffs,
     LocalIntegrators::Diffusion::compute_penalty(dinfo1,dinfo2,deg1,deg2));
  }
}
```

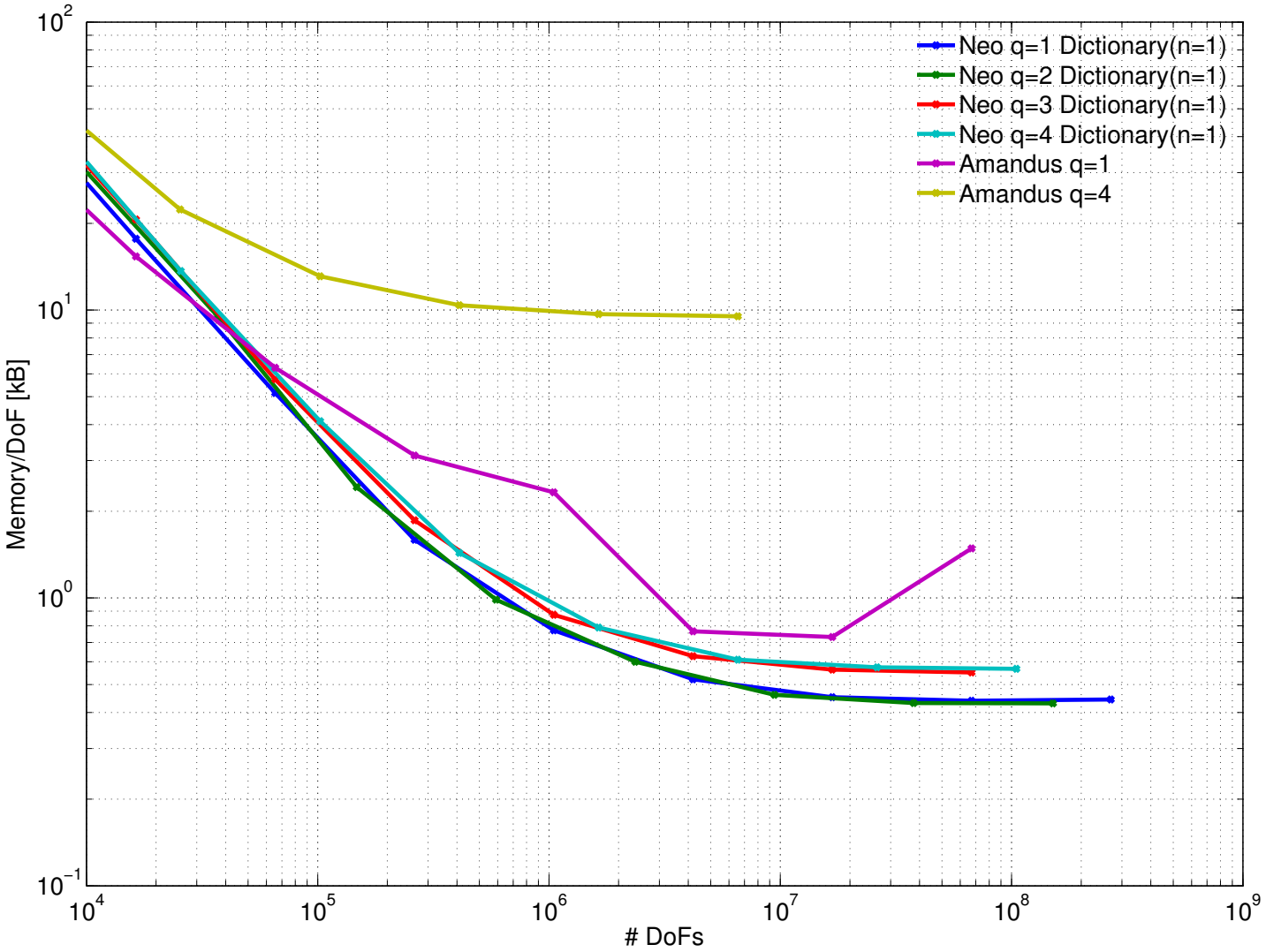
Usage - Code example - Boundary

```
template <int dim, bool same_diagonal>
void MatrixIntegrator<dim, same_diagonal>::boundary (...) const
{
  std::vector<std::vector<std::vector<double> > > coeffs;
  for (auto b=0; b<n_blocks; ++b )
  {
    const auto &fev = info.fe_values(dinfo.block_info->base_element(b));
    const auto deg = fev.get_fe().tensor_degree();
    const auto n_quads = fev.n_quadrature_points;
    const auto n_components = fev.get_fe().n_components();
    //setup coeffs
    [...]
