deal.II Workshop @ Durham University

Welcome

Peter Munch[1], Anne Reinarz[2]

[1]Institute of Mathematics, Technical University of Berlin, Germany
[2]Department of Computer Science, Durham University, United Kingdom

April 3, 2025

Part 1:

Motivation

Introduction

- deal.II[1]: mathematical software for finite-element analysis, written in C++
- origin in Heidelberg 1998: Wolfgang Bangerth, Ralf Hartmann, Guido Kanschat
- 370 contributors + principal developer team with 13 active members
- more than 2,500 publications (on and with deal.II)
- freely available under Apache-2.0 with LLVM-exception or LGPL-2.1-or-later
- yearly releases; current release: 9.6
- features comprise: matrix-free implementations, parallelization (MPI, threading via TBB & Taskflow, SIMD, GPU support), discontinuous Galerkin methods, AMR via p4est, particles, wrappers for PETSc and Trilinos, ...
- many other libraries: e.g., MFEM, DUNE, FEniCS, libMesh, ...

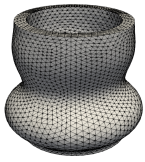
[1] successor of DEAL: **D**ifferential **E**quations **A**nalysis **L**ibrary

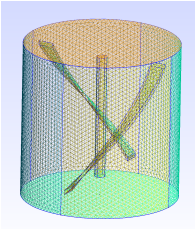
Why use deal.II?

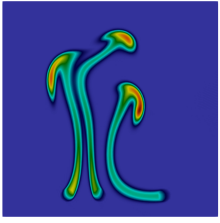
Why use ~~deal.II~~ open-source libraries?

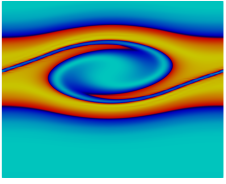
Why use deal.II? Because it's useful!

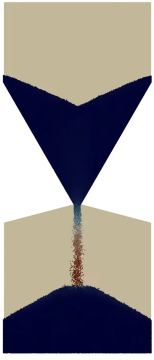
- solve complex PDEs needed by engineers

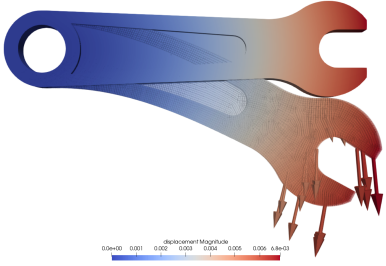


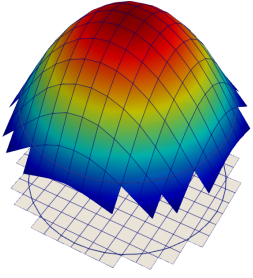






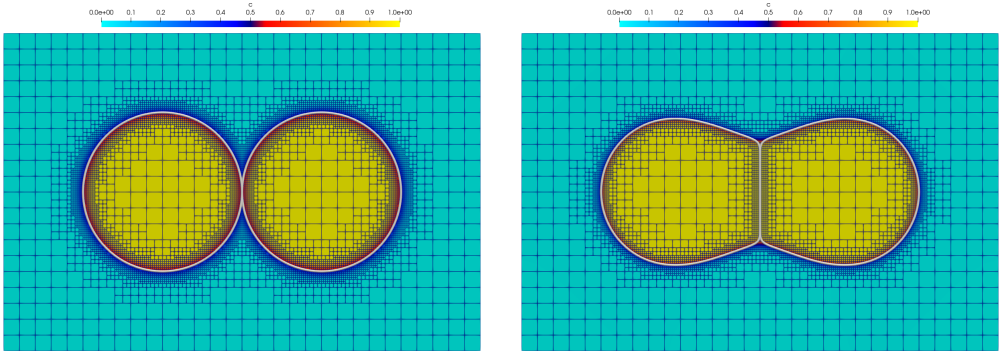






Why use deal.II? Because it's useful! (cont.)

Applications: phase-field solvers for sintering processes

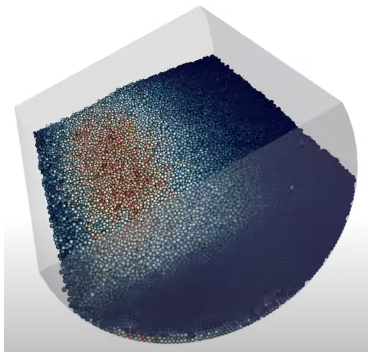


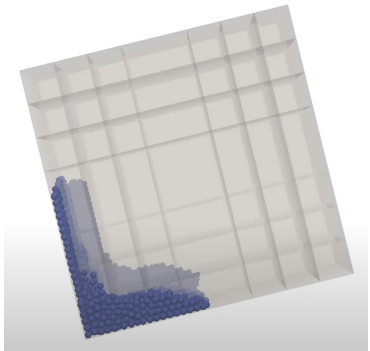
by Vladimir Ivannikov (Hereon)

with adaptive mesh refinement (AMR) to resolve high gradients in solution

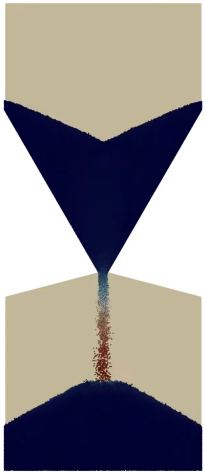
Why use deal.II? Because it's useful! (cont.)

Applications: DEM simulations for chemical process engineering[2]





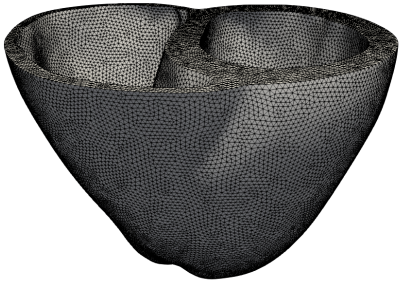
by Bruno Blais (Polytechnique Montreal)

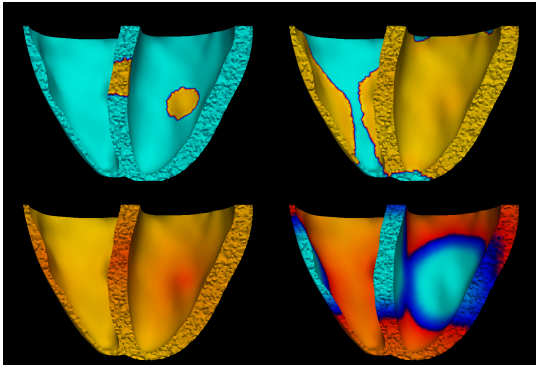


[2] https://lethe-cfd.github.io/lethe/

Why use deal.II? Because it's useful! (cont.)

Applications: Monodomain model for cardiac electrophysiology in a left+right ventricle geometry[3]



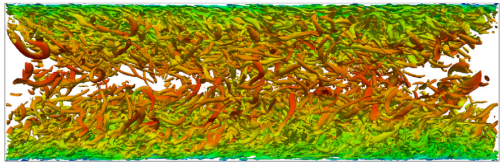


by Pasquale Africa (Polimi)

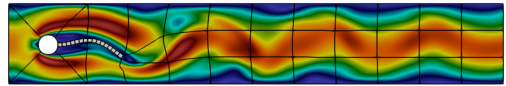
[3] https://lifex.gitlab.io/

Why use deal.II? Because it's useful! (cont.)

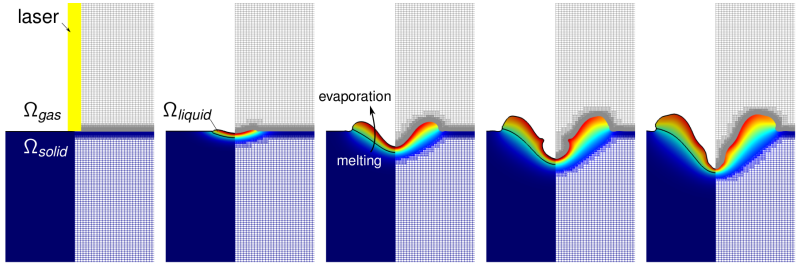
- compressible & incompressible NS



- fluid-structure interaction



- simulation of melt-pool processes



B. Krank et al. ['17], N. Fehn ['21], M. Schreter-Fleischhacker et al ['24, '25]

Why use deal.II? Because it's useful! (cont.)

- as means of research in HPC, numerical math and numerical linear algebra
- as teaching tool

Finite element methods in scientific computing: 1

1.4K views • 1 year ago

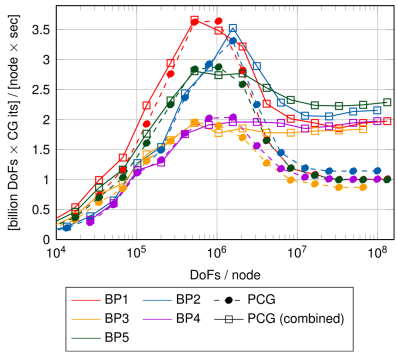
W Wolfgang Bangerth

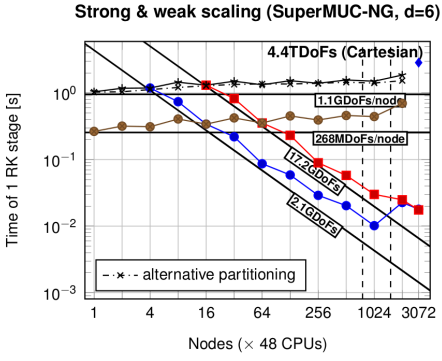
An introduction to the finite element method for the numerical solution of partial

... Wolfgang's lectures on YouTube

Why use deal.II? Because it's fast!

- state-of-the-art matrix-free algorithms → node-level performance

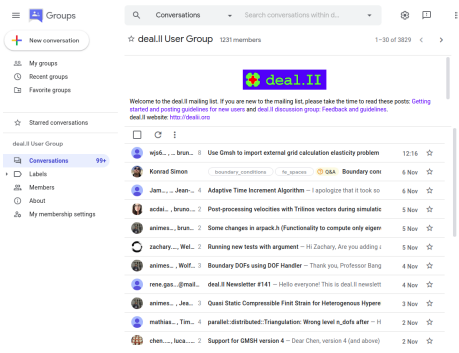




- excellent parallel scalability
  - largest simulation: $4.4 \cdot 10^{13}$ DoFs
  - full machine runs on SuperMUC-NG ($\approx$ 6k nodes, $\approx$ 150k cores)
- GPU support via Kokkos (WIP)

Why use deal.II? Because it's open and active!

- open development on GitHub
- active community with workshops



- daily PRs by developers and users (bug fixes, improvements, additions)
- regularly major new features, e.g., simplex (2021), CutFEM (2022), multigrid (2021/2023), particles (2020), 24 new tutorials over the last 5 years, ...

Why use deal.II? Because it's active! (cont.)

Winner of "SIAM/ACM Prize in Computational Science and Engineering 2025"

Why use deal.II? Because it's fun!

https://www.youtube.com/shorts/GI_jfsO0ZeM

Part 2:

Organization

Time table

	Thursday	**Friday**
9:00–10:00	welcome	lecture 2: solvers
10:00–11:00	introduction into deal.II (for beginners)	coding
11:00–12:00		
12:00–13:00	lunch	lunch
13:00–14:00	lecture 1: matrix-free	lecture 3: applications
14:00–15:00	coding (open end)	coding (open end)
15:00–16:00		
19:00–	workshop dinner	

Coding

- bring your code with you
- do a tutorial
- ask questions and help each other!

Introduction in deal.II

- based on a 3-day workshop at Helmholtz-Zentrum Hereon for material scientists
- summarized in 90-120 minutes
- extra material: Stokes equations
- hands-on session is moved to coding session

Lectures

- 3 lectures
- lecture 1: introduction into matrix-free computations
- lecture 2: adaptive mesh refinement & linear solvers
- lecture 3: applications
  - Lethe-CFD: matrix-free computation and multigrid for process engineering
  - additive manufacturing: simulation of melt-pool processes
  - solid-state sintering
  - cut Galerkin difference methods
  - computational plasma physics
  - space-time finite-element computations

... using deal.II in non-standard cases

Lectures (cont.)

not covered:

- coupled multiphysics problems
- GPU programming
- particles
- ...

Goals of the workshop

- learn about deal.II
- learn about advanced topics
- get answers to questions by a deal.II developer
- meet other deal.II users
- have fun and code

Part 3:

Introduction round

... other slide set

deal.II Workshop @ Durham University

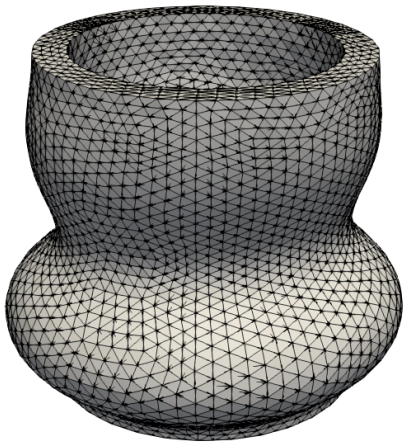
Introduction I: Overview, FEM basics, mesh handling

Peter Munch[1]

[1] Institute of Mathematics, Technical University of Berlin, Germany

April 3, 2025

Motivation



$$\mathrm{Div}(\underline{\boldsymbol{F}} \cdot \underline{\boldsymbol{S}}(\underline{\boldsymbol{E}})) + \rho_0 \hat{\underline{\boldsymbol{b}}} = 0$$

FEM

How can deal.II help?

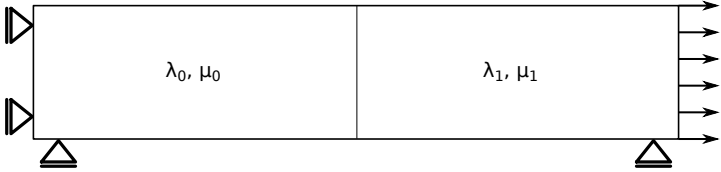
Organization: timetable

Topics:

- Part I: introduction into FEM, overview of deal.II, mesh handling
- Part II: Poisson problem
- Part III: solid mechanics
- Part IV: fluid mechanics (new)

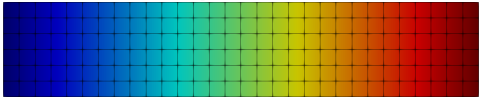
Organization: goal at the end of the third and fourth part

III: Tension rod:

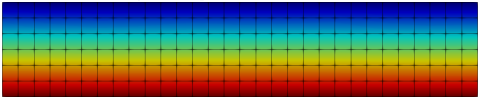


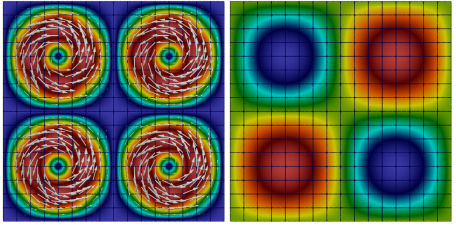
... with the result:

x displacement:



y displacement:





IV: Stokes problem (manufactured solution):

Part 1:

A short introduction into finite element methods

Model problem: Poisson problem

Strong form of the Poisson problem:

$$
\begin{aligned}
-\nabla \cdot \nabla u &= f && \text{in } \Omega = (0,1) \times (0,1), \\
u &= h && \text{on } \Gamma_D = \{x = 0, y \in (0,1)\}, \\
\nabla u(x,y) \cdot \underline{\boldsymbol{n}} &= g && \text{on } \Gamma_N = \{x = 1, y \in (0,1)\}, \\
\nabla u(x,y) \cdot \underline{\boldsymbol{n}} &= 0 && \text{else.}
\end{aligned}
$$

Steps:

a. definition of the function spaces
b. derivation of the weak form
c. spatial discretization + computation of the element stiffness matrix
d. assembly and set-up of the linear equation system

Definition of the function spaces

- $\mathcal{V}_{\hat{u}}(t, \Omega) = \{u(\cdot, t) \in \mathcal{H}^1(\Omega) : u = \hat{u} \text{ on } \Gamma_D\}$ for the solution
- $\mathcal{V}_0(\Omega) = \{v \in \mathcal{H}^1(\Omega) : v = 0 \text{ on } \Gamma_D\}$ for the test function

Derivation of the weak form

1. step: multiplication with the test function v and integration over Ω

$$-\int_{\Omega} v(\underline{\boldsymbol{x}})\nabla\cdot(\nabla u(\underline{\boldsymbol{x}},t))\mathrm{d}\underline{\boldsymbol{x}} = \int_{\Omega} v(\underline{\boldsymbol{x}})f\mathrm{d}\underline{\boldsymbol{x}}$$

2. step: integration by parts:

$$\int_{\Omega} v(\underline{\boldsymbol{x}})\nabla\cdot(\nabla u(\underline{\boldsymbol{x}}))\mathrm{d}\underline{\boldsymbol{x}} = \int_{\Gamma} v(\underline{\boldsymbol{x}})(\nabla u(\underline{\boldsymbol{x}}))\cdot\underline{\boldsymbol{n}}\mathrm{d}\Gamma - \int_{\Omega}\nabla v(\underline{\boldsymbol{x}})\cdot\nabla u(\underline{\boldsymbol{x}})\mathrm{d}\underline{\boldsymbol{x}}$$

3. step: exploitation of the boundary conditions (with: $\Gamma = \Gamma_D \cup \Gamma_N$):

$$\int_{\Gamma} v(\underline{\boldsymbol{x}})\nabla u(\underline{\boldsymbol{x}},t)\cdot\underline{\boldsymbol{n}}\mathrm{d}\Gamma = \int_{\Gamma_D} v(\underline{\boldsymbol{x}})\nabla u(\underline{\boldsymbol{x}},t)\cdot\underline{\boldsymbol{n}}\mathrm{d}\Gamma + \overbrace{\int_{\Gamma_N} v(\underline{\boldsymbol{x}})g\mathrm{d}\Gamma}^{0}$$

Derivation of the weak form (cont.)

Weak form

Find $u(\underline{\boldsymbol{x}}) \in \mathcal{V}_{\hat{u}}$ such that for all $v(\underline{\boldsymbol{x}}) \in \mathcal{V}_0(\Omega)$:

$$\int_\Omega \nabla v(\underline{\boldsymbol{x}}) \cdot \nabla u(\underline{\boldsymbol{x}}) \mathrm{d}\underline{\boldsymbol{x}} = \int_\Omega v(\underline{\boldsymbol{x}}) f \mathrm{d}\underline{\boldsymbol{x}} + \int_{\Gamma_N} v(\underline{\boldsymbol{x}}) g \mathrm{d}\Gamma$$

in compact notation:

$$(\nabla v(\underline{\boldsymbol{x}}),\ \nabla u(\underline{\boldsymbol{x}}))_\Omega = (v(\underline{\boldsymbol{x}}), f)_\Omega + (v(\underline{\boldsymbol{x}}), g)_{\Gamma_N}$$

Discretization

- decompose computational domain into cells $\Omega = \bigcup \Omega^{(e)}$
- use scalar Lagrange finite element $\mathcal{Q}_k$:

$$(\nabla v, \nabla u)_{\Omega_e} \approx \sum_q (\nabla v, \nabla u) \cdot |J| \times w \quad \rightarrow \quad \mathbf{K}_{ij}^{(e)} = \sum_q (\nabla N_{iq}, \nabla N_{jq}) \cdot |J_q| \times w_q$$

$$(v, f)_{\Omega_e} \quad \rightarrow \quad \mathbf{f}_i^{(e)} = \sum_q (N_{iq}, f) \cdot |J_q| \times w_q$$

$$(v, g)_{\Gamma_{e,N}} \quad \rightarrow \quad \mathbf{g}_i^{(e)} = \sum_q (N_{iq}, g) \cdot |J_q| \times w_q$$

... with N shape functions in real space, mapping & quadrature

- loop over all cells, assemble system matrix and right-hand-side vector, and solve system

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} + \boldsymbol{g}$$

Requirements to FEM library

The solution of a PDE with a FEM library (like deal.II):

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} + \boldsymbol{g} \quad \text{with} \quad \mathbf{K}_{ij}^{(e)} = \sum_q (\nabla N_{iq}, \nabla N_{jq}) \cdot |J_q| \times w_q$$

$$\mathbf{f}_i^{(e)} = \sum_q (N_{iq}, f) \cdot |J_q| \times w_q$$

$$\mathbf{g}_i^{(e)} = \sum_q (N_{iq}, g) \cdot |J_q| \times w_q$$

requires:

- mesh handling
- finite elements, quadrature rules, mapping rules
- assembly procedure
- linear solver

Part 2:

Short overview of deal.II

Introduction

- deal.II[1]: mathematical software for finite-element analysis, written in C++
- origin in Heidelberg 1998: Wolfgang Bangerth, Ralf Hartmann, Guido Kanschat
- 370 contributors + principal developer team with 13 active members
- more than 2,500 publications (on and with deal.II)
- freely available under Apache-2.0 with LLVM-exception or LGPL-2.1-or-later license
- yearly releases; current release: 9.6 (2024)
- features comprise: matrix-free implementations, parallelization (MPI, threading via TBB & Taskflow, SIMD, GPU support), discontinuous Galerkin methods, AMR via p4est, particles, wrappers for PETSc and Trilinos, ...

[1] successor of DEAL: **D**ifferential **E**quations **A**nalysis **L**ibrary

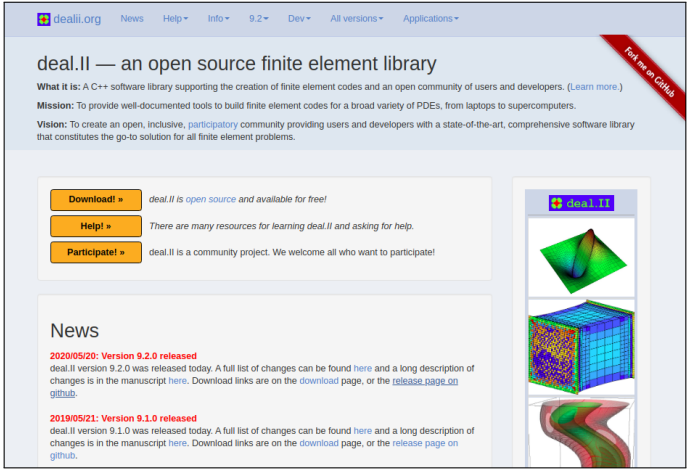
Introduction (cont.)

Publications describing the design of and recent development in deal.II:

D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. The deal.II finite element library: Design, features, and insights. *Computers and Mathematics with Applications*. 2020. DOI: https://doi.org/10.1016/j.camwa.2020.02.022

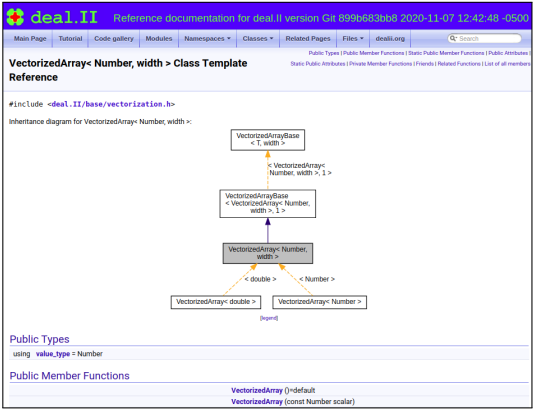
P. C. Africa, D. Arndt, W. Bangerth, B. Blais, T. C. Clevenger, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, S. Kinnewig, M. Kronbichler, M. Maier, P. Munch, M. Schreter-Fleischhacker, J. P. Thiele, B. Turcksin, D. Wells, and V. Yushutin. The deal.II Library, Version 9.6. *Journal of Numerical Mathematics.* 2024. DOI: https://doi.org/10.1515/jnma-2024-0137

Official webpage

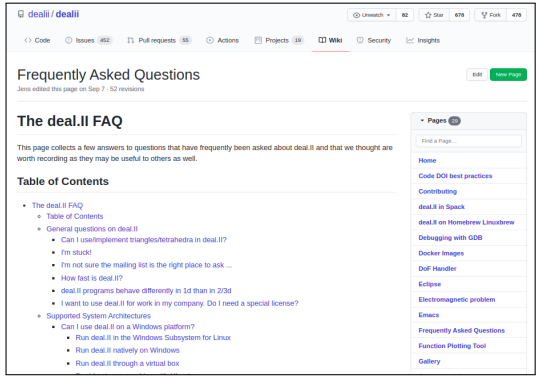


... www.dealii.org

Documentation



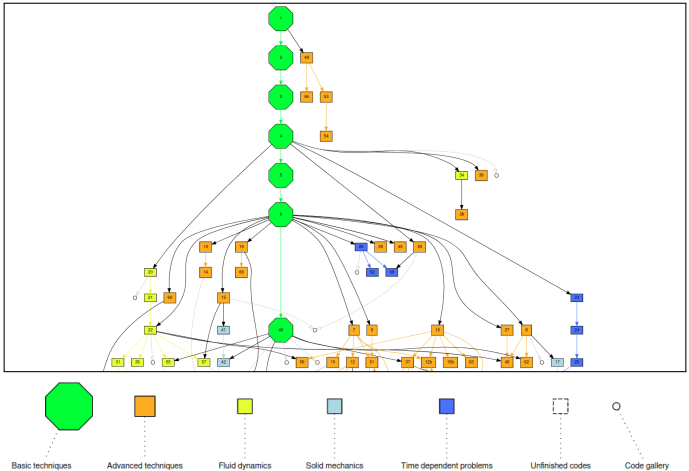
Extensive Doxygen documentation



GitHub Wiki

Documentation (cont.)

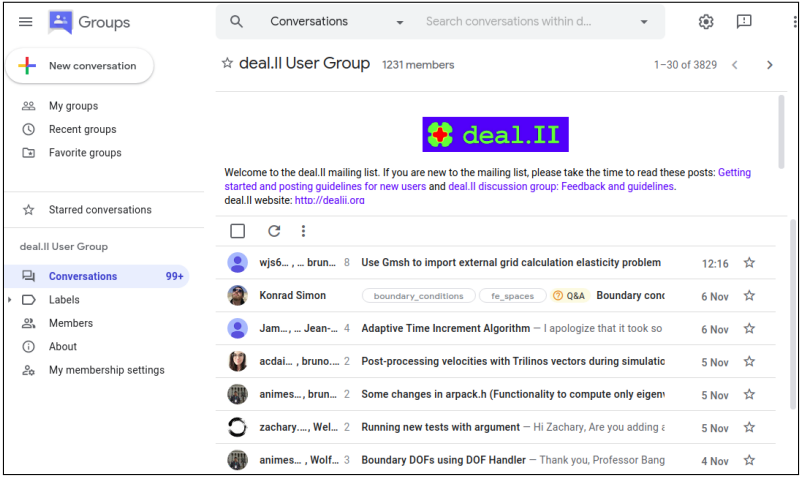
87 tutorials and code gallery:



... further 10 tutorials: work in progress

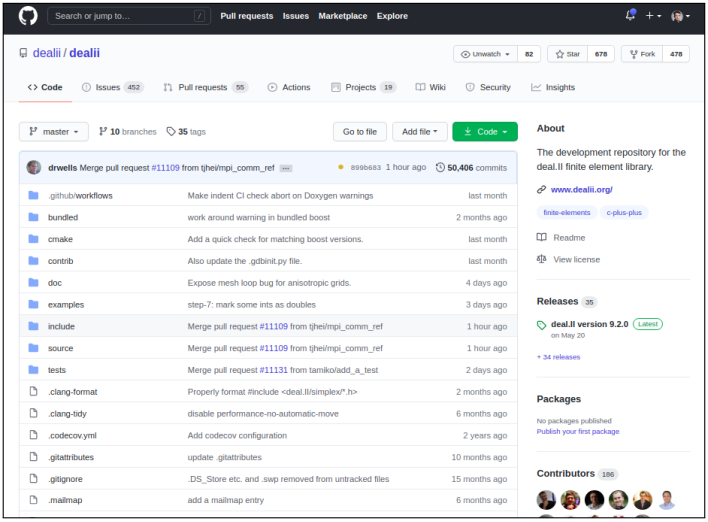
Forum

deal.II user group:



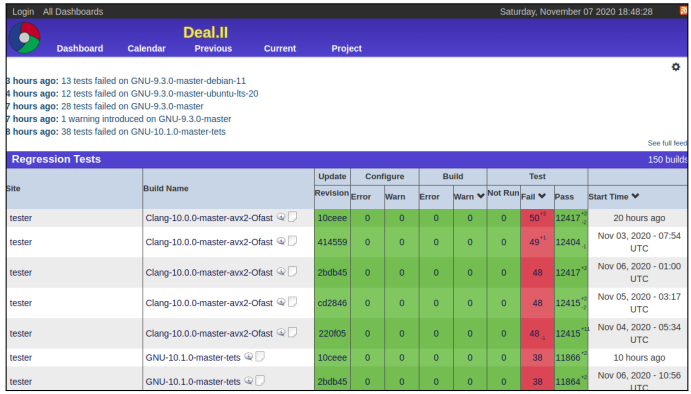
... Q&A by users and developers!

Development on GitHub



- issues
- pull requests
- GitHub actions $\rightarrow$ CI
- required: approval by ≥ 1 principal developer

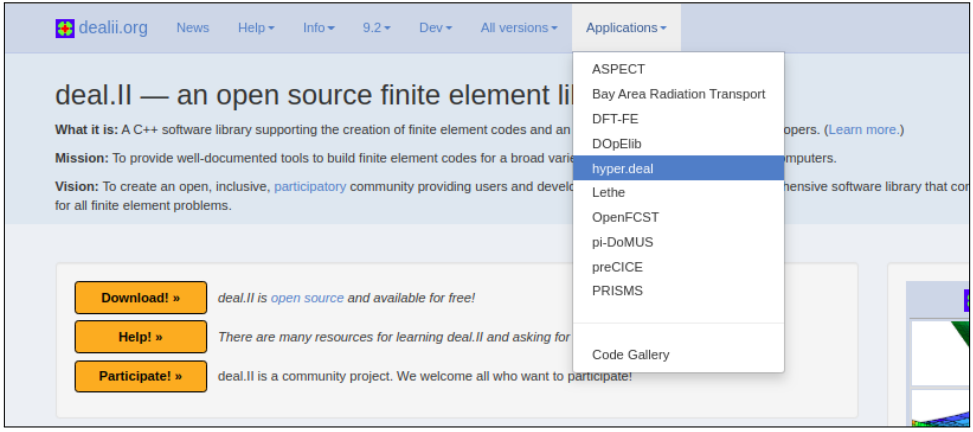
Continuous integration



... more than 17,000 tests run for different compilers/hardware/configurations

Applications

Some deal.II-based user codes/libraries are open source as well:



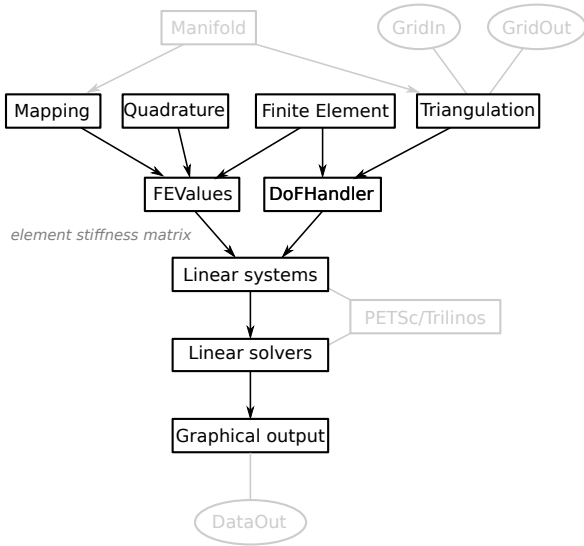
... motivation for further development

Main modules

needed from a FEM library:

- mesh handling
- finite elements
- quadrature rules
- mapping rules
- assembly procedure
- linear solver

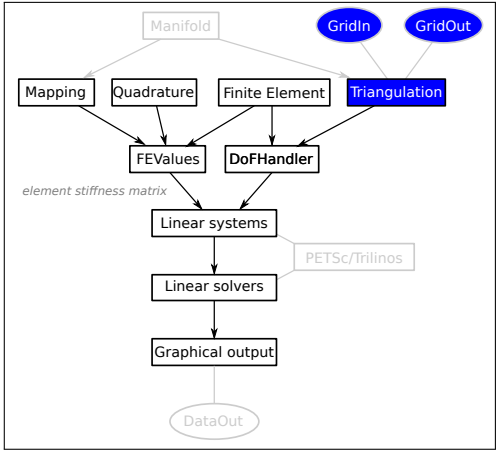
deal.II main modules $\rightarrow$



Part 3:

Mesh handling in deal.II

Mesh handling



Triangulation

- meshes are called `Triangulation`
- meshes can be created with functions in the `GridGenerator` namespace →
- meshes can be read from files
- meshes can be written to files

```
#include <deal.II/grid/tria.h>

int main()
{
  using namespace dealii;

  Triangulation<2> tria;
  GridGenerator::subdivided_hyper_cube(tria, 3);

  // output properties
  std::cout << tria.n_cells() << std::endl;
}
```

more information about `Triangulation` and `GridGenerator`:
https://www.dealii.org/developer/doxygen/deal.II/index.html

Cells

- meshes consist of cells
- it is possible to loop over all cells of a mesh
- cell properties (e.g., material ID) can be get/set →

  see: `CellAccessor`
- vertices, lines, and faces of cells can be accessed

  see: `TriaAccessor`

```
#include <deal.II/grid/tria.h>

int main()
{
  using namespace dealii;

  Triangulation<2> tria;
  // ...

  for(auto & cell : tria.active_cell_iterators())
  {
    std::cout << cell->matrial_id() << std::endl;
    std::cout << cell->face(0)->boundary_id()
                                    << std::endl;
  }
}
```

note “`operator->()`”: we are working with iterators (here: `TriaIterator`)

Task 1a: reading and writing meshes

- read the mesh “beam.msh” with `GridIn::read()`
- write the mesh to “task-1a-grid.vtk” with `GridOut::write_vtk()`
- take a look at the mesh in Paraview
- what properties are visualized?

... don’t forget to include the needed header files!

Optional:

- write the mesh to “task-1a-data.vtk” with `DataOut::write_vtk()`
- create a mesh in Gmsh
- try out other mesh formats

Task 1a: reading and writing meshes (cont.)

Getting started with Linux:

- open a terminal and get/compile the code:

```
git clone https://github.com/peterrum/dealii-durham-workshop-2025.git
cd dealii-durham-workshop-2025
cmake .
make task-1a-empty
```

- run the program:

```
./task-1a-empty
```

- for visualization use Paraview:

```
paraview
```

... in a new tab or new terminal

Task 1b: working with meshes

Loop over all cells and boundary faces and

- print the center of each cell
- assign material IDs to cells
- assign boundary IDs to faces

... hint: check results in Paraview!

Task 1c: working with distributed meshes

deal.II has different parallel meshes:

- `parallel::shared::Triangulation`
- `parallel::distributed::Triangulation` (built around p4est)
- `parallel::fullydistributed::Triangulation`

Check the documentation and extend the code of Task 1a.

deal.II Workshop @ Durham University

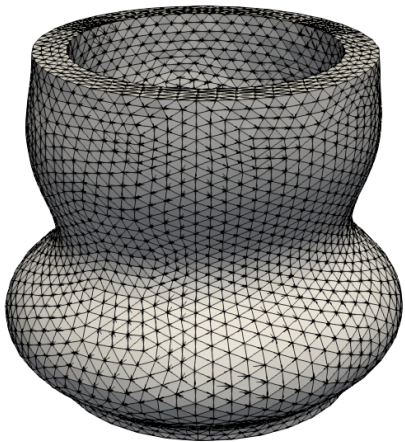
Introduction II: Poisson problem

Peter Munch[1]

[1] Institute of Mathematics, Technical University of Berlin, Germany

April 3, 2025

Motivation



$$\mathrm{Div}(\underline{\boldsymbol{F}} \cdot \underline{\boldsymbol{S}}(\underline{\boldsymbol{E}})) + \rho_0 \hat{\underline{\boldsymbol{b}}} = 0$$

FEM

How can deal.II help?

Organization: timetable

Topics:

- Part I: introduction into FEM, overview of deal.II, mesh handling
- Part II: Poisson problem
- Part III: solid mechanics, particles
- Part IV: fluid mechanics (new)

Part 1:

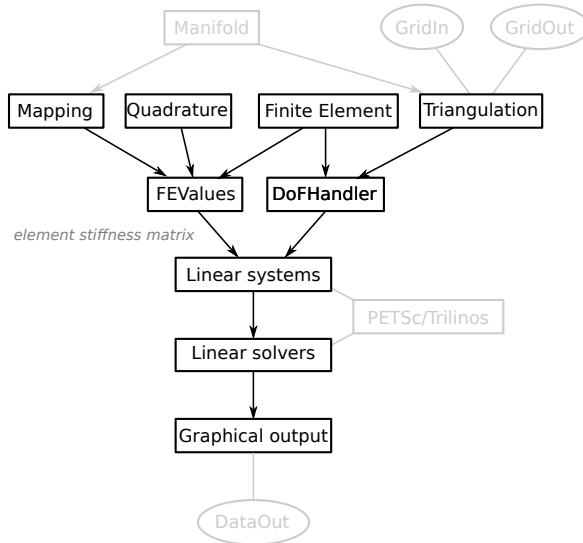
Wrap-up of part I

Main modules

needed from a FEM library:

- mesh handling
- finite elements
- quadrature rules
- mapping rules
- assembly procedure
- linear solver

deal.II main modules $\rightarrow$



Task 1a: solution

```
#include <deal.II/grid/tria.h>
#include <deal.II/grid/grid_in.h>
#include <deal.II/grid/grid_out.h>
#include <deal.II/numerics/data_out.h>
#include <fstream>

using namespace dealii;

const int dim = 2;

int
main()
{
  Triangulation<dim> tria;

  // read mesh with GridIn
  GridIn<dim> grid_in(tria);
  grid_in.read("beam.msh");

  // write mesh with GridOut in VTK format
  std::ofstream output_1("task-1a-grid.vtk");
  GridOut grid_out;
  grid_out.write_vtk(tria, output_1);
  }
}
```

Task 1b: solution

```
#include <deal.II/grid/tria.h>
#include <deal.II/grid/grid_in.h>
#include <deal.II/grid/grid_out.h>
#include <fstream>

using namespace dealii;
const int dim = 2;

int
main()
{
  Triangulation<dim> tria;

  // ...
  for (const auto &cell : tria.active_cell_iterators()) {
      std::cout << cell->center() << std::endl;

      cell->set_material_id(cell->center()[0] > 2.5);

      for (const auto &face : cell->face_iterators())
        if (face->at_boundary()) {
            if (face->center()[0] == 0.0) face->set_boundary_id(0);
            else if (face->center()[0] == 5.0) face->set_boundary_id(1);
            else if (face->center()[1] == 0.0) face->set_boundary_id(2);
            else if (face->center()[1] == 1.0) face->set_boundary_id(3);
          }
    }
  // ...
}
```

Part 2:

Solving the Poisson problem with deal.II

Poisson problem

Strong form of the Poisson problem:

$$
\begin{aligned}
-\nabla \cdot \nabla u &= f && \text{in } \Omega = (0,1) \times (0,1), \\
u &= h && \text{on } \Gamma_D = \{x = 0, y \in (0,1)\}, \\
\nabla u(x,y) \cdot \underline{\boldsymbol{n}} &= g && \text{on } \Gamma_N = \{x = 1, y \in (0,1)\}, \\
\nabla u(x,y) \cdot \underline{\boldsymbol{n}} &= 0 && \text{else.}
\end{aligned}
$$

Steps:

a. definition of the function spaces
b. derivation of the weak form
c. spatial discretization + computation of the element stiffness matrix
d. assembly and set-up of the linear equation system

Poisson problem (cont.)

Solve:

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} + \boldsymbol{g}$$

with:

$$\mathbf{K}_{ij}^{(e)} = \sum_q (\nabla N_{iq}, \nabla N_{jq}) \cdot |J_q| \times w_q,\ \mathbf{f}_i^{(e)} = \sum_q (N_{iq}, f) \cdot |J_q| \times w_q,\ \mathbf{g}_i^{(e)} = \sum_q (N_{iq}, g) \cdot |J_q| \times w_q$$

requires:

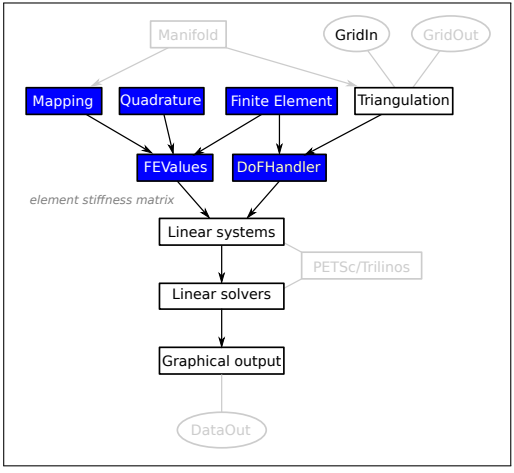
- mesh handling ✔
- finite elements, quadrature rules, mapping rules
- assembly procedure
- linear solver

Finite element, quadrature, mapping

Example configuration:

- finite element (`FE_Q`)
- quadrature (`QGauss`)
- mapping (`MappingQ1`)

```
#include <deal.II/fe/fe_q.h>
#include <deal.II/fe/mapping_q.h>
#include <deal.II/base/quadrature_lib.h>

int main()
{
  using namespace dealii;

  MappingQ1<2> mapping;
  FE_Q<2>      fe (degree);
  QGauss<2>    quad (n_q_points_1D);
}
```



DoFHandler

- responsible for degrees of freedom
- initialization: Triangulation + FiniteElement
- loop over cells as in the case of Triangulation

see: `DoFCellAccessor`

```
#include <deal.II/dofs/dof_handler.h>
#include <deal.II/fe/fe_q.h>
#include <deal.II/grid/tria.h>

int main()
{
  using namespace dealii;

  Triangulation<2> tria;
  // ...

  FE_Q<2> fe (/**/);

  DoFHandler<2> dofs(tria);
  dofs.distribute_dofs(fe);

  for(auto & cell : dofs.active_cell_iterators())
  {
    // ...
  }
}
```

Constraints

The `AffineConstraint` class can be used to prescribe relationships of DoFs:

$$x_i = \sum_j a_{ij} x_j + b_i$$

... constraints are considered during assembly!

Following utility function can be used:

- `VectorTools::interpolate_boundary_values()` → DBC
- `DoFTools::make_periodicity_constraints()` → PBC
- `DoFTools::make_hanging_node_constraints()` → AMR

Note: can be used for multi-point constraints (MPC)

FEValues

motivation:

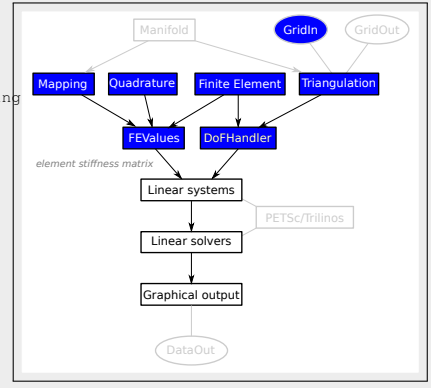
$$\mathbf{K}_{ij}^{(e)} = \sum_q (\nabla N_{iq}, \nabla N_{jq}) \cdot |J_q| \times w_q$$

- `FEValues` provides information at the cell quadrature points
- `UpdateFlags` determines what is needed
  - `update_values` $\rightarrow \underline{\underline{\boldsymbol{N}}}$
  - `update_gradients` $\rightarrow \nabla \underline{\underline{\boldsymbol{N}}}$
- for faces: `FEFaceValues`

```
#include <deal.II/dofs/dof_handler.h>
#include <deal.II/fe/fe_q.h>
#include <deal.II/grid/tria.h>

int main()
{
  using namespace dealii;

  Triangulation<2> tria;
  // ...

  MappingQ1<2> mapping;
  FE_Q<2> fe (/**/);
  QGauss quad (/**/);

  DoFHandler<2> dofs(tria);

  FEValues eval(mapping, fe, quad,
    update_values | update_quadrature_points);

  for(auto & cell : dofs.active_cell_iterators())
  {
    fe_values.reinit(cell);

    cout << eval.shape_value(0, 0) << endl;
    cout << eval.quadrature_point (0) << endl;
  }
}
```

Example

```
const unsigned int dim = 2, degree = 3, n_global_refinements = 3;

Triangulation<dim> tria; GridGenerator::hyper_cube(tria, 0, 1, true);
tria.refine_global(n_global_refinements);

FE_Q<dim> fe(degree); QGauss<dim> quad(degree + 1); MappingQ1<dim> mapping;

DoFHandler<dim> dof_handler(tria);
dof_handler.distribute_dofs(fe);

// deal with boundary conditions
AffineConstraints<double> constraints;
VectorTools::interpolate_boundary_values(
  mapping, dof_handler, 0, Functions::ZeroFunction<dim>(), constraints);
constraints.close();

// initialize vectors and system matrix
Vector<double>       x(dof_handler.n_dofs()), b(dof_handler.n_dofs());
SparseMatrix<double> A;
SparsityPattern      sparsity_pattern;

DynamicSparsityPattern dsp(dof_handler.n_dofs());
DoFTools::make_sparsity_pattern(dof_handler, dsp); sparsity_pattern.copy_from(dsp); A.reinit(sparsity_pattern);
```




```
// assemble right-hand side and system matrix
FullMatrix<double>                  cell_matrix;
Vector<double>                      cell_rhs;
std::vector<types::global_dof_index> local_dof_indices;

FEValues<dim> fe_values(mapping, fe, quad, update_default /*TODO*/);
```

$$\sum_q (\nabla N_{iq}, \nabla N_{jq}) \cdot |J_q| \times w_q, \quad \sum_q (N_{iq}, f) \cdot |J_q| \times w_q$$

Example (cont.)

```
// loop over all cells
for (const auto &cell : dof_handler.active_cell_iterators())
  {
    fe_values.reinit(cell);

    const unsigned int dofs_per_cell = cell->get_fe().dofs_per_cell;
    cell_matrix.reinit(dofs_per_cell, dofs_per_cell);
    cell_rhs.reinit(dofs_per_cell);

    // loop over cell dofs
    for (const auto q : fe_values.quadrature_point_indices())
      {
        for (const auto i : fe_values.dof_indices())
          for (const auto j : fe_values.dof_indices())
            cell_matrix(i, j) += 0.0; // TODO

        for (const unsigned int i : fe_values.dof_indices())
          cell_rhs(i) += 0.0; // TODO
      }

    local_dof_indices.resize(cell->get_fe().dofs_per_cell);
    cell->get_dof_indices(local_dof_indices);

    constraints.distribute_local_to_global(cell_matrix, cell_rhs, local_dof_indices, A, b);
  }
```

$$\sum_q (\nabla N_{iq}, \nabla N_{jq}) \cdot |J_q| \times w_q \to \mathbf{K}_{ij}^{(e)}$$

$$\sum_q (N_{iq}, f) \cdot |J_q| \times w_q \to \mathbf{f}_i^{(e)}$$

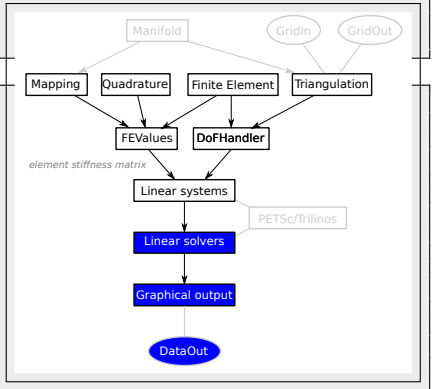
$$\mathop{\mathbf{A}}_{e} \mathbf{K}^{(e)}, \mathop{\mathbf{A}}_{e} \mathbf{f}^{(e)}$$

Example (cont.)

```
// solve linear equation system
ReductionControl         reduction_control(100, 1e-10, 1e-4);
SolverCG<Vector<double>> solver(reduction_control);
solver.solve(A, x, b, PreconditionIdentity());

printf("Solved in %d iterations.\n", reduction_control.last_step());

constraints.distribute(x);
```

$$\mathbf{K}\mathbf{x} = \mathbf{f} \rightarrow \mathbf{x} = \mathbf{K}^{-1}\mathbf{f}$$

```
// output results
DataOutBase::VtkFlags flags;
flags.write_higher_order_cells = true;

DataOut<dim> data_out;
data_out.set_flags(flags);
data_out.attach_dof_handler(dof_handler);
data_out.add_data_vector(dof_handler, x, "solution");
data_out.build_patches(mapping, degree + 1);

std::ofstream output("solution.vtu");
data_out.write_vtu(output);
```



Task 2

- task 2a) compute element stiffness matrix and right-hand side for $g = h = 0$
- task 2b) set $g = 1$ ... *hint: take a look at* `Functions` *namespace*
- task 2c) set $h = 1$ ... *hint: use* `FEFaceValues`

Optional:

- make g and h depend on $\underline{\boldsymbol{x}}$
- play with solver and preconditioner
- implement mass-matrix operator (v, u) and Helmholtz operator $(v, u) + (\nabla v, \nabla u)$
- make the code work for triangles
  (hints: `FE_SimplexP`, `QGaussSimplex`, `MappingFE(FE_SimplexP(1))`)

Task 2: hint

mass matrix operator: $\mathbf{K}_{ij}^{(e)} = \sum_q (N_{iq}, N_{jq}) \cdot |J_q| \times w_q$

```
fe_values.reinit(cell);

for(const auto i : fe_values.dof_indices ())
  for(const auto j : fe_values.dof_indices ())
    for(const auto q : fe_values.quadrature_point_indices ())
      matrix(i,j) += fe_values.shape_value(i, q) * fe_values.shape_value(j, q) * fe_values.JxW(q);
```

deal.II Workshop @ Durham University

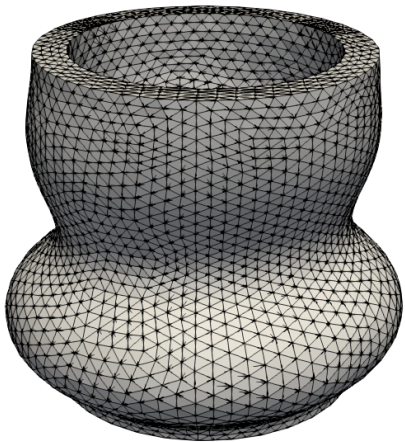
Introduction III: solid mechanics

Peter Munch[1]

[1] Institute of Mathematics, Technical University of Berlin, Germany

April 3, 2025

Motivation



$$\mathrm{Div}(\underline{\boldsymbol{F}} \cdot \underline{\boldsymbol{S}}(\underline{\boldsymbol{E}})) + \rho_0 \hat{\underline{\boldsymbol{b}}} = 0$$

FEM

How can deal.II help?

Organization: timetable

Topics:

- Part I: introduction into FEM, overview of deal.II, mesh handling
- Part II: Poisson problem (heat-conduction problem)
- Part III: solid mechanics
- Part IV: fluid mechanics (new)

Part 1:

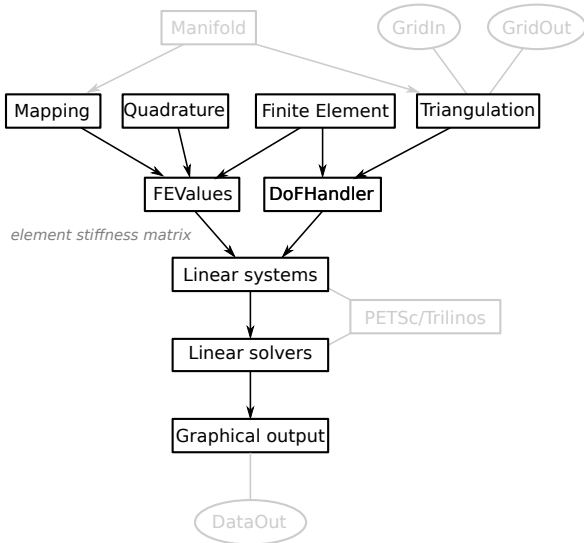
Wrap-up of part II

Main modules

needed from a FEM library:

- mesh handling
- finite elements
- quadrature rules
- mapping rules
- assembly procedure
- linear solver

deal.II main modules $\rightarrow$



Task 2: solution

Solve:

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} + \boldsymbol{g}$$

with:

$$\mathbf{K}_{ij}^{(e)} = \sum_q (\nabla N_{iq}, \nabla N_{jq}) \cdot |J_q| \times w_q,\ \mathbf{f}_i^{(e)} = \sum_q (N_{iq}, f) \cdot |J_q| \times w_q,\ \mathbf{g}_i^{(e)} = \sum_q (N_{iq}, g) \cdot |J_q| \times w_q$$

tasks:

- a) implement element stiffness matrix and right-hand-side vector
- b) modify DBC such that $h \neq 0$
- c) implement NBC such that $g \neq 0$

Task 2: solution (cont.)

Task 2a with $f(\underline{\boldsymbol{x}}) = ||\underline{\boldsymbol{x}}||$:

- initialization of `FEValues`:

```
FEValues<dim> fe_values(mapping, fe, quad,
                        update_values | update_gradients | update_JxW_values | update_quadrature_points);
```

- computation of element stiffness matrix and right-hand-side vector :

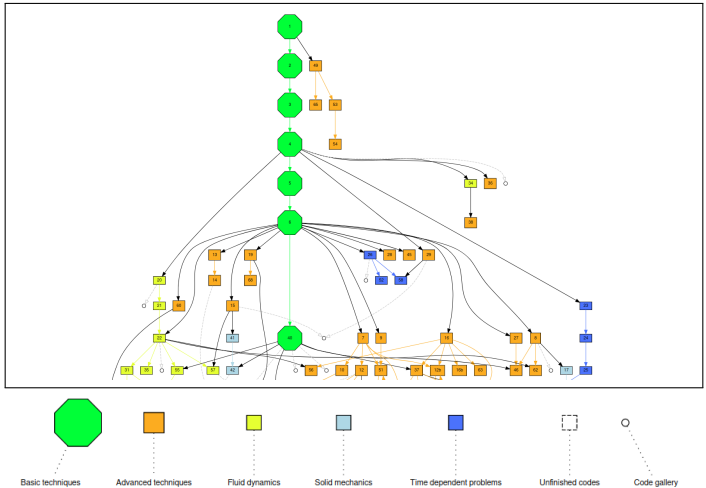
```
      // loop over cell dofs
      for (const auto q : fe_values.quadrature_point_indices())
        {
          for (const auto i : fe_values.dof_indices())
            for (const auto j : fe_values.dof_indices())
              cell_matrix(i, j) +=
                (fe_values.shape_grad(i, q) * // grad phi_i(x_q)
                 fe_values.shape_grad(j, q) * // grad phi_j(x_q)
                 fe_values.JxW(q));           // dx

          for (const unsigned int i : fe_values.dof_indices())
            cell_rhs(i) += (fe_values.shape_value(i, q) *           // phi_i(x_q)
                            fe_values.quadrature_point(q).norm() * // f(x_q)=||x_q||
                            fe_values.JxW(q));                     // dx
        }
```

Part 2:

Solid mechanics in deal.II

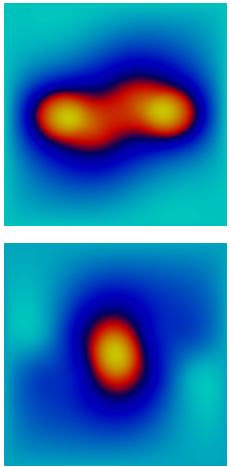
Further examples

Many tutorials and code-gallery programs give good starting points for solid mechanics:



Further examples (cont.)

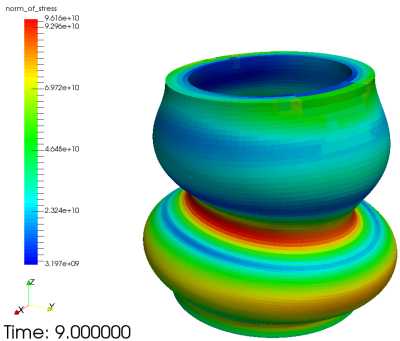
Tutorial: step-8



- linear elasticity
- dimension-independent
- Hooke's law
- parallelization in step-17 with PETSc

Further examples (cont.)

Tutorial: step-18

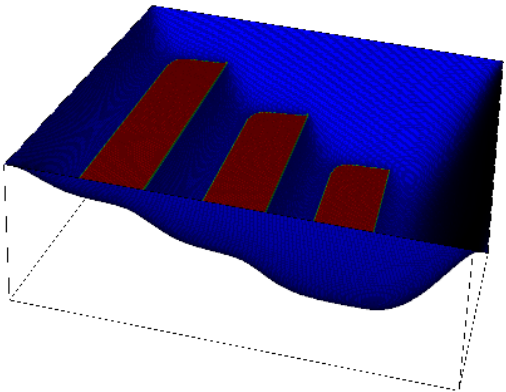


- quasistatic but time-dependent elasticity problem for large deformations with a Lagrangian mesh-movement approach
- buckling

Warning: The model considered here has little to do with reality!

Further examples (cont.)

Tutorial: step-41

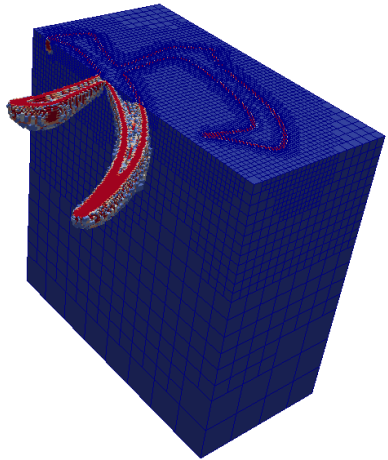


- elastic clamped membrane is deflected by gravity force, but is also constrained by an obstacle

... *aka obstacle problem*

Further examples (cont.)

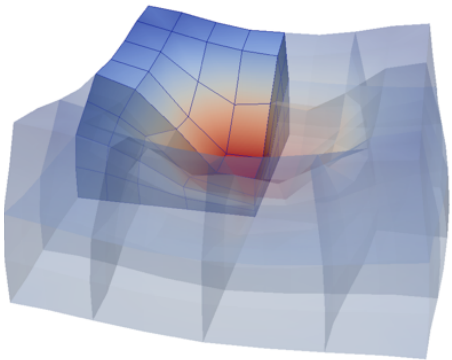
Tutorial: step-42



- deformation by rigid obstacle (contact problem)
- elastoplastic material behavior with isotropic hardening

Further examples (cont.)

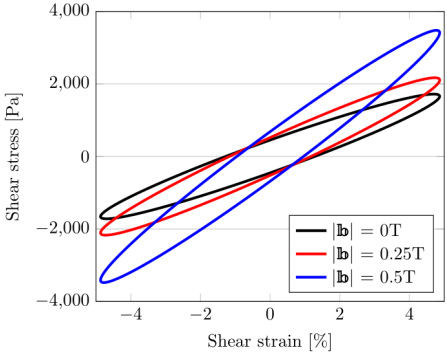
Tutorial: step-44



- three-field formulation
- fully nonlinear (geometrical and material) response of an isotropic continuum body
- quasi-incompressible neo-Hookean
- locking-free

Further examples (cont.)

Tutorial: step-71 (WIP)[1]

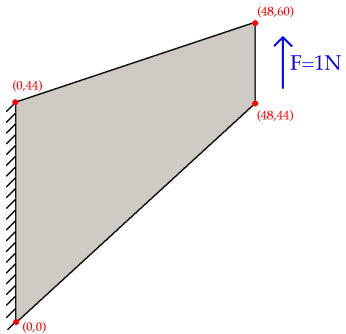


- automatic differentiation
- magneto-elastic constitutive law
- magneto-viscoelastic constitutive law

[1] https://github.com/dealii/dealii/pull/10392

Further examples (cont.)

Tutorial: step-73 (WIP)[2]



- automatic and symbolic differentiation
- finite-strain elasticity
- Cook's membrane

[2] https://github.com/dealii/dealii/pull/10394

Further examples (cont.)

Code gallery:

- elastoplastic torsion
- goal-oriented mesh adaptivity in elastoplasticity problems
- linear elastic active skeletal muscle model
- nonlinear poro-viscoelasticity
- quasistatic finite-strain compressible elasticity
- quasistatic finite-strain quasi-incompressible viscoelasticity
- linear elastoplasticity (WIP)[3] ▷ *history variables:* `CellDataStorage`

[3] https://github.com/dealii/code-gallery/pull/62

Part 3:

Theory

Strong form

- geometrically nonlinear elasticity (reference configuration):

$$\mathrm{Div}(\underline{\boldsymbol{F}} \cdot \underline{\boldsymbol{S}}(\underline{\boldsymbol{E}})) + \hat{\underline{\boldsymbol{b}}}_0 = 0 \qquad \text{with } \rho_0 = 1$$

  with deformation gradient $\underline{\boldsymbol{F}}$, Green-Lagrange strain $\underline{\boldsymbol{E}}$, 2nd Piola-Kirchhoff stress $\underline{\boldsymbol{S}}$

- geometrically linear elasticity:

$$\mathrm{Div}(\underline{\sigma}) + \hat{\underline{\boldsymbol{b}}} = 0$$

  with $\underline{\sigma} = \underline{\boldsymbol{C}} : \underline{\varepsilon}$ and $\underline{\varepsilon} = \frac{1}{2}\left(\nabla \underline{\boldsymbol{u}} + \nabla \underline{\boldsymbol{u}}^T\right)$

Discrete weak form

Discrete weak form (geometrically linear elasticity):

$$\underline{\underline{\boldsymbol{K}}}\,\underline{\boldsymbol{u}} = \underline{\boldsymbol{F}} \quad \text{with} \quad \underline{\underline{\boldsymbol{K}}}_{ij}^{(e)} = \int_{\Omega^{(e)}} \underline{\boldsymbol{B}}_i : \underline{\underline{\boldsymbol{C}}} : \underline{\boldsymbol{B}}_j \, \mathrm{d}\Omega \quad \text{and} \quad \underline{\boldsymbol{F}}_i^{(e)} = \int_{\Gamma^{(e)}} \underline{\boldsymbol{N}}_i \cdot \underline{\boldsymbol{t}} \, \mathrm{d}\Gamma + \int_{\Omega^{(e)}} \underline{\boldsymbol{N}}_i \cdot \underline{\boldsymbol{f}} \, \mathrm{d}\Omega$$

with $\underline{\boldsymbol{B}}_i = \frac{1}{2}\left(\nabla \underline{\boldsymbol{N}}_i + \nabla \underline{\boldsymbol{N}}_i^T\right)$.

Modifications compared to Poisson problem:

- $\underline{\boldsymbol{u}}$ is vectorial
- computation of $\underline{\underline{\boldsymbol{C}}}$ (Hooke's law)
- computation of $\underline{\boldsymbol{B}}$

more common notation:

$$\underline{\boldsymbol{B}} = \begin{bmatrix} \frac{\partial}{\partial x_1} & 0 \\ 0 & \frac{\partial}{\partial x_2} \\ \frac{\partial}{\partial x_2} & \frac{\partial}{\partial x_1} \end{bmatrix} \left[\begin{array}{cc|c|cc} \underline{\boldsymbol{N}}_0 & 0 & \cdots & \underline{\boldsymbol{N}}_k & 0 \\ 0 & \underline{\boldsymbol{N}}_0 & & 0 & \underline{\boldsymbol{N}}_k \end{array}\right]$$

... with index related to node

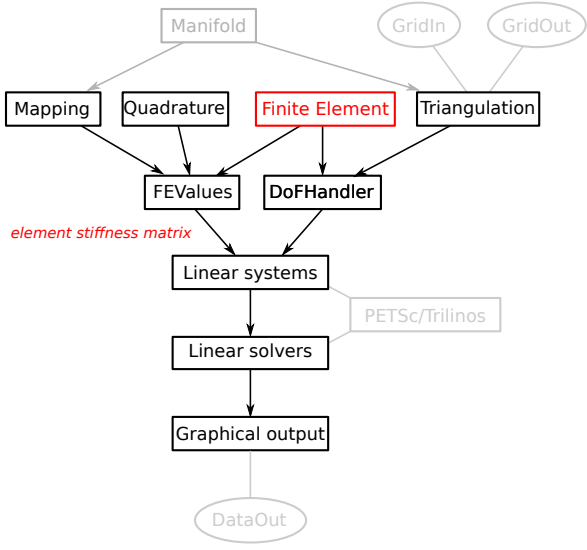
Vectorial finite element

Describe $\vec{u} \in \mathbb{R}^d$ as a **system of scalar Lagrange finite elements**:

$$\underbrace{[\mathcal{Q}_p^d, \ldots, \mathcal{Q}_p^d]}_{\times d}$$

in code:

```
FESystem<dim> fe(FE_Q<dim>(degree), dim);
```



Elastic stiffness tensor

Compute tensor $C_{ijkl} = \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) + \lambda\,\delta_{ij}\delta_{kl}$:

```
template <int dim>
SymmetricTensor<4, dim>
get_stress_strain_tensor(const double lambda, const double mu)
{
  SymmetricTensor<4, dim> tmp;
  for (unsigned int i = 0; i < dim; ++i)
    for (unsigned int j = 0; j < dim; ++j)
      for (unsigned int k = 0; k < dim; ++k)
        for (unsigned int l = 0; l < dim; ++l)
          tmp[i][j][k][l] = (((i == k) && (j == l) ? mu : 0.0) +
                             ((i == l) && (j == k) ? mu : 0.0) +
                             ((i == j) && (k == l) ? lambda : 0.0));
  return tmp;
}
```

... using SymmetricTensor<4, dim>

Strain tensor

Compute tensor $\underline{\boldsymbol{B}}_{iq}^{(e)} = \frac{1}{2}\left(\nabla \underline{\boldsymbol{N}}_i^{(e)}(\underline{\boldsymbol{x}}_q) + \nabla \underline{\boldsymbol{N}}_i^{(e)}(\underline{\boldsymbol{x}}_q)^T\right)$:

```
template <int dim>
inline SymmetricTensor<2, dim>
get_strain(const FEValues<dim> &fe_values,
           const unsigned int   shape_func,
           const unsigned int   q_point)
{
  SymmetricTensor<2, dim> tmp;

  for (unsigned int i = 0; i < dim; ++i)
    tmp[i][i] = fe_values.shape_grad_component(shape_func, q_point, i)[i];

  for (unsigned int i = 0; i < dim; ++i)
    for (unsigned int j = i + 1; j < dim; ++j)
      tmp[i][j] = (fe_values.shape_grad_component(shape_func, q_point, i)[j] +
                   fe_values.shape_grad_component(shape_func, q_point, j)[i]) /
                  2;

  return tmp;
}
```

... using SymmetricTensor<2, dim>

Strain tensor (cont.)

Compute tensor $\underline{\boldsymbol{B}}_{iq}^{(e)} = \frac{1}{2}\left(\nabla \underline{\boldsymbol{N}}_i^{(e)}(\underline{\boldsymbol{x}}_q) + \nabla \underline{\boldsymbol{N}}_i^{(e)}(\underline{\boldsymbol{x}}_q)^T\right)$:

```
template <int dim>
inline SymmetricTensor<2, dim>
get_strain(const FEValues<dim> &fe_values,
           const unsigned int   shape_func,
           const unsigned int   q_point)
{
  return fe_values[FEValuesExtractors::Vector()].symmetric_gradient(shape_func, q_point);
}
```

... using SymmetricTensor<2, dim>

On SymmetricTensor

The class SymmetricTensor allows working in tensor notation with the performance of the Voigt notation[4] due to reduced memory consumption and specialized functions (e.g., double contraction).

E.g., internal representation of SymmetricTensor<2, 3>[5]:

$$\begin{bmatrix} \varepsilon_{00} & \varepsilon_{01} & \varepsilon_{02} \\ \varepsilon_{10} & \varepsilon_{11} & \varepsilon_{12} \\ \varepsilon_{20} & \varepsilon_{21} & \varepsilon_{22} \end{bmatrix} \quad \leftrightarrow \quad \begin{bmatrix} \varepsilon_{00} & \varepsilon_{11} & \varepsilon_{22} & \varepsilon_{01} & \varepsilon_{02} & \varepsilon_{12} \end{bmatrix}$$

[4] https://www.dealii.org/developer/doxygen/deal.II/namespacePhysics_1_1Notation.html

[5] https://github.com/dealii/dealii/blob/8e208ae9dca8349c52b230514722463eb6fd51f4/include/deal.II/base/symmetric_tensor.h#L2419-L2425

Part 4:

Task

Example: beam

Example: beam (cont.)

```
const unsigned int dim = 2, degree = 1, n_refinements = 0;

// create mesh, select relevant FEM ingredients, and set up DoFHandler
Triangulation<dim> tria;
GridGenerator::subdivided_hyper_rectangle(
  tria, {10, 2}, Point<dim>(0, 0), Point<dim>(1, 0.2), true /*automatically set BIDs*/);
tria.refine_global(n_refinements);

FESystem<dim>        fe(FE_Q<dim>(degree), dim);
QGauss<dim>          quad(degree + 1);
QGauss<dim - 1>      face_quad(degree + 1);
MappingQGeneric<dim> mapping(1);

DoFHandler<dim> dof_handler(tria);
dof_handler.distribute_dofs(fe);

// Create constraint matrix
AffineConstraints<double> constraints;
VectorTools::interpolate_boundary_values(dof_handler,
                                         0 /*left face*/,
                                         Functions::ConstantFunction<dim>(std::vector<double>{0.0, 0.0}),
                                         constraints);
constraints.close();

// compute traction
Tensor<1, dim> traction; traction[0] = +0e9; traction[1] = -1e9;

// compute stress strain tensor
const auto stress_strain_tensor = get_stress_strain_tensor<dim>(9.695e10, 7.617e10);
```

Example: beam (cont.)

```
// initialize vectors and system matrix
Vector<double>       x(dof_handler.n_dofs()), b(dof_handler.n_dofs());
SparseMatrix<double> A;
SparsityPattern      sparsity_pattern;

DynamicSparsityPattern dsp(dof_handler.n_dofs());
DoFTools::make_sparsity_pattern(dof_handler, dsp);
sparsity_pattern.copy_from(dsp);
A.reinit(sparsity_pattern);

// assemble right-hand side and system matrix
FEValues<dim> fe_values(mapping, fe, quad, update_gradients | update_JxW_values);

FEFaceValues<dim> fe_face_values(mapping, fe, face_quad, update_values | update_JxW_values);

FullMatrix<double>                   cell_matrix;
Vector<double>                       cell_rhs;
std::vector<types::global_dof_index> local_dof_indices;
```

Example: beam (cont.)

```
// loop over all cells
for (const auto &cell : dof_handler.active_cell_iterators())
  {
    if (cell->is_locally_owned() == false)
      continue;

    fe_values.reinit(cell);

    const unsigned int dofs_per_cell = cell->get_fe().dofs_per_cell;
    cell_matrix.reinit(dofs_per_cell, dofs_per_cell);
    cell_rhs.reinit(dofs_per_cell);

    // loop over cell dofs
    for (unsigned int i = 0; i < dofs_per_cell; ++i)
      for (unsigned int j = 0; j < dofs_per_cell; ++j)
        for (unsigned int q = 0; q < fe_values.n_quadrature_points; ++q)
          {
            const auto eps_phi_i = get_strain(fe_values, i, q);
            const auto eps_phi_j = get_strain(fe_values, j, q);

            cell_matrix(i, j) += (eps_phi_i * stress_strain_tensor * eps_phi_j ) * fe_values.JxW(q);
          }
```

$$\int_{\Omega^{(e)}} \underline{\boldsymbol{B}}_i^T : \underline{\underline{\boldsymbol{C}}} : \underline{\boldsymbol{B}}_j \, \mathrm{d}\Omega \underline{\boldsymbol{u}}$$

Example: beam (cont.)

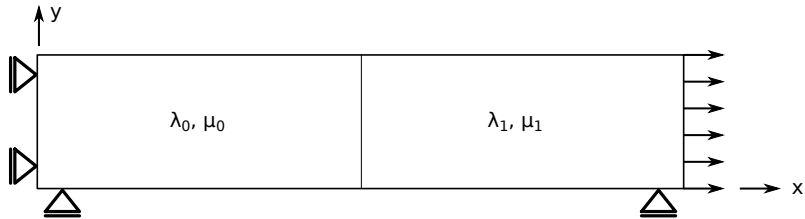
```
    // loop over all cell faces and their dofs
    for (const auto &face : cell->face_iterators())
      {
        // we only want to apply NBC on the right face
        if (!face->at_boundary() || face->boundary_id() != 1)
          continue;

        fe_face_values.reinit(cell, face);

        for (unsigned int q = 0; q < fe_face_values.n_quadrature_points; ++q)
          for (unsigned int i = 0; i < dofs_per_cell; ++i)
            cell_rhs(i) += fe_face_values.shape_value(i, q) *
                           traction[fe.system_to_component_index(i).first] *
                           fe_face_values.JxW(q);
      }

    local_dof_indices.resize(cell->get_fe().dofs_per_cell);
    cell->get_dof_indices(local_dof_indices);

    constraints.distribute_local_to_global(
      cell_matrix, cell_rhs, local_dof_indices, A, b);
  }
```

$$\int_{\Gamma^{(e)}} \underline{\boldsymbol{N}}_i^T \cdot \underline{\boldsymbol{t}} \, \mathrm{d}\Gamma$$

Example: beam (cont.)

```
// solve linear equation system
ReductionControl         reduction_control;
SolverCG<Vector<double>> solver(reduction_control);
solver.solve(A, x, b, PreconditionIdentity());

printf("Solved in %d iterations.\n", reduction_control.last_step());

constraints.distribute(x);

// output results
DataOut<dim> data_out;
data_out.attach_dof_handler(dof_handler);
x.update_ghost_values();
data_out.add_data_vector(dof_handler, x, "solution",
  std::vector<DataComponentInterpretation::DataComponentInterpretation>(
    dim, DataComponentInterpretation::component_is_part_of_vector));
data_out.build_patches(mapping, degree + 1);

std::ofstream output("solution.vtu");
data_out.write_vtu(output);
```

Tasks



Extend “task-3a-empty.cc” (beam) to simulate a torsion rod:

- symmetric boundary condition on the left and bottom face
- force in x-direction on the right face
- vary material parameters of the rod (left vs. right) - see also Task 1b

deal.II Workshop @ Durham University

Introduction IV: fluid mechanics

Peter Munch[1], Laura Prieto Saavedra[2]

[1]Institute of Mathematics, Technical University of Berlin, Germany
[2]Polytechnique Montréal, Université de Montréal, Canada

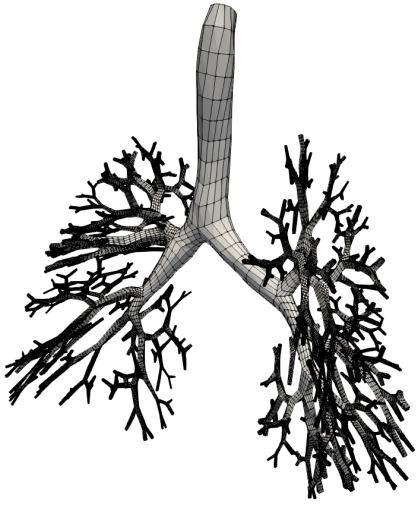
April 3, 2025

Organization: timetable

Topics:

- Part I: introduction into FEM, overview of deal.II, mesh handling
- Part II: Poisson problem (heat-conduction problem)
- Part III: solid mechanics
- Part IV: fluid mechanics (new)

Computational fluid dynamics



Solve Navier-Stokes equation:

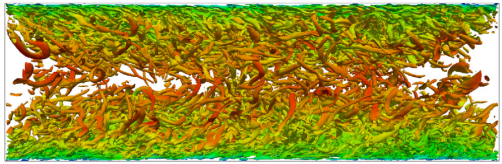
$$\frac{\partial \vec{u}}{\partial t} + \nabla \cdot (\vec{u} \otimes \vec{u}) - \nu \nabla^2 \vec{u} + \nabla p = \vec{f}$$

$$\nabla \cdot \vec{u} = 0$$

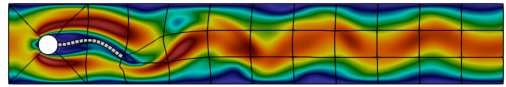
Kronbichler et al. ['21]

Computational fluid dynamics (cont.)

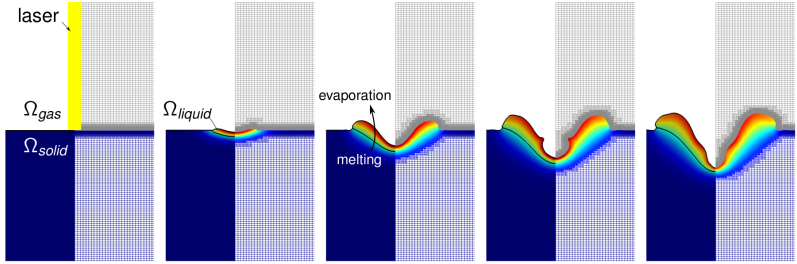
- compressible & incompressible NS



- fluid-structure interaction



- simulation of melt-pool processes



B. Krank et al. [’17], N. Fehn [’21], M. Schreter-Fleischhacker et al [’24, ’25]

Computational fluid dynamics (cont.)

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t}+(u\cdot\nabla)\mathbf{u}\right)=-\nabla p+\nabla\cdot\tau_{\mu}+\rho\mathbf{g}+\mathbf{f}$$

$$\nabla\cdot\mathbf{u}=0$$

(Runge–Kutta method)

Dual splitting (Karniadakis et al. ['91])

- convection step
- pressure Poisson step
- projection step
- viscous step

Coupled approach

Linearization $\rightarrow$ block system:

$$K=\begin{bmatrix} A & B \\ C & 0 \end{bmatrix},\ P=\begin{bmatrix} A & B \\ 0 & S \end{bmatrix}$$

with $S=-CA^{-1}B$

Challenges:

- systems to solve: Poisson operator, mass matrix, Helmholtz operator

Part 1:

Proxy problem: Stokes problem

Strong form

We solve the Stokes problem on a domain $\Omega \in [-1,+1]^d$ and with given Dirichlet boundary conditions:

$$
\begin{aligned}
-\nabla \cdot 2\varepsilon(\boldsymbol{u}) + \nabla p &= \boldsymbol{f} && x \in \Omega, \\
\nabla \cdot \boldsymbol{u} &= 0 && x \in \Omega, \\
\boldsymbol{u} = \boldsymbol{g}_{\mathrm{D}}(\boldsymbol{x}),\ \nabla p &= \boldsymbol{0} && x \in \Gamma,
\end{aligned}
$$

where $\boldsymbol{u}$ is velocity, p is pressure, $\boldsymbol{f}$ is a source term, and deformation-rate tensor $\varepsilon(\boldsymbol{u}) := \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^\top)$.

Weak form

Corresponding weak form:

$$F_{\text{Stokes}}(\boldsymbol{u}, p) = (\varepsilon(\boldsymbol{v}), 2\varepsilon(\boldsymbol{u})) - (\nabla \cdot \boldsymbol{v}, p) + (q, \nabla \cdot \boldsymbol{u}) - (\boldsymbol{v}, \boldsymbol{f}) = 0.$$

Challenges: inf-sub instability, resulting system of linear equation has a saddle-point form.

Approach 1: use stable mixed finite elements for $\boldsymbol{u}$ and p space (e.g., $\mathcal{Q}_k^d \mathcal{Q}_{k-1}$).

Weak form incl. stabilization

Approach 2: add stabilization of the form

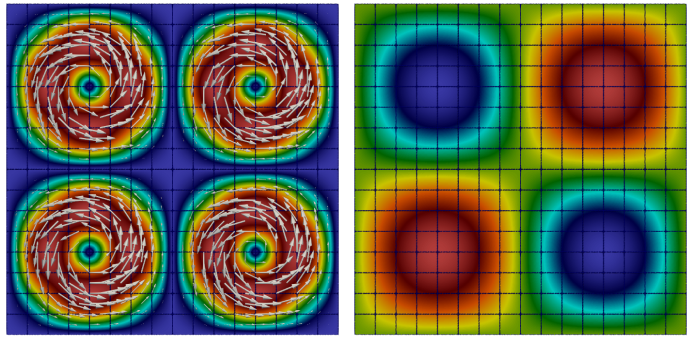
$$F^{\text{stab}}_{\text{Stokes}}(\boldsymbol{v}, p) = F_{\text{Stokes}}(\boldsymbol{v}, p) + \sum_c \tau_e (\nabla q,\ \nabla p - \boldsymbol{f})_{\Omega^c} = 0, \qquad \tau_e = h_e,$$

which allows using equal-order $\mathcal{Q}_k^d \mathcal{Q}_k$ polynomials.

Part 2:

Task

Problem statement



Method of manufactured solution:

$$\begin{pmatrix} u_0 \\ u_1 \\ p \end{pmatrix} = \begin{pmatrix} \sin(\pi x) \cdot \sin(\pi x) \cdot \cos(\pi y) \cdot \sin(\pi y) \\ -\cos(\pi x) \cdot \sin(\pi x) \cdot \sin(\pi y) \cdot \sin(\pi y) \\ \sin(\pi x) \cdot \sin(\pi y) \end{pmatrix}.$$

Tasks

Extend “task-4-empty.cc” to solve the Stokes problem:

- computation of element stiffness matrix
- computation of element right-hand-side vector

Hint: use ‘FEValuesExtractors::Vector‘ AND ‘FEValuesExtractors::Scalar‘.

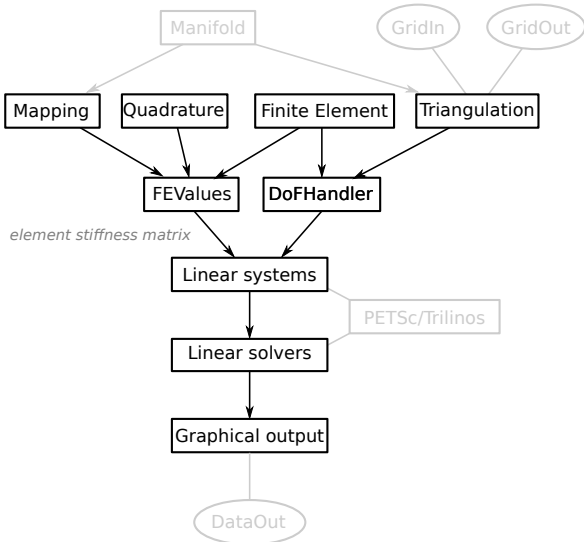
Note: source function is already implemented:

$$
\begin{aligned}
f_0 = &+ 2\pi^2(\sin(\pi x)\sin(\pi x) - \cos(\pi x)\cos(\pi x))\sin(\pi y)\cos(\pi y) \\
&+ 4\pi^2 \sin(\pi x)\sin(\pi x)\sin(\pi y)\cos(\pi y) + \pi \sin(\pi y)\cos(\pi x), \\
f_1 = &- 2pi^2(\sin(\pi y)\sin(\pi y) - \cos(\pi y)\cos(\pi y))\sin(\pi x)\cos(\pi x) \\
&- 4\pi^2 \sin(\pi x)\sin(\pi y)\sin(\pi y)\cos(\pi x) + \pi \sin(\pi x)\cos(\pi y).
\end{aligned}
$$

Part 3:

Outlook

Outlook



ASPECT

advanced topics:

- physical models
- performance
- (block) preconditioning
- discontinuous Galerkin (DG)
- Navier–Stokes equations, e.g.,

deal.II Workshop @ Durham University

Lecture 1: matrix-free computations

Peter Munch[1]

in collaboration with Martin Kronbichler, Katharina Kormann, . . .

[1] Institute of Mathematics, Technical University of Berlin, Germany

April 3, 2025

Part 1:

Motivation for matrix-free computations

Motivation – for high order

Solution of partial differential equations (e.g., Poisson problem):

$$-\Delta u = f \qquad \text{on } \Omega \text{ with appropriate BCs on } \Gamma$$

Discretization (e.g., FDM, FEM) leads to system of linear equations, which needs to be solved:

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$$

Observation 1: $\mathrm{NNZ}(\boldsymbol{A}) \gg \mathrm{size}(\boldsymbol{u}) + \mathrm{size}(\boldsymbol{f})$ for high-order polynomial degrees k.

Question 1: Do we need $\boldsymbol{A}$?

Motivation – for high order (cont.)

Observation 2: To solve problems with # DoFs $\gg 10k$, we normally use iterative solvers, e.g., CG, GMRES, FGMRES, which do not need $\boldsymbol{A}$ but only the action of an operator $\mathcal{A}$ on a vector $\boldsymbol{u}$ (aka `vmult`):

$$\mathcal{A}(\boldsymbol{u}) = \boldsymbol{A}\boldsymbol{u}$$

and an (optional) preconditioner $\mathcal{P}^{-1}(\boldsymbol{u})$:

$$\mathcal{P}^{-1}(\boldsymbol{u}) \approx \boldsymbol{A}^{-1}\boldsymbol{u}.$$

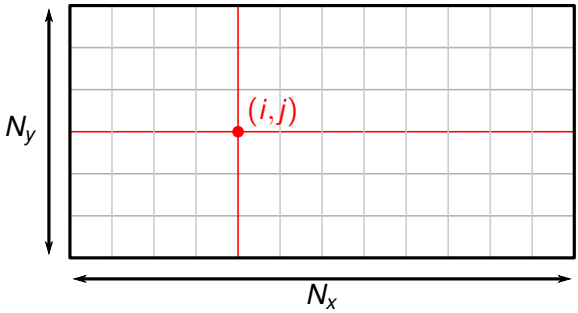
Question 2: How does efficient and flexible $\mathcal{A}(\boldsymbol{u})$, which returns exactly $\boldsymbol{A}\boldsymbol{u}$, look like?
Question 3: Can you construct $\mathcal{P}^{-1}(\boldsymbol{u})$ without $\boldsymbol{A}$?

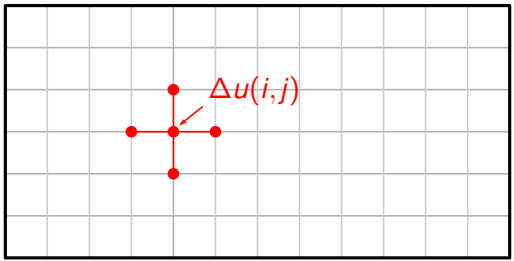
Part 2:

Matrix-free computations for finite difference methods

Matrix-free FDM

Discretization of the domain with structured mesh with $N_x \times N_y$ points:





Approximation of Δu at point (i,j), e.g., with:

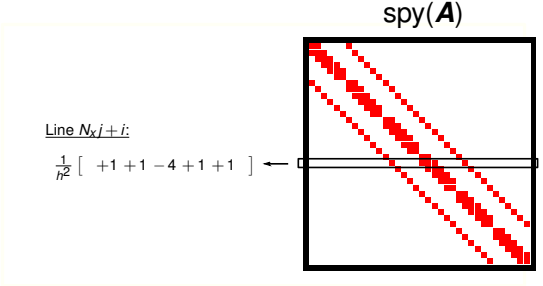
$$\Delta u_{ij} \approx \frac{u_{i,j-1} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1}}{h^2}$$

Matrix-free FDM (cont.)

Approximation of Δu at point (i, j), e.g., with:

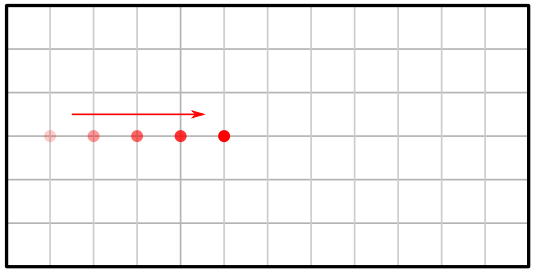
$$\Delta u_{ij} \approx \frac{u_{i,j-1} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1}}{h^2} \quad \leftrightarrow \quad \frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ (aka. stencil)}$$

Matrix-based approach $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{u}$:



... assemble sparse matrix $\boldsymbol{A} \in \mathbb{R}^{(N_x N_y)\times(N_x N_y)}$

Matrix-free approach $\boldsymbol{v} = \mathcal{A}(\boldsymbol{u})$:



... loop over points and apply stencil on the fly

Part 3:

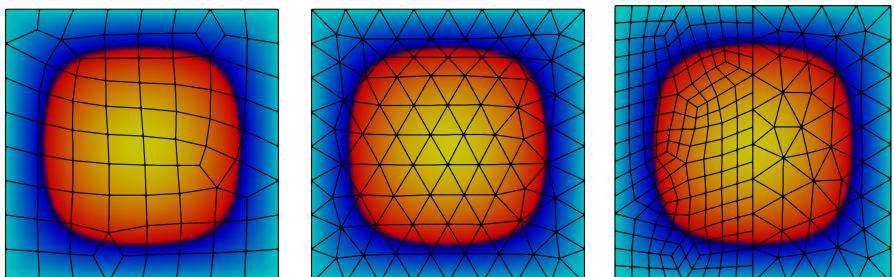
Matrix-free computations for finite element methods

Finite element methods (recap)

Solution of the weak form:

$$-\Delta u = f \qquad \leftrightarrow \qquad (\nabla v, \nabla u)_\Omega = (v, f)_\Omega \quad \text{with} \quad (a, b)_\Omega = \int a \cdot b \, \mathrm{d}\Omega$$

on a domain discretized with cells of arbitrary shape:



Finite element methods (recap - cont.)

- introduce cells & use scalar Lagrange finite element $\mathcal{Q}_k$
- replace global integration and global shape functions by local ones:

$$(\nabla v, \nabla u)_\Omega = \sum_e (\nabla(\boldsymbol{R}v), \nabla(\boldsymbol{R}u))_{\Omega_e} = \sum_e (\nabla v_e, \nabla u_e)_{\Omega_e} \approx \sum_e \sum_q (\nabla v_e, (|J|w)\nabla u_e)$$

... with $u_e(\xi) = \sum N_i(\xi) u_i^{(e)}$, N_i *shape function* i *in real space, mapping & quadrature*

- loop over all cells, assemble system matrix

$$\underbrace{\left(\sum_e \boldsymbol{R}^T \boldsymbol{A}_e \boldsymbol{R}\right)}_{\boldsymbol{A}} \boldsymbol{u} = \boldsymbol{f} \qquad \text{with} \qquad \mathbf{A}_{ij}^{(e)} = \sum_q (\nabla N_{iq}, (|J_q| w_q) \nabla N_{jq})$$

... and right-hand-side vector, and solve system

Question 4: Can we eliminate $\boldsymbol{A}$?

Matrix-free finite element methods

Given

$$\left(\sum_e \boldsymbol{R}^T \boldsymbol{A}_e \boldsymbol{R}\right) \boldsymbol{u} = \boldsymbol{f},$$

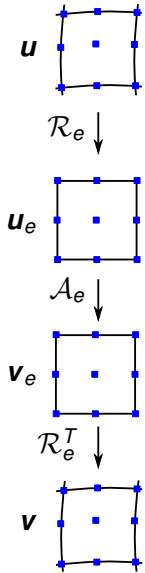
a matrix-free implementation can be derived by loop switching:

$$\sum_e \boldsymbol{R}^T (\boldsymbol{A}_e (\boldsymbol{R}\boldsymbol{u})) = \boldsymbol{f} \qquad \leftrightarrow \qquad \boxed{\sum_e \mathcal{R}^T \circ \mathcal{A}_e \circ \mathcal{R} \circ \boldsymbol{u} = \boldsymbol{f}}$$

with:

- $\mathcal{R}$ gather operation: $\boldsymbol{u}_e \leftarrow \mathcal{R} \circ \boldsymbol{u}$
- $\mathcal{A}_e$ application of cell integral: $\boldsymbol{v}_e \leftarrow \mathcal{A}_e \circ \boldsymbol{u}_e$; e.g., $\boldsymbol{v}_e \leftarrow \boldsymbol{A}_e \boldsymbol{u}_e$
- $\mathcal{R}^T$ scatter operation: $\boldsymbol{v} \leftarrow \mathcal{R}^T \circ \boldsymbol{v}_e$

Question 5: Do we need an explicit representation of $\boldsymbol{A}_e$?



Matrix-free finite element methods (cont.)

The effect of the element stiffness matrix can be evaluated in 3 steps:

$$\sum_q (\nabla v_e, (|J|w)\nabla u_e) = \sum_q (\nabla_{\vec{\xi}} v_e, (J^{-1}|J|wJ^{-T})\nabla_{\vec{\xi}} u_e)$$

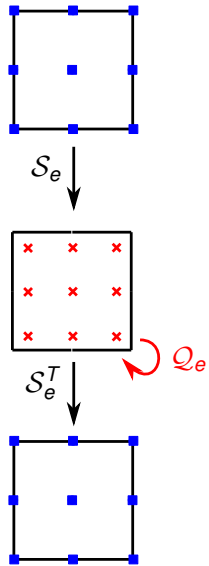
with $\nabla_\xi = (N^T_{,\xi_1}, N^T_{,\xi_2})^T$

In operator form:

$$\boldsymbol{v}_e = \mathcal{A}_e(\boldsymbol{u}_e) = \mathcal{S}^T \circ \mathcal{Q} \circ \mathcal{S} \circ \boldsymbol{u}_e$$

with:

- $\mathcal{S}^{(T)}$ transformation of DoF values to values/gradients at q. points
- $\mathcal{Q}$ block-diagonal operation with independent operations on each quadrature point

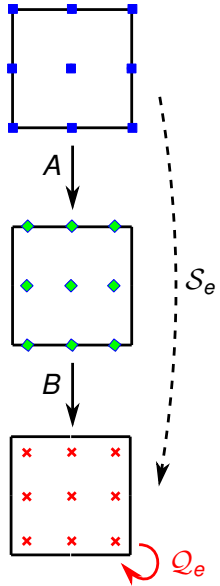


Matrix-free finite element methods (cont.)

For tensor-product elements,

$$N = N^{1D} \otimes N^{1D} \quad N_{,\xi_1} = N^{1D} \otimes N^{1D}_{,\xi} \quad N_{,\xi_2} = N^{1D}_{,\xi} \otimes N^{1D}.$$

Basis transformation of the form $v = (B \otimes A)u$ can be efficiently performed via sum factorization.

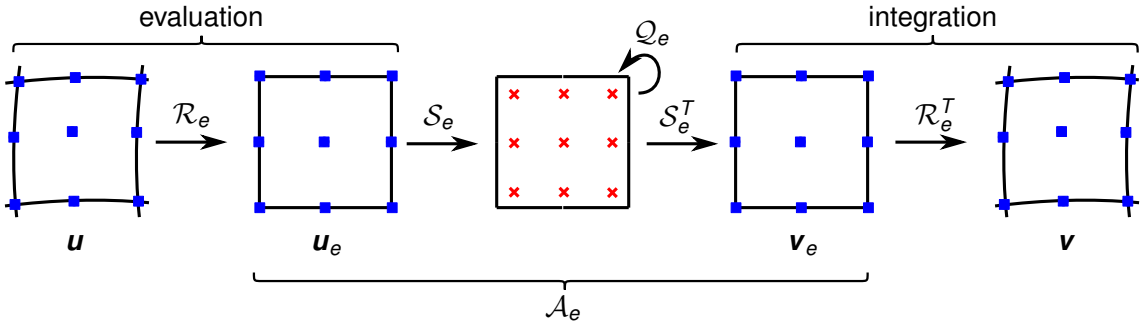


Matrix-free finite element methods (cont.)

Putting everything together gives a 5-step algorithm:

$$\boldsymbol{v} = \sum_e \mathcal{R}^T \circ \mathcal{A}^{(e)} \circ \mathcal{R} \circ \boldsymbol{u} = \boxed{\sum_e \mathcal{R}^T \circ \mathcal{S}^T \circ \mathcal{Q} \circ \mathcal{S} \circ \mathcal{R} \circ \boldsymbol{u} = \boldsymbol{v}}$$

... $\mathcal{R}^{(T)}/\mathcal{S}^{(T)}/\mathcal{Q}$: interchangeable depending on the given operator/physics



Summary & historical background

We get:

	memory consumption	number of flops
matrix-based	$\mathcal{O}(k^d)$	$\mathcal{O}(k^d)$
matrix-free	$\mathcal{O}(1)$	$\mathcal{O}(dk)$

... normalized by the number of degrees of freedom

Origins in spectral element community:

- *Orszag, S.A., 1979. Spectral methods for problems in complex geometrics. In Numerical methods for partial differential equations (pp. 273-305). Academic Press.*
- *Deville, M.O., Fischer, P.F., Fischer, P.F. and Mund, E.H., 2002. High-order methods for incompressible fluid flow (Vol. 9). Cambridge University Press.*

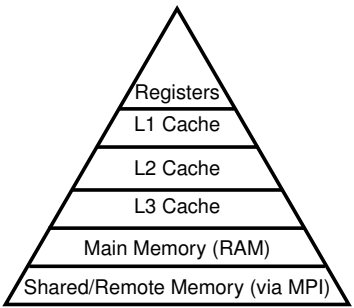
First application for FEM:

- *Brown, J., 2010. Efficient nonlinear solvers for nodal high-order finite elements in 3D. Journal of Scientific Computing, 45(1), pp.48-63.*
- *Kronbichler, M. and Kormann, K., 2012. A generic interface for parallel cell-based finite element operator application. Computers & Fluids, 63, pp.135-147.*

Summary & historical background (cont.)

Different phases of the matrix-free algorithms target different parts of the memory hierarchy and operate on different working sets:

▷ c: components; k: degree; d: dimension

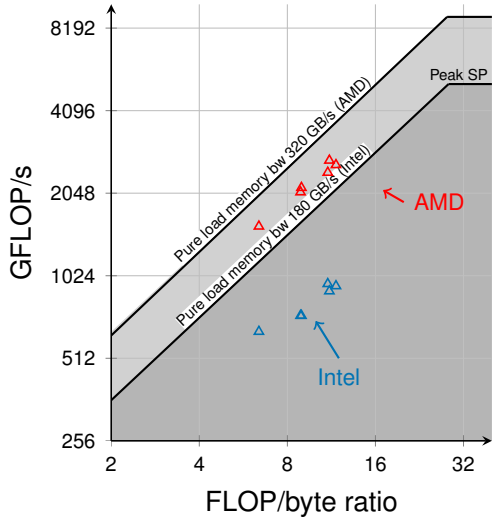


steps	working set	work
quad.	$O(c(d+1))$	$O(c^2(d+1))$
eval.	$O(c(d+1)(k+1)^d)$	$O(cd(k+1)^{d+1})$
R_K	$O(c(k+1)^d)$	

Arithmetic intensity:
$O(d(k+1))$ vs. 0.25
SpMV

Overall goal: minimize **time to solution**.

Summary & historical background (cont.)



Intel Cascade Lake Xeon Gold 6230: 2×20 cores / AMD EPYC 7713: 2×64 cores

Part 4:

Hands-on session

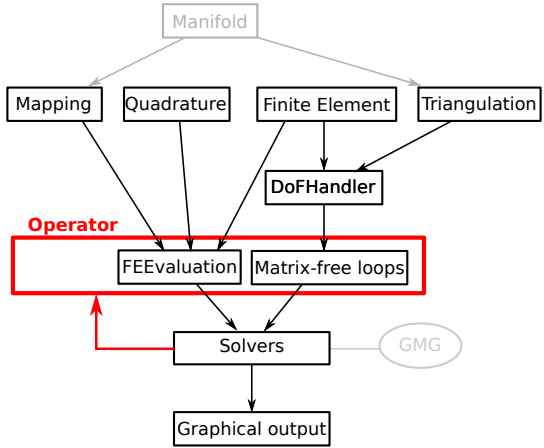
Fast matrix-free operator evaluation

idea: rely on operator evaluations

```
vmult(dst, src)
```

features:

- sum factorization
- vectorization via SIMD (CPUs)
- GPU support



Example

Matrix-free operator evaluation of $(\nabla v, \nabla u)_\Omega$:

```
matrix_free.template cell_loop<Vector<Number>, Vector<Number>>(
  [&](const auto & mf, auto &dst, const auto &src, const auto cells) {
    FEEvaluation<dim, -1, 0, n_components, Number, VectorizedArrayType> phi(mf);

    for (unsigned int cell = cells.first; cell < cells.second; ++cell)
      {
        phi.reinit(cell);
        phi.gather_evaluate(src, EvaluationFlags::gradients);

        for (unsigned int q = 0; q < phi.dofs_per_cell; ++q) //
          phi.submit_gradient(phi.get_gradient(q), q);       // =>

        phi.integrate_scatter(EvaluationFlags::gradients, dst);
      }
  }, dst, src);
```

... full code: `matrix_free.cc`

$$\longrightarrow \mathcal{J}_q^{-1} |\mathcal{J}_q| w_q \mathcal{J}_q^{-T} \nabla \hat{u}_q \qquad (\nabla v, \nabla u)_{\Omega^{(e)}}$$

$$\uparrow \qquad\qquad \downarrow$$

$$\sum_q (\nabla_\xi v, \mathcal{J}_q^{-1} |\mathcal{J}_q| w_q \mathcal{J}_q^{-T} \nabla_\xi u)$$

Part 5:

Challenges & current trends

Challenges & current trends

Matrix-free methods in FEM are efficient and are supported by many open-source libraries (e.g., deal.II), however, are not standard yet. Current research efforts are targeted towards:

- improving usability (e.g., by code generation) and standardization of the matrix-free interface (see also CEED initiative)
- development of matrix-free “off-the-shelf” building blocks, e.g., (block) preconditioner
- performance optimization & hardware portability
- porting of applications & identification of ways to express algorithms suitable for MF
- application in the context of (complex coupled) multiphysics solvers

Challenge 1: Usability and standardization

- code generation (+configuration files) *... UFL/AD/SD/Firedrake/DUNE*

```
a = dot(grad(v), grad(u))*dx
```

- make matrix-free programming directly in C++ easier *... deal.II style*

```
phi.reinit(cell);
phi.gather_evaluate(src, EvaluationFlags::gradients);
for (const auto q : phi.quadrature_point_indices())
  phi.submit_gradient(phi.get_gradient(q), q);
phi.integrate_scatter(EvaluationFlags::gradients, dst);
```

... by providing easy-to-use helper classes

- attempt to standardize via CEED (see challenge 6)

Challenge 2: Redundant computations

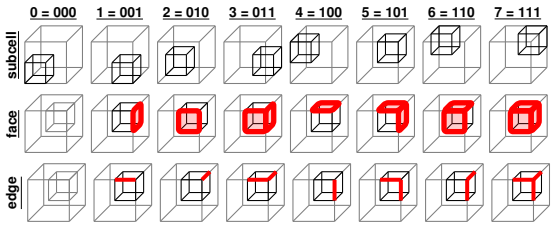
Matrix-free operator evaluation is efficient if the additional work, which has to be executed in each operator evaluation, is not excessive. In contrast to matrix-based algorithms, the work is restricted to the assembly phase and is performed only once.

Examples:

- application of hanging-node constraints
- expensive evaluation routines at quadrature-point level $\rightarrow$ finite-strain hyperelastic material

Challenge 2: Redundant computations: hanging-node constraints

Problem: constraint matrix is locally dense for hanging-node constraint $\mathcal{O}(p^4)$.
Challenge for matrix-free alternative: 137 possible refinement configurations!



Algorithm:

- update DoF map $\mathcal{G}_e$
- determine refinement configuration:

$$(subcell, face, edge) \rightarrow 8 \text{ bits}$$

- split up applications of general and hanging-node constraints
- apply hanging-node constraints via sum factorization

Munch, P., Ljungkvist, K. and Kronbichler, M., 2022. Efficient Application of Hanging-Node Constraints for Matrix-Free High-Order FEM Computations on CPU and GPU. In International Conference on High Performance Computing (pp. 133-152). Springer, Cham.

Challenge 2: Redundant computations: finite-strain hyperelastic material

Finite-strain hyperelastic material:

$$(\boldsymbol{v}, \boldsymbol{C}\boldsymbol{g}_s + \boldsymbol{G}\boldsymbol{g}) \quad \text{with} \quad \boldsymbol{g} = \nabla \boldsymbol{u},\ \boldsymbol{g}_s = \left(\nabla \boldsymbol{g} + \nabla \boldsymbol{g}^T\right)/2$$

Caching strategies:

v1: $\boldsymbol{C}, \boldsymbol{G} \in \mathcal{O}(d^4)$

v2: $c_1, c_2, \boldsymbol{G} \in \mathcal{O}(d^2) \quad \rightarrow \quad \boldsymbol{C}\boldsymbol{g}_s = c_1 \boldsymbol{g}_s + c_2 \text{tr}(\boldsymbol{g}_s)\boldsymbol{I}$

v3: $c_1 \in \mathcal{O}(1) \rightarrow \ \boldsymbol{C}\boldsymbol{g}_s = \left(c_1 \boldsymbol{g}_s + c_2 \text{tr}(\boldsymbol{g}_s)\boldsymbol{I}\right)/J,\ \boldsymbol{G} = \tau/J,\ \tau = \mu \boldsymbol{F} \cdot \boldsymbol{F}^T - c_1 \boldsymbol{I},\ J = \det(\boldsymbol{F}),\ ...$

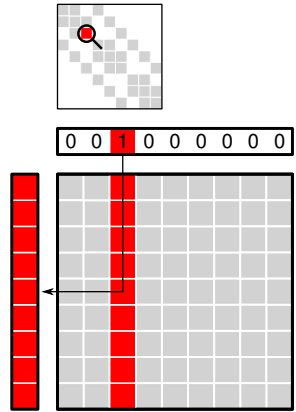
Davydov, D., Pelteret, J.P., Arndt, D., Kronbichler, M. and Steinmann, P., 2020. A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid. International Journal for Numerical Methods in Engineering, 121(13), pp.2874-2895.

Challenge 3: Preconditioning

Given the operators $\mathcal{A}$ and $\mathcal{A}_e$, can we simply *reconstruct* the diagonal of the matrix and the matrix representation?

The i-th columns of the local element matrix can be determined via matrix-free operator evaluations:

$$\text{column}_i(A_K) = \mathcal{A}_K(e_i)$$



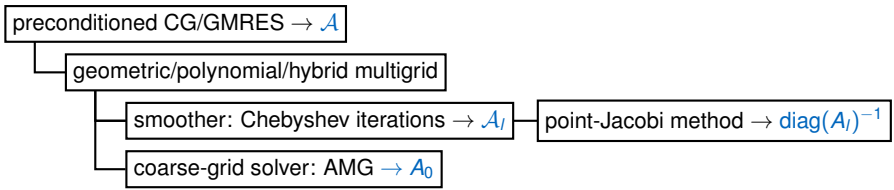
... and use that to setup traditional solvers.

Challenge 3: Preconditioning (cont.)

- preconditioned conjugate gradient methods around point Jacobi for Helmholtz problems

  Kronbichler, M., Sashko, D. and Munch, P., 2022. Enhancing data locality of the conjugate gradient method for high-order matrix-free finite-element implementations. arXiv preprint arXiv:2205.08909.

- geometric/polynomial/hybrid multigrid for elliptic problems



Fehn, N., Munch, P., Wall, W.A. and Kronbichler, M., 2020. Hybrid multigrid methods for high-order discontinuous Galerkin discretizations. Journal of Computational Physics, 415, p.109538.

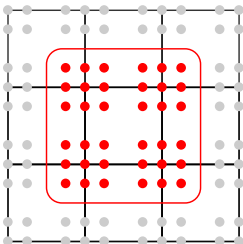
Munch, P., Heister, T., Saavedra, L.P. and Kronbichler, M., 2022. Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations. arXiv preprint arXiv:2203.12292.

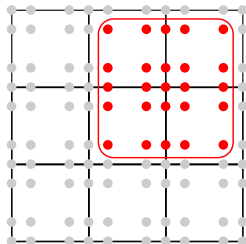
Challenge 3: Preconditioning (cont.)

- point-Jacobi preconditioner might not be robust, e.g., anisotropic meshes
- alternative: additive/multiplicative overlapping/non-overlapping Schwarz methods

$$A = \sum_e R_e^T A_e R_e \quad \leftrightarrow \quad \boxed{P^{-1} = \sum_b R_b^T A_b^{-1} R_b \quad \text{w.} \quad A_b = R_b A R_b^T}$$

problem: A not given $\rightarrow$ *domain decomposition (geometrically motivated)*



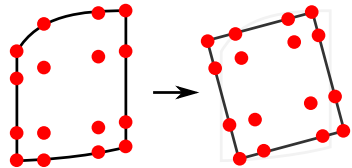


element-centered vs. vertex-star patch

possible domain solvers: fast-diagonalization method (Cartesian approx.), PCG, ...

Challenge 3: Preconditioning (cont.)

Fast diagonalization method (FDM) → Cartesian approximation



$$
\begin{aligned}
A_b &= M_2 \otimes M_1 \otimes K_0 + M_2 \otimes K_1 \otimes M_0 + K_2 \otimes M_1 \otimes M_0 \\
&= T_2 \otimes T_1 \otimes T_0 (\Lambda_2 \otimes I \otimes I + I \otimes \Lambda_1 \otimes I + I \otimes I \otimes \Lambda_0) T_2^T \otimes T_1^T \otimes T_0^T \quad \text{... with EVs/EWs } \Lambda / T
\end{aligned}
$$

with explicit inverse application:

$$
\boldsymbol{v} = A_b \boldsymbol{u} = \sum_b R_b^T T_2 \otimes T_1 \otimes T_0 (\Lambda_2 \otimes I \otimes I + I \otimes \Lambda_1 \otimes I + I \otimes I \otimes \Lambda_0)^{-1} T_2^T \otimes T_1^T \otimes T_0^T R_b \boldsymbol{u}
$$

... can be expressed as matrix-free loop!

Challenge 3: Preconditioning (cont.)

Lottes, J.W. and Fischer, P.F., 2005. Hybrid multigrid/Schwarz algorithms for the spectral element method. Journal of Scientific Computing, 24(1), pp.45-78.

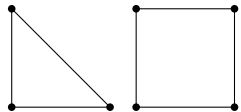
Witte, J., Arndt, D. and Kanschat, G., 2021. Fast tensor product Schwarz smoothers for high-order discontinuous Galerkin methods. Computational Methods in Applied Mathematics, 21(3), pp.709-728.

Brubeck, P.D. and Farrell, P.E., 2021. A scalable and robust vertex-star relaxation for high-order FEM. arXiv preprint arXiv:2107.14758.

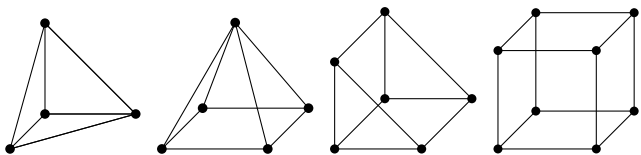
Challenge 4: Different cell shapes and element types

Meshes in practice also contain non-tensor-product cells:

- reference-cell types in 2D: triangle, quadrilateral



- reference-cell types in 3D: tetrahedron, pyramid, wedge/prism, hexahedron



... do not allow using sum factorization → more expensive!

Is it possible to generate hex-dominated meshes?

Challenge 4: Different cell shapes and element types (cont.)

- discontinuous Galerkin methods $\rightarrow$ computation of fluxes at internal faces

  Kronbichler, M. and Kormann, K., 2019. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM Transactions on Mathematical Software (TOMS), 45(3), pp.1-40.

- Hermite(-like) elements *... also: Ivy Weber*

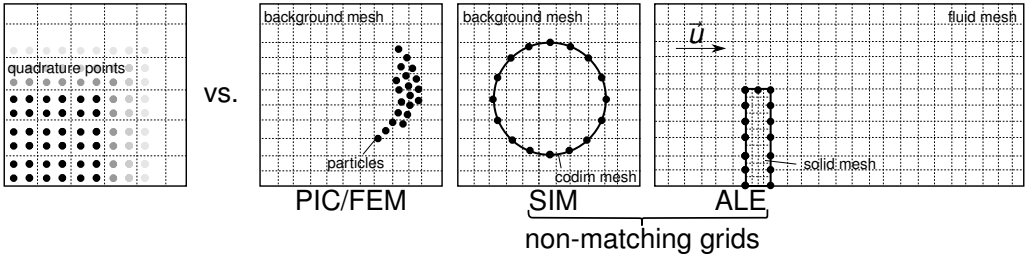
  Kronbichler, M., Kormann, K., Fehn, N., Munch, P. and Witte, J., 2019. A Hermite-like basis for faster matrix-free evaluation of interior penalty discontinuous Galerkin operators. arXiv preprint arXiv:1907.08492.

- H^{div} and H^{curl} elements (e.g., Raviart-Thomas and Nédélec elements)

  Pazner, W., Kolev, T. and Dohrmann, C., 2022. Low-order preconditioning for the high-order finite element de Rham complex. arXiv preprint arXiv:2203.02465.

  Wik, N., 2022. High-performance implementation of H (div)-conforming elements for incompressible flows.

Challenge 5: Non-matching grids and coupling of fields



E.g., computation of surface-tension force in the sharp-interface method (SIM):

$$(\boldsymbol{v}, \kappa \boldsymbol{n})_\Gamma \approx \sum_q \boldsymbol{v}(\boldsymbol{x}_q) \cdot (\kappa(\boldsymbol{x}_q)\, \boldsymbol{n}(\boldsymbol{x}_q))\, (JxW)_q$$

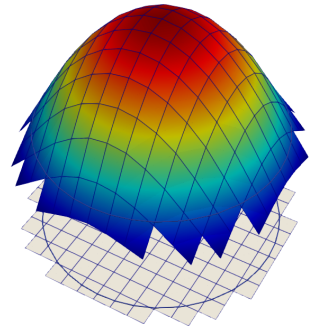
red: on Γ

Arbitrary Lagrangian Eulerian (ALE): communicate displacement ($\boldsymbol{u}$) and force (σ) between non-matching grids, to be used as boundary conditions. Are mortar methods possible?

... Niklas Fehn/David Schneider

Challenge 5: Non-matching grids and coupling of fields (cont.)

- CutFEM/CutDG



Find $u_h \in V_\Omega^h$ such that

$$a_h(u_h, v_h) = L_h(v_h), \quad \forall v_h \in V_\Omega^h,$$

where

$$a_h(u_h, v_h) = (\nabla u_h, \nabla v_h)_\Omega - (\partial_n u_h, v_h)_\Gamma - (u_h, \partial_n v_h)_\Gamma + \left(\frac{\gamma_D}{h} u_h, v_h\right)_\Gamma,$$

$$L_h(v_h) = (f, v)_\Omega + \left(u_D, \frac{\gamma_D}{h} v_h - \partial_n v_h\right)_\Gamma.$$

... Max Bergbauer/Martin Kronbichler/Simon Sticko

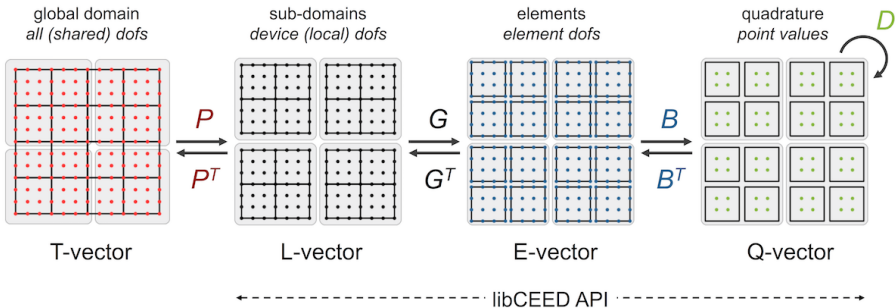
- element birth/death $\rightarrow$ metal powder bed fusion AM *... B. Turcksin/S. Proell*

Challenge 6: Perfomance & hardware portability

Matrix-free algorithms are applicable both to CPU- and GPU-based systems, but require different hardware-aware optimizations.

Challenge 6: Perfomance & hardware portability (cont.)

$$A = P^T G^T B^T D B G P$$



Philosophy of libCEED:

- low-level library: gathering G and basis change B are optimized for hardware + JIT
- high-level library (like MFEM, NEK-RS): handle parallelization and provide off-the-shelf quadrature functors

Challenge 6: Perfomance & hardware portability (cont.)

Fischer, P., Min, M., Rathnayake, T., Dutta, S., Kolev, T., Dobrev, V., Camier, J.S., Kronbichler, M., Warburton, T., Świrydowicz, K. and Brown, J., 2020. Scalability of high-performance PDE solvers. The International Journal of High Performance Computing Applications, 34(5), pp.562-586.

Kolev, T., Fischer, P., Min, M., Dongarra, J., Brown, J., Dobrev, V., Warburton, T., Tomov, S., Shephard, M.S., Abdelfattah, A. and Barra, V., 2021. Efficient exascale discretizations: High-order finite element methods. The International Journal of High Performance Computing Applications, 35(6), pp.527-552.

Anderson, R., Andrej, J., Barker, A., Bramwell, J., Camier, J.S., Cerveny, J., Dobrev, V., Dudouit, Y., Fisher, A., Kolev, T. and Pazner, W., 2021. MFEM: A modular finite element methods library. Computers & Mathematics with Applications, 81, pp.42-74.

Fischer, P., Kerkemeier, S., Min, M., Lan, Y.H., Phillips, M., Rathnayake, T., Merzari, E., Tomboulides, A., Karakus, A., Chalmers, N. and Warburton, T., 2021. NekRS, a GPU-Accelerated Spectral Element Navier-Stokes Solver. arXiv preprint arXiv:2104.05829.

Challenge 6: Perfomance & hardware portability (cont.)

GPU implementation in deal.II:

Ljungkvist, K., 2017. Finite element computations on multicore and graphics processors (Doctoral dissertation, Acta Universitatis Upsaliensis).

Kronbichler, M. and Ljungkvist, K., 2019. Multigrid for matrix-free high-order finite element computations on graphics processors. ACM Transactions on Parallel Computing (TOPC), 6(1), pp.1-32.

Munch, P., Ljungkvist, K. and Kronbichler, M., 2022. Efficient Application of Hanging-Node Constraints for Matrix-Free High-Order FEM Computations on CPU and GPU. In International Conference on High Performance Computing (pp. 133-152). Springer, Cham.

Challenge 6: Perfomance & hardware portability (cont.)

CPU implementation in deal.II:

- modern CPU hardware has SIMD vector units (AVX-512: processes 8 doubles at once)
- programs have to use these units, by special instructions, to reach high performance
- auto-vectorization by compilers for sum factorization is not suitable
- explicit vectorization is needed
- vectorization within a cell vs. vectorization over cells (deal.II $\rightarrow$ "batch")
- instead of working with `double`, one works with `VectorizedArray<double>`
- challenge: code path diverges for cells in the same batch
  - solution: categorization, masking, creating batches on the fly
  - examples: CutFEM/CutDG, shock capturing ... *Maximilian Bergbauer/David Schneider*

Part 6:

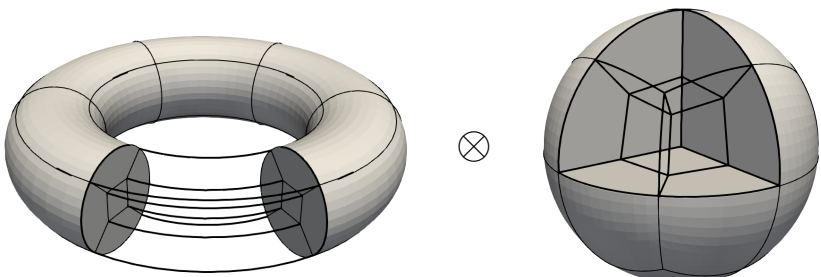
Projects

Application 1: Computational plasma physics with hyper.deal

Vlasov equation: non-linear, high-dimensional, hyperbolic PDE *up to:* 5.6×10^{12} *DoFs*

$$\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} f + \vec{a}(t, f, \vec{x}, \vec{v}) \cdot \nabla_{\vec{v}} f = 0 \qquad \vec{a}(t, f, \vec{x}, \vec{v}) = \frac{q}{m} \left(\vec{E}(t, \vec{x}) + \vec{v} \times \vec{B}(t, \vec{x}) \right)$$

... discretized with high-order DG solved on “tokamak”-like geometry:

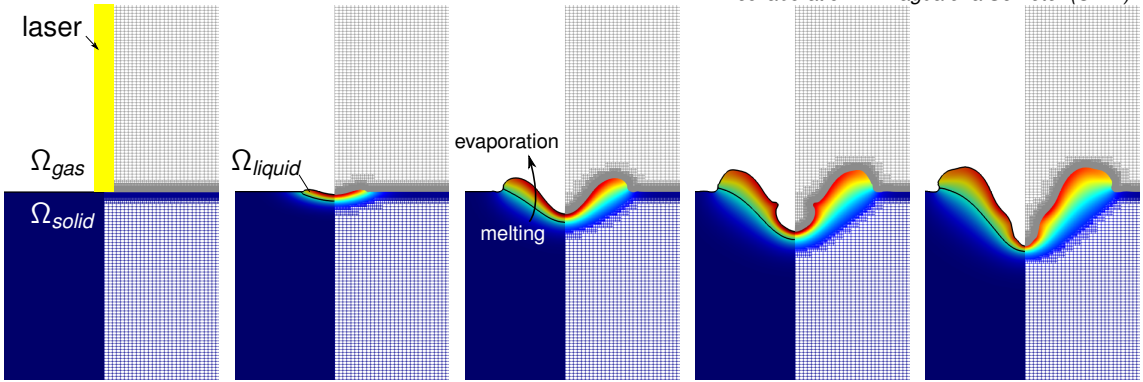


Munch, P., Kormann, K. and Kronbichler, M., 2021. hyper. deal: An efficient, matrix-free finite-element library for high-dimensional partial differential equations. ACM Transactions on Mathematical Software (TOMS).

Application 2: Melt-pool modeling

Simulation of melt-pool process:

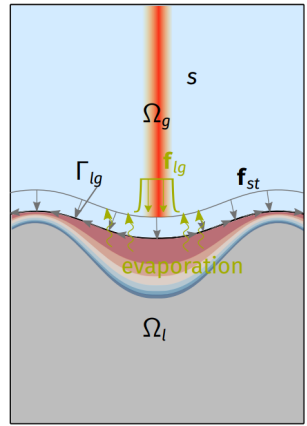
in collaboration w. Magdalena Schreter (UIBK)



... resolution of high temperature gradients and changes in material parameters

Application 2: Melt-pool modeling (cont.)

- mass: $\nabla \cdot \mathbf{u} = -\frac{\dot{\rho}}{\rho}$ — evaporative mass flux
- momentum: $\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\,\mathbf{u}\right) = -\nabla p + \eta\Delta\mathbf{u} + \rho\,\mathbf{g} + \mathbf{f}_{st} + \mathbf{f}_{lg}$ — recoil pressure
- level set: $\frac{\partial \phi}{\partial t} + \mathbf{u}_\Gamma \nabla\phi = 0$
- energy: $\frac{\partial\left(\rho\, c_p\, T\right)}{\partial t} + \nabla\cdot\left(\rho\, c_p\, T\,\mathbf{u}\right) = \nabla\cdot(k\nabla T) + s + s_{lg}$ — evaporative heat flux



Application 3: Computational fluid mechanics

- adaflo: two-phase flow solver/(variable coefficient) incompressible NS/level set/PF

  Kronbichler, M., Diagne, A. and Holmgren, H., 2018. A fast massively parallel two-phase flow solver for microfluidic chip simulation. The International Journal of High Performance Computing Applications..

- ExaDG: incompressible + compressible NS, fluid-structure interaction (FSI), ...

  Krank, B., Fehn, N., Wall, W.A. and Kronbichler, M., 2017. A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow. Journal of Computational Physics.

  Arndt, D., Fehn, N., Kanschat, G., Kormann, K., Kronbichler, M., Munch, P., Wall, W.A. and Witte, J., 2020. ExaDG: High-order discontinuous Galerkin for the exa-scale. In Software for Exascale Computing-SPPEXA 2016-2019 (pp. 189-224). Springer, Cham.

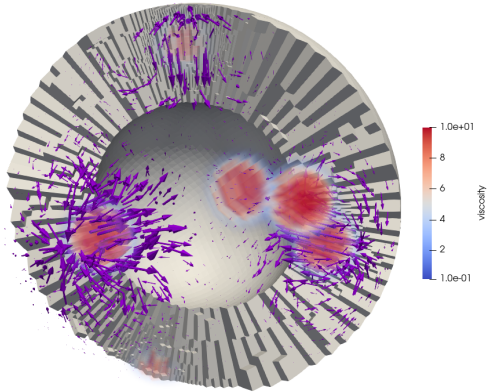
  Kronbichler, M., Fehn, N., Munch, P., Bergbauer, M., Wichmann, K.R., Geitner, C., Allalen, M., Schulz, M. and Wall, W.A., 2021, November. A next-generation discontinuous Galerkin fluid dynamics solver with application to high-resolution lung airflow simulations. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-15).

Application 4: Geosciences with aspect: variable-viscosity Stokes problem

Solve:

$$-\nabla \cdot (2\eta\varepsilon(\boldsymbol{u})) + \nabla p = f$$
$$\nabla \cdot \boldsymbol{u} = 0$$

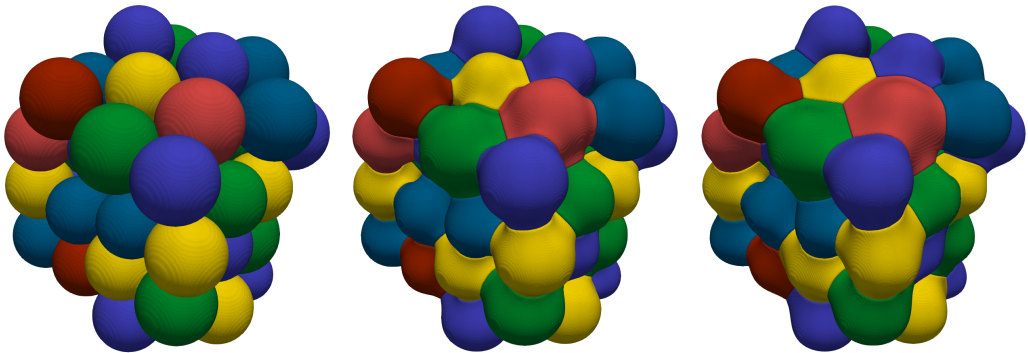
with a Q2-Q1 Taylor-Hood discretization of velocity $\vec{u}$, pressure p, and viscosity $\eta(x)$.



Clevenger, T.C. and Heister, T., 2021. Comparison between algebraic and matrix-free geometric multigrid for a Stokes problem on adaptive meshes with variable viscosity. Numerical Linear Algebra with Applications.

Application 5: Modeling of solid-state-sintering processes

in collaboration w. Vladimir Ivannikov (Helmholtz-Zentrum Hereon)



... 49 particles, 8 components, 40 million DoFs

Application 5: Modeling of solid-state-sintering processes (cont.)

- coupling of Cahn-Hilliard equations (conserved quantity) and Allen-Cahn equations (for each particle) $\rightarrow$ time-dependent non-linear system to be solved with Newton solver
- (linearized) weak form of sintering problem $\rightarrow$ to be solved with GMRES:

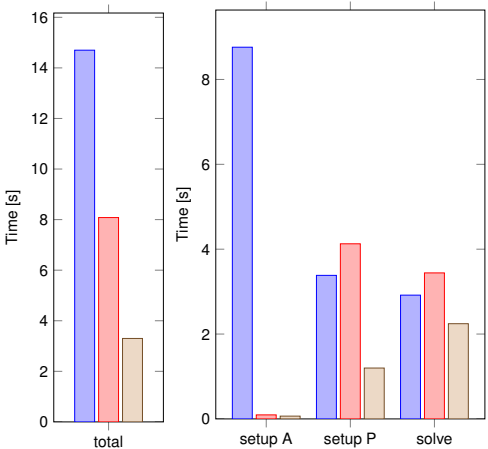
$$\mathcal{A} = \begin{pmatrix} (v_0, \frac{1}{\Delta t} u_0) + (\nabla v_0, M' \mu' u_0) + (\nabla v_0, M \nabla u_0) & + & (\nabla v_0, M \nabla u_1) & + & (v_0, \frac{\partial M}{\partial \eta_0} \mu'' u_2) & + & (v_0, \frac{\partial M}{\partial \eta_1} \mu'' u_3) \\ (v_1, f'' u_0) + (\nabla v_1, \kappa_c \nabla u_0) & + & (v_1, -u_1) & + & (v_1, \frac{\partial^2 f}{\partial \eta_0^2} u_2) & + & (v_1, \frac{\partial^2 f}{\partial \eta_1^2} u_3) \\ (v_2, L \frac{\partial^2 f}{\partial c \partial \eta_0} u_0) & & & + & (v_2, (\frac{1}{\Delta t} + L \frac{\partial^2 f}{\partial \eta_0^2}) + (\nabla v_2, L \kappa_c \nabla u_2) & + & (v_2, L \frac{\partial^2 f}{\partial \eta_0 \eta_1} u_3) \\ (v_3, L \frac{\partial^2 f}{\partial c \partial \eta_1} u_0) & & & + & (v_3, L \frac{\partial^2 f}{\partial \eta_1 \eta_0} u_2) & + & (v_3, (\frac{1}{\Delta t} + L \frac{\partial^2 f}{\partial \eta_1^2}) u_3) + (\nabla v_3, L \kappa_c \nabla u_3) \end{pmatrix}$$

  with:
  - Cahn-Hilliard block and Allen-Cahn block $\rightarrow$ efficient preconditioner (for each block?)
  - L and κ_c constant
  - with f (free energy) and M (mobility), dependent on old solution
- Allen-Cahn equations can be merged if particles are not adjacent $\rightarrow$ grain tracking

Application 5: Modeling of solid-state-sintering processes (cont.)

Goal is the minimization of runtime T:

$$T = N_T \cdot T_{P,setup} + N_T \cdot \overline{N}_N \cdot T_{A,setup} + N_T \cdot \overline{N}_N \cdot \overline{N}_L \cdot (T_{A,vmult} + T_{P,vmult} + T_{vector\ update}) + X$$



Comparison (two-particle case):

- matrix-based + ILU
- matrix-free + ILU
- matrix-free + block precon. (ILU, diag.)

Part 7:

Conclusions & outlook

Conclusions & outlook

Matrix-free FEM computation is fast:

- fast matrix-vector multiplication ($k > 1$)
- reduced costs for setup (no sparsity pattern, assembly)

Until it will become mainstream:

- resolve challenges: usability, preconditioning, portability, ...
- convince by success in practice

deal.II Workshop @ Durham University

Lecture 2: adaptive mesh refinement & linear solvers

Peter Munch[1]

[1] Institute of Mathematics, Technical University of Berlin, Germany

April 4, 2025

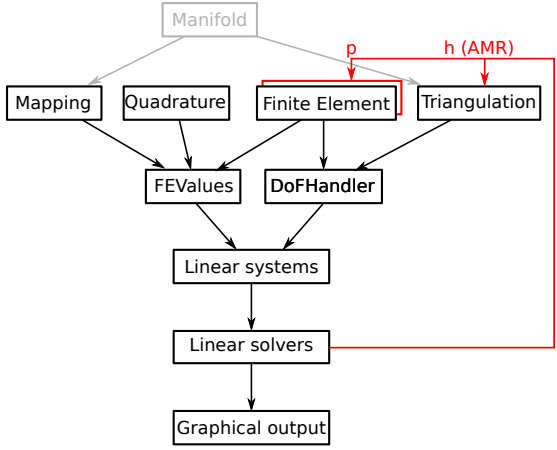
Part 1:

Adaptive mesh refinement

Main modules

support of:

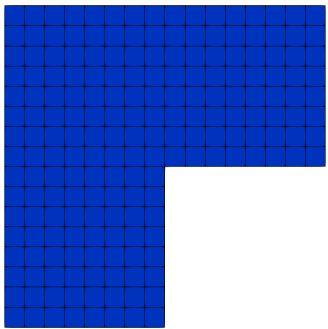
- FE collections
- hanging nodes
- distributed hp-adaptivity

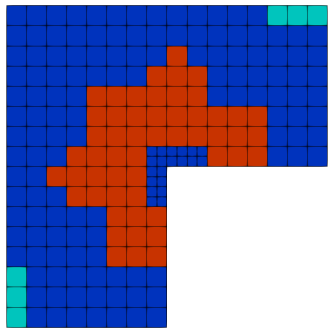


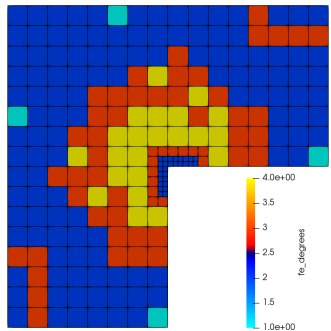
Example

3 cycles of hp-adaptivity:

▷ *colors: polynomial degree* $1 \leq k \leq 4$







... solved with a distributed matrix-free p-multigrid algorithm

Refinement strategies

- geometrical knowledge

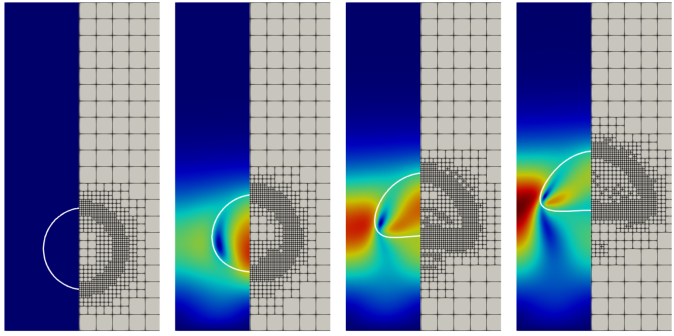


Figure: Rising bubble: refinement around bubble interface

- error estimation: e.g., Kelly error estimator $\rightarrow$ jump of gradients at faces

$$\nu_K^2 = \sum_{F \in \partial K} c_F \int_{\partial K_F} \left[\left[a \frac{\partial u_h}{\partial n} \right] \right]^2$$

Refinement strategies: implementation details

manual flagging

```
for (auto cell :
    tria.active_cell_iterators())
  if (/*some criterion*/)
    cell->set_refine_flag();
  else if (/*some criterion*/)
    cell->set_coarsen_flag();
```

Kelly error estimation

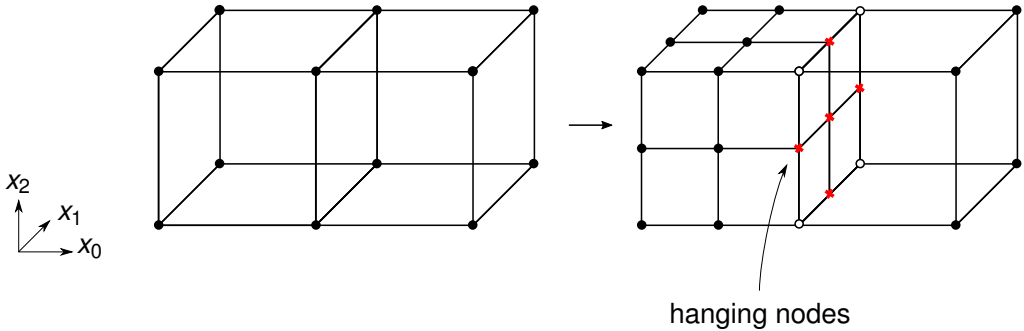
```
KellyErrorEstimator<dim>::estimate(
  dof_handler, quad, {},
  solution, estimated_error_per_cell);

GridRefinement::
    refine_and_coarsen_fixed_number(
  tria, estimated_error_per_cell, .1, .4);
```

```
tria.execute_coarsening_and_refinement();
```

mesh coarsening/refinement

Challenge: application of hanging-node constraints

Example: 2 coarse cells; scalar, linear Lagrange elements ($k = 1$); non-conformal refinement



- task: guarantee H^1-conformity
- normally via constraint matrix: $x_i = C_{ij} x_j + b_i$

... locally dense $\mathcal{O}(k^{2(d-1)})$

Shephard, M.S., 1984. Linear multipoint constraints applied via transformation as part of a direct stiffness assembly process. IJNME.

Challenge: application of hanging-node constraints (cont.)

The quadratic form under constraints is:

$$f(\tilde{x}) = \frac{1}{2}\tilde{x}^\top \tilde{A}\tilde{x} - \tilde{x}^\top \tilde{b} \qquad \text{with} \qquad \tilde{x} = Cx$$

and transformed:

$$f(x) = \frac{1}{2}x^\top C^\top \tilde{A} C\tilde{x} - x^\top C^\top \tilde{b} \qquad \rightarrow \qquad \underbrace{C^\top \tilde{A} C}_{:=A} x = \underbrace{C^\top \tilde{f}}_{:=f} \qquad \rightarrow \qquad Ax = f$$

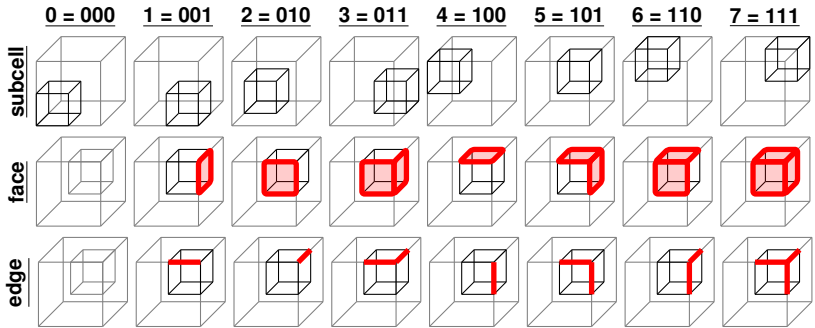
... with $\tilde{\square}$ indicating the unconstrained system

In practice, constraints are applied during assembly via `AffineConstraints::distribute_local_to_global()`:

$$A = \sum_e R_e^\top C_e^\top A_e C_e R_e \qquad \text{and} \qquad A = \sum_e R_e^\top C_e^\top f_e.$$

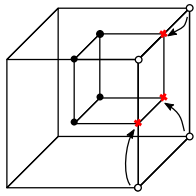
Fast application of hanging-node constraints

137 refinement configurations, which can be characterized by:



... triple (subcell, face, edge)

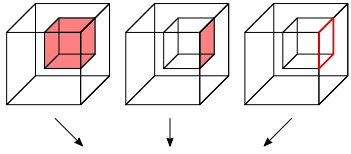
Fast application of hanging-node constraints (cont.)



1) update DoF map

↓

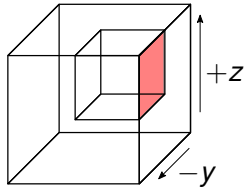
consider mesh orientation



2) encode refinement config.

↓

137 configurations in 3D



3) inplace interpolation

↓

CPU/GPU-specific

↓

decode/vectorization

Fischer, Paul F. et al. "Spectral element methods for transitional flows in complex geometries." JSC. 2002.

Ljungkvist, K. "Matrix-free finite-element computations on graphics processors with adaptively refined unstructured meshes." In SpringSim (HPC). 2017

PM, K. Ljungkvist, and M. Kronbichler, "Efficient application of hanging-node constraints for matrix-free high-order FEM computations on CPU and GPU", ISC, 2022.

Part 2:

Linear solvers: direct solvers

Gauss elimination vs. LU factorization

Goal: directly solve

$$Ax = b.$$

Approaches:

- Gauss(–Jordan) elimination
- LU factorization: ▷ $A = A^\top$: Cholesky decomposition

$$\underbrace{Ax = (LU)x = b}_{\text{setup}}, \qquad \underbrace{Ly = b,\ Ux = y}_{\text{solve}}$$

Implementation details

Variants in deal.II:

- dense direct solvers
  - `FullMatrix`
  - `LAPACKFullMatrix`
- sparse direct solvers
  - `TrilinosWrappers::SolverDirect`
  - `PETScWrappers::SparseDirectMUMPS`

Example:

```
TrilinosWrappers::SparseMatrix matrix;
Vector<double> solution, rhs;

TrilinosWrappers::SolverDirect solver;
solver.initialize(matrix);
solver.solve(solution, rhs);
```

Part 3:

Linear solvers: iterative solvers

Implementation details

Goal: iteratively solve

$$Ax = b.$$

Variants:

- `SolverRelaxation`
- `SolverRichardson`
- `SolverCG`
- `SolverGMRES`
- ...

We need: A and P^{-1} that implement

```
void vmult(VT & dst, const VT & src) const;
```

Example:

Richardson solver: $x^{(k+1)} = x^{(k)} + \omega P^{-1}\left(b - Ax^{(k)}\right)$

```
void SolverRichardson<VectorType>::solve(
  const MatrixType         &A,
  VectorType               &x,
  const VectorType         &b,
  const PreconditionerType &preconditioner)
{

  while (conv == SolverControl::iterate)
    {
      A.vmult(r, x);
      r.sadd(-1., 1., b);
      preconditioner.vmult(d, r);
      x.add(additional_data.omega, d);

      ++iter;
    }
}
```

Implementation details (cont.)

```
void vmult(VT & dst, const VT & src) const;
```

- can be implemented by users (e.g., in a matrix-free way)
- A: `FullMatrix`, `SparseMatrix`, `SparseMatrix`, `TrilinosWrappers::SparseMatrix`, ...
- P^{-1}: use off-the-shelf implementations also interfacing to PETSc and Trilinos
  - `DiagonalMatrix`, `PreconditionRelaxation`, `PreconditionChebyshev`
  - `TrilinosWrappers::PreconditionILU/PreconditionAMG`
  - `PreconditionMG`

Example:

```
TrilinosWrappers::PreconditionAMG preconditioner;
TrilinosWrappers::SparseMatrix matrix;
Vector<double> solution, rhs;

ConvergenceControl convergence_control;
SolverCG<Vector<double>> solver(convergence_control);
solver.solve(solver, solution, rhs, preconditioner);
```

Multigrid: general algorithm

Solve the system of linear equations $A(\boldsymbol{x}) = \boldsymbol{b}$:

- presmoothing:

$$\boldsymbol{x} \leftarrow S(\boldsymbol{x})$$

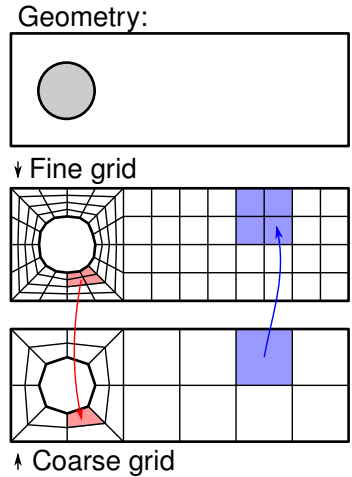
- recursive coarse-grid correction:

$$A_c(\boldsymbol{v}) = R(\boldsymbol{b} - A(\boldsymbol{x})) \quad \text{and} \quad \boldsymbol{x} \leftarrow \boldsymbol{x} + P(\boldsymbol{v})$$

e.g., with $A_c = RAP$ (Galerkin multiplication)

- postsmoothing:

$$\boldsymbol{x} \leftarrow S(\boldsymbol{x})$$

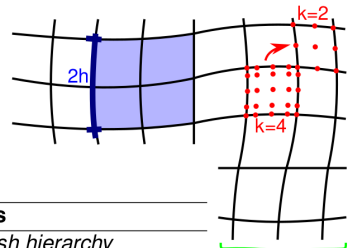


How should one define the levels?

Multigrid: level definitions

Multigrid: Options

- *h*: cell width
- *k*: polynomial order of shape functions
- *l*: multigrid level



	Pros	Cons
h-GMG	+ matrix-free	- *mesh hierarchy*
p-GMG	+ matrix-free + no mesh hierarchy	- high-order needed
AMG	+ flexible (out-of-the-box) + *geometric flexibility* + anisotropic problems	- matrix-based - not suited for DG - not suited for high-order

- hybrid multigrid: nesting of different variants of multigrid (Fehn et al. ['20])
- AMG: $A_c = RA_fP$ (Galerkin multiplication)
- normally $R = P^T$ (with P: pseudo projection)

Part 4:

Geometric multigrid for locally refined meshes[1]

[1] P. Munch, K. Ljungkvist, M. Kronbichler, Efficient application of hanging-node constraints for matrix-free high-order FEM computations on CPU and GPU, 2022 ISC High Performance, Hamburg, Germany, 2022

Part 1:

Introduction

Introduction

- recent additions to the multigrid infrastructure in the open-source FEM library `deal.II`
- multigrid is possible through wrappers to external libraries (ML, MueLu, BoomerAMG) or through `deal.II`'s geometric multigrid infrastructure
- for locally refined meshes, we now support two strategies: "local smoothing" (old) and "global coarsening" (new)
- global coarsening: not limited to geometric multigrid, but can also be used to implement *p*-multigrid, auxiliary-space idea, ...

The presentation concentrates on CPUs. Algorithms are also applicable to GPUs!

Introduction (cont.)

This presentation is loosely based on:

P. Munch, T. Heister, L. Prieto Saavedra, M. Kronbichler.
Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations. submitted. arXiv:2203.12292. 2022.

Part 2:

Multigrid methods for adaptively refined meshes

General multigrid algorithm

Solve the system of linear equations $\mathcal{A}(\boldsymbol{x}) = \boldsymbol{b}$:

- presmoothing:

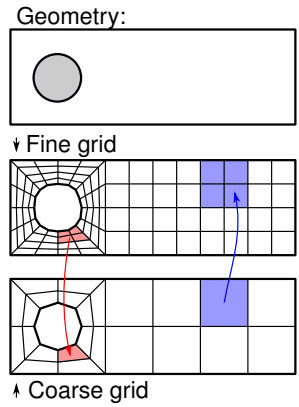
$$\boldsymbol{x} \leftarrow \mathcal{S}(\boldsymbol{x})$$

- recursive coarse-grid correction:

$$\mathcal{A}_c(\boldsymbol{v}) = \mathcal{R}(\boldsymbol{b} - \mathcal{A}(\boldsymbol{x})) \quad \text{and} \quad \boldsymbol{x} \leftarrow \boldsymbol{x} + \mathcal{P}(\boldsymbol{v})$$

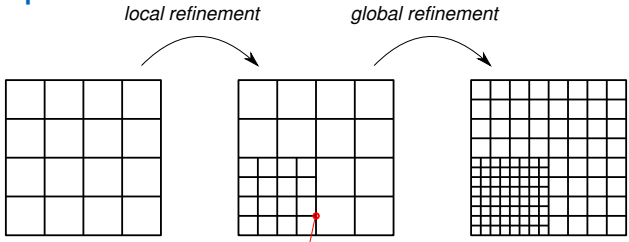
- postsmoothing:

$$\boldsymbol{x} \leftarrow \mathcal{S}(\boldsymbol{x})$$



How should one define the levels?

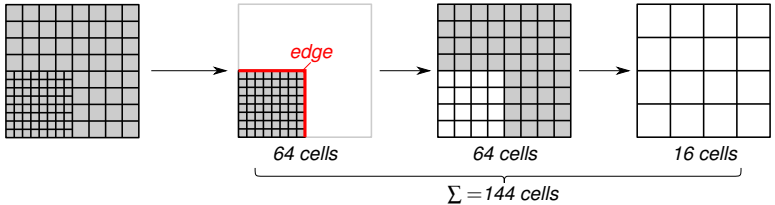
Example of adaptive mesh



- at hanging nodes: maintain H^1 regularity of the tentative solution (force solution representation of refined side to be matching polynomial representation of coarse side)
- apply (hanging-node) constraints via $x_i = \sum_j c_{ij} x_j + b_j$ (constraint matrix)
- hanging nodes are “motivation” for development of different geometric multigrid variants

Shephard, M.S., 1984. Linear multipoint constraints applied via transformation as part of a direct stiffness assembly process. IJNME.

Multigrid variant 1: local smoothing (LS)

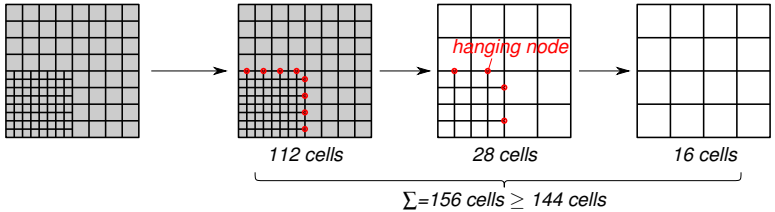


- internal interface/“edge”: homogeneous/inhomogeneous DBC during pre-/postsmoothing
- uses refinement levels + first-child policy[1] $\rightarrow$ memory-efficient, efficient transfer
- smoothers designed for uniform meshes, e.g., patch smoothers, are applicable
- available in `deal.II` for a long time

Brandt, A. 1977. Multi-level adaptive solutions to boundary-value problems. Mathematics of computation.
Janssen, B. and Kanschat, G., 2011. Adaptive multilevel methods with local smoothing for H^1-and H^{curl}-conforming high order finite element methods. SIAM JSC.
Clevenger, T.C. et. al., 2021. A flexible, parallel, adaptive geometric multigrid method for FEM. ACM TOMS.

[1] first child policy: owner of parent cell is owner of first child

Multigrid variant 2: global coarsening (GC)



- repartitioning of levels is common → good load balance
- hanging-node constraints need to be considered during smoothing
- new in `deal.II` since release 9.3 (June 2021)

Becker, R. and Braack, M., 2000. Multigrid techniques for finite elements on locally refined meshes. Numerical linear algebra with applications, 7(6), pp.363-379.
Becker, R., Braack, M. and Richter, T., 2007. Parallel multigrid on locally refined meshes. RFDT.
Rudi, J. et. al., 2015. An extreme-scale implicit solver for complex PDEs: highly heterogeneous flow in earth's mantle. SC'15.

Part 3:

Implementation

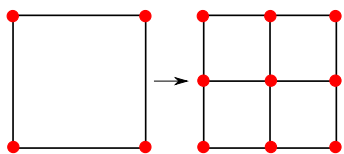
Matrix-free transfer operators

Express prolongation as a matrix-free loop:

$$\boldsymbol{x}^{(f)} = \mathcal{P}^{(f,c)} \circ \boldsymbol{x}^{(c)} \quad \leftrightarrow \quad \boxed{\boldsymbol{x}^{(f)} = \sum_{e \in \{cells\}} \mathcal{S}_e^{(f)} \circ \mathcal{W}_e^{(f)} \circ \mathcal{P}_e^{(f,c)} \circ \mathcal{C}_e^{(c)} \circ \mathcal{G}_e^{(c)} \circ \boldsymbol{x}^{(c)}}$$

with:

- $\mathcal{C}_e^{(c)} \circ \mathcal{G}_e^{(c)}$: gather values and apply constraints
- $\mathcal{P}_e^{(f,c)}$: prolongation on coarse cell (see figure)
- $\mathcal{W}_e^{(f)}$: consider valence
- $\mathcal{S}_e^{(f)}$: add into $\boldsymbol{x}^{(f)}$ (coarse-side identification $\rightarrow$ communication)



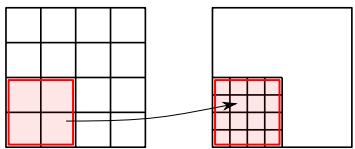
$\mathcal{Q}^1$

Matrix-free transfer operators (cont.)

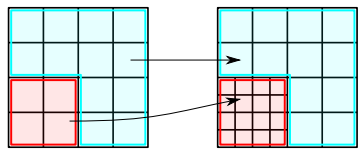
Enabling vectorization over elements

$$\boldsymbol{x}^{(f)} = \sum_{c} \sum_{e \in \{e | \mathcal{C}(e) = c\}} \mathcal{S}_e^{(f)} \circ \mathcal{W}_e^{(f)} \circ \mathcal{P}_c^{(f,c)} \circ \mathcal{C}_e^{(c)} \circ \mathcal{G}_e^{(c)} \circ \boldsymbol{x}^{(c)}.$$

by categorization of element-prolongation types:



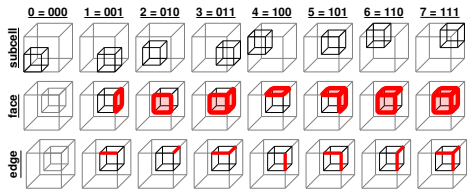
LS: 1 category



GC: 2 categories

Efficient application of hanging-node constraints

Problem: constraint matrix is locally dense for hanging-node constraint $\mathcal{O}(p^4)$.
Challenge for matrix-free alternative: 137 possible refinement configurations!



Algorithm:

- update DoF map $\mathcal{G}_e$
- determine refinement configuration:

  $$(\mathit{subcell}, \mathit{face}, \mathit{edge}) \rightarrow 8 \text{ bits}$$

- split up application of general and hanging-node constraints
- apply hanging-node constraints via sum factorization

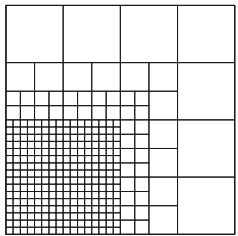
Fischer, Paul F. et al. "Spectral element methods for transitional flows in complex geometries." JSC. 2002.
Munch, Peter et al. "Efficient application of hanging-node constraints for matrix-free high-order FEM computations on CPU and GPU." ISC High Performance 2022.

Part 4:

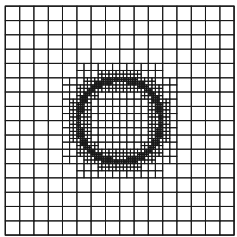
Results

Problem

Solve 3D Poisson problem with constant right-hand-side function and DBC on:



octant $L = 5$



sphere $L = 7$

configuration:

- PCG with 1 V-cycle GMG
- relative tolerance: 10^{-4}
- $p = 1$ and $p = 4$
- mixed precision (double, MG: float)
- coarse-grid solver: AMG via ML
- weight $2\times$ of cell with hanging nodes
- smoother: Chebyshev iteration (degree 3) around a point-Jacobi method

Metrics based on geometrical information

- estimating the workload imbalance of the smoother:
  - serial workload $\mathcal{W}_s$: sum of the number of cells on all levels
  - parallel workload $\mathcal{W}_p$: sum of the max. number of cells owned by any process on each level
  - parallel workload efficiency $\eta_w = \mathcal{W}_s/(\mathcal{W}_p \cdot p)$
- estimating the efficient of the inter-grid transfer operators:
  - vertical communication efficiency η_v: share of fine cells that have the same owning process as their corresponding parent coarse cell

Example:

	1 process		192 processes					
	LS	GC	LS			GC		
	$\mathcal{W}_s$	$\mathcal{W}_s$	$\mathcal{W}_p$	η_w	η_v	$\mathcal{W}_p$	η_w	η_v
octant ($L = 9$)	1.9e+7	1.9e+7	1.6e+5	64%	99%	1.0e+5	98%	38%
sphere ($L = 9$)	2.5e+6	2.5e+6	3.5e+4	36%	99%	1.5e+4	93%	84%

Serial and moderately parallel simulations

Timings on SuperMUC-NG (2 sockets, each with 24 cores of Intel Xeon Skylake, AVX-512):

	1 process				192 processes			
	LS		GC		LS		GC	
	$\#i$	$t[s]$	$\#i$	$t[s]$	$\#i$	$t[s]$	$\#i$	$t[s]$
octant ($L = 9$, $p = 1$)	4	2.2e+1	3	1.8e+1	4	2.3e-1	3	1.3e-1
sphere ($L = 9$, $p = 1$)	5	3.7e+0	4	4.3e+0	5	6.3e-2	4	3.5e-2

Observations:

- GC tends to need less iterations
- LS has a higher throughput per iteration in serial (less overhead due to HN)
- GC has a higher throughput per iteration in parallel (due to better workload)

Results indicate importance of good work balance on (expensive) fine levels!

Serial and moderately parallel simulations

Timings on SuperMUC-NG (2 sockets, each with 24 cores of Intel Xeon Skylake, AVX-512):

	1 process				192 processes			
	LS		GC		LS		GC	
	$\#i$	$t[s]$	$\#i$	$t[s]$	$\#i$	$t[s]$	$\#i$	$t[s]$
octant ($L = 9$, $p = 1$)	4	2.2e+1	3	1.8e+1	4	2.3e-1	3	1.3e-1
sphere ($L = 9$, $p = 1$)	5	3.7e+0	4	4.3e+0	5	6.3e-2	4	3.5e-2

Observations:

- GC tends to need less iterations
- LS has a higher throughput per iteration in serial (less overhead due to HN)
- GC has a higher throughput per iteration in parallel (due to better workload)

Results indicate importance of good work balance on (expensive) fine levels!

Serial and moderately parallel simulations

Timings on SuperMUC-NG (2 sockets, each with 24 cores of Intel Xeon Skylake, AVX-512):

	1 process				192 processes			
	LS		GC		LS		GC	
	$\#i$	$t[s]$	$\#i$	$t[s]$	$\#i$	$t[s]$	$\#i$	$t[s]$
octant ($L = 9$, $p = 1$)	4	2.2e+1	3	1.8e+1	4	2.3e-1	3	1.3e-1
sphere ($L = 9$, $p = 1$)	5	3.7e+0	4	4.3e+0	5	6.3e-2	4	3.5e-2

Observations:

- GC tends to need less iterations
- LS has a higher throughput per iteration in serial (less overhead due to HN)
- GC has a higher throughput per iteration in parallel (due to better workload)

Results indicate importance of good work balance on (expensive) fine levels!

Serial and moderately parallel simulations

Timings on SuperMUC-NG (2 sockets, each with 24 cores of Intel Xeon Skylake, AVX-512):

	1 process				192 processes			
	LS		GC		LS		GC	
	$\#i$	$t[s]$	$\#i$	$t[s]$	$\#i$	$t[s]$	$\#i$	$t[s]$
octant ($L = 9$, $p = 1$)	4	2.2e+1	3	1.8e+1	4	2.3e-1	3	1.3e-1
sphere ($L = 9$, $p = 1$)	5	3.7e+0	4	4.3e+0	5	6.3e-2	4	3.5e-2

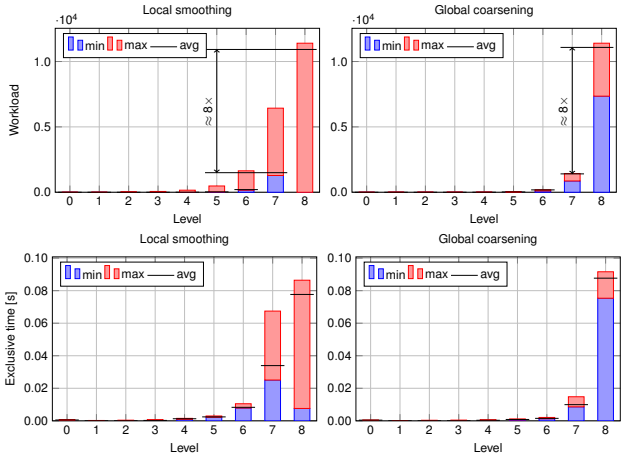
Observations:

- GC tends to need less iterations
- LS has a higher throughput per iteration in serial (less overhead due to HN)
- GC has a higher throughput per iteration in parallel (due to better workload)

Results indicate importance of good work balance on (expensive) fine levels!

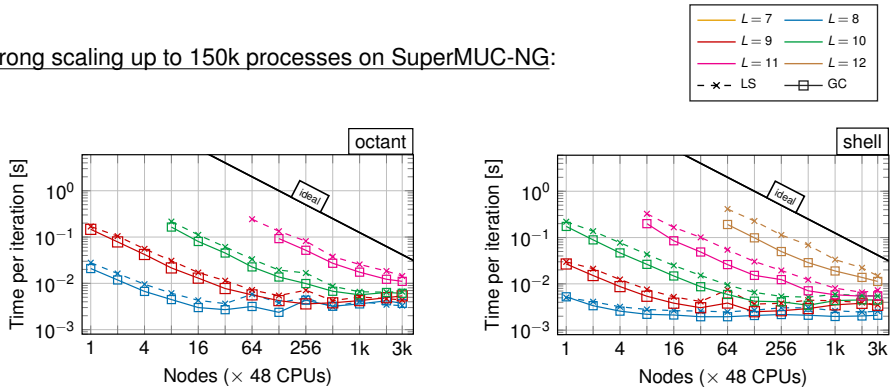
Serial and moderately parallel simulations (cont.)

Workload and execution time per level, e.g., for octant ($L = 8$):



Large-scale simulations

Strong scaling up to 150k processes on SuperMUC-NG:



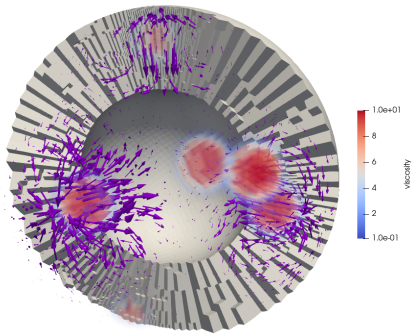
speedup (GC vs. LS): up to $\times 2.4$

Part 5:

Applications

Geosciences: variable-viscosity Stokes problem

nsinker benchmark



Solve:

$$-\nabla \cdot (2\eta\varepsilon(\mathbf{u})) + \nabla p = f$$
$$\nabla \cdot \mathbf{u} = 0$$

with a Q2-Q1 Taylor-Hood discretization of velocity $\vec{u}$, pressure p, and viscosity $\eta(x)$.

Geosciences: variable-viscosity Stokes problem (cont.)

The resulting linear system

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix} \qquad \text{with preconditioner:} \quad P^{-1} = \begin{pmatrix} A & B^T \\ 0 & -\hat{S} \end{pmatrix}^{-1}$$

- is solved with IDR(2)
- $\hat{S}$: mass matrix weighted by the viscosity
- inverses of A and $\hat{S}$ are each approximated by a single V-cycle of GMG
- 96 coarse cells
- mantle-convection code `ASPECT`[2]

[2] `https://aspect.geodynamics.org/`

Geosciences: variable-viscosity Stokes problem (cont.)

Preliminary results:

		global coarsening			local smoothing		
L	#DoFs/1e6	#it	solve/s	v-cycle/s	#it	solve/s	v-cycle/s
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
12	1441.4	28	22.17	0.141	29	50.85	0.436
13	2896.7	28	37.38	0.266	26	99.07	0.971
14	5861.9	29	77.94	0.515	26	193.19	2.060

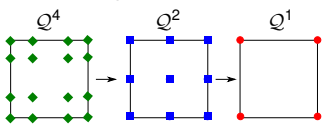
Observation:

- LS: workload efficiency $\sim 20\%$
- GC vs. LS 3x/4x faster (total/V-cycle)

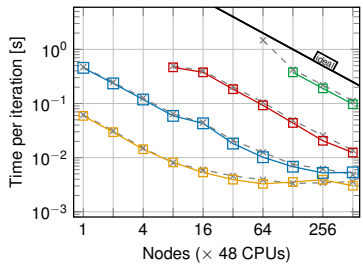
p-multigrid & hp-multigrid

General:

- applicable for high orders
- easy to implement based on GC by replacing the local transfer operator



... e.g., with bisection: $p_{l-1} = \lfloor p_l/2 \rfloor$



Observation:

- faster than pure GC: local transfer with good load balance

Part 6:

Conclusions & outlook

Conclusions & outlook

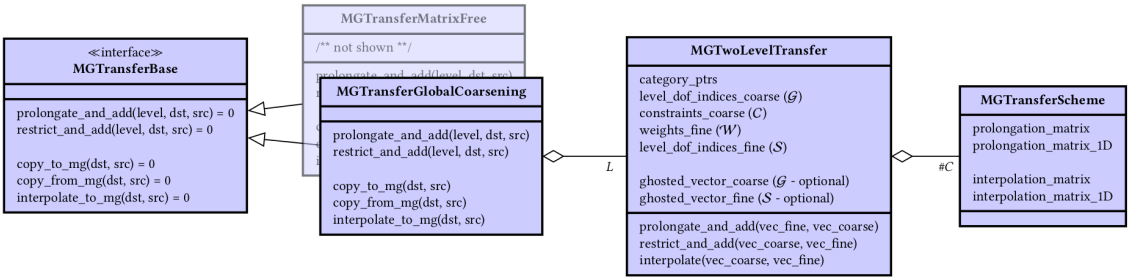
Conclusions

- global coarsening:
  - has a better parallel behavior than local smoothing
  - needs efficient hanging-node-constraint kernels
  - is easily applicable to p-multigrid, hp-adaptivity, DG, simplices, ...
  - repartitioned multigrid levels make the setup more difficult
- local smoothing: more smoothers, since nohanging-node constraints have to be applied
- useful metrics: workload efficiency and vertical efficiency
- freely available via `deal.II`

Outlook

- improve usability and setup routines
- apply to mixed meshes
- investigation of smoothers for GC

Implementation details

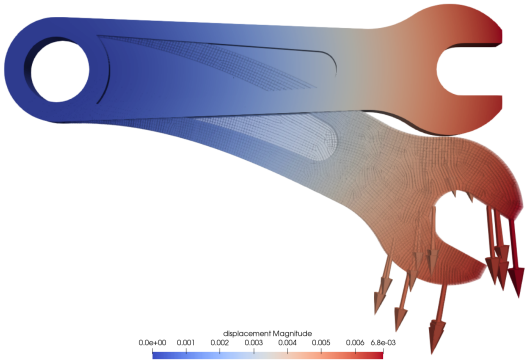


Update:

- `MGTransferGlobalCoarsening` has been generalized and is about to replace `MGTransferMatrixFree` (for local smoothing)
- `MGTransferGlobalCoarsening` now also supports non-nested meshes

Implementation details (cont.)

Example: non-nested multigrid



multigrid levels

Feder, M., Heltai, L., Kronbichler, M. and Munch, P., 2024. Matrix-free implementation of the non-nested multigrid method. arXiv preprint arXiv:2412.10910.

Part 5:

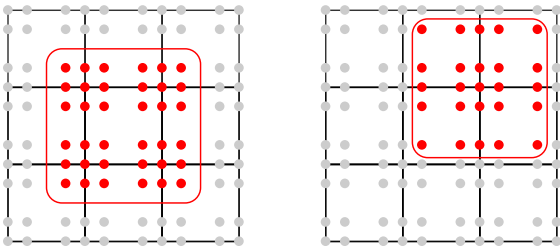
Additive Schwarz smoothers

Patch smoothers

- point-Jacobi preconditioner might not be robust, e.g., anisotropic meshes
- alternative: additive/multiplicative overlapping/non-overlapping Schwarz methods

$$A = \sum_e R_e^T A_e R_e \quad \leftrightarrow \quad \boxed{P^{-1} = \sum_b R_b^T A_b^{-1} R_b \quad \text{w.} \quad A_b = R_b A R_b^T}$$

problem: A not given → domain decomposition (geometrically motivated)

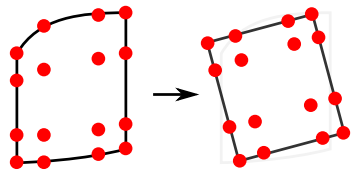


element-centered vs. vertex-star patch

possible domain solvers: fast-diagonalization method (Cartesian approx.), PCG, ...

Fast diagonalization method

Fast diagonalization method (FDM) → Cartesian approximation



$$A_b = M_2 \otimes M_1 \otimes K_0 + M_2 \otimes K_1 \otimes M_0 + K_2 \otimes M_1 \otimes M_0$$
$$= T_2 \otimes T_1 \otimes T_0 (\Lambda_2 \otimes I \otimes I + I \otimes \Lambda_1 \otimes I + I \otimes I \otimes \Lambda_0) T_2^T \otimes T_1^T \otimes T_0^T \quad \text{... with EVs/EWs } \Lambda / T$$

with explicit inverse application:

$$\boldsymbol{v} = A_b \boldsymbol{u} = \sum_b R_b^T T_2 \otimes T_1 \otimes T_0 (\Lambda_2 \otimes I \otimes I + I \otimes \Lambda_1 \otimes I + I \otimes I \otimes \Lambda_0)^{-1} T_2^T \otimes T_1^T \otimes T_0^T R_b \boldsymbol{u}$$

... can be expressed as matrix-free loop!

Implementation details

Relevant classes in `deal.II`:

- `SparseMatrixTools::restrict_to_full_matrices()`

  $A_P = R_P A R_P^T$

- `TensorProductMatrixCreator::create_laplace_tensor_product_matrix()`

  compute T_i, Λ_i via generalized eigendecomposition $K_i T_i = M_i T_i \Lambda_i$

- `TensorProductMatrixSymmetricSum`

  $\left(A_i^{\text{cart}}\right)^{-1} = T_1 \otimes T_0 (\Lambda_1 \otimes I + I \otimes \Lambda_0)^{-1} T_1^\top \otimes T_0^\top$

Performance optimizations of ASM described in: *Munch, P. and Kronbichler, M., 2024. Cache-optimized and low-overhead implementations of additive Schwarz methods for high-order FEM multigrid computations. The International Journal of High Performance Computing Applications, 38(3), pp.192-209.*

Literature

Lottes, J.W. and Fischer, P.F., 2005. Hybrid multigrid/Schwarz algorithms for the spectral element method. Journal of Scientific Computing, 24(1), pp.45-78.

Witte, J., Arndt, D. and Kanschat, G., 2021. Fast tensor product Schwarz smoothers for high-order discontinuous Galerkin methods. Computational Methods in Applied Mathematics, 21(3), pp.709-728.

Brubeck, P.D. and Farrell, P.E., 2021. A scalable and robust vertex-star relaxation for high-order FEM. arXiv preprint arXiv:2107.14758.

deal.II Workshop @ Durham University

Lecture 3: applications

Peter Munch[1]

[1] Institute of Mathematics, Technical University of Berlin, Germany

April 4, 2025

Table of content

Part 1:

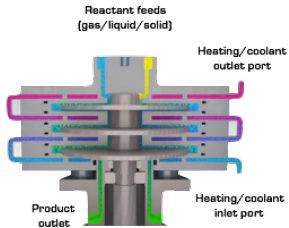
Lethe-CFD[1]

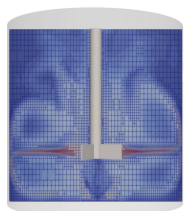
[1] L. Prieto Saavedra, P. Munch, B. Blais, Geometric multigrid methods for a matrix-free stabilized solver for the incompressible Navier-Stokes equations, 16th World Congress on Computational Mechanics (WCCM) / 4th Pan American Congress on Computational Mechanics (PANACM), Vancouver, British Columbia, Canada, July 21-26, 2024.

Context and motivation: process-intensified equipment

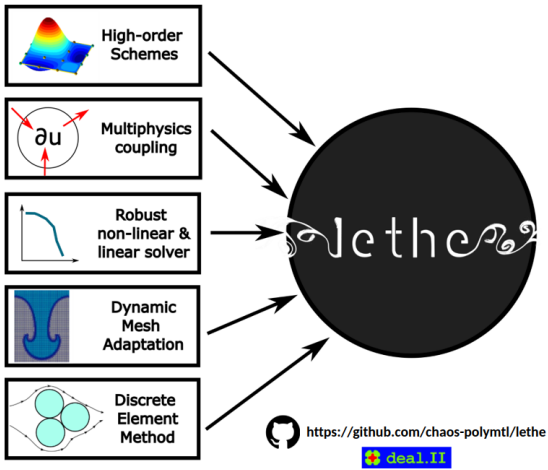
- Simulation of complex flow problems in chemical engineering is still a challenge: **rotating flows, transient flows** $\rightarrow \mathrm{Re} = [10^3, 10^5]$.
- Localized physical phenomena require very fine meshes $\rightarrow$ alternative: **locally refined meshes**.
- The algorithms for these very large simulations are memory bandwidth bounded, which led us to implement a **matrix-free solver**.
- Several challenges arise, among them: the development of a **matrix-free linear solver.**

Main goal: evaluate the efficiency of different **multigrid methods for locally refined meshes as preconditioners** for an incompressible stabilized Navier-Stokes matrix-free solver.





1. Open-source software: Lethe



- Based on the deal.II library (Arndt et al., 2023)
- Created by Blais et al. (2020)
- 35 contributors
- User guide (> 50 examples)
- Tests (> 350 tests)
- Used in 5 countries

2. Governing equations and discretization

Incompressible Navier-Stokes:

$$\nabla \cdot \boldsymbol{u} = 0 \text{ in } (0, T] \times \Omega$$

$$\partial_t \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \frac{1}{\rho}\nabla p - \nabla \cdot \mathbb{D}(\boldsymbol{u}) = \boldsymbol{f} \text{ in } (0, T] \times \Omega$$

- Continuous Galerkin method.
- Tensor elements: quadrilateral and hexahedra.
- Newton's method for the non-linearity.

2. Governing equations and discretization (Cont.)

Weak form:

$$F(\boldsymbol{u}, p) := (q, \nabla \cdot \boldsymbol{u})_\Omega + (\boldsymbol{v}, \partial_t \boldsymbol{u})_\Omega + (\boldsymbol{v}, (\boldsymbol{u} \cdot \nabla)\boldsymbol{u})_\Omega - (\nabla \cdot \boldsymbol{v}, p)_\Omega + \nu(\nabla \boldsymbol{v}, \nabla \boldsymbol{u})_\Omega + (\boldsymbol{n} \cdot \boldsymbol{v}, p)_{\partial\Omega} - (\boldsymbol{v}\boldsymbol{n}, \nu \nabla \boldsymbol{u})_{\partial\Omega} - (\boldsymbol{v}, \boldsymbol{f})_\Omega = 0$$

Linear system for each Newton iteration:

$$\underbrace{\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^T & 0 \end{bmatrix}}_{\boldsymbol{F}'} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{P} \end{bmatrix} = - \underbrace{\begin{bmatrix} \boldsymbol{R}_u \\ \boldsymbol{R}_p \end{bmatrix}}_{\boldsymbol{R}}$$

This discretization has three limitations:

- Ladyzhenskaya–Babuška–Brezzi condition.
- Instabilities when Re increases.
- Saddle-point problem.

→ **we overcome these by using stabilization techniques**

2. Governing equations and discretization (Cont.)

$$F + \underbrace{\sum_k (\tau \nabla \cdot \boldsymbol{q}, \mathrm{R_m})_{\Omega_k}}_{\text{PSPG term}} + \underbrace{\sum_k (\tau (\boldsymbol{u} \cdot \nabla)\boldsymbol{v}, \mathrm{R_m})_{\Omega_k}}_{\text{SUPG term}} = 0 \quad \mathrm{R_m} : \text{momentum strong residual}$$

$$\tau = \left[\left(\frac{1}{\Delta t} \right)^2 + \left(\frac{2\|\boldsymbol{u}\|}{h} \right)^2 + 9 \left(\frac{4\nu}{h^2} \right)^2 \right]^{-1/2} \qquad h = \left(\frac{6 h_k}{\pi} \right)^{\frac{1}{3}}$$

Modified linear system:

$$\underbrace{\begin{bmatrix} \widehat{A} & \widehat{B} \\ C & D \end{bmatrix}}_{F'} \begin{bmatrix} U \\ P \end{bmatrix} = - \underbrace{\begin{bmatrix} \widehat{R}_u \\ \widehat{R}_p \end{bmatrix}}_{R}$$

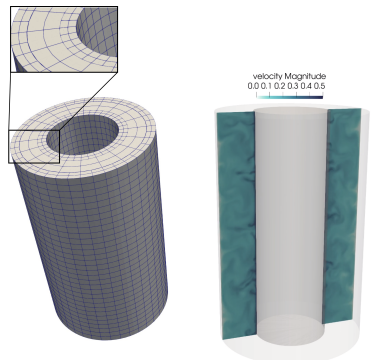
- We solve this problem in a fully-coupled monolithic way.
- At each nonlinear iteration, we solve the system using a preconditioned GMRES method.

→ **how to implement such a solver efficiently? Which preconditioner to use?**

4. Numerical tests: turbulent Taylor-Couette

Complex turbulent flow problem:

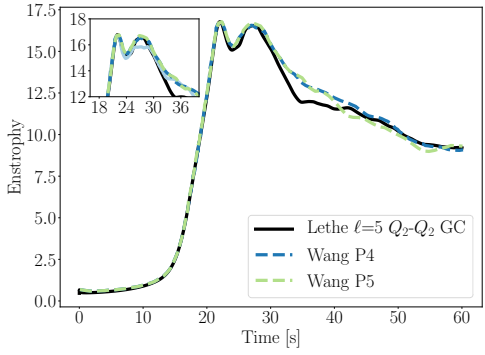
- Annular flow between two coaxial cylinders.
- Inner cylinder with a fixed angular velocity.
- Outer cylinder is static.
- Curved walls.
- Transient: BDF2, fixed CFL=1.
- $\mathrm{Re} = 4000$.
- $Q_p Q_p$ elements with $p = 1, 2, 3$.
- **Global static mesh refinement** ℓ with one additional refinement next to the walls.
- Simulation time: 60s.



$\rightarrow$ **allow us to study serial and large-scale parallel behavior of the preconditioners**

4. Numerical tests: turbulent Taylor-Couette - validation

The quality of the results is evaluated through the enstrophy and compared to the reference results from Wang and Jourdan (2021).



- For the next analysis, we focus on the measurements over the interval $(15, 30]$.

Part 2:

AM: simulation of melt-pool processes[2]

2

Part 3:

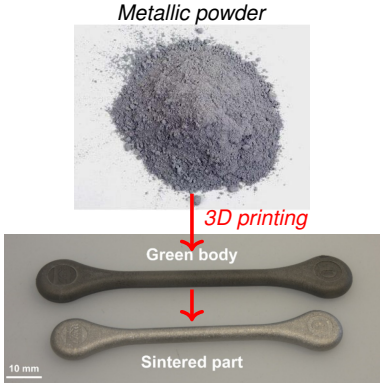
Solid-state sintering[3]

[3] P. Munch, V. Ivannikov, M. Kronbichler, Application and tailoring of matrix-free algorithms to many-particle solid-state-sintering phase-field implementations, SIAM Conference on Computational Science and Engineering (CSE23), Amsterdam, Netherlands, February 26 - March 3, 2023.

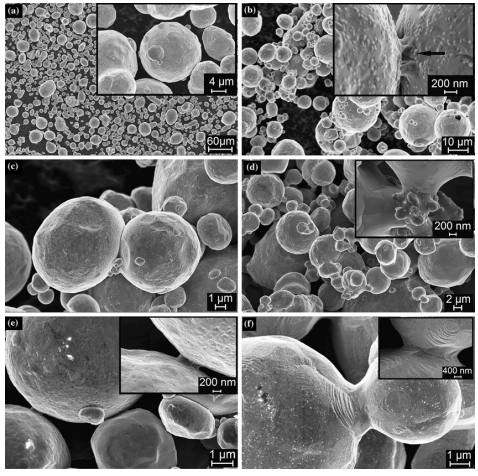
Part 1:

Motivation

Sintering of metallic powders



Sintering of metallic powders is a process in which particles *chemically bond* to themselves in order to form a coherent shape when exposed to a *high temperature*.

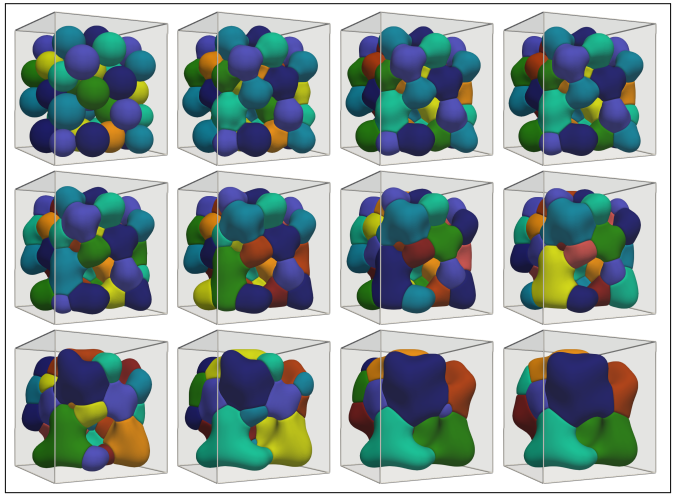


Typical microstructures of the sintered parts

Aman, Y., et. al., 2012. Pressure-less spark plasma sintering effect on non-conventional necking process during the initial stage of sintering of copper and alumina.

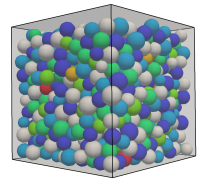
Simulation challenges of SSS processes

49 particles; colors: order parameters; 15,000s



challenges:

I. topological changes
II. size distribution
III. 2D vs. 3D
IV. thousands of particles



DEM vs. PF

Ivannikov, V., et. al., 2021. Capturing shrinkage and neck growth with phase field simulations of the solid state sintering. MSMSE.
Ivannikov, V., et. al., 2022. Coupling the discrete element method and solid state diffusion equations for modeling of metallic powders sintering. CPM.

Part 2:

A phase-field approach & discretization

Governing equations

Set of one Cahn-Hilliard (CH) equation and one Allen-Cahn (AC) equation for each particle:

$$\text{CH}: \quad \frac{\partial c}{\partial t}(\mathbf{x},t) = \nabla \cdot \left[M \nabla \frac{\delta F}{\delta c} \right],$$

$$\text{AC}: \quad \frac{\partial \eta_i}{\partial t}(\mathbf{x},t) = -L \frac{\delta F}{\delta \eta_i} \quad \text{for} \quad 1 \leq i \leq N.$$

local support of particles allows:
η_i of non-neighboring particles $\rightarrow$ order parameter
in experiments: 8-12 order parameters (#c)

with (conserved) concentration $0 \leq c \leq 1$, non-conserved $0 < \eta_i \leq 1$ within particle i, and

- free energy (based on Landau-type polynomial f):

$$F(c,\eta_i) = \int_\Omega \left[f(c,\eta_i) + \frac{1}{2}\kappa_c |\nabla c|^2 + \sum_i^N \frac{1}{2}\kappa_p |\nabla \eta_i|^2 \right] \mathrm{d}\Omega$$

- strong coupling
- $\mathcal{O}(N^2)$ complexity

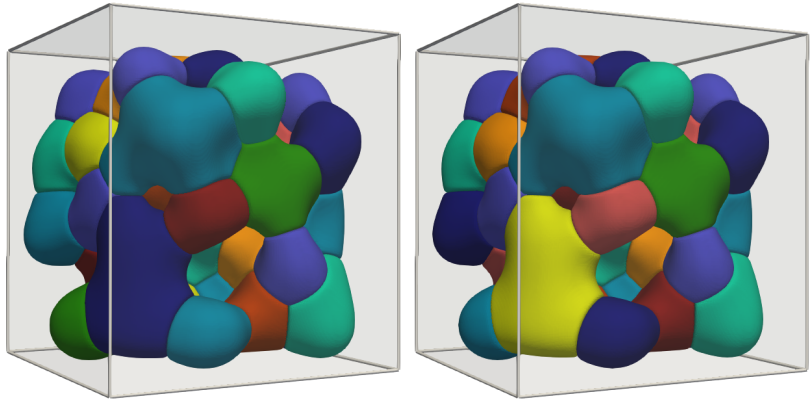
- scalar mobility with $\phi = c^3(10 - 15c + 6c^2)$:

$$M(c,\eta_i) = M_{\text{vo}}\phi + M_{\text{va}}(1-\phi) + M_{\text{s}}c^2(1-c)^2 + M_{\text{gb}} \sum_{i=1}^{N} \sum_{j \neq i}^{N} \eta_i \eta_j,$$

Order parameters

Example:

49 particles; colors: order parameters; iso-contour $c = 0.5$



topological changes $\rightarrow$ reassignment of particles/changing number of order parameters

Permann, C.J., Tonks, M.R., Fromm, B. and Gaston, D.R., 2016. Order parameter re-mapping algorithm for 3D phase field model of grain growth using FEM. CMS.

Weak form

The resulting weak form with chemical potential $\mu = \delta F / \delta c$:

$$\left(v_c,\ \frac{\partial c}{\partial t}\right) = -\left(\nabla v_c,\ M \nabla \mu\right),$$

$$\left(v_\mu,\ \mu\right) = \left(v_\mu,\ \frac{\partial f}{\partial c}\right) + \left(\nabla v_\mu,\ \kappa_c \nabla c\right),$$

$$\left(v_{\eta_i},\ \frac{\partial \eta_i}{\partial t}\right) = -\left(v_{\eta_i},\ L\frac{\partial f}{\partial \eta_i}\right) - \left(\nabla v_{\eta_i},\ L\kappa_p \nabla \eta_i\right) \quad \text{for} \quad 1 \leq i \leq N.$$

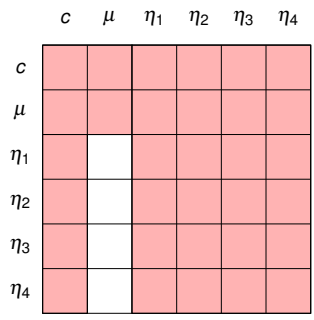
10k particles with 10k scalar DoFs per p. needs:

$\sim$ 10k$\times$10k$\times$14 $>$ 1 billion DoFs

Discretization & implementation details:

- adaptive mesh refinement (`deal.II/p4est`)
- BDF-2 with adaptive time steps
- linear FEM (w. multiple components)
- Newton solver w. Jacobian or JF (`NOX`)
- GMRES with coarse tolerances
- preconditioner lagging (`IFPACK`)
- one component corresponds to one block in a block vector $\rightarrow$ easy to remap grains and add/remove blocks

Jacobian & preconditioning



block sparsity pattern $\mathcal{O}(c^2)$

$\mathcal{O}(c^2)$ memory consumption motivates:

- matrix-free application of J
- Jacobian-free Newton-Krylov (JFNK) $\rightarrow$ residual

  ↯ coupling $\rightarrow \mathcal{O}(c^2)$ computation on q-point level

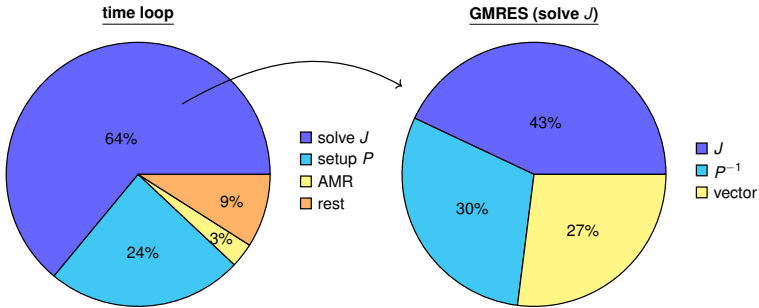
short comments on preconditioning:

* favorite: avg/max free energy

Brown, J., 2010. Efficient nonlinear solvers for nodal high-order finite elements in 3D. Journal of Scientific Computing, 45, pp.48-63.
Pernice, M. and Walker, H.F., 1998. NITSOL: A Newton iterative solver for nonlinear systems. SIAM Journal on Scientific Computing, 19(1), pp.302-318.
Brune, P.R., Knepley, M.G., Smith, B.F. and Tu, X., 2015. Composing scalable nonlinear algebraic solvers. SIAM Review, 57(4), pp.535-565.

Performance overview

49 particles; default settings; Intel Cascade Lake Xeon Gold 6230



Approx. 30% (=64%·43%) of simulation time is spent on application of Jacobian J.

Part 3:

Fast operator evaluation

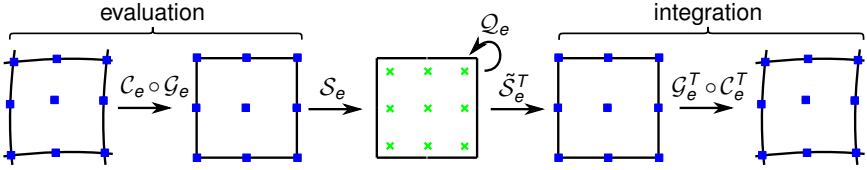
Generic fast operator evaluation

Fast operator evaluation as a loop over all cells:

$$\boldsymbol{v} = \mathcal{A}(\boldsymbol{u}) \quad \leftrightarrow \quad \boxed{\boldsymbol{v} = \sum_{e \in \{cells\}} \mathcal{G}_e^T \circ \mathcal{C}_e^T \circ \mathcal{S}_e^T \circ \mathcal{Q}_e \circ \mathcal{S}_e \circ \mathcal{C}_e \circ \mathcal{G}_e \circ \boldsymbol{u}}$$

- SIMD vectorization
- sum factorization
- ...

with:



... $\mathcal{G}_e$ gathering, $\mathcal{C}_e$ constraints, $\mathcal{S}_e$ interpolation, $\mathcal{Q}_e$ operation on quadrature points

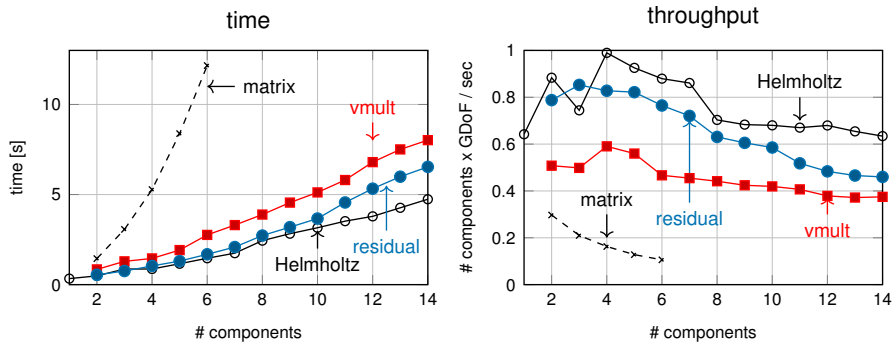
Extension to multiple components:

- $\mathcal{G}_e$, $\mathcal{C}_e$, $\mathcal{S}_e \rightarrow$ for each component
- $\mathcal{Q}_e \rightarrow$ combine components

Kronbichler, M. and Kormann, K., 2012. A generic interface for parallel cell-based finite element operator application. Computers & Fluids, 63, pp.135-147.
Kronbichler, M. and Kormann, K., 2019. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM TOMS, 45(3), pp.1-40.

Behavior for increasing number of components

uniform mesh; Intel Xeon Gold 6230



Observations:

- performance drops with increasing $\#c \rightarrow \mathcal{O}(c^2)$ complexity + caches/registers
- evaluation of residual 15% more expensive than that of Helmholtz operator
- evaluation of Jacobian 30% more expensive than that of residual
- overall JFNK the fastest approach

Behavior for increasing number of components (cont.)

Comparison of residual and Jacobian evaluation of generic sintering operator.

	sintering (residual)				sintering (vmult)			
#c	D/s	r/D	w/D	F/D	D/s	r/D	w/D	F/D
2	0.79	4.1	0.8	78	0.51	37.6	2.0	87
3	0.85	4.4	1.4	75	0.50	29.6	2.3	87
4	0.83	4.8	1.9	73	0.59	25.6	2.5	85
5	0.82	5.0	2.2	73	0.56	23.3	2.7	89
6	0.76	5.2	2.4	73	0.47	21.7	2.8	90
7	0.72	5.2	2.6	73	0.45	20.5	2.8	91
8	0.63	5.3	2.6	74	0.44	19.7	2.9	95
9	0.61	5.3	2.7	74	0.42	18.9	2.9	96
10	0.59	5.3	2.7	75	0.42	18.3	3.0	97
11	0.52	5.3	2.8	76	0.41	17.8	3.0	98
12	0.48	5.2	2.8	**77**	0.38	**17.4**	3.0	**99**
13	0.47	5.2	2.8	77	0.37	17.0	3.1	100
14	0.46	5.2	2.9	78	0.37	16.7	3.1	101

Helmholtz operator: 52 FLOPs/DoF

D/s: throughput in [GDoFs/sec]
r/D: read data per DoF in [Double/DoF]
w/D: written data per DoF in [Double/DoF]
F/D: work [FLOPs/DoF]

Opt. I & II: constant expressions & apply mobility/free energy

Algorithm 1: Cell loop considering *all* vector blocks.

```
1 if n_blocks = n_blocks^static then
2     for cell ∈ cells do
3         for b = 1 to n_blocks do
4             read from block b of source vector
5         perform cell integral → A_e
6         for b = 1 to n_blocks do
7             write to block b of destination vector
8 else
9     not shown
```

- fixed loop bounds

Algorithmic reformulations w. goal: $\mathcal{O}(c^2) \to \mathcal{O}(c)$.
E.g.:

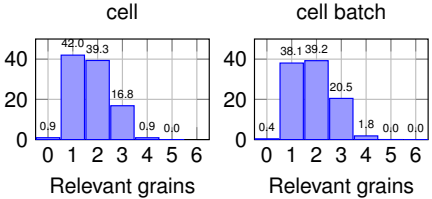
$$\sum_{j,i\neq j} (\eta_i \eta_j) u_j = \eta_i \sum_{j,i\neq j} \eta_j u_j = \eta_i(\alpha - \eta_i u_i)$$

with $\alpha = \sum_i \eta_i u_i$.

- precompute factors: $\sum_i \eta_i^2, \sum_i \eta_i^3$,
- exploit symmetry: $\sum_{i=1}^{g} \sum_{j=1, i\neq j}^{g} \eta_i \eta_j = 2 \sum_{i=1}^{g} \sum_{j=1}^{i-1} \eta_i \eta_j$.
- exploit tensor-product structure: $(\mathbf{n}_a \otimes \mathbf{n}_b)\mathbf{v} = \mathbf{n}_a(\mathbf{n}_b \cdot \mathbf{v})$,

Optimization III: Working on locally relevant particles

- motivation: local support of particles



- work on cell level on locally relevant particles, by modifying gather/scatter to only work on relevant blocks
- parameter: cut-off tolerance (e.g., 10^{-5})

Algorithm 2: Cell loop considering only vector blocks *relevant for the current cell.*

```
1  for cell ∈ cells do
2      blocks ← relevant blocks of cell
3      for b ∈ blocks do
4          read from block b of source vector
5      if |blocks| = n^static_blocks then
6          perform cell integral → A_e
7      else
8          not shown
9      for b ∈ blocks do
10         write to block b of destination vector
```

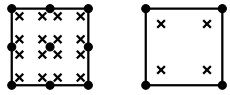
- speedup $> 2\times$
- redundant computations/operations and high memory consumption ↯
- alternative: sparse block vectors*, "*hp*-adapt." → challenges: sparsity pattern, prec., ...

Kim, S.G., Kim, D.I., Kim, W.T. and Park, Y.B., 2006. Computer simulations of two-dimensional and three-dimensional ideal grain growth. Physical Review.

Davydov, D. and Kronbichler, M., 2020. Algorithms and data structures for matrix-free finite element operators with MPI-parallel sparse multi-vectors. ACM TOPC.

Optimization IV & V & VI

Idea: Process 2^d child cells together.
Example:



Observation: high L1-cache misses.

Solution:

Algorithm 3: Layer-by-layer approach*.

```
1 interpolate in x-direction
2 interpolate in y-direction
3 interpolate in z-direction
4 compute derivative in z-direction
5 foreach xy-plane do
6     compute derivative in x-direction
7     compute derivative in y-direction
8     process quadrature points
```

Generic implementation: $J(\mathbf{u}_{\text{lin}})\mathbf{p} \approx J'(\mathbf{u}_{\text{lin}})\mathbf{p} = \frac{\mathcal{F}(\mathbf{u}_{\text{lin}}+\beta\mathbf{p})-\mathcal{F}(\mathbf{u}_{\text{lin}})}{\beta}$,

Cache-efficient implementation: interleave cell loop $v \leftarrow \mathcal{F}(u^{\text{lin}})$ with vector updates*:

$$\underline{\textbf{pre:}}\ u_i^{\text{lin}} \leftarrow u_i^{\text{lin}} + \beta p_i, \quad \underline{\textbf{post:}} \begin{cases} v_i \leftarrow (v_i + r_i^{\text{lin}})/\beta \\ u_i^{\text{lin}} \leftarrow u_i^{\text{lin}} - \beta p_i, \end{cases}$$

Kronbichler, M. and Kormann, K., 2019. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM TOMS, 45(3)
Kronbichler, M., Sashko, D. and Munch, P., 2022. Enhancing data locality of the conjugate gradient method for high-order matrix-free finite-element

Part 4:

Outlook & conclusions

Outlook & conclusions

Outlook: add more involved physics:

- tensorial mobility $\mathbf{M} \in \mathbb{R}^{N_{\mathrm{OP}} \times N_{\mathrm{OP}}}$

$$\mathbf{M}(c, \eta_i) = M_{\mathrm{vo}} \phi \mathbf{I} + M_{\mathrm{va}} (1-\phi) \mathbf{I} + M_{\mathrm{s}} c^2 (1-c)^2 \mathbf{T}_s + M_{\mathrm{gb}} \sum_{i=1}^{N} \sum_{j \neq i}^{N} \eta_i \eta_j \mathbf{T}_{ij}$$

$$\text{with } \mathbf{T}_s = \mathbf{I} - \mathbf{n}_s \otimes \mathbf{n}_s,\ \mathbf{T}_{ij} = \mathbf{I} - \mathbf{n}_{ij} \otimes \mathbf{n}_{ij},\ \mathbf{n}_s = \frac{\nabla c}{|\nabla c|},\ \mathbf{n}_{ij} = \frac{\nabla \eta_i - \nabla \eta_j}{|\nabla \eta_i - \nabla \eta_j|}.$$

- advection term $\rightarrow$ rigid body motion

$$\frac{\partial c}{\partial t}(\mathbf{x}, t) = \nabla \cdot \left[M \nabla \frac{\delta F}{\delta c} \right] - \nabla \cdot \left(c \sum \mathbf{v}_i \right) \qquad \frac{\partial \eta_i}{\partial t}(\mathbf{x}, t) = -L \frac{\delta F}{\delta \eta_i} - \nabla \cdot (\eta_i \mathbf{v}_i).$$

preliminary results: same tricks regarding fast operator evaluation and preconditioning

Outlook & conclusions (cont.)

Outlook (cont.):

- comparison with experiments → magnesium implants → GCS: 48 million core h
- generalize functionalities for block systems and integrate them in `deal.II`
- physics-based preconditioner for Cahn–Hilliard block

Implementation is freely available via `hpsint`[1], an open-source project for sintering applications targeting HPC systems and based on `deal.II`.

Other additive-manufacturing projects based on deal.II's matrix-free infrastructure:

- Meier, C., Fuchs, S.L., Much, N., Nitzler, J., Penny, R.W., Praegla, P.M., Proell, S.D., Sun, Y., Weissbach, R., Schreter, M. and Hodge, N.E., 2021. Physics-based modeling and predictive simulation of powder bed fusion additive manufacturing across length scales. GAMM.
- Proell, S.D., Munch, P., Wall, W.A. and Meier, C., 2023. A highly efficient computational framework for fast scan-resolved simulations of metal additive manufacturing processes on the scale of real parts. arXiv.
- DeWitt, S., Rudraraju, S., Montiel, D., Andrews, W.B. and Thornton, K., 2020. PRISMS-PF: A general framework for phase-field modeling with a matrix-free finite element method. npj CM.

[1] `https://github.com/hpsint/hpsint`

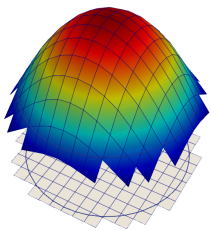
Part 4:

Cut Galerkin difference methods[4]

[4] P. Munch, Cut Galerkin difference methods and more, 11th deal.II Users and Developers Workshop, Fort Collins, Colorado, USA, August 12 - 16, 2024.

deal.II at Uppsala University (cont.)

① CutFEM/DG → immersed domains



Challenge: small cuts → stabilization

② Time stepping

Time-step restriction for Lagrange elements:

$$\Delta t = \mathcal{O}\left(\frac{1}{p^{\alpha}}\right) \quad \text{with } \alpha > 1.$$

No time-step restriction for:

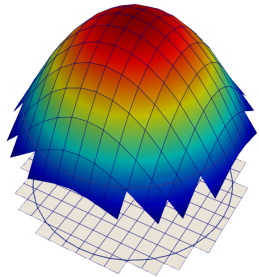
- Hermite elements
- Galerkin difference methods

①+② CutFEM + Hermite elements/Galerkin difference methods?

Part 1:

CutFEM

Problem statement



Example: solve Poisson problem on Ω, using a background mesh

$$a_h(u_h, v_h) = L_h(v_h), \quad \forall v_h \in V_\Omega^h,$$

where

$$a_h(u_h, v_h) = (\nabla u_h, \nabla v_h)_\Omega - (\partial_n u_h, v_h)_\Gamma - (u_h, \partial_n v_h)_\Gamma + \left(\frac{\gamma_D}{h} u_h, v_h\right)_\Gamma,$$

$$L_h(v_h) = (f, v)_\Omega + \left(u_D, \frac{\gamma_D}{h} v_h - \partial_n v_h\right)_\Gamma.$$

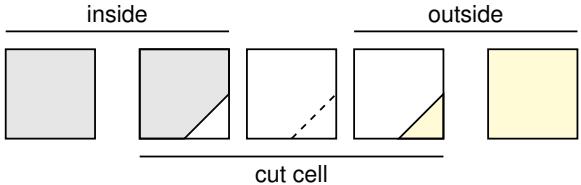
- stabilization for small cut cells: $A_h(u_h, v_h) := A_h(u_h, v_h) + \gamma_A h^{-2} j(v, u_h)$ with, e.g., :

$$j(v, u_h) = \sum_{F \in \mathcal{F}_\Gamma} \sum_{k=1}^{p} h^{2k+1} \left\langle \left[\partial_n^k v\right], \left[\partial_n^k u_h\right] \right\rangle \qquad \rightarrow \text{ ghost penalty}$$

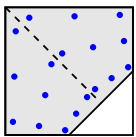
- extension: two domains, two-phase flow $\rightarrow$ moving interface

Types of integration

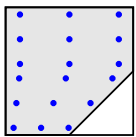
- cell can be inside/cut/outside



- similar for faces
- options for definition of quadrature rules:



Carraro & Wetterauer, 2015



Saye, 2015

Workflow in deal.II

full example: step-85

- define level-set field

```
Functions::SignedDistance::Sphere<dim> signed_distance_sphere;
```

- categorize cells

```
NonMatching::MeshClassifier<dim> mesh_classifier(/*...*/);
mesh_classifier.reclassify();
```

- perform different integrals depending on the category and position of cut

```
NonMatching::FEValues<dim> nm_fe_values(/*...*/, mesh_classifier,/*...*/ , ls);
for (const auto &cell : dof_handler.active_cell_iterators())
  {
    non_matching_fe_values.reinit(cell);
    if (const auto fe_values = non_matching_fe_values.get_inside_fe_values())
      {
        // continue as normal
      }
  }
```

- similar on surface (NM::FEImmersedSurfaceValues) and faces (NM::FEInterfaceValues).

Outlook

Alternatively, integration on a set of unstructured quadrature rules can be performed in a matrix-free way, using FEPointEvaluation[1]

```
FEPointEvaluation<dim> phi(/*...*/);

phi.reinit(/*...*/);

phi.evaluate(buffer, EvaluationFlags::value);

for(const auto q : phi.quadrature_point_indices ())
  phi.submit_value(phi.get_value(q), q);

phi.integrate(buffer, EvaluationFlags::value);
```

Notes:

- can exploit tensor-product structure of shape functions to reach high performance
- to be used together with DoFCