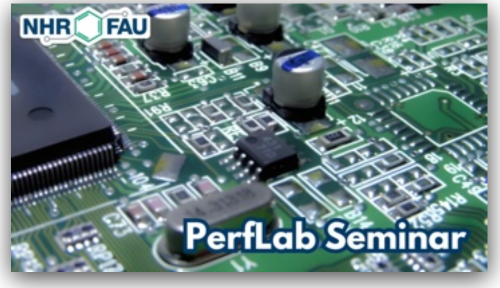
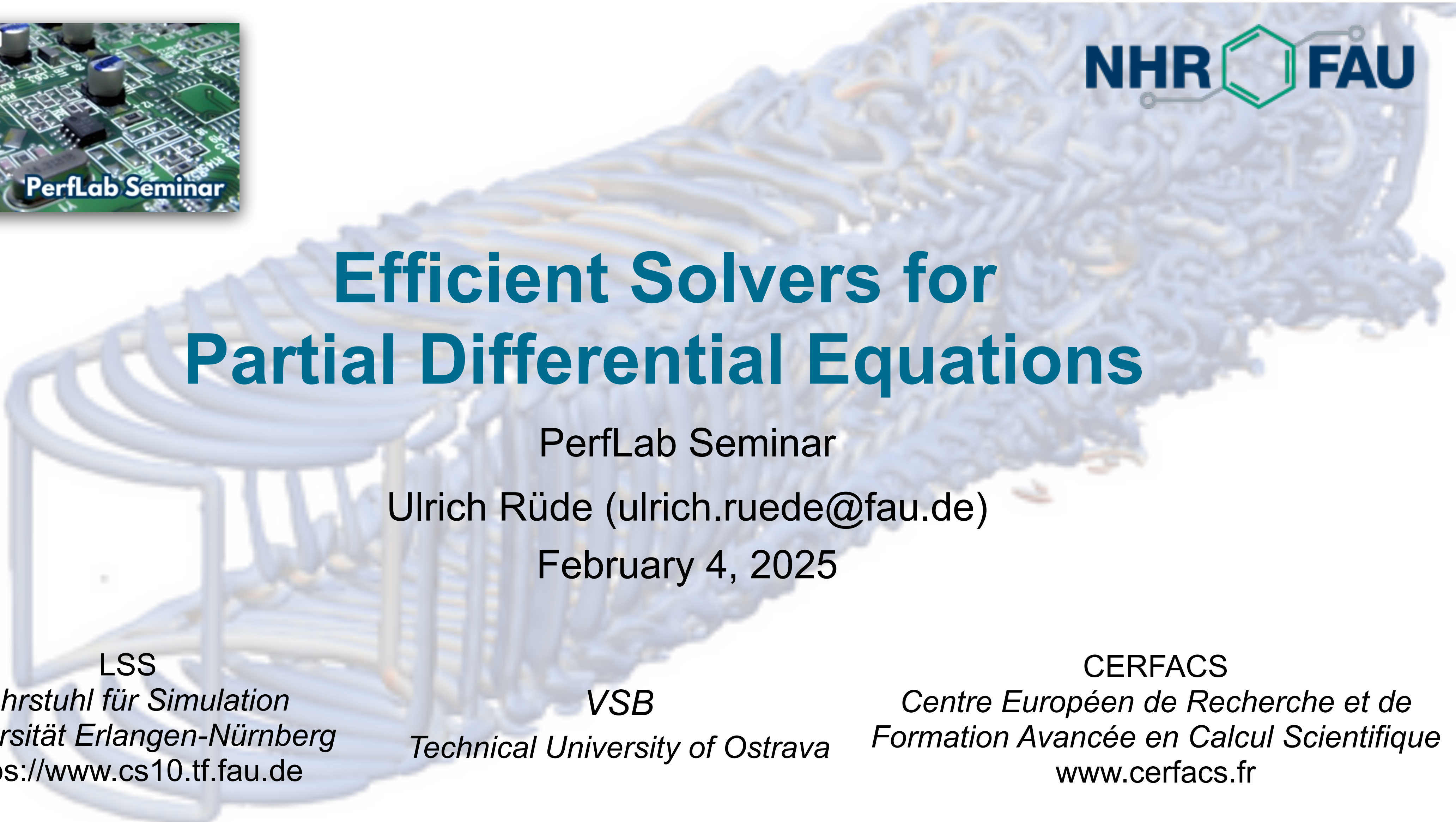




Efficient Solvers for Partial Differential Equations

PerfLab Seminar

Ulrich Rüde (ulrich.ruede@fau.de)

February 4, 2025

LSS
Lehrstuhl für Simulation
Universität Erlangen-Nürnberg
https://www.cs10.tf.fau.de

VSB
Technical University of Ostrava

CERFACS
Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique
www.cerfacs.fr

Preamble:

What is the fastest solver for Poisson's equation?

The context:

- Scientific Computing is about efficient methods
- Numerical algorithms require a tradeoff between accuracy and cost
  - If accuracy is irrelevant, cheap algorithms are trivial to find
  - If cost is irrelevant, accuracy is trivial to achieve

Setting accuracy in relation to cost:

- We need metrics for
  - cost (algorithmic complexity)
  - accuracy (magnitude of error)
- Both are surprisingly unclear
  - Cost: counting #unknowns, counting #FLOPS, memory consumption, run time, energy consumption, ….
  - Accuracy: Residual vs. error? Which norm?
    Often not the solution is needed, but a functional thereof, …
- All this makes a difference in what is needed
- The new kid on the block:
  - **Deep Learning (for PDE)**
    When your natural intelligence fails, use an artificial one!

Making the question more specific:

- When teaching linear algebra we insist that students learn:
  - Gaussian elimination costs $\sim \frac{2}{3}n^3$ FLOPS
- But for PDE? Let's focus on:
  - Poisson's equation unit square with
  - 5-point discretization of the Laplace operator
    - at this stage we thus avoid the discussion of accuracy
  - Complexity metric: FLOPS
- With this: What is the cost of solving the discretized Poisson equation on a grid with $N = n_x \times n_y = n^2$ unknowns?
  - … what is the best algorithm known today?
  - … what is the answer for 3D? … or more general equations?
    … more advanced discretization techniques?
- In any case: I insist on the constant, multiplying the dominating term
- When the complexity is (almost) linear, the constant is the critical quantity

The model problem:

$$\begin{aligned} -\Delta u &= f \quad \text{in} \quad \Omega := (0,1)\times(0,1) \\ u &= 0 \quad \text{on} \quad \partial\Omega \end{aligned}$$

- Let's restrict ourselves to Poisson's equation
- Smooth enough rhs data
- Initially we'll even simplify to the unit interval (1D) for easier illustration
- Then focus on unit square with homogeneous Dirichlet BC

This is the fruitfly for studying PDE solvers

- But is this a problem of practical relevance?
    - Yes and No
    - Most applications require generalizations, e.g. other domains, other bc, variable coefficients
    - but this simple problem captures fundamental features that characterizes elliptic PDE: The need for global data exchange.

The 1D model problem

$$-u'' = f(x) \text{ in } (0,1) \qquad \begin{aligned} u'(0) &= 0 && \text{(Neumann condition)} \\ u(1) &= \beta && \text{(Dirichlet condition)} \end{aligned}$$

- Another (dimension independent) way to write this:

$$\operatorname{div} \operatorname{grad} u = f$$

- The 1D differential operator with the given boundary conditions has the eigenfunctions

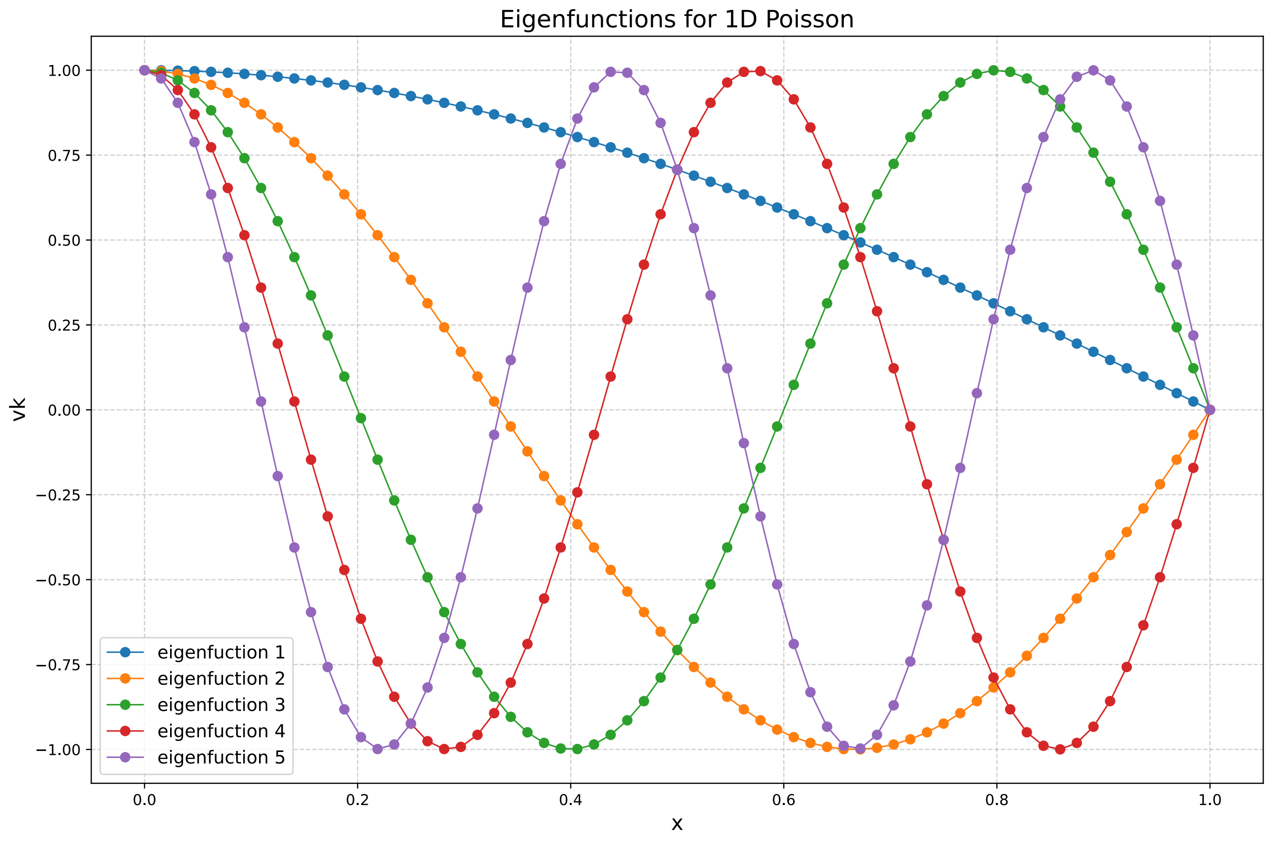
$$v_k(x) = \cos(\tfrac{2k+1}{2}\pi x) \quad \text{for } k = 0, 1, 2, 3, \ldots$$

Visualization of the first 5 eigenfunctions

$$v_k(x) = \cos(\tfrac{2k+1}{2}\pi x) \quad \text{for } k = 0, 1, 2, 3, \ldots$$

At the right boundary we have a homogeneous Dirichlet condition

At the left a homogeneous Neumann condition



Setting up the discretization

- The matrix (without 1/h^2 factor)
  - has tridiagonal structure
  - is diagonally dominant
  - Is symmetric positive definite
  - Neumann condition on left end, Dirichlet condition on right end

```
Apoisson
```

```
array([[ 1., -1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [-1.,  2., -1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0., -1.,  2., -1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0., -1.,  2., -1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0., -1.,  2., -1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0., -1.,  2., -1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0., -1.,  2., -1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0., -1.,  2., -1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  2., -1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  2., -1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  2., -1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  2.]])
```

Let us look into solution algorithms

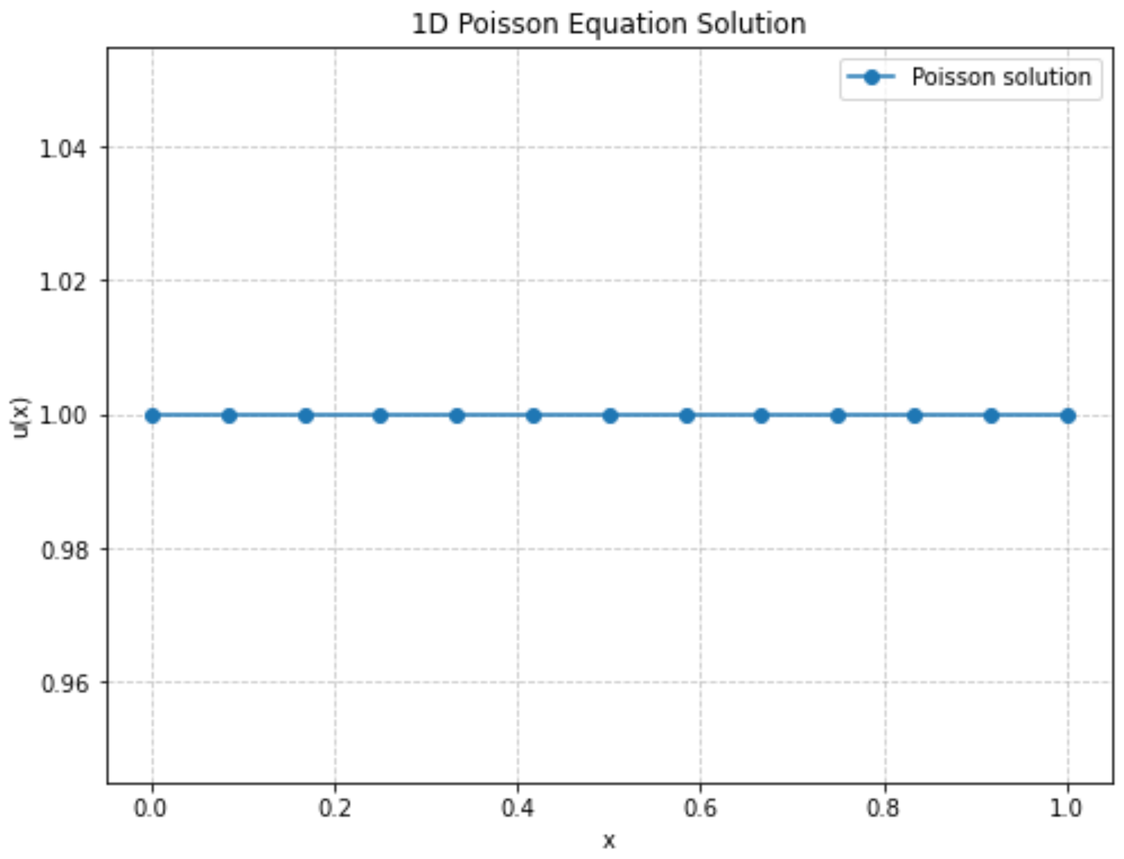
- The best way in 1D is to use a tridiagonal Cholesky factorization (in this case it recovers discrete div and grad)
  - has O(N) complexity
  - suffers form sequentiality in the factorization and also in the fwd-bwd substitution
  - On a parallel system, better use cyclic reduction as elimination order
- Here let us consider Gauss-Seidel and SOR as first iterative solvers.
- Note that iterative solvers are not an efficient choice for the 1D Poisson eqn.
- Obvious change to make for SOR
- Note that for these tests we do not worry about efficiency, e.g. exploiting the tridiagonal structure of the matrix

```
# Next we study iterative methods
# We start with Gauss-Seidel
def gs1d_fwdstep(A, u, f):
    """
    Eexcutes one step of fwd Gauss-Seideö for the 1D Poisson equation -u'
    with Dirichlet boundary conditions u(1) = 1 and Neumann boundary cond

    Parameters:
        u (ndarray): approximate solution, Dirichlet condition at u[0]
        f (ndarray): right hand side
        A matrix

    Returns:
        u (ndarray): Numerical solution at the grid points.
    """

    n_pts= u.size
    h = 1.0/(n_pts)  # Grid spacing
    # print(n_points, h)

    for i in range(0, n_pts):
        old_u = u[i]
        res= f[i] - A[i,:] @ u
        u[i] = u[i] + 1/A[i,i] * res
    return u
```

```
# Set rhs and set initial value for u
h=1/npts
u= np.zeros(npts)
f= -np.zeros(npts)*h**2
# Add Dirichlet condition value at right end in f
f[-1]=1
u,f
```

Visualization of the exact solution

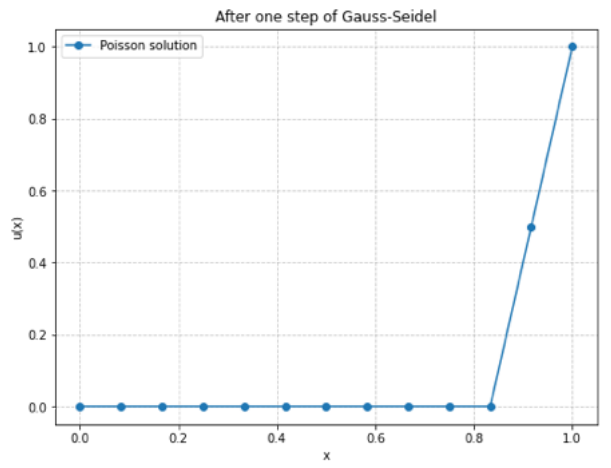
```
# Try out again exaact solution with direct solver
u_el2= np.linalg.solve(Apoisson, f)
plot_1d(u_el2, 1)
```



The Dirichlet value (= 1) from the right the end bc is „propagated“ to the left across the whole domain

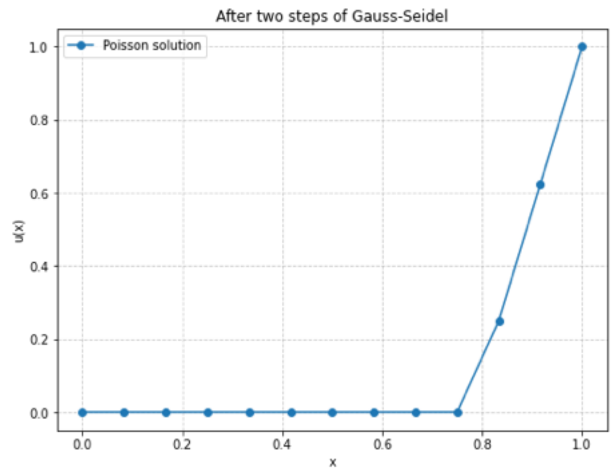
What does Gauss-Seidel do?

```
u1= sor1d_fwdstep(Apoisson,u,f)
plot_1d(u1,1.0, tit= "After one step of Gauss-Seidel")
```



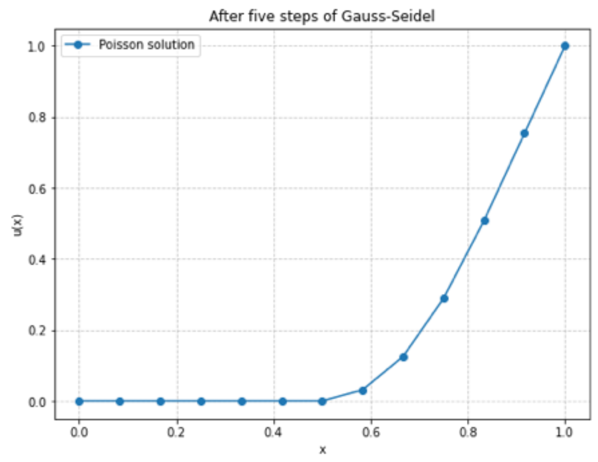
1st step of GS

```
u2= sor1d_fwdstep(Apoisson,u1,f)
plot_1d(u1,1.0, tit= "After two steps of Gauss-Seidel")
```



2nd step of GS

```
un= u2
un= sor1d_fwdstep(Apoisson,un,f)
un= sor1d_fwdstep(Apoisson,un,f)
un= sor1d_fwdstep(Apoisson,un,f)
plot_1d(u1,1.0, tit= "After five steps of Gauss-Seidel")
```



5 steps of GS

- In each step, the information from the Dirichlet point propagates by one mesh point towards the interior of the domain.
- There is a **computational speed-of-light** for propagating the information

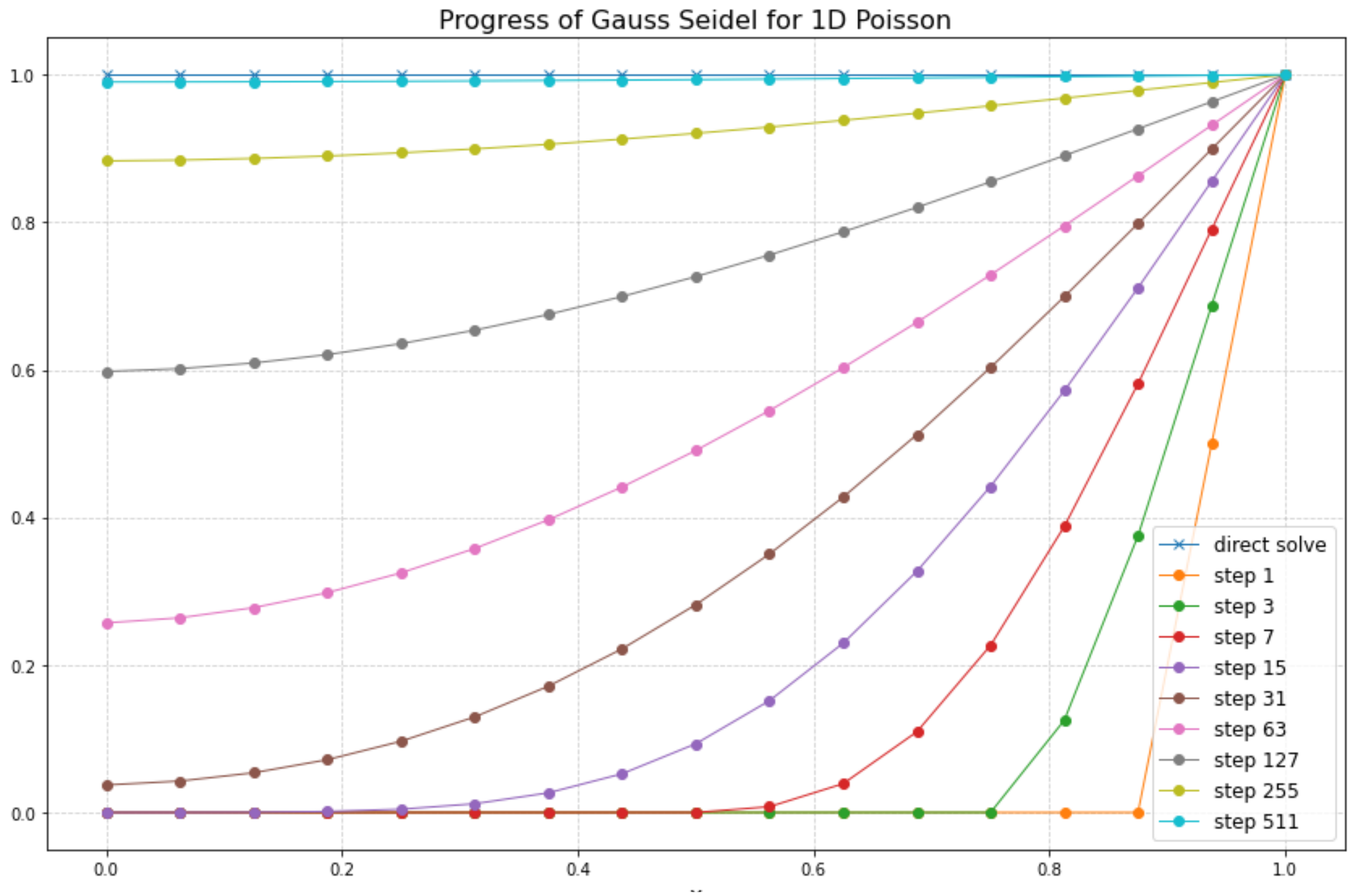
Now more systematically

- Using a grid with 16(+1) points
- Loop taking progressively more and more Gauss-Seidel steps

```python
#setup a larger problem
npts= 16
x = np.linspace(0, 1, npts+1)
Agrad= np.eye(npts)-np.eye(npts,npts,1)
Adiv= Agrad.T
Apoisson= Adiv @ Agrad
# Create a suitable right hand side. Initialize with 0
f= np.zeros(npts)
# Set right end to 1, corresponding to an eliminated Dirichlet condition
f[-1]= 1
u= np.zeros(npts)
uelim= np.linalg.solve(Apoisson, f)
uelim= np.append(uelim, [1])
```

```python
# trying several steps of Gauss-Seidel

plt.figure(figsize=(12, 8))

plt.plot(x, uelim, label='direct solve', marker = 'x', linewidth=1)

j_start= 0
j_end= 1
for ii in range(9):
    # print(i)
    j_start= j_end
    j_end= 2*j_end
    for jj in range(j_start, j_end):
        # print(jj)
        tmp= gs1d_fwdstep(Apoisson, u,f)
        tmp= np.append(tmp, [1])
    plt.plot(x, tmp, label='step '+str(jj), marker = 'o', linewidth=1)


# Add labels, legend, and grid
plt.title('Progress of Gauss Seidel for 1D Poisson', fontsize=16)
plt.xlabel('x', fontsize=14)
plt.ylabel('u', fontsize=14)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()

# Show the plot
plt.show()
```

Visualization of results



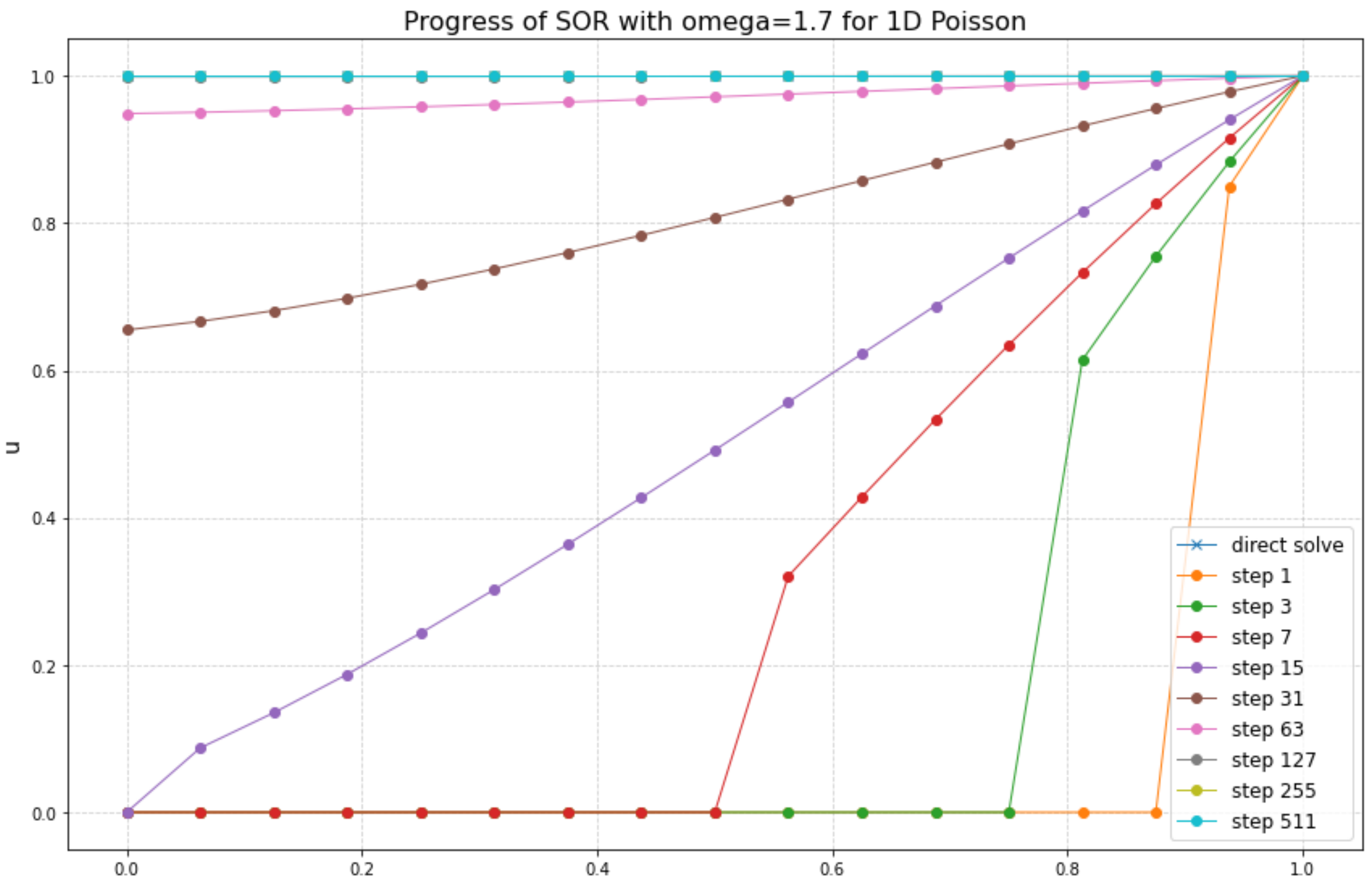
- Information propagates by one mesh point per Gauss-Seidel iteration
- However, there is „**additional slowness**“
- 15 (Npts) iterations are by far not enough
- Only when the number of iterations is roughly as large as the square of the number of mesh points, the solution becomes „qualitatively correct“
- But even with 511 GS iterations, the remaining error remains clearly visible
- The number of iterations must be as large as

$$\mathcal{O}(\kappa(A))$$

How can this be improved?

- Reversing the order of grid traversal (right to left)
  - Helps less than one would hope, depends on special case, and speeds up only initially, but not in the long „asymptotic tail“
- More successfully, we can try:
  - Over relaxation, SOR
  - Conjugate gradients
  - Both can improve the number of iterations to $\mathcal{O}(\sqrt{\kappa(A)})$

Visualization, now with over relaxation parameter omega=1.7



- We see a quite significant speedup
- From about 100 iterations onwards, the solution visually overlaps with the exact one
- Of course, we can next explore what the best omega would be
- could be determined experimentally or analytically

Instead, let us take a look at CG

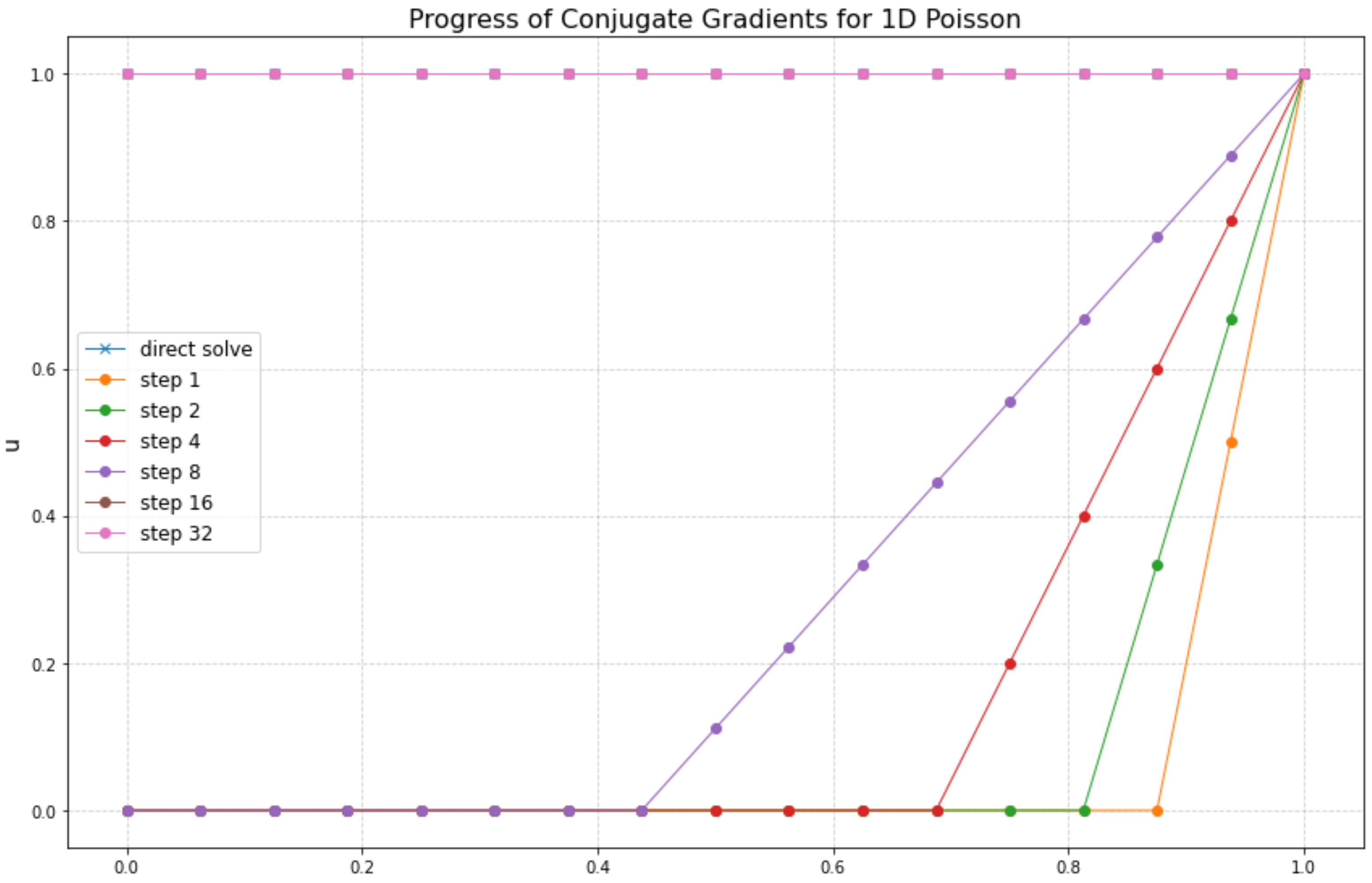
- Taking a CG routine from the internet
- We'll use the maxit parameter to study progress of CG throughout the iterations

CG test driver routine

```
plt.figure(figsize=(12, 8))

plt.plot(x, uelim, label='direct solve', marker = 'x', linewidth=1)

j_end= 1
for ii in range(6):
    uz, info= cg(Apoisson, f, max_iter= j_end) #stopping CG early
    ucg= np.append(uz,[1])
    plt.plot(x, ucg, label='step '+str(j_end), marker = 'o', linewidth=1)
    j_end= 2*j_end

# Add labels, legend, and grid
plt.title('Progress of Conjugate Gradients for 1D Poisson', fontsize=16)
plt.xlabel('x', fontsize=14)
plt.ylabel('u', fontsize=14)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()

# Show the plot
plt.show()
```

```
def cg (A, b, x0=None, tol=1e-10, max_iter=None):
    """
    Solves the symmetric positive definite system Ax = b using the Conjug
    Parameters:
        A (numpy.ndarray): Symmetric positive definite matrix.
        b (numpy.ndarray): Right-hand side vector.
        x0 (numpy.ndarray): Initial guess for the solution (default is ze
        tol (float): Convergence tolerance (default is 1e-10).
        max_iter (int): Maximum number of iterations (default is len(b)).
    Returns:
        x (numpy.ndarray): Solution vector.
        info (dict): Dictionary with additional info (e.g., iteration cou
    """
    n = len(b)
    if x0 is None:
        x0 = np.zeros(n)
    if max_iter is None:
        max_iter = n

    x = x0
    r = b - A @ x  # Residual
    p = r.copy()   # Search direction
    rs_old = r @ r # Dot product of residual with itself

    info = {
        'iterations': 0,
        'residual_norm': np.linalg.norm(r)
    }

    for i in range(max_iter):
        Ap = A @ p
        alpha = rs_old / (p @ Ap)
        x = x + alpha * p
        r = r - alpha * Ap
        rs_new = r @ r

        # Check convergence
        if np.sqrt(rs_new) < tol:
            info['iterations'] = i + 1
            info['residual_norm'] = np.sqrt(rs_new)
            return x, info

        p = r + (rs_new / rs_old) * p
        rs_old = rs_new

    # If we reach max_iter without convergence
    info['iterations'] = max_iter
    info['residual_norm'] = np.sqrt(rs_old)
    return x, info
```

CG results



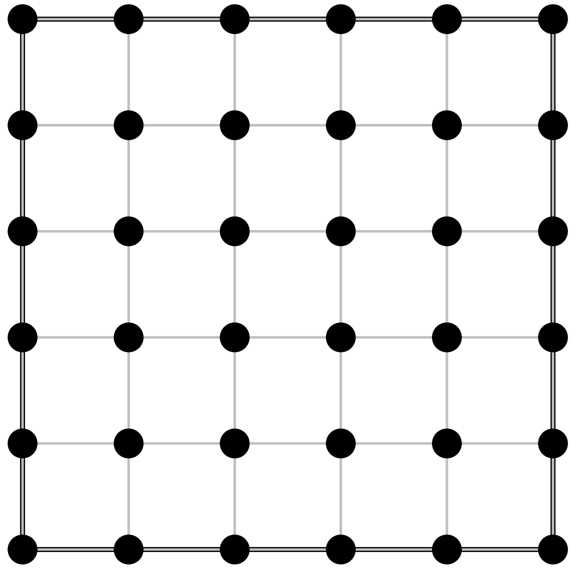
- CG can be understood as a clever implementation of combining successive iterates.
- Among all linear combinations it finds the best one (in terms of the energy norm)
- From iteration 16 onwards, CG has reached the exact solution, since a linear combination of 16 previous iterates is enough to represent the exact solution
- But also CG is subject to the speed-of-info limitation
- While iterate 16 is „perfect“, iterate 15 is still „completely wrong“

What comes next

- We see that all iterative schemes suffer from the „propagation speed limit“
  - Gauss-Seidel, SOR, CG, GMRES, etc. are all slow
  - All are subject to the limit that info can only be transported by one mesh point per iteration
  - Because of this incompressible CFD solvers based on using CG have implicitly a nonphysiscal „speed of sound“
- What can help?
  - Obvious answer:
  - Multigrid
  - In 1D, multigrid reduces to cyclic reduction and becomes a direct solver
- Thus we will now leave the 1D toy problem and look at the situation in 2D

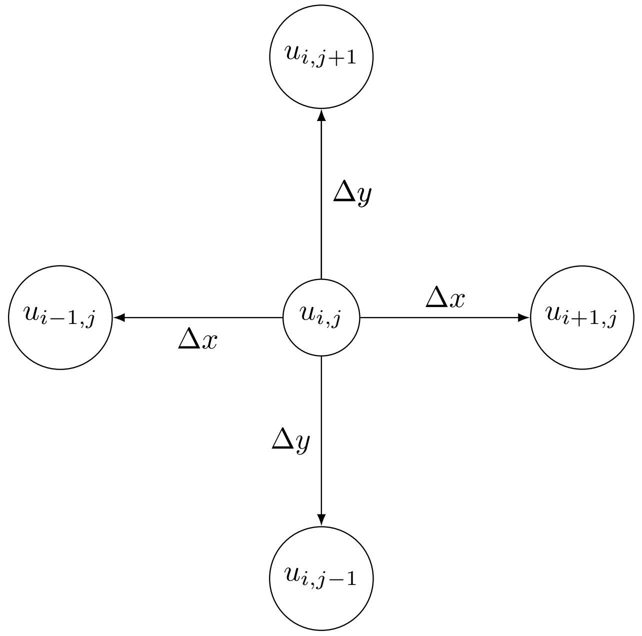
Discretization

- The Poisson equation must first be discretized, and this can be done in many different ways
  - FD, FE, FV, spectral
  - h-refined meshes, p-refinement, AMR
  - FE: continuous or discontinuous
  - Mixed formulations based on splitting the second order PDE in a system of first order … FOSLS
- We will here stay as simple as possible and use uniform cartesian meshes
  - Uniform mesh width h
  - n cells grid lines in x and y direction
  - n+1 grid lines
  - N= $(n-1)^2$ „true“ unknowns



Second order finite difference discretization

- We will not consider whether and when other (e.g. higher order) discretization can lead to more accurate solutions (in same compute time)
- We begin with the standard 5 point FD stencil
  - equivalent to FE discretization with triangles (splitting each square along one of the diagonals)
  - note that with proper scaling all eqns, the stencil can be executed with a minimal number of operations



$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}$$

$$\Delta u = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{h^2} + \mathcal{O}(h^2)$$

Matrix structure

- The discretization leads to matrix structures like

$$\left[\begin{array}{ccc|ccc|ccc}
4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\
\hline
-1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\
0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\
0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\
\hline
0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4
\end{array}\right]$$

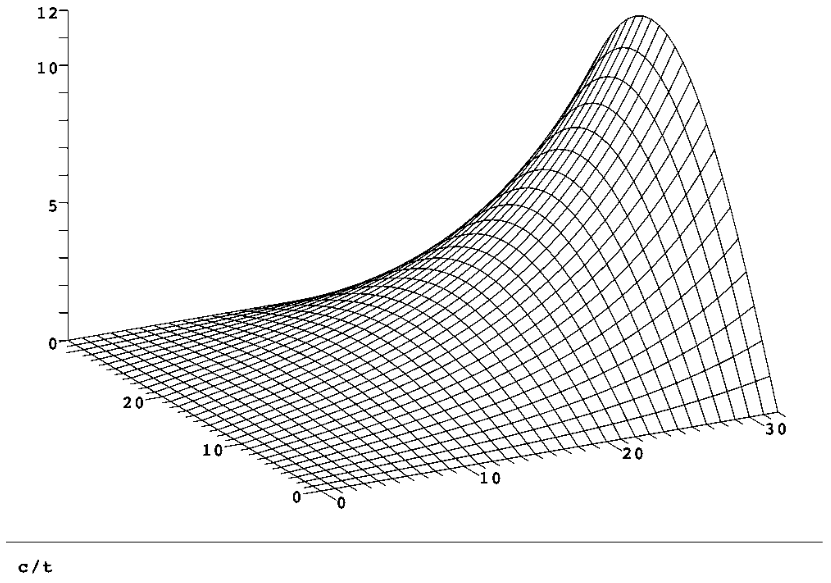
- With N unknowns, a banded solver will need O(N^2) operations
- Nested dissection can reduce this to O($N^{1.5}$)
- The condition number is

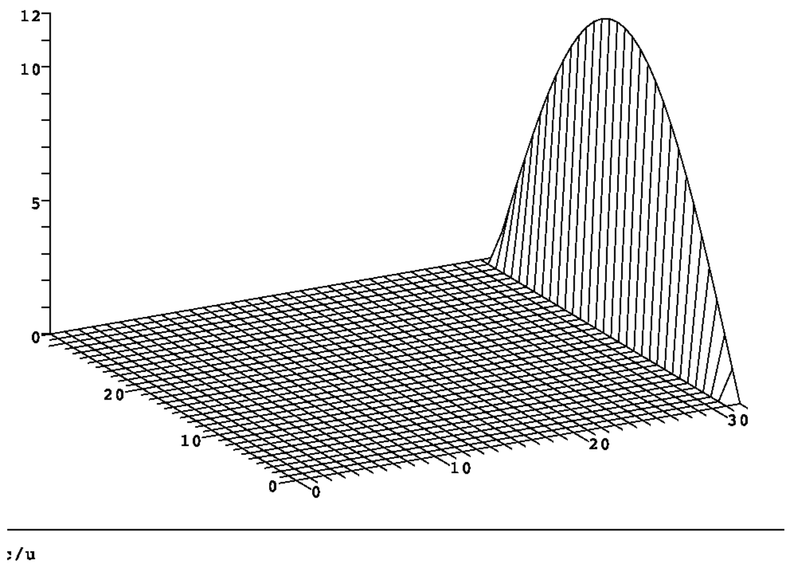
$$\kappa = O(h^{-2}) = O(N)$$

- The condition number will determine how many iterations are needed.

Graphical Illustration (Visualization)

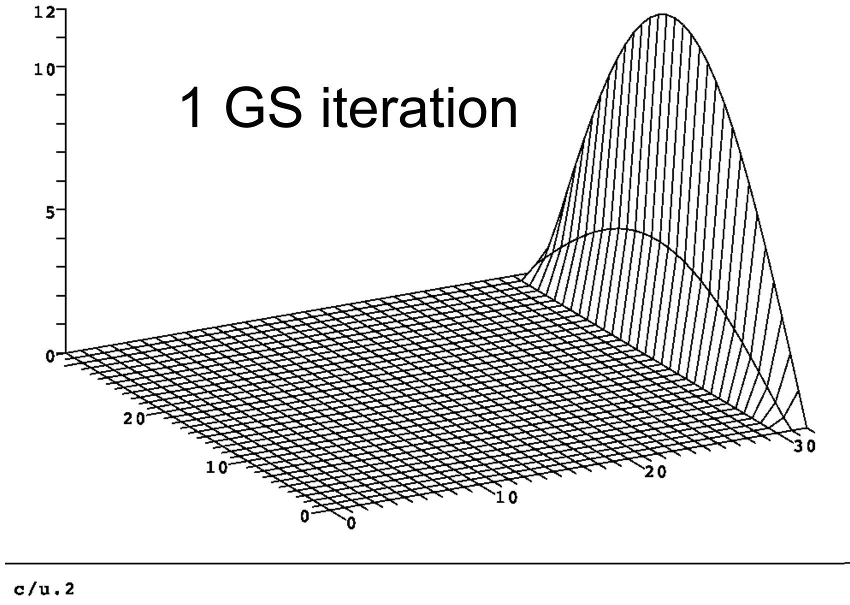
$$u = \sinh(x)\sin(y)$$

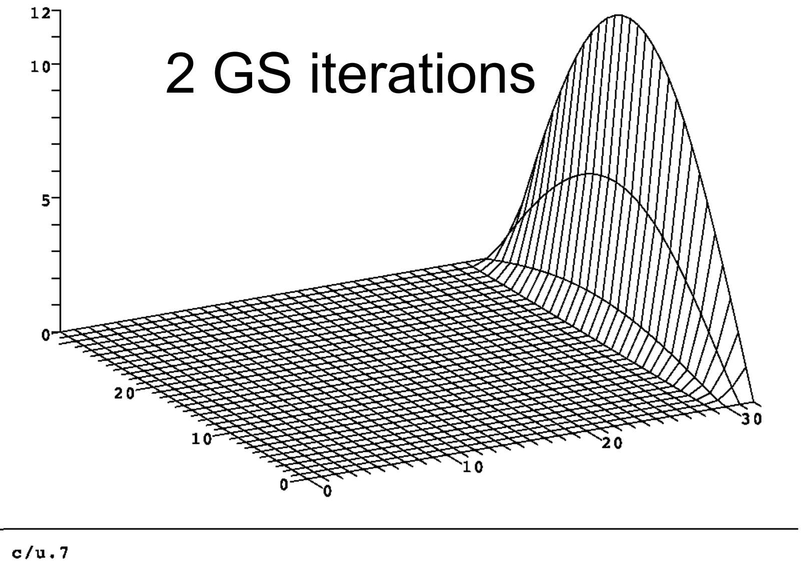




- Exact Solution (of PDE)
- Boundary values to start the iteration

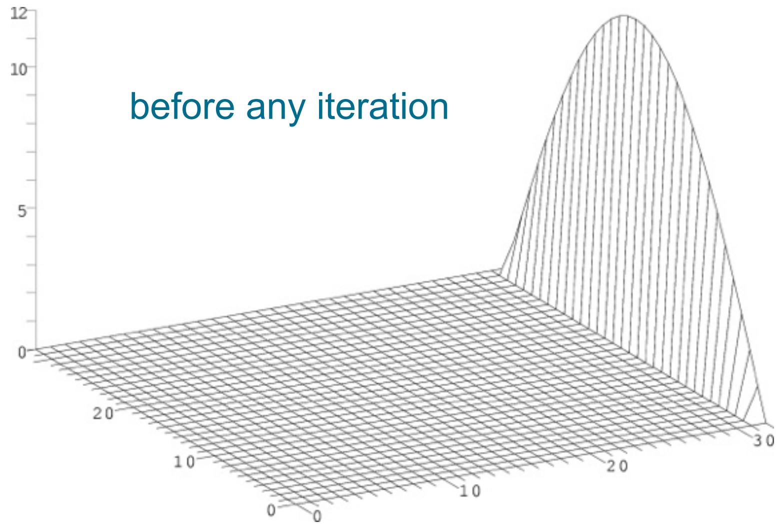
Visualization of Iterations

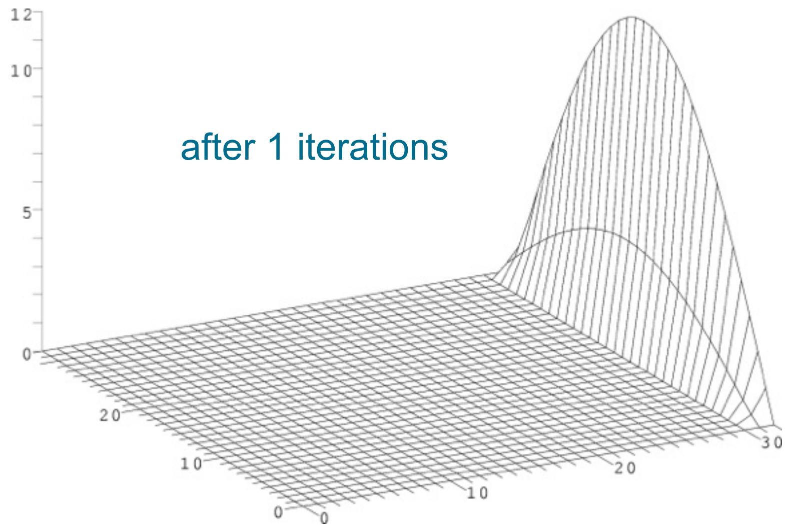


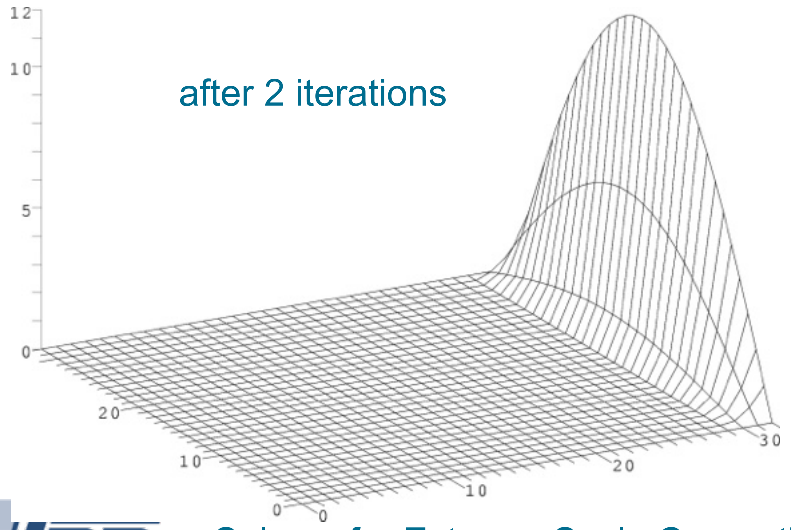


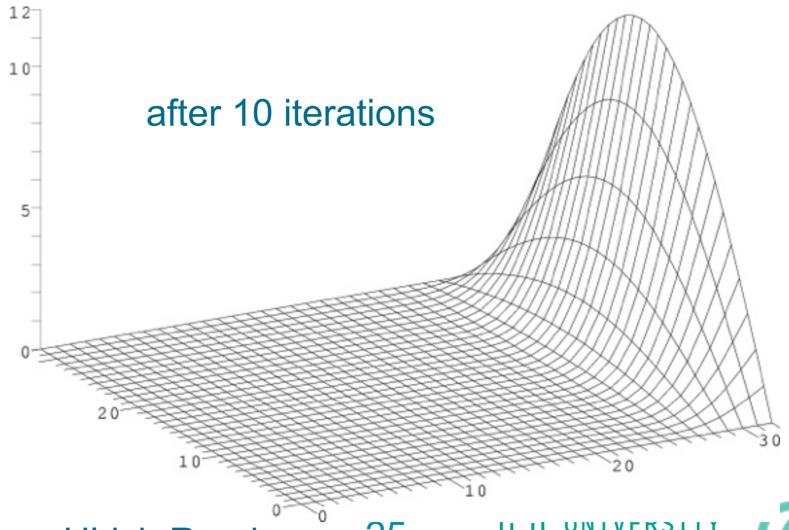
- View of (approximate) solution after first and
- after second Gauss-Seidel Iteration

Visualization of Convergence

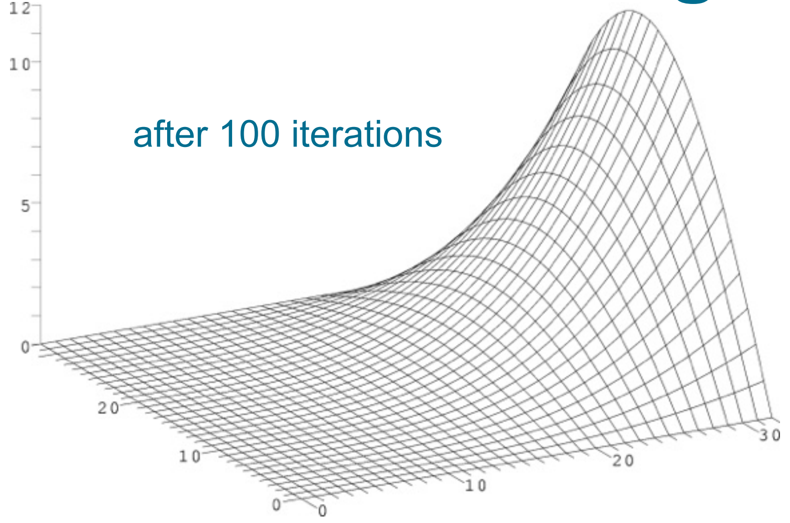

before any iteration

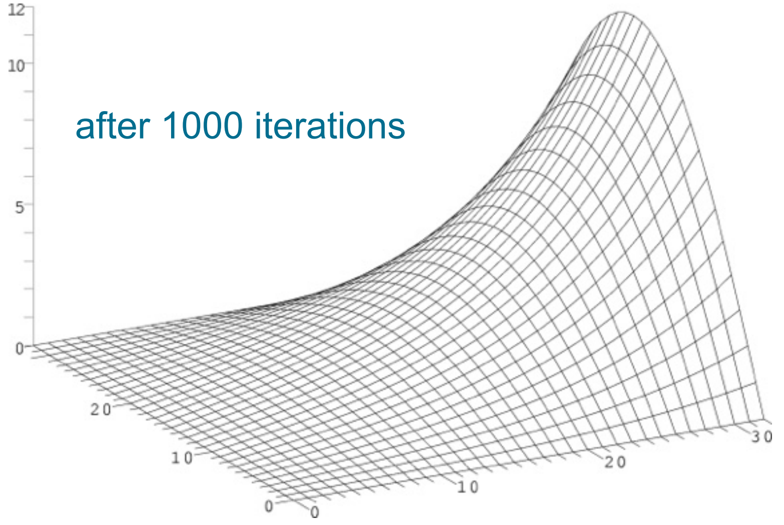

after 1 iterations

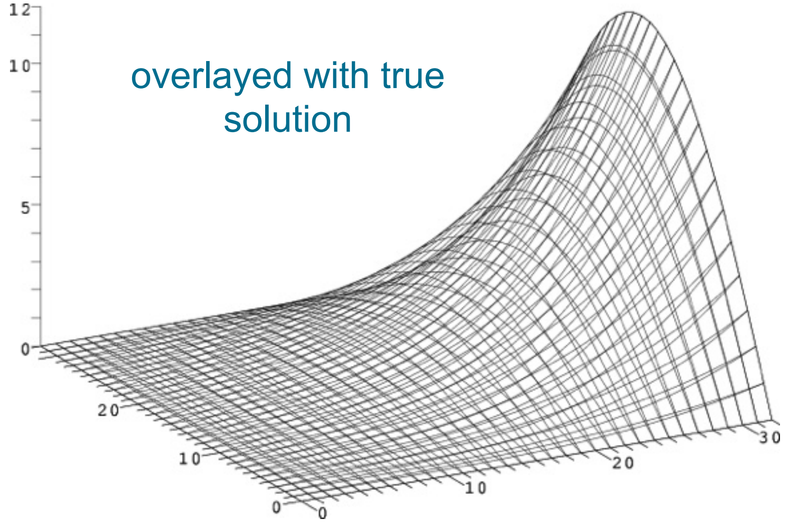

after 2 iterations

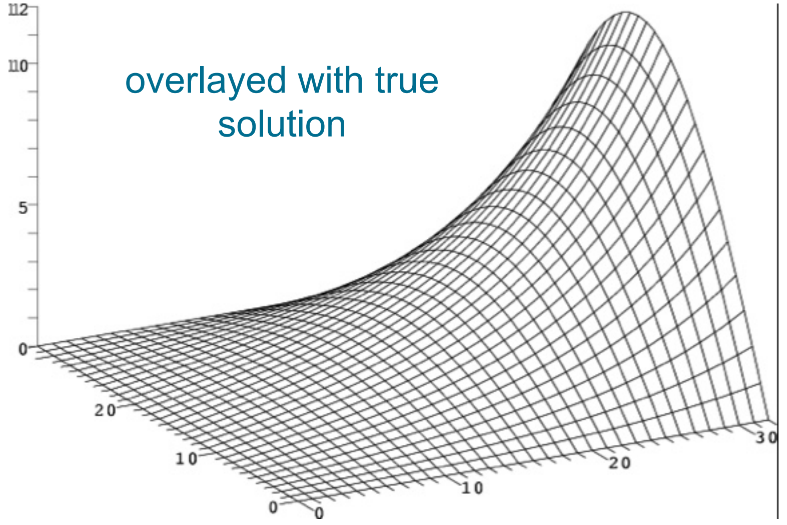

after 10 iterations

Visualization of Convergence



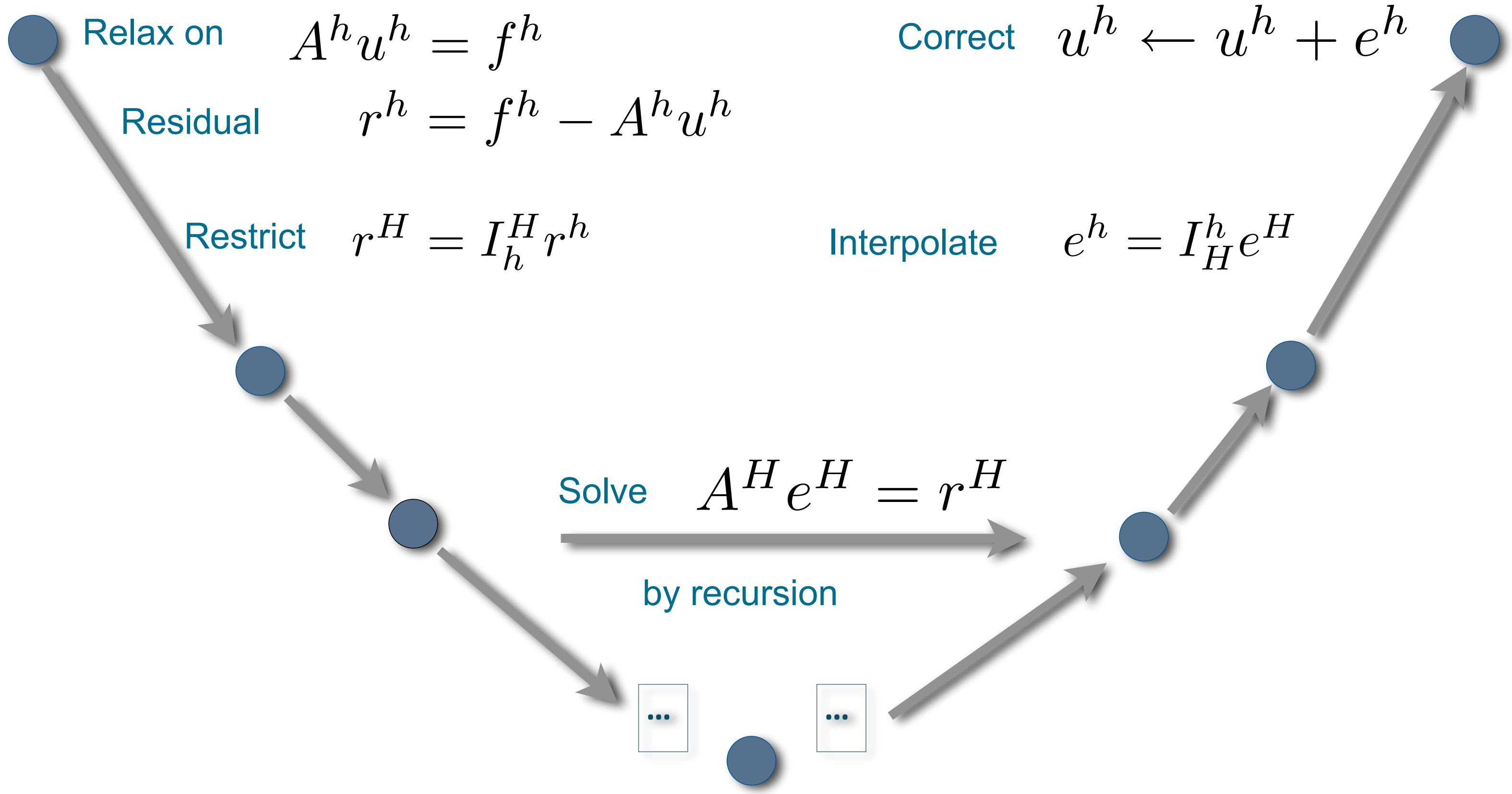






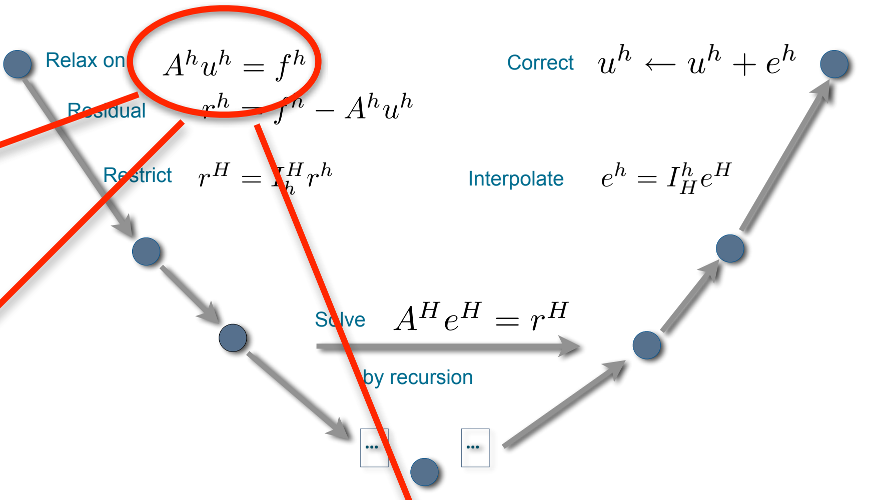
Geometric Multigrid: V-cycle

Goal: solve $A^h u^h = f^h$ using a hierarchy of grids



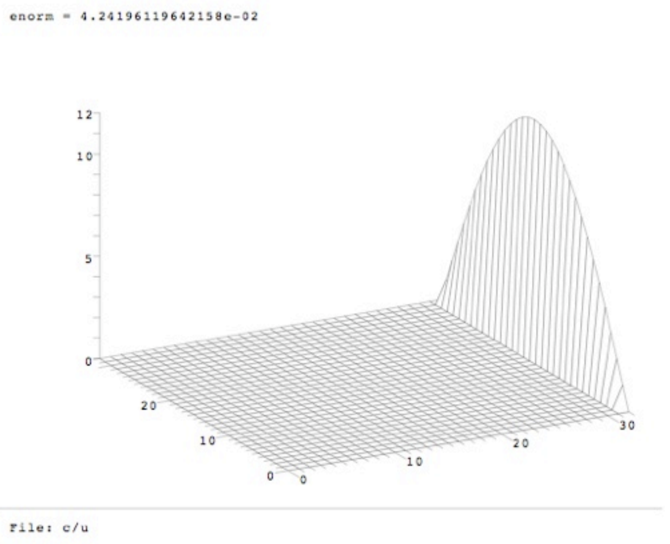
Relax on $A^h u^h = f^h$
Residual $r^h = f^h - A^h u^h$
Restrict $r^H = I_h^H r^h$
Solve $A^H e^H = r^H$
by recursion
…
…
Interpolate $e^h = I_H^h e^H$
Correct $u^h \leftarrow u^h + e^h$

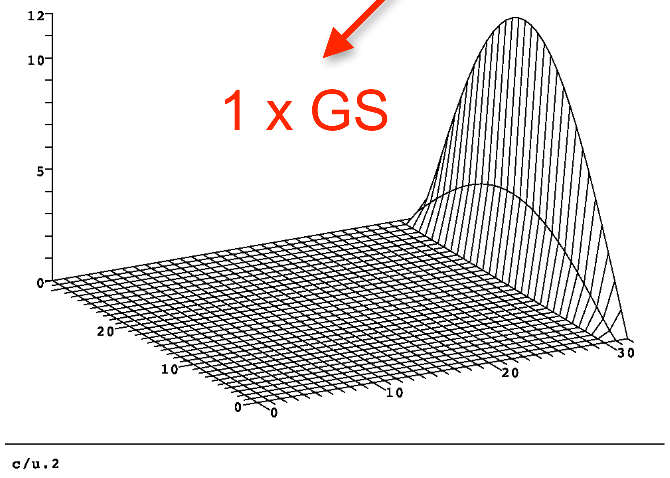
Relax 2 times

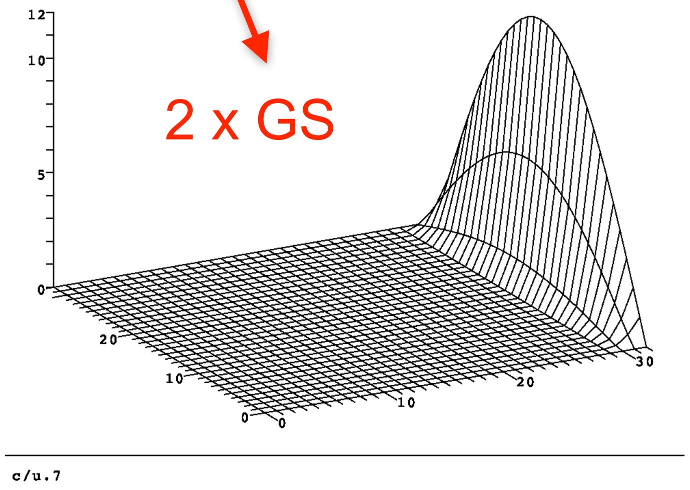


Relax on $A^h u^h = f^h$
Residual $r^h = f^h - A^h u^h$
Restrict $r^H = I_h^H r^h$
Solve $A^H e^H = r^H$
by recursion
...
...
Interpolate $e^h = I_H^h e^H$
Correct $u^h \leftarrow u^h + e^h$

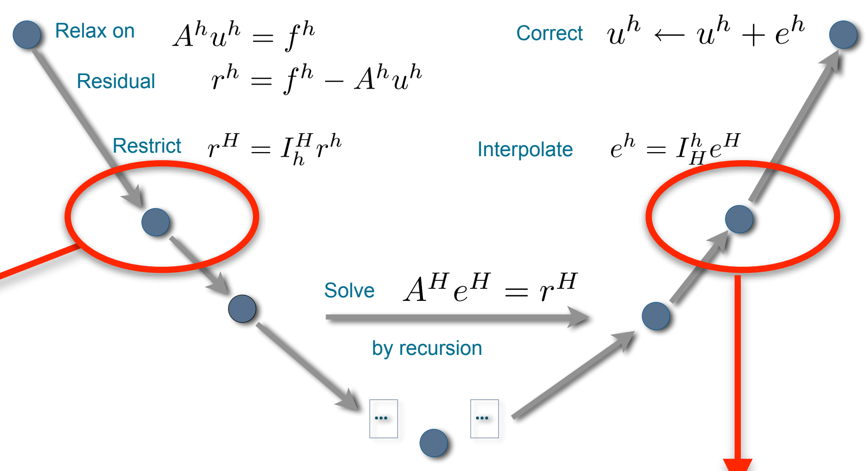
Initialized with 0 plus BC







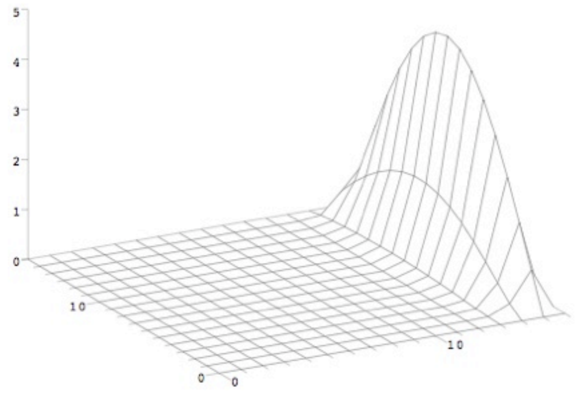
Compute residual
Restrict
Relax on coarser grid
Recursion



Relax on $A^h u^h = f^h$
Residual $r^h = f^h - A^h u^h$
Restrict $r^H = I_h^H r^h$
Solve $A^H e^H = r^H$
by recursion
...
...
Interpolate $e^h = I_H^h e^H$
Correct $u^h \leftarrow u^h + e^h$

enorm = 4.67013507655193e-02

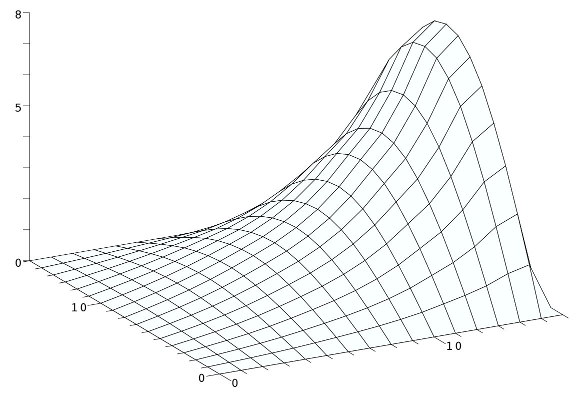
After 2x GS on next coarser grid



File: c/c.4/u.5

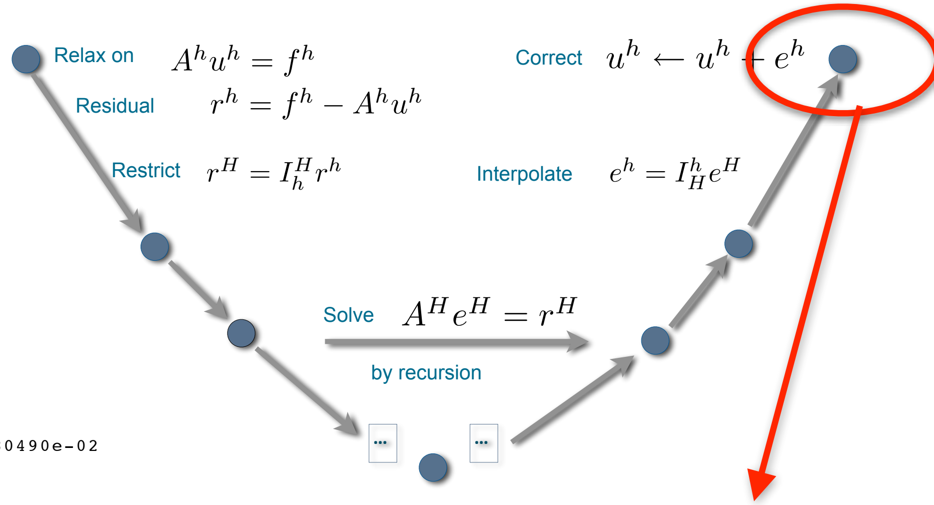
enorm = 1.38371516388512e-01

After V-cycle on next coarser grid



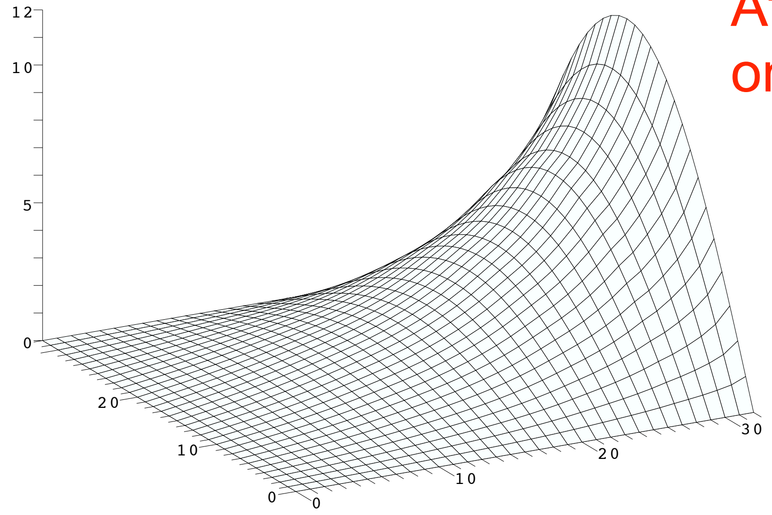
File: c/c.4/u.26

Interpolate correction
Correct fine grid solution
Post-Smooth



Relax on $A^h u^h = f^h$
Residual $r^h = f^h - A^h u^h$
Restrict $r^H = I_h^H r^h$
Solve $A^H e^H = r^H$
by recursion
Interpolate $e^h = I_H^h e^H$
Correct $u^h \leftarrow u^h + e^h$
...
...

```
enorm = 9.11869751280490e-02
```



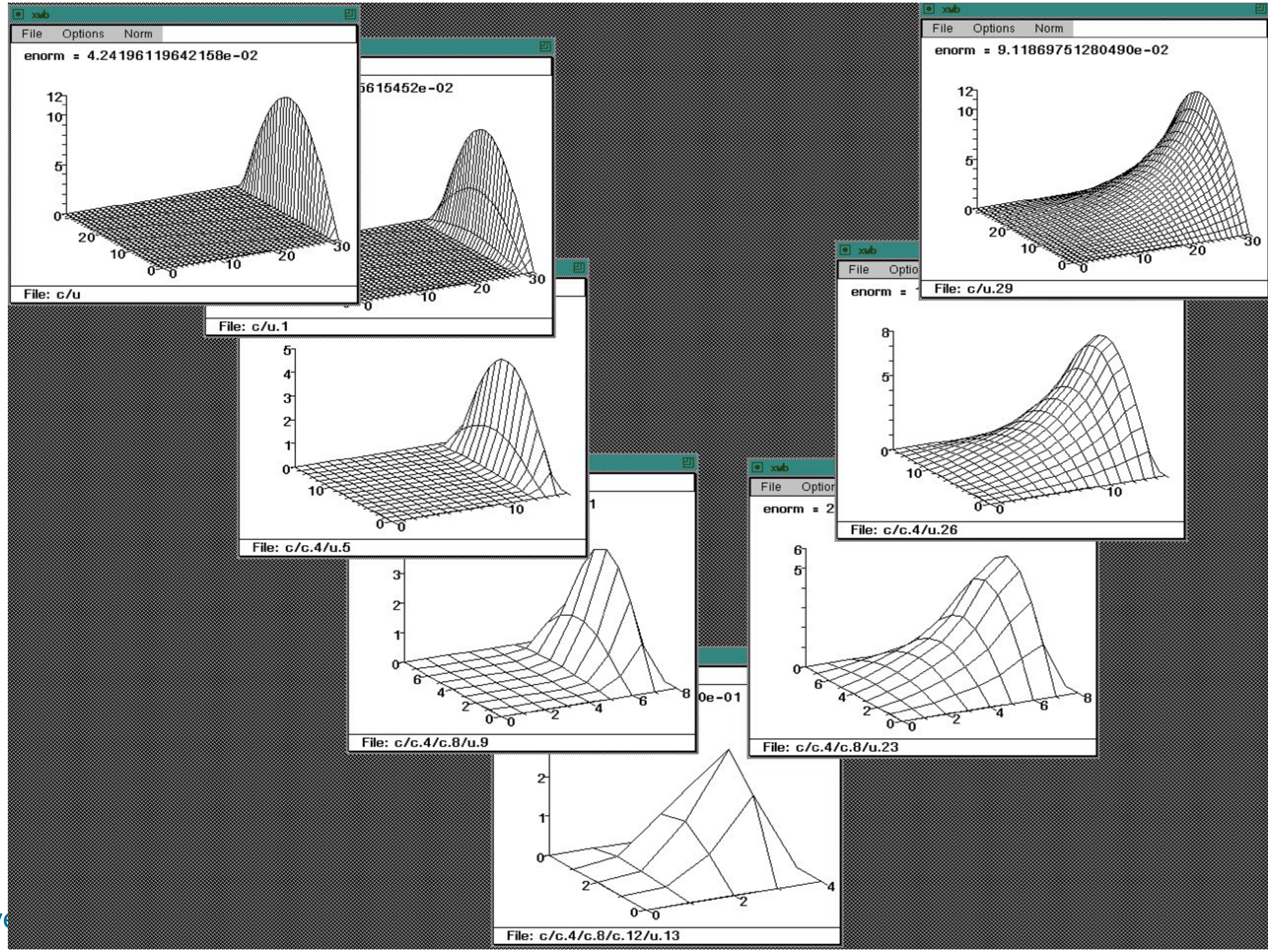
After V-cycle on finest grid

```
File: c/u.29
```

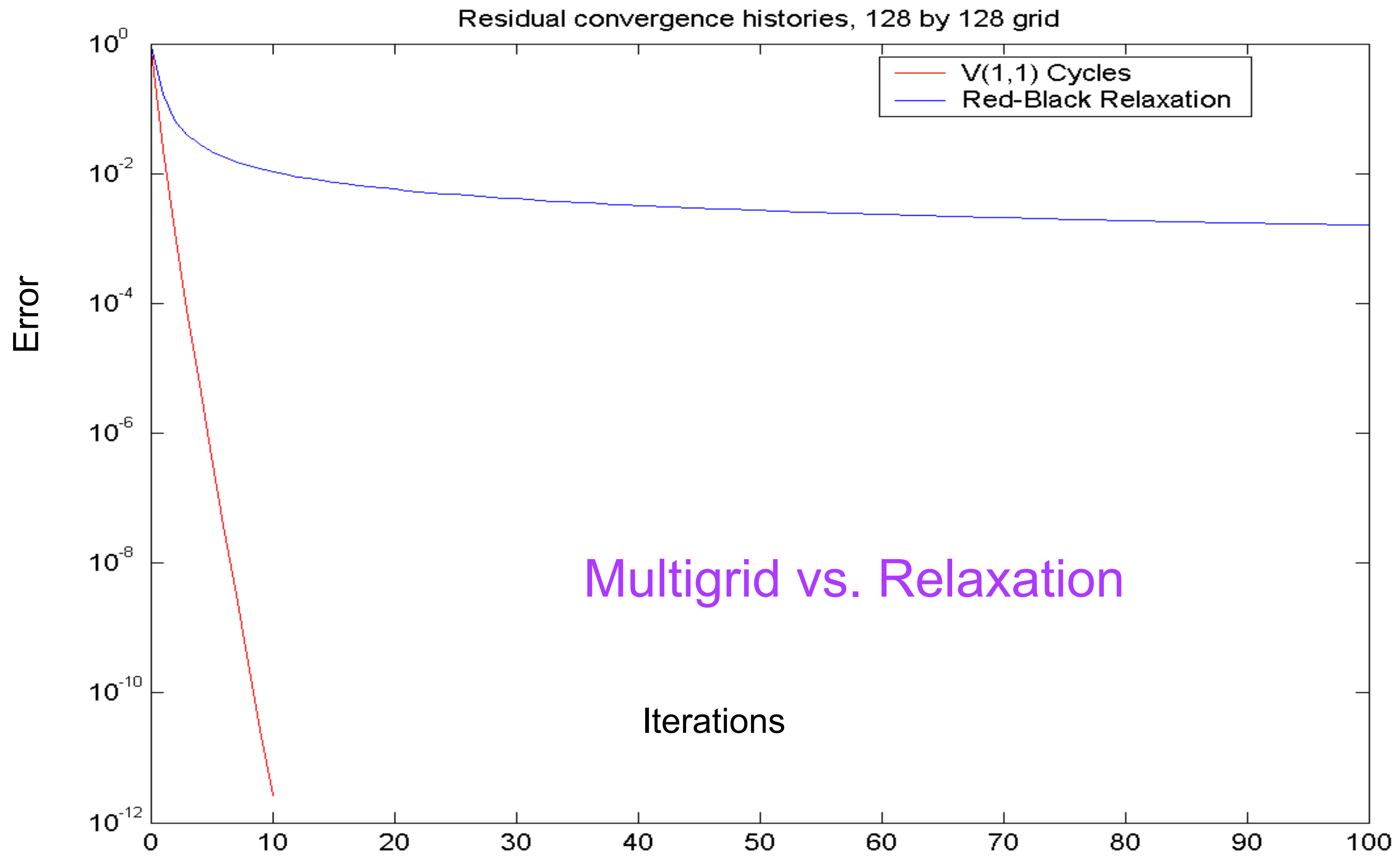
All steps of a multigrid V-cycle illustrated in one picture.

Even with only one V(2,1)-cycle, the result is qualitatively already quite good.

It is still an iterative method, and for convergence, the complete cycle must be iterated.



Relaxation and Multigrid compared



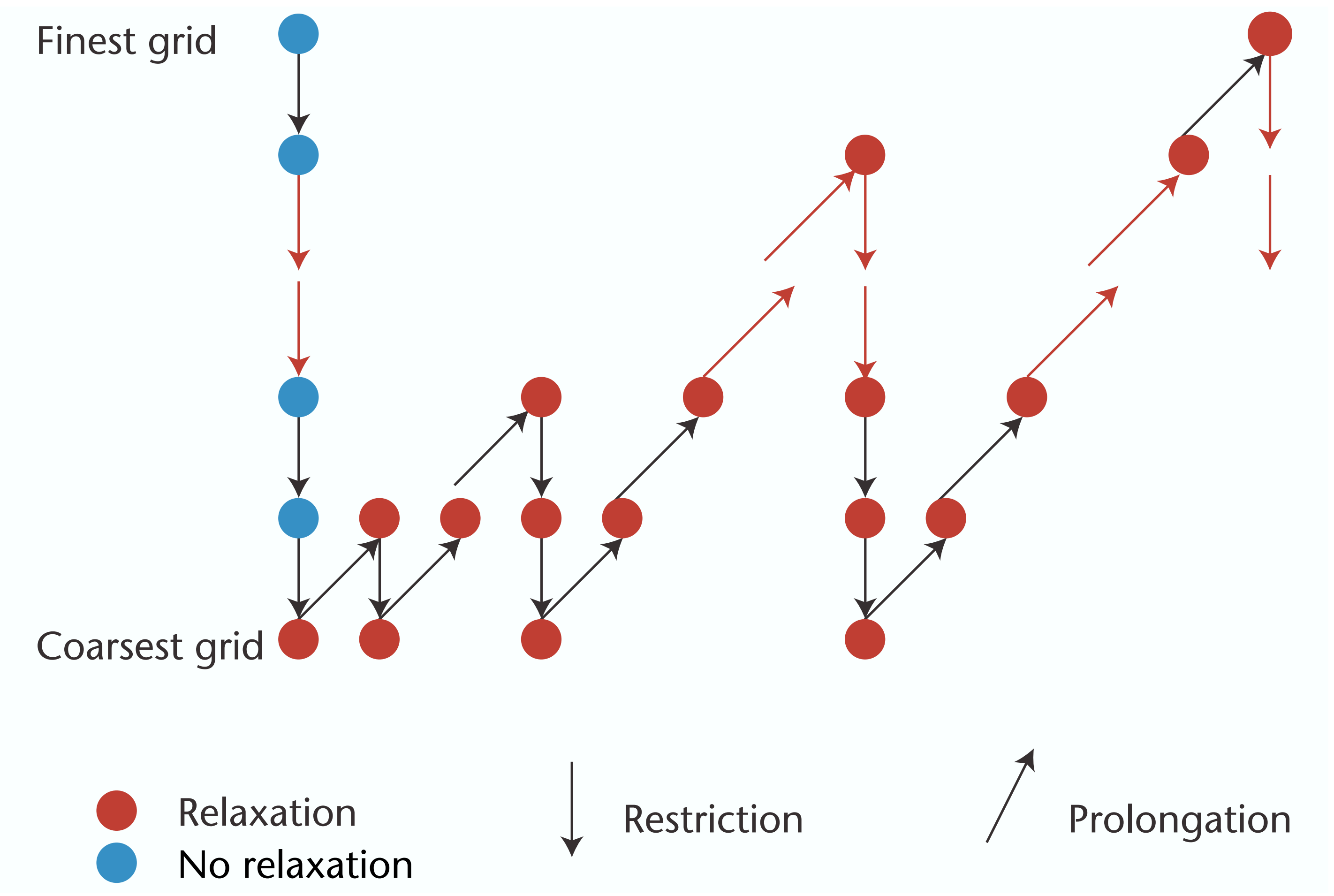
The Full Multi-Grid (FMG) Algorithm (nested iteration)

The multigrid V-cycle is an iterative method, and hence it requires an initial guess for the solution. This initial approximation can be obtained from a coarser grid, and so on recursively.

The FMG algorithm combines the grid-refinement approach with the V-cycle.

For many problems, FMG with just a single V-cycle per level suffices to reduce the error below truncation level. In this case, only $O(N)$ operations are required overall.

F-cycle, FMG, and V-cycle

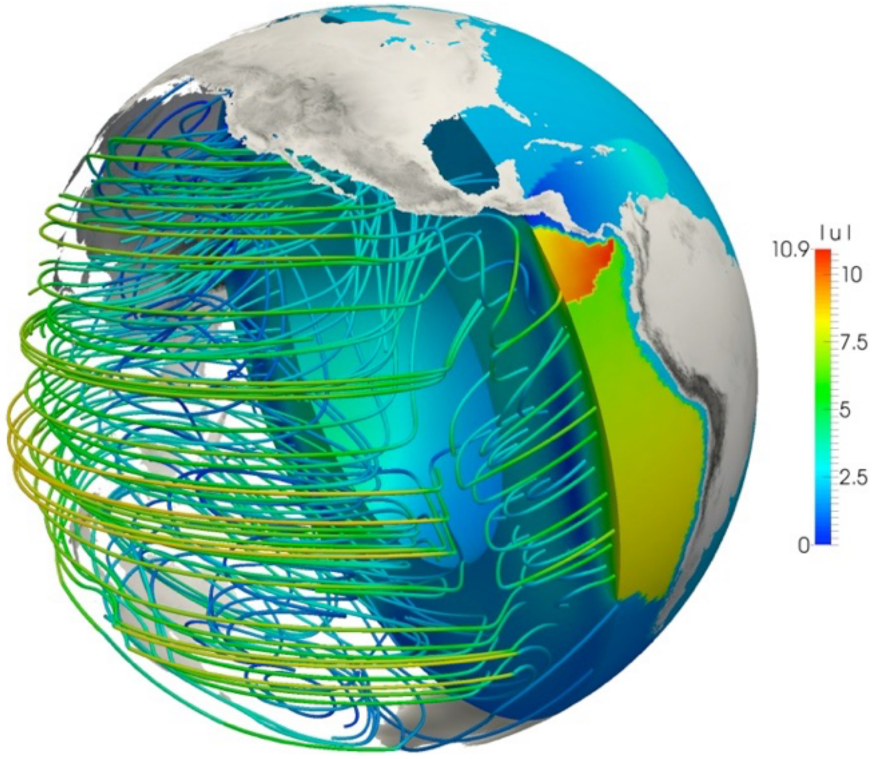


Multigrid summarized

- It alternates between
  - **Smoothing**, i.e. a Gauss-Seidel-like iteration with the goal to contribute the high frequency modes
  - **Coarse grid correction,** computed recursively, with the goal to contribute the low frequency modes
- The recursion leads to a V-cycle structure, alternatively W-cycle, when doing two coarse-grid corrections
- The overall cost is only a moderate factor more than processing on the finest grid (geometric series of flop count)
- It can be shown that the converge rate is smaller than 1 and independent of the mesh size
  - not depending on condition number
  - A fixed number of iterations is sufficient to compute the result with prescribed accuracy (but when the mesh gets finer more accuracy might be needed)
- The method can still be improved as „**Full Multigrid** (FMG
  - FMG can compute the solution to a (simple) PDE in cost proportional to the number of unknowns
  - The accuracy automatically increases when going to finer meshes

Solving large linear systems with multigrid:

An excursion to Earth Mantle Convection

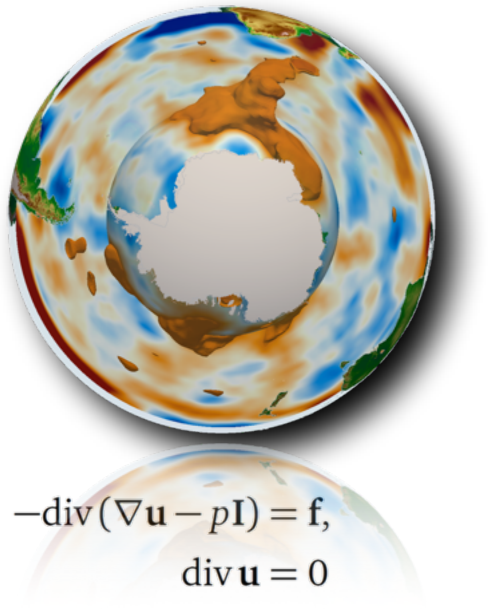


Simple Earth Mantle convection models:
Stokes equation coupled with energy transport

$$-\nabla \cdot (2\eta\epsilon(\mathbf{u})) + \nabla p = \rho(T)\mathbf{g},$$
$$\nabla \cdot \mathbf{u} = 0,$$
$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T - \nabla \cdot (\kappa \nabla T) = \gamma.$$

$\mathbf{u}$	velocity
p	dynamic pressure
T	temperature
ν	viscosity of the material
$\epsilon(\mathbf{u}) = \frac{1}{2}(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)$	strain rate tensor
ρ	density
$\kappa, \gamma, \mathbf{g}$	thermal conductivity, heat sources, gravity vector

Gmeiner, Waluga, Stengel, Wohlmuth, UR: Performance and Scalability of Hierarchical Hybrid Multigrid Solvers for Stokes Systems, SIAM J. Scientific Comp., 2015.



Stokes equation: $-\operatorname{div}(\nabla\mathbf{u} - p\mathbf{I}) = \mathbf{f}, \quad \operatorname{div}\mathbf{u} = 0$

FEM Discretization:

$$\mathbf{a}(\mathbf{u}_l, \mathbf{v}_l) + \mathbf{b}(\mathbf{v}_l, p_l) = \mathbf{L}(\mathbf{v}_l) \qquad \forall\, \mathbf{v}_l \in \mathbf{V}_l,$$
$$\mathbf{b}(\mathbf{u}_l, q_l) - \mathbf{c}(p_l, q_l) = 0 \qquad \forall\, q_l \in Q_l,$$

with: $\mathbf{a}(\mathbf{u},\mathbf{v}) := \int_\Omega \nabla\mathbf{u} : \nabla\mathbf{v}\,dx, \quad \mathbf{b}(\mathbf{u},q) := -\int_\Omega \operatorname{div}\mathbf{u}\cdot q\,dx$

Schur-complement formulation:

$$\begin{bmatrix} \mathbf{A}_l & \mathbf{B}_l^\top \\ \mathbf{0} & \mathbf{C}_l + \mathbf{B}_l\mathbf{A}_l^{-1}\mathbf{B}_l^\top \end{bmatrix} \begin{bmatrix} \underline{\mathbf{u}}_l \\ \underline{p}_l \end{bmatrix} = \begin{bmatrix} \underline{\mathbf{f}}_l \\ \mathbf{B}_l\mathbf{A}_l^{-1}\underline{\mathbf{f}}_l \end{bmatrix}$$

Mantle Convection

Why Mantle Convection?

- driving force for plate tectonics
- mountain building and earthquakes

Why Exascale?

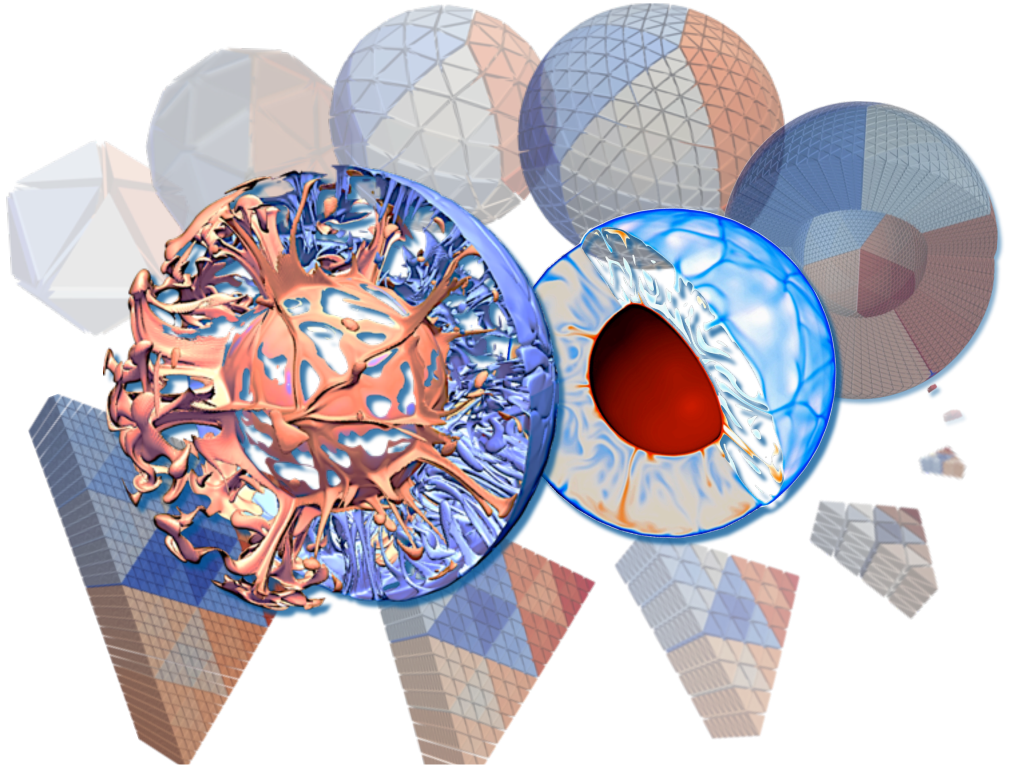
- mantle has 10^{12} km^3
- inversion and UQ blow up cost

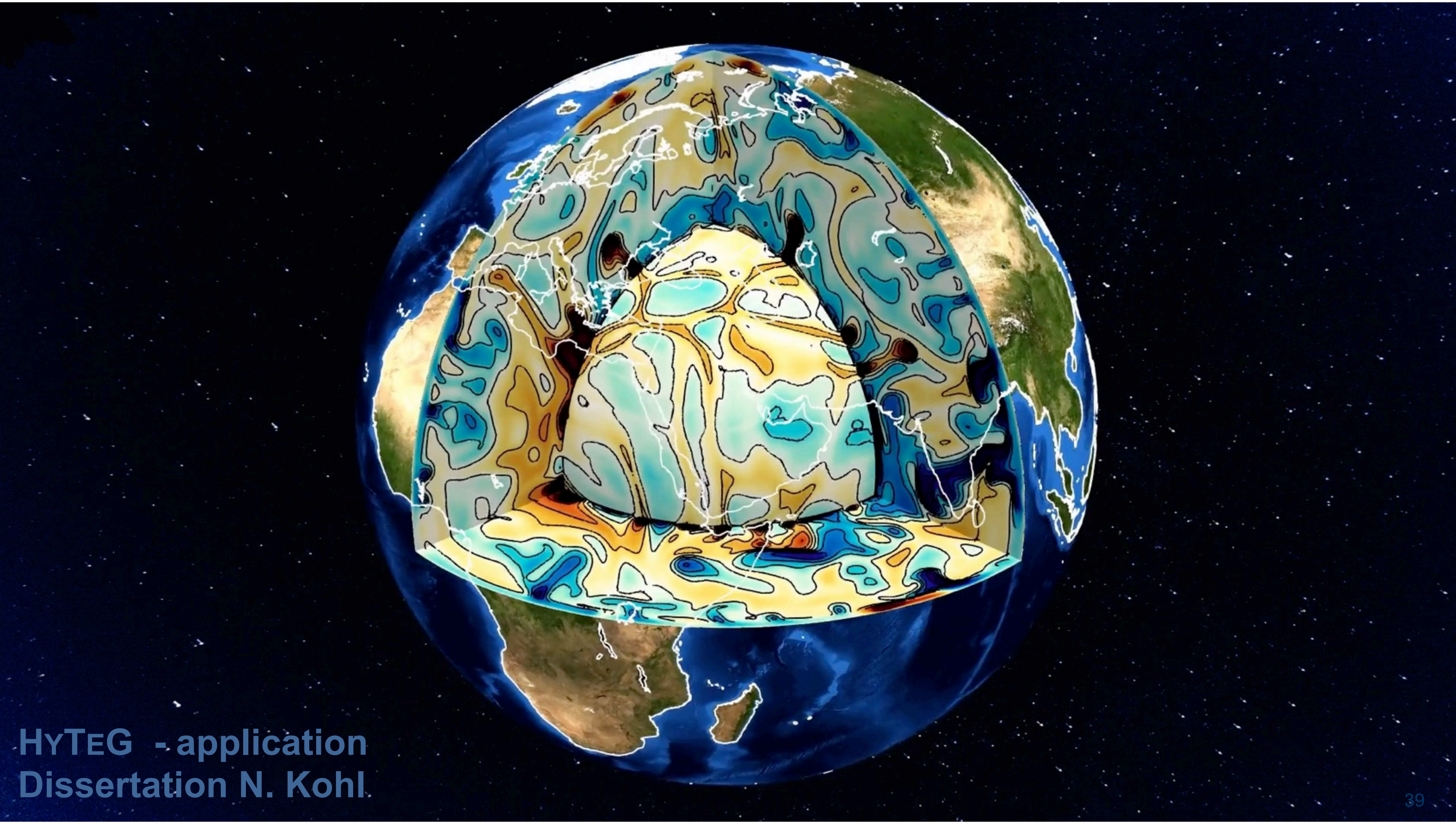
Why TERRANEO

- implementation based on HyTeG
- scalable and fast
- sustainable framework

Challenges

- **computer sciences:** software design for exascale systems
- **mathematics:** HPC performance oriented metrics
- **geophysics:** model complexity and uncertainty
- **bridging disciplines:** integrated co-design

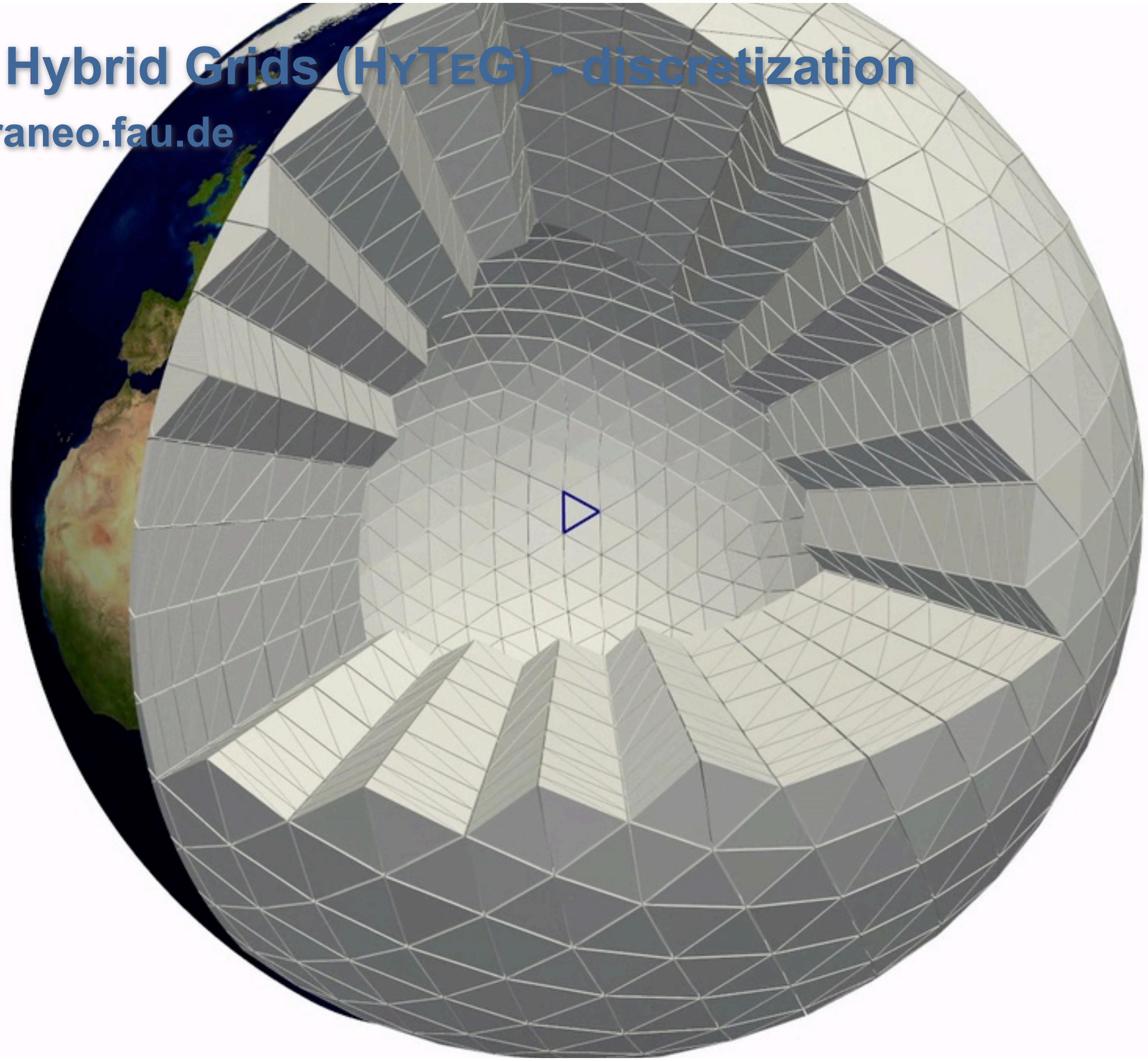




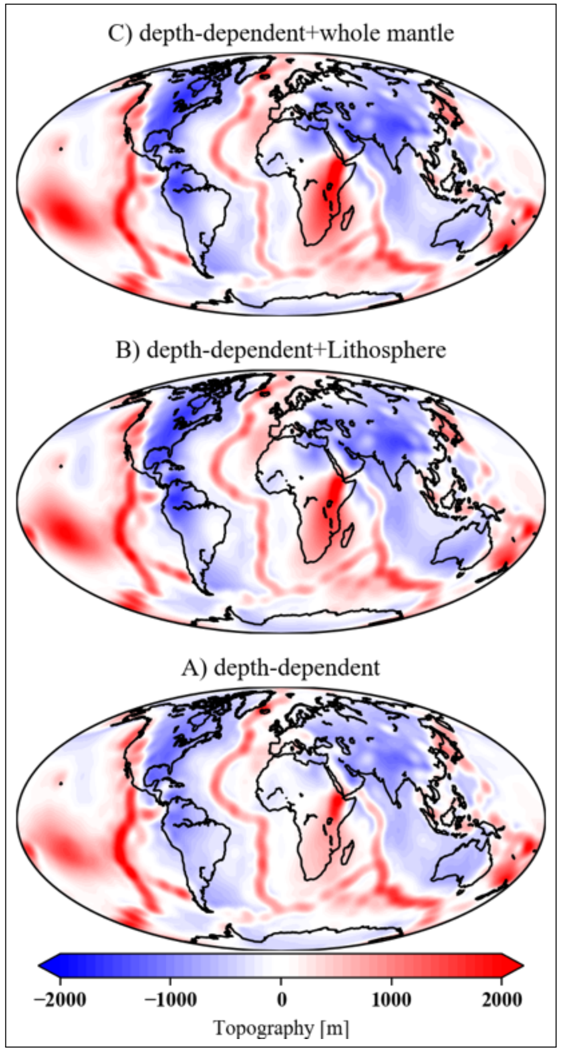
HyTeG - application
Dissertation N. Kohl

Hierarchical Hybrid Grids (HyTeG) - discretization

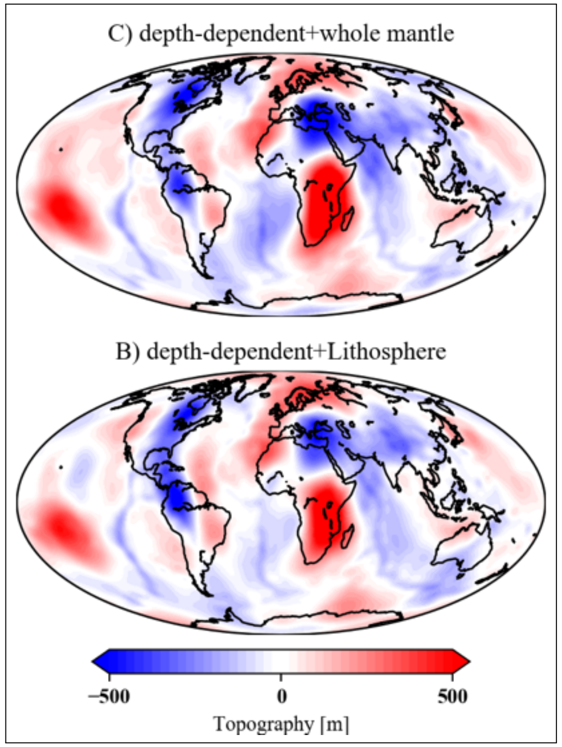
full video on **terraneo.fau.de**



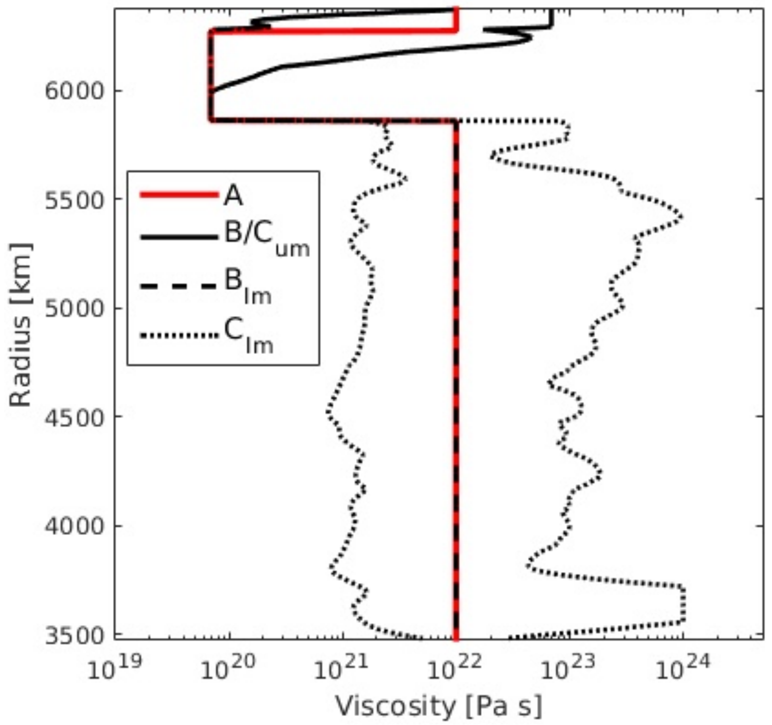
Geophysical in-silico experiment: dynamical topography



C) depth-dependent+whole mantle
B) depth-dependent+Lithosphere
A) depth-dependent
−2000
−1000
0
1000
2000
Topography [m]



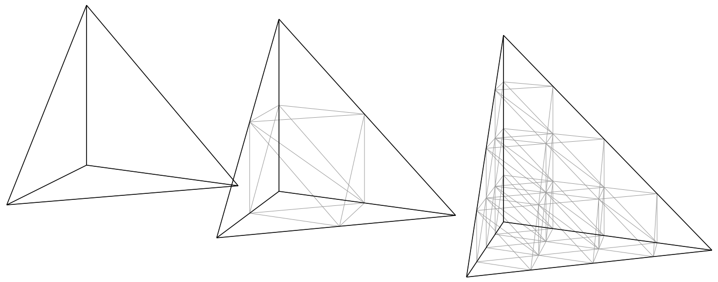
C) depth-dependent+whole mantle
B) depth-dependent+Lithosphere
−500
0
500
Topography [m]



- Global dynamical topography depending on different assumptions
- radial viscosity variations

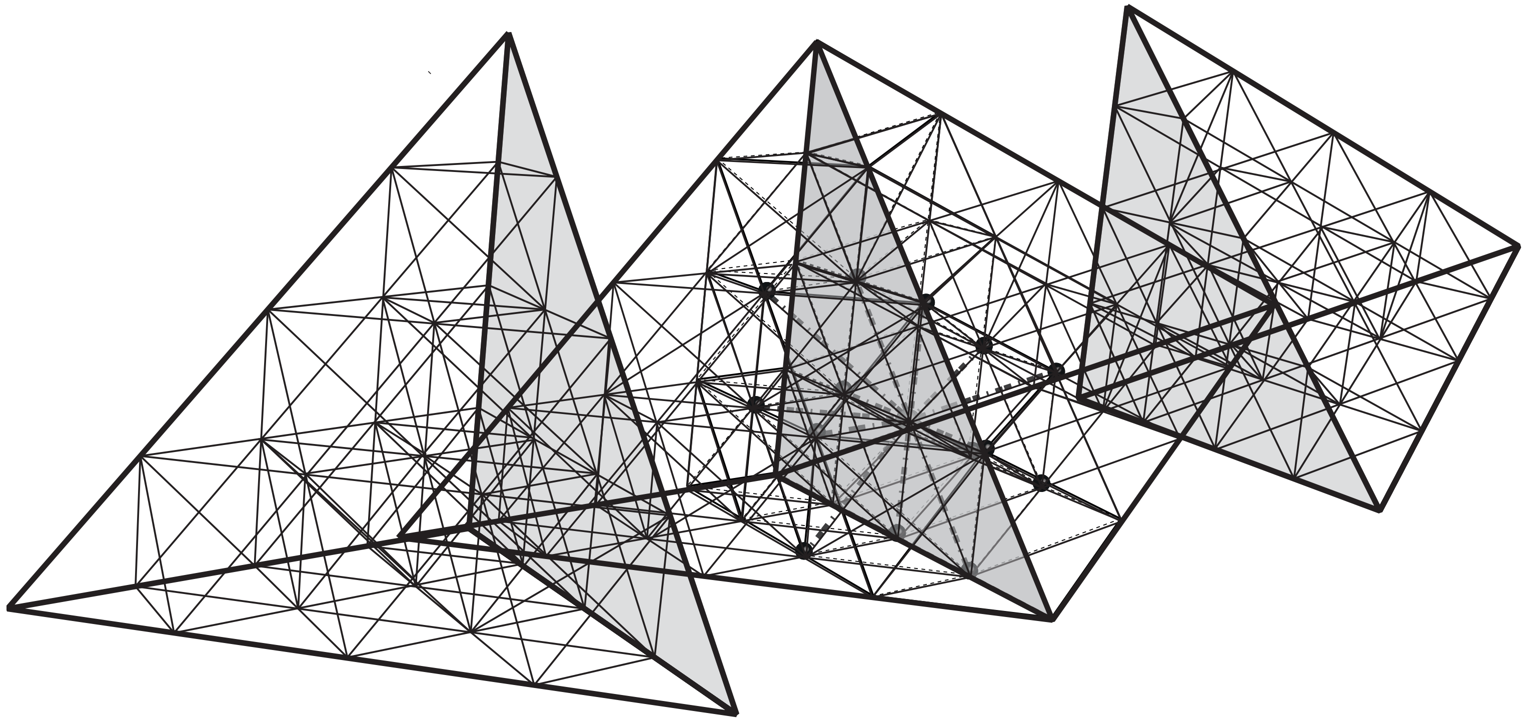
Weismüller, J., Gmeiner, B., Ghelichkhan, S., Huber, M., John, L., Wohlmuth, B., ... & Bunge, H. P. (2015). Fast asthenosphere motion in high-resolution global mantle flow models. *Geophysical Research Letters*, *42*(18), 7429-7435.

HyTeG: A matrix-free architecture for FE



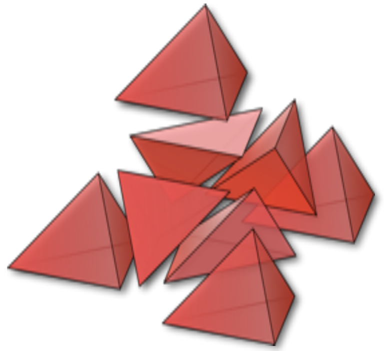
Structured refinement of an unstructured base mesh

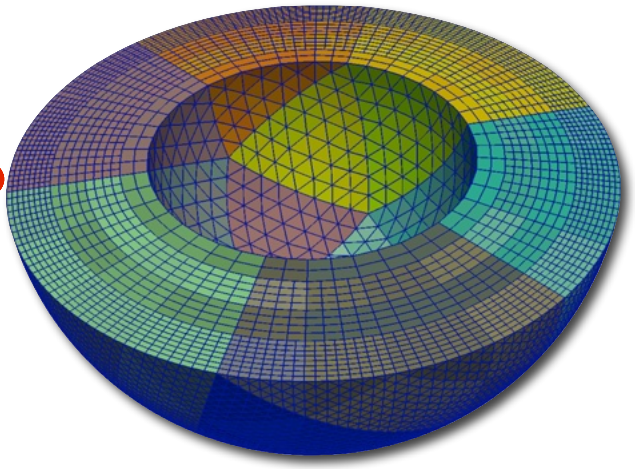
Geometrical Hierarchy: Volume, Face, Edge, Vertex



Hierarchical Hybrid Grids (HHG) and Multigrid (HyTeG)

- Parallelize multigrid for tetrahedral finite elements
  - partition domain
  - parallelize all operations on all grids
  - use clever data structures
  - matrix free implementation
- Coarse grids
  - agglomeration?
  - sequential dependency in grid hierarchy
- Elliptic problems always require global communication and thus coarser grids for the global data transport


Bey‘s Tetrahedral Refinement



B. Bergen, F. Hülsemann, UR, G. Wellein: „Is 1.7× 10^{10} unknowns the largest finite element system that can be solved today?“, SuperComputing, 2005.

Gmeiner, UR, Stengel, Waluga, Wohlmuth: Towards Textbook Efficiency for Parallel Multigrid, Journal of Numerical Mathematics: Theory, Methods and Applications, 2015

Algorithms Matter!

- Solution of Laplace equation in 3D wit $N=n^3$ unkowns
- Direct methods:
  - banded: $\sim n^7 = N^{2.33}$
  - nested dissection: $\sim n^6 = N^2$
- Iterative Methods:
  - Jacobi: $\sim 50\ n^5 = 50\ N^{1.66}$
  - CG: $\sim 100\ n^4 = 100\ N^{1.33}$
  - Full Multigrid: $\sim 200\ n^3 = 200\ N$

Energy per FLOP: 1nJ				
Computer Generation	gigascale: 10^9	terascale: 10^{12}	petascale: 10^{15}	exascale: 10^{18}
problem size: DoF=N	10^6	10^9	10^{12}	10^{15}
Direct method: $1*N^2$	0.278 Wh	278 kWh	278 GWh	278 PWh
Krylov method: $100*N^{1.33}$	10 Ws	28 Wh	278 kWh	2.77 GWh
Full Multigrid: 200 N	0.2 Ws	0.056 Wh	56 Wh	56 kWh
TerraNeo prototype (est. for Juqueen)	0.13 Wh	30 Wh	27 kWh	?

Exploring the limits

Gmeiner et al. 2016, A quantitative **performance study for Stokes** solvers at the extreme scale, Journal of Computational Science.

- matrix-free multigrid with Uzawa smoother
- optimized for minimal memory consumption

- 10^{13} Unknowns correspond to 80 TByte for the solution vector
- Juqueen had ~450 TByte memory
- matrix free implementation essential

nodes	threads	DoFs	iter	time	time w.c.g.	time c.g. in %
5	80	$2.7 \cdot 10^9$	10	685.88	678.77	1.04
40	640	$2.1 \cdot 10^{10}$	10	703.69	686.24	2.48
320	5 120	$1.2 \cdot 10^{11}$	10	741.86	709.88	4.31
2 560	40 960	$1.7 \cdot 10^{12}$	9	720.24	671.63	6.75
20 480	327 680	$1.1 \cdot 10^{13}$	9	776.09	681.91	12.14

Algorithms for saddle point systems

Benzi, M., Golub, G. H., & Liesen, J. (2005). Numerical solution of saddle point problems. *Acta numerica*, *14*, 1-137.
Rozložník, M. (2018). *Saddle-point problems and their iterative solution*. Basel: Birkhäuser.

- Monolithic multigrid

Gmeiner, B., Rüde, U., Stengel, H., Waluga, C., & Wohlmuth, B. (2015). Towards textbook efficiency for parallel multigrid. *Numerical Mathematics: Theory, Methods and Applications*, *8*(1), 22-46.

Drzisga, D., John, L., Rude, U., Wohlmuth, B., & Zulehner, W. (2018). On the analysis of block smoothers for saddle point problems. *SIAM Journal on Matrix Analysis and Applications*, *39*(2), 932-960.

Kohl, N., & Rüde, U. (2022). Textbook efficiency: massively parallel matrix-free multigrid for the Stokes system. *SIAM Journal on Scientific Computing*, *44*(2), C124-C155.

- Exploiting block structure and/or Schur complement formulation

Darrigrand, V., Dumitrasc, A., Kruse, C., & Rüde, U. (2023). Inexact inner–outer Golub–Kahan bidiagonalization method: A relaxation strategy. *Numerical Linear Algebra with Applications*, *30*(5), e2484.
Dumitrasc, A., Kruse, C., & Rüde, U. (2024). Deflation for the off-diagonal block in symmetric saddle point systems. *SIAM Journal on Matrix Analysis and Applications*, *45*(1), 203-231.

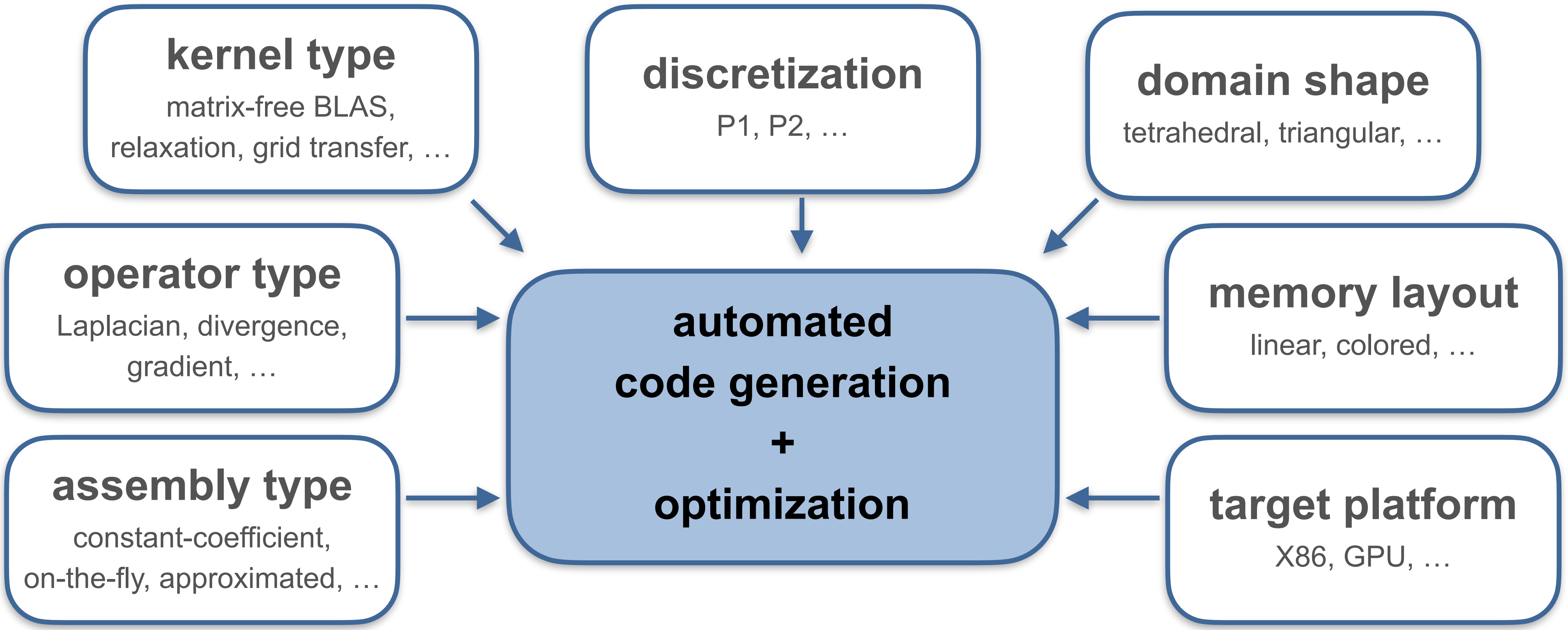
HYTEG

Hybrid Tetrahedral Grids

Automatic Code Generation for Multigrid Metaprogramming

The HyTeG framework - code generation

Combinatorial explosion leads to many different kernels and would require **an enormous manual implementation and optimization effort**!


kernel type
matrix-free BLAS, relaxation, grid transfer, …
discretization
P1, P2, …
domain shape
tetrahedral, triangular, …
operator type
Laplacian, divergence, gradient, …
automated code generation + optimization
memory layout
linear, colored, …
assembly type
constant-coefficient, on-the-fly, approximated, …
target platform
X86, GPU, …

Performance Analysis and Code Optimization

Measurements

- Fritz Supercomputer at NHR@FAU
- Matrix-vector multiplication (without communication)
- Single socket: Intel Xeon Platinum 8360Y (”Ice Lake”)
- 36 cores per socket
- LIKWID performance monitoring and benchmarking suite

Generated optimizations

- Symmetry (S)
- Inter-element vectorization (V)
- Loop invariants (I)
- Cubes loop strategy (C)
- Under-integration (U)
- Fused quadrature loops (fQ)
- Tabulation (T)

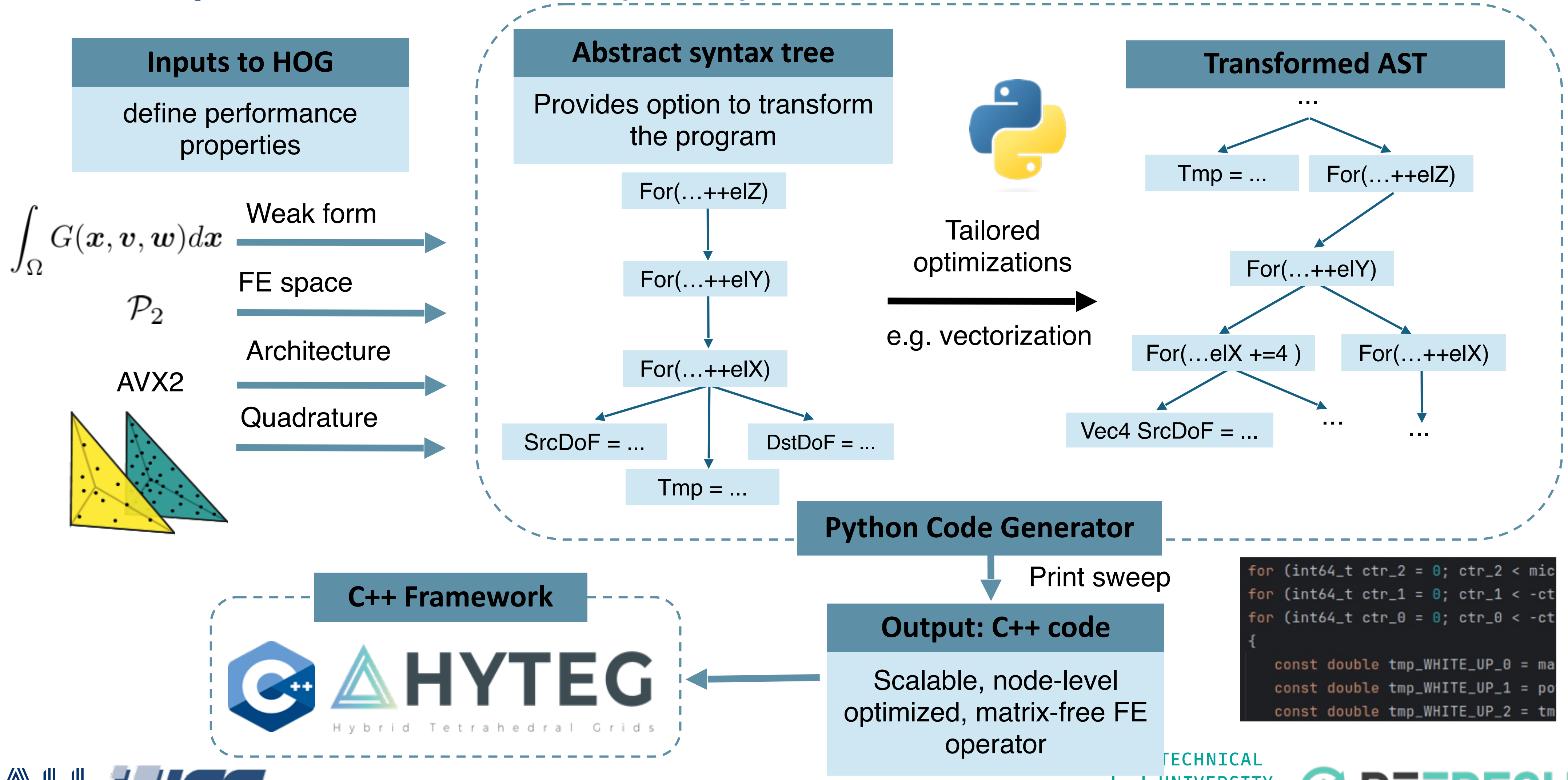
Optimization search

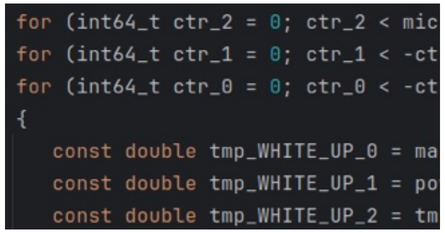
- Generate all combinations
- Determine the **set of most effective optimizations**

Optimization path

For the fastest operator:
Roofline analysis of
optimization path

HyTeG Operator Generator (HOG)



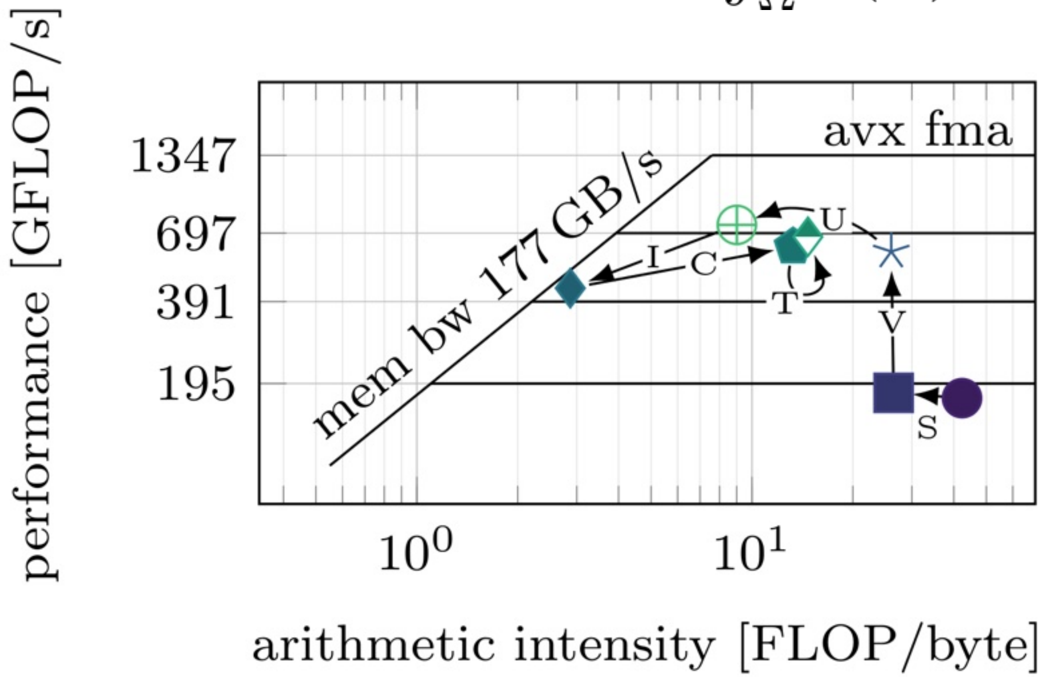


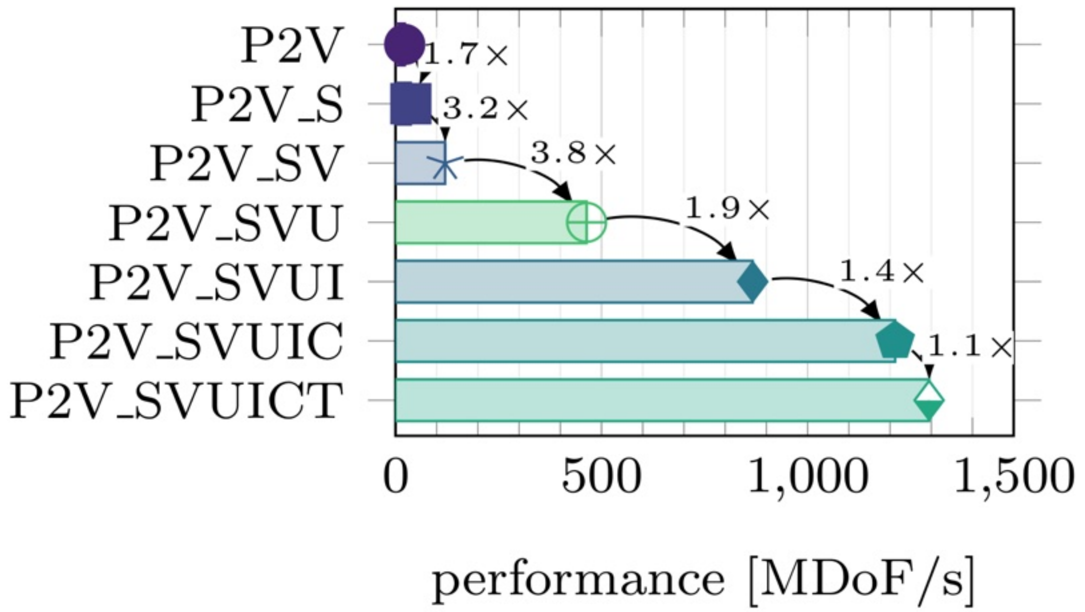
```
for (int64_t ctr_2 = 0; ctr_2 < mic
for (int64_t ctr_1 = 0; ctr_1 < -ct
for (int64_t ctr_0 = 0; ctr_0 < -ct
{
   const double tmp_WHITE_UP_0 = ma
   const double tmp_WHITE_UP_1 = po
   const double tmp_WHITE_UP_2 = tm
```

Optimization Path: P2V

- Symmetry (S)
- Inter-element vectorization (V)
- Loop invariants (I)
- Cubes loop strategy (C)
- Under-integration (U)
- Fused quadrature loops (fQ)
- Tabulation (T)

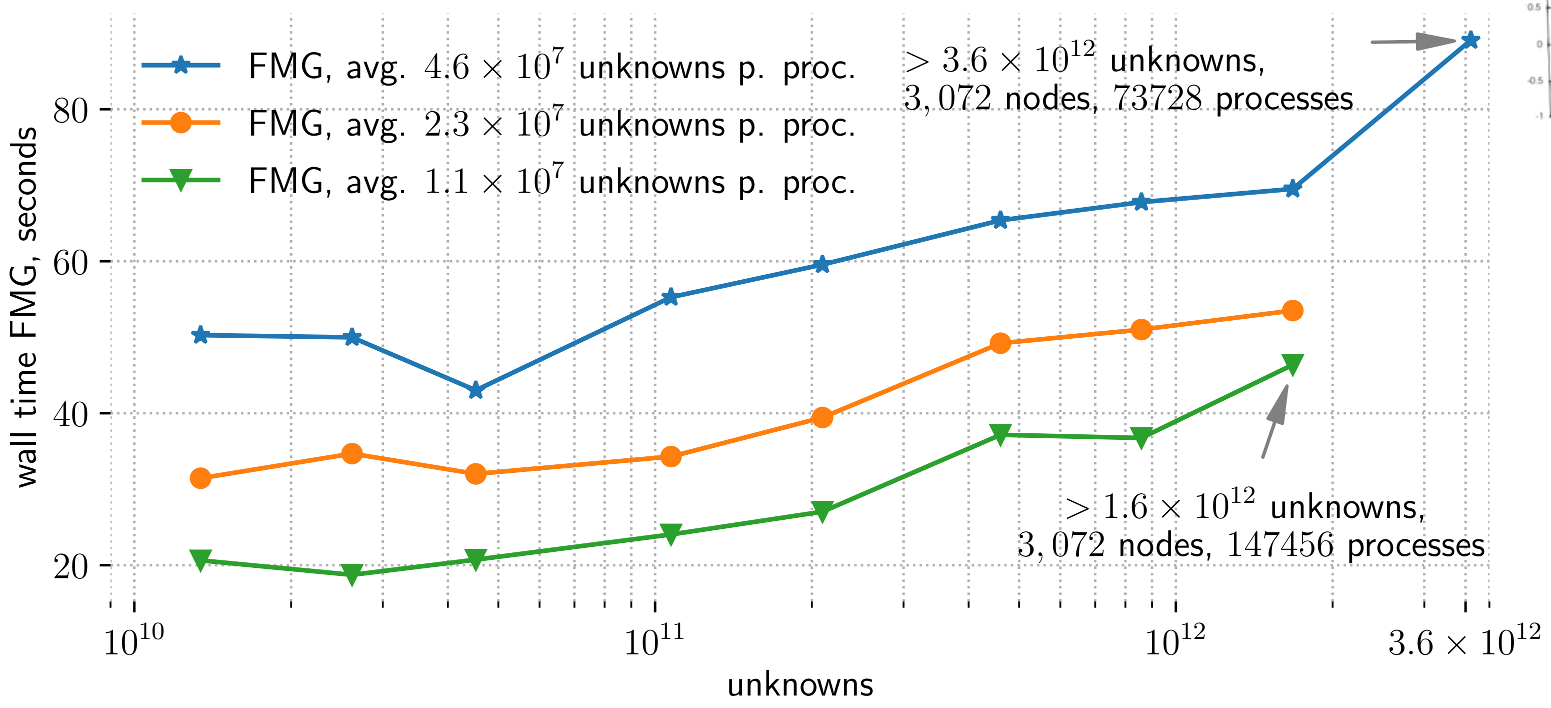
Operator P2V: $\int_\Omega k(\boldsymbol{x})\nabla v \cdot \nabla w, \mathcal{P}_2$

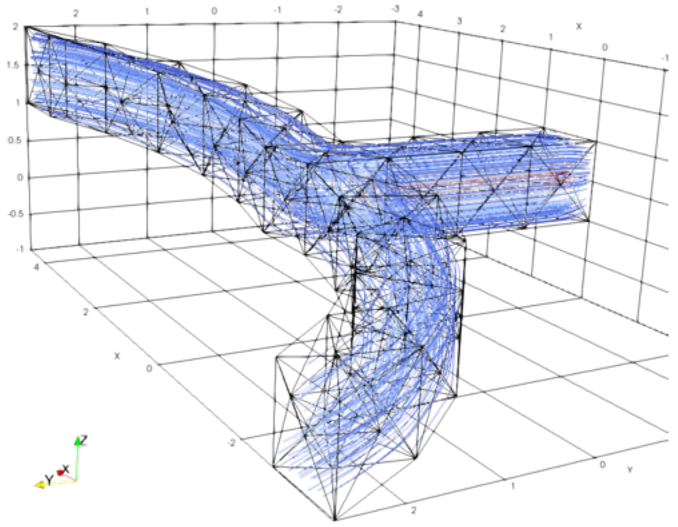




- Starting point: already compute-bound
- Series of opts reducing arithmetic intensity
- Compute-intense **P2V becomes memory-bound** with P2V_SVUI
- Cubes loop applicable -> more speed-up
- **58x accumulated speed-up, 50% peak, 1.4 GDoF/s**

HyTeG: Scaling for the Stokes Problem





Kohl, N., & Rüde, U. (2022). **Textbook efficiency**: massively parallel matrix-free multigrid for the Stokes system. *SIAM Journal on Scientific Computing*, *44*(2), C124-C155.

Kohl, N., Mohr, M., Eibl, S., & Rüde, U. (2022). A Massively Parallel Eulerian-Lagrangian Method for Advection-Dominated Transport in Viscous Fluids. *SIAM Journal on Scientific Computing*, *44*(3), C260-C285.

$$\frac{\mathfrak{W}(\mathrm{MG})}{\mathfrak{W}(A)} < 10$$

Textbook Multigrid Efficiency

Textbook Multigrid Efficiency (TME)

„Textbook multigrid efficiency means solving a discrete PDE problem with a computational effort that is only a small (less than 10) multiple of the operation count associated with the discretized equations itself.“
[Brandt, 98]

This is a programmatic claim - not a theorem.
For which types of PDE is it achievable?

Work unit (WU)

- Linear system $Ax = b.$
- Work unit (WU) to apply operator: $1\mathrm{WU} := \mathfrak{W}(A)$
  - or perform one sweep of relaxation
- TME achieved, if work for MG solver(!) less than 10 WU:

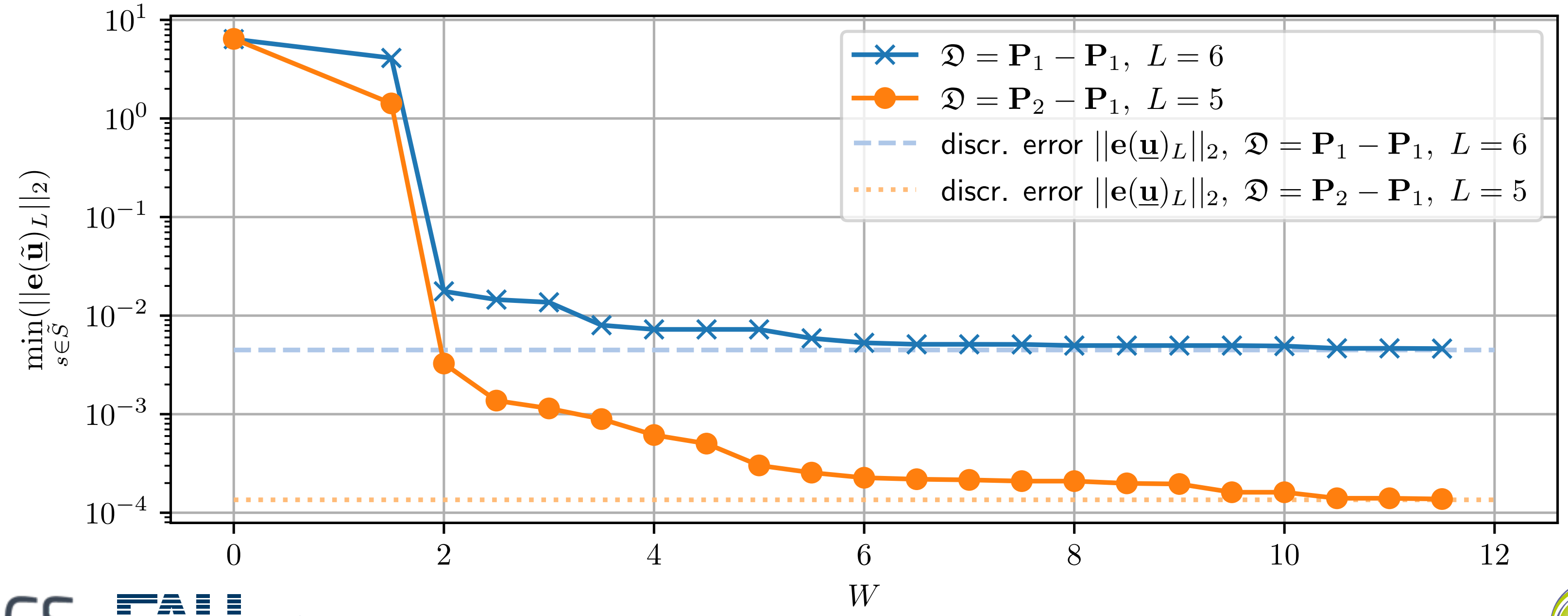
$$\frac{\mathfrak{W}(\mathrm{MG})}{\mathfrak{W}(A)} < 10$$

- TME defined wrt. to underlying differential equation
- TME is (much!) more ambitious than asymptotic optimality or mesh independent convergence of an iterative solver
- TME requires to quantify the constant
  - Hard to assess theoretically
  - But systematic numerical studies possible

Cost comparison for Stokes with stabilized P1-P1 vs. P2-P1

$$\lim_{\ell\to\infty} \frac{\mathfrak{W}(\mathbf{A}_\ell^{\mathbf{P}_2-\mathbf{P}_1})}{\mathfrak{W}(\mathbf{A}_{\ell+1}^{\mathbf{P}_1-\mathbf{P}_1})} = \frac{23}{12}, \qquad \lim_{\ell\to\infty} \frac{\mathfrak{W}(\mathbf{B}_\ell^{\mathbf{P}_2-\mathbf{P}_1})}{\mathfrak{W}(\mathbf{B}_{\ell+1}^{\mathbf{P}_1-\mathbf{P}_1})} = \frac{13}{24} \qquad \lim_{\ell\to\infty} \frac{\mathfrak{W}(\mathcal{A}_\ell^{\mathbf{P}_2-\mathbf{P}_1})}{\mathfrak{W}(\mathcal{A}_{\ell+1}^{\mathbf{P}_1-\mathbf{P}_1})} = \frac{9}{10}$$

- A WU for P2-P1 and for P1-P1 are roughly equivalent
- Velocity error after an FMG iteration with parameterization chosen to achieve minimal error



With this let’s come back to:

What is the fastest solver for Poisson’s equation?

References from the stone age of multigrid research

[ST] Stüben, K., & Trottenberg, U. Multigrid methods: Fundamental algorithms, model problem analysis and applications, in vol. 960 of Lecture Notes in Mathematics. Springer Verlag, 1982

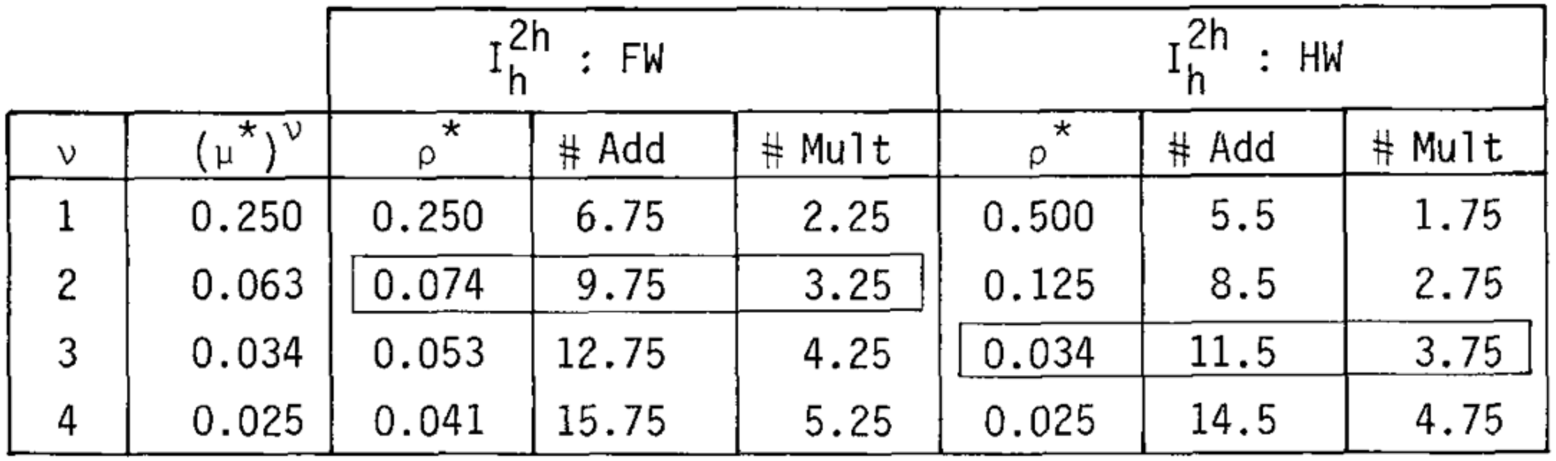
This is in the proceedings of the 1st European conf on multigrid methods that was held in Köln in 1981.
This volume also contains Brandt's original „Multigrid Guide".

[Hac] W. Hackbusch: Multi-grid methods and applications, 1985, Springer Berlin, ISBN 3-540-12761-5

[Gri] M. Griebel. Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen Transformations-Mehrgitter-Methode. Technical Report, SFB Bericht 342/4/90 A, Institut für Informatik, TU München, 1990

This is the dissertation of the author, submitted and defended in 1989

Work estimates from [ST] for 5-pt discretization of Poisson's eq 2-grid-method with red-black Gauss-Seidel smoothers



		I_h^{2h} : FW			I_h^{2h} : HW		
ν	$(\mu^*)^\nu$	ρ^*	# Add	# Mult	ρ^*	# Add	# Mult
1	0.250	0.250	6.75	2.25	0.500	5.5	1.75
2	0.063	0.074	9.75	3.25	0.125	8.5	2.75
3	0.034	0.053	12.75	4.25	0.034	11.5	3.75
4	0.025	0.041	15.75	5.25	0.025	14.5	4.75

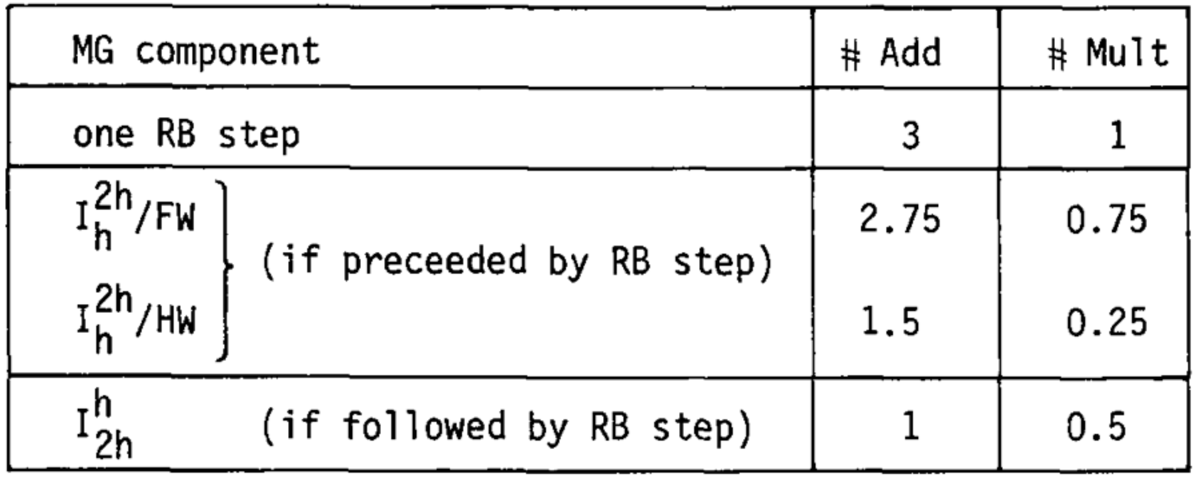
Table 8.1a: μ^*, ρ^* and computational work $W_h^{2h}/\mathcal{N}_h$ in case of smoothing by RB relaxation (for 5-point Laplace discretization)

Once the dust has been wiped off, this is still healthy, good, solid numerics

$\mu^*(\nu)$	smoothing factor
$\rho^*(\nu)$	Asymptotic 2-grid convergence factor

Half-weighting restriction (HW) $I_h^{2h} \hat{=} \frac{1}{8}\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix}_h^{2h}$

work optimization

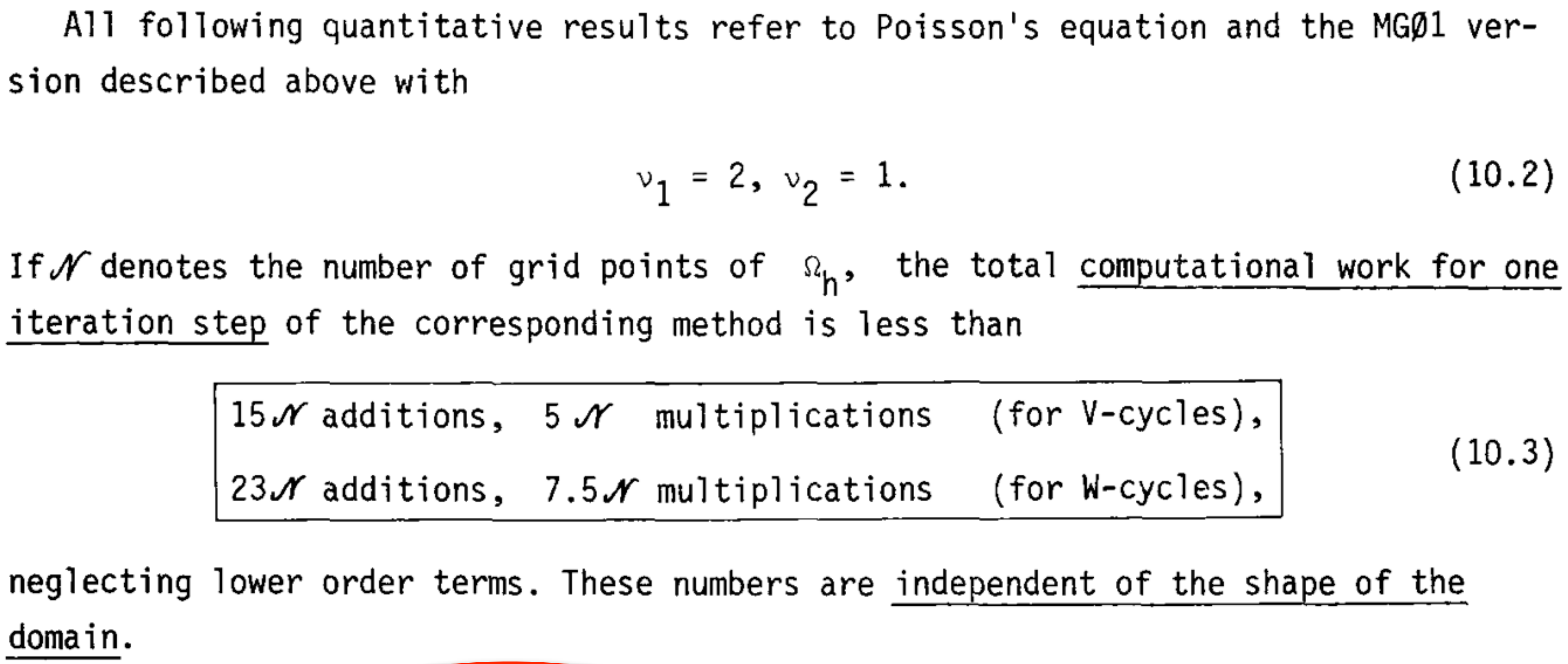


MG component	# Add	# Mult
one RB step	3	1
I_h^{2h}/FW (if preceeded by RB step)	2.75	0.75
I_h^{2h}/HW (if preceeded by RB step)	1.5	0.25
I_{2h}^h (if followed by RB step)	1	0.5

Table 8.1c: Operation count for the individual MG components used in Tables 8.1a and 8.1b (number of operations per point of Ω_h). The numbers given for I_h^{2h} and I_{2h}^h include the work needed for the computation of the defect and adding the correction, respectively.

- The RB-relaxation
  - overwrites all red points using only black points, so we need only to interpolate to black points
  - Makes the residual vanish on all black points: we can exploit this to use only black points to compute the restriction

And what is achieved by this



All following quantitative results refer to Poisson's equation and the MGØ1 version described above with

$$\nu_1 = 2, \nu_2 = 1. \tag{10.2}$$

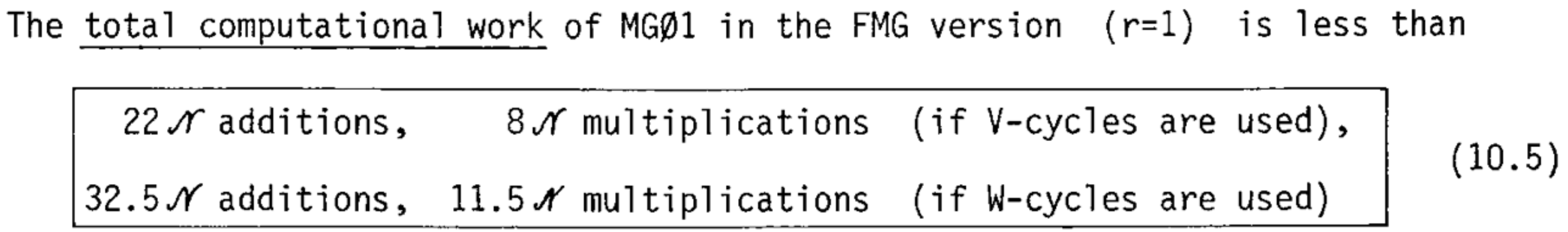
If $\mathcal{N}$ denotes the number of grid points of Ω_h, the total computational work for one iteration step of the corresponding method is less than

$$\boxed{\begin{array}{lll} 15\mathcal{N} \text{ additions}, & 5\,\mathcal{N} \text{ multiplications} & \text{(for V-cycles)}, \\ 23\mathcal{N} \text{ additions}, & 7.5\mathcal{N} \text{ multiplications} & \text{(for W-cycles)}, \end{array}} \tag{10.3}$$

neglecting lower order terms. These numbers are independent of the shape of the domain.

V(2,1)-cycle: 20 Flops/unknown
W(2,1)-cycle 30.5 Flops/unknown

… and if we use full multigrid (FMG)?



The total computational work of MGØ1 in the FMG version (r=1) is less than

$$\boxed{\begin{array}{ll} 22\mathcal{N} \text{ additions}, & 8\mathcal{N} \text{ multiplications (if V-cycles are used)}, \\ 32.5\mathcal{N} \text{ additions}, & 11.5\mathcal{N} \text{ multiplications (if W-cycles are used)} \end{array}} \qquad (10.5)$$

- Summarizing: We should be solving the 2D Poisson equation
  - to discretization error accuracy
  - **with 30 Flops per unknown!**
  - in the model case, FMG-V(2,1) cycles are enough to achieve asymptotic optimality

So, what is the cost of solving the discrete Poisson equation?

- What is the best constant published?
  - For Poisson 2D, second order:
    #Flops ~ **30 n** (Stüben, 1982)
- assume computer with 1 PetaFLOPS, $n=10^9$
  - expected time to solution: Poisson 2D
    $30*10^{-6}$ sec (microseconds!)
- standard computational practice in 2025 misses this by **several orders of magnitude**!
- What is the reason for this **gap** between theory and practice?
- Do we need a failure analysis?
- Related questions:
  - Cost of complex discretizations?
  - Has the deflation of computational cost lured us into mis-developments?

Conclusion and Outlook

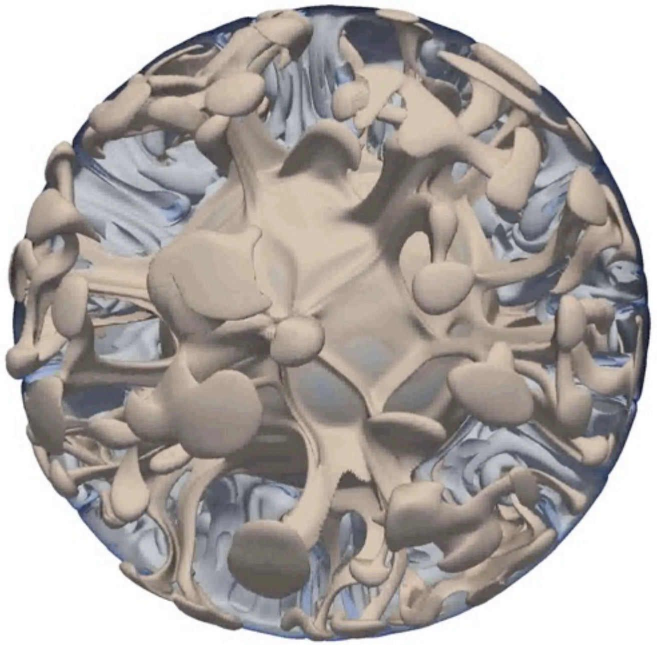
Multigrid scales!

HHG (since 2000):

- prototype implementation reaching 10^{13} DOF

HyTeG (since 2018):

- sustainable, flexible software architecture
- implements core concepts of HHG
- advanced discretizations



However, the efficiency seems still suboptimal when compared with the plain old-fashioned algorithms/software from 1982

Links:

- terraneo.fau.de
- https://i10git.cs.fau.de/hyteg/hyteg