    auto &M = dinfo.matrix(b*n_blocks + b).matrix;
    LocalIntegrators::Diffusion::nitsche_matrix<dim>
    (M, fev, coeffs,
     LocalIntegrators::Laplace::compute_penalty(dinfo, dinfo, deg, deg));
  }
}
```

Laplace - Memory BlockJacobi



Laplace - Memory Dictionary



Radiative Transport

$$\Omega \cdot \nabla_x u(x, \Omega) + \frac{1}{\epsilon} \int \sigma_s(x, \Omega')(u(x, \Omega) - u(x, \Omega'))\, d\Omega' = \epsilon$$

Model particle distributions u_i along angles Ω_i, $i \leq |\Omega|$

- Convection in direction Ω_i

$$\Omega_i \cdot \nabla_x u_i(x, \Omega)$$

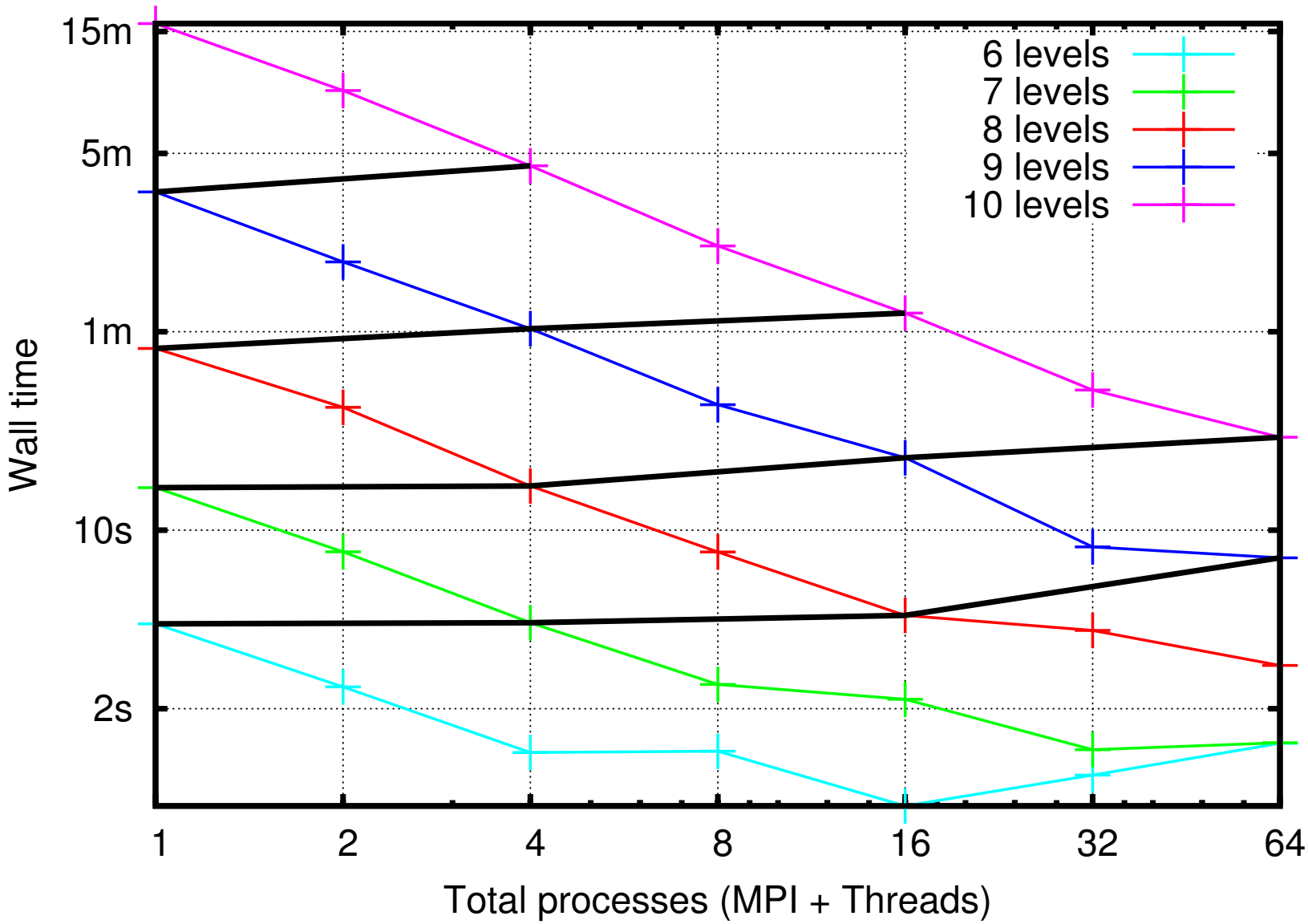
- Scaterring

$$\int \sigma_s(x, \Omega')(u(x, \Omega) - u(x, \Omega'))\, d\Omega'$$

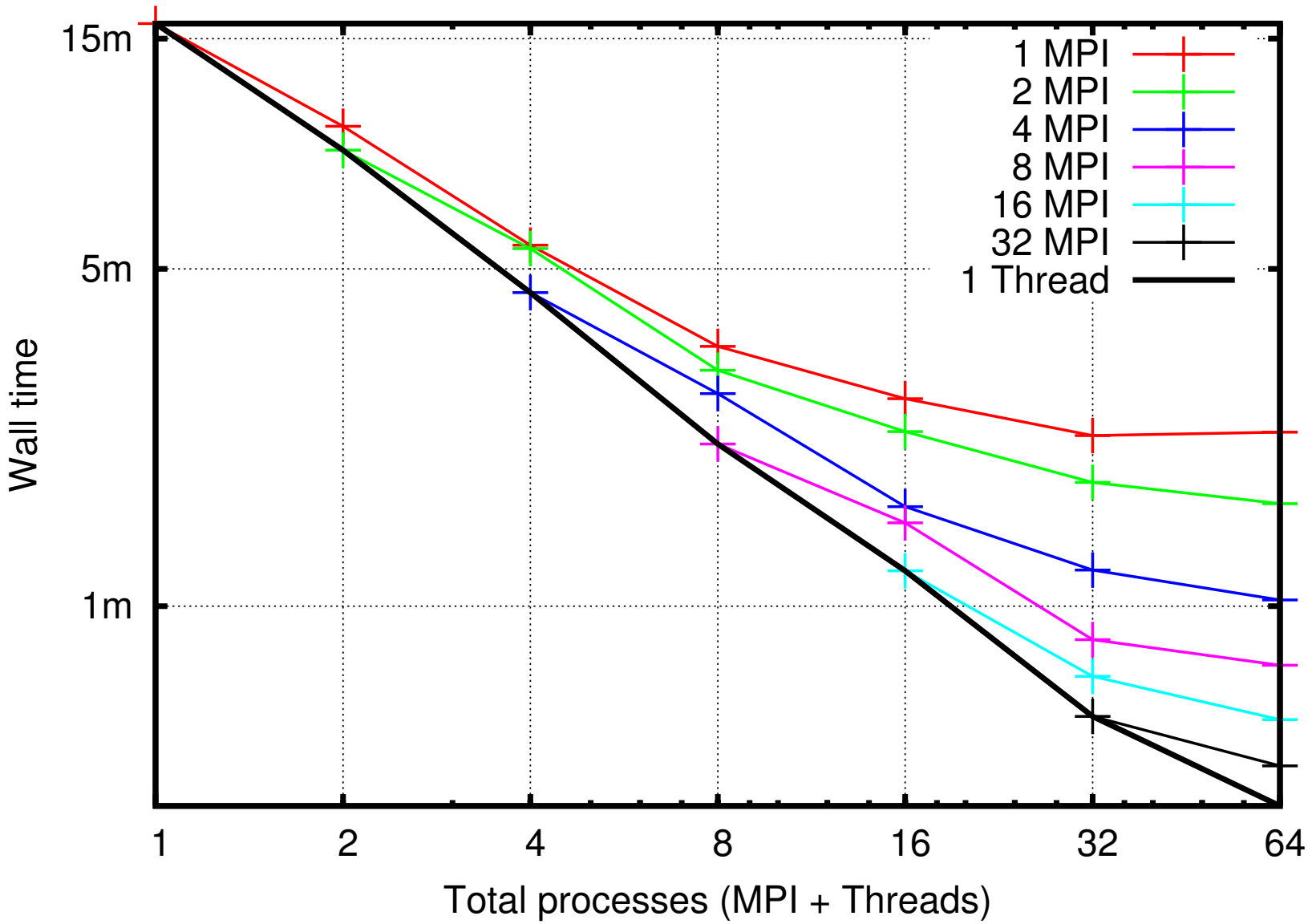
- Source s

In the following: $\sigma_s \equiv 1$ $|\Omega| = 4$ $\sigma_s \equiv 1$ $\epsilon = 10^{-2}$

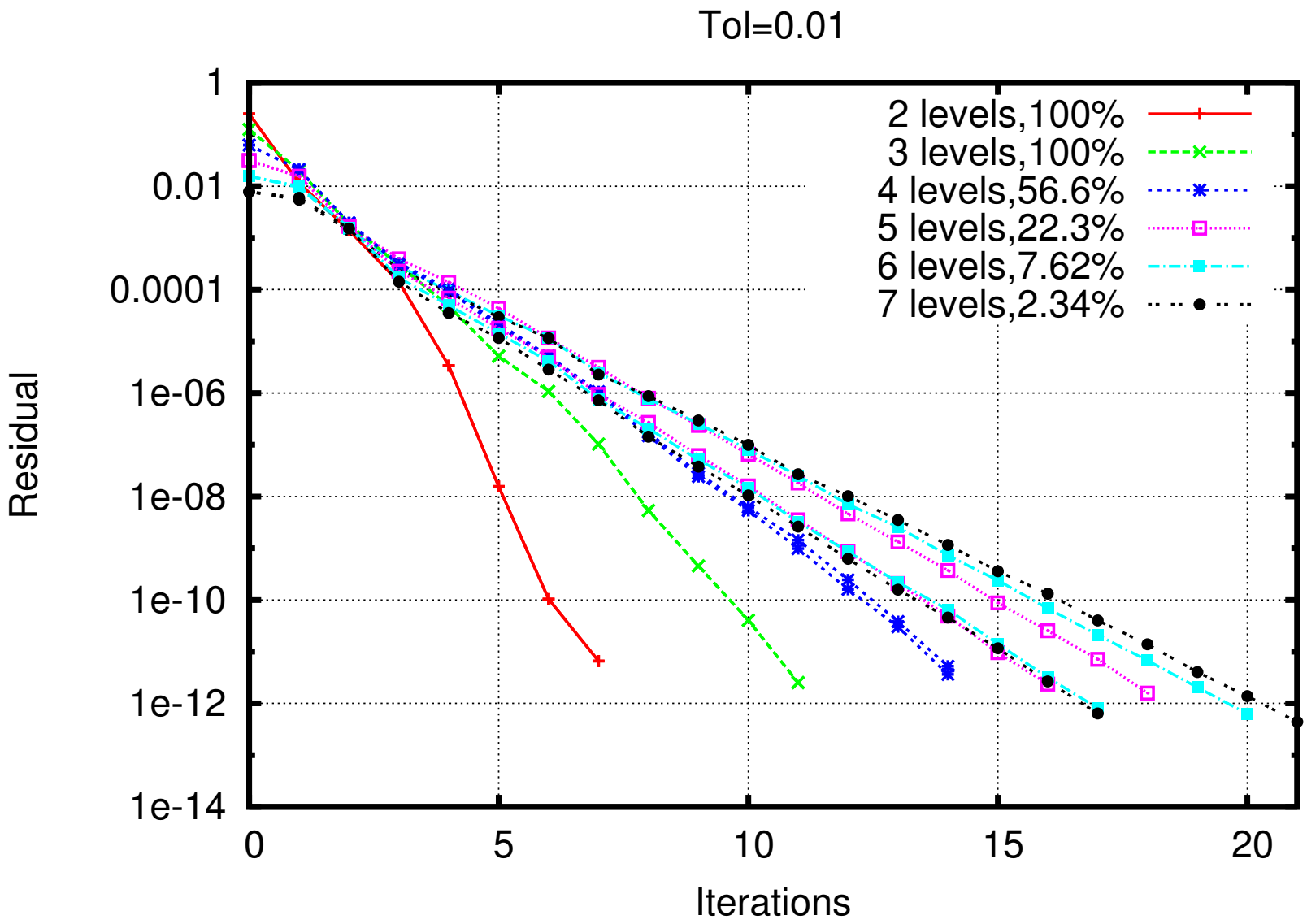
Transport - Weak and Strong Scaling - $4^{level} \cdot ||\Omega|| \cdot 3$ DoFs



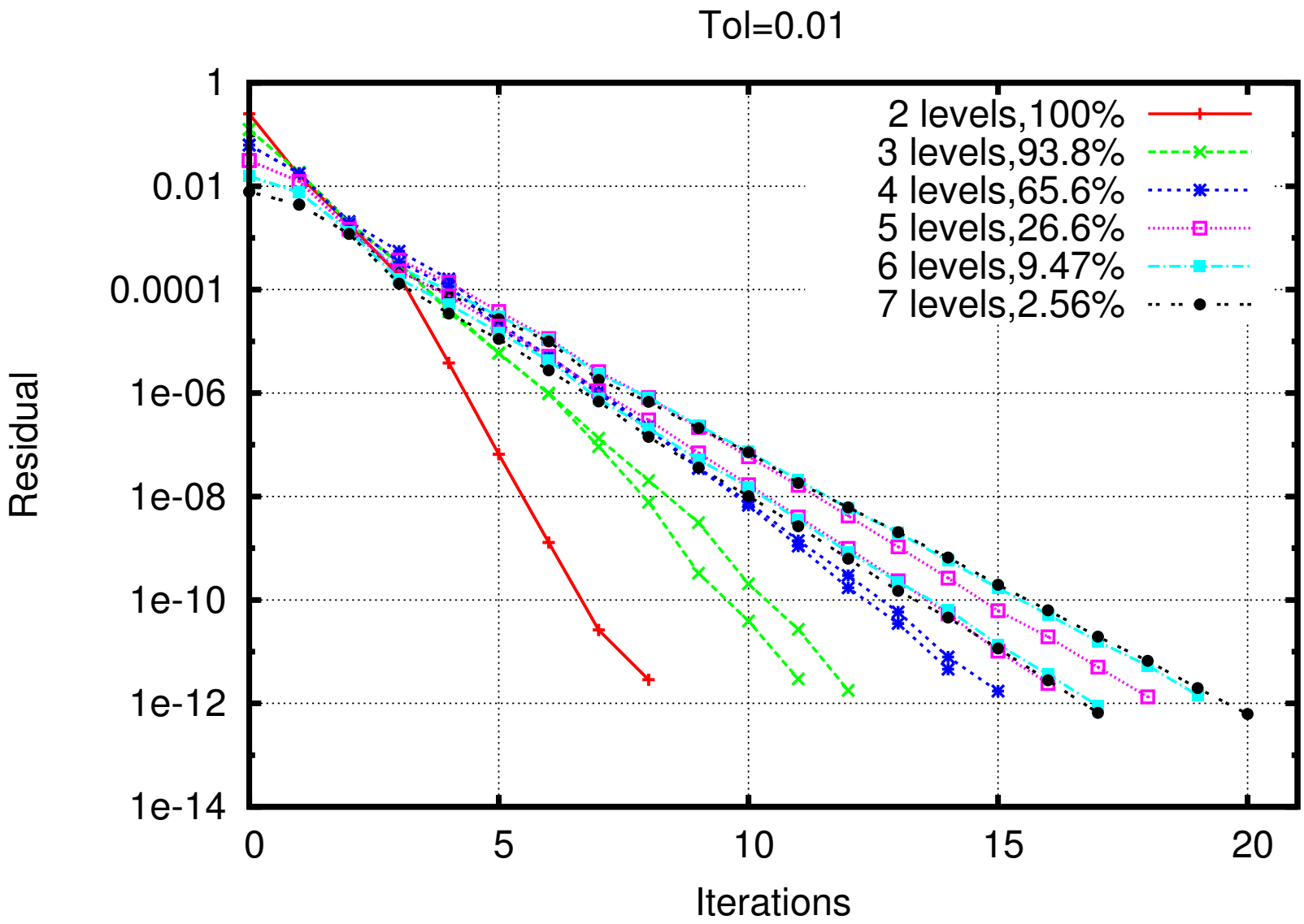
Transport - Threads vs. MPI Scaling - 12,582,912 DoFS



Transport - Dictionary linear



Transport - Dictionary quadratic



Transport - Iterations 2d

#levels	ϵ = 1.0	0.1	0.01	0.001	0.0001	1e-05
3	9	10	10	9	8	7
4	15	12	14	14	14	14
5	21	12	15	15	15	15
6	30	14	15	15	15	15
7	47	18	13	15	15	15
8	71	24	12	15	15	15
9	106	34	12	14	14	14

Transport - Iterations 3d

#levels	ϵ 1.0	0.1	0.01	0.001	0.0001	1e-05
3	12	12	12	11	10	10
4	18	14	16	16	16	16
5	26	15	18	18	18	18
6	36	18	18	19	19	19
7	52	22	17	19	19	19
8	77	29	15	18	18	18

Summary

Features:

- Matrix-Free
- Adaptive Multigrid
- Schwarz smoothers
- Parallelization

Results

- Always prefer MPI over Thread paralleization
- Convincing scaling
- Schwarz smoothers compatible with Matrix-Free approach
- Schwarz smoothers effective also for radiative transport

Thank you for your attention!

References


W. Bangerth et al. “The `deal.II` Library, Version 8.4”. In: *Journal of Numerical Mathematics* 24 (2016).


Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. “`p4est`: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees”. In: *SIAM Journal on Scientific Computing* 33.3 (2011), pp. 1103–1133. DOI: 10.1137/100791634.


Bärbel Janssen and Guido Kanschat. “Adaptive Multilevel Methods with Local Smoothing for H^1- and H^{curl}-Conforming High Order Finite Element Methods”. In: *SIAM Journal on Scientific Computing* 33.4 (2011), pp. 2095–2114.


Martin Kronbichler and Katharina Kormann. “A generic interface for parallel cell-based finite element operator application”. In: *Computers & Fluids* 63 (2012), pp. 135 –147. ISSN: 0045-7930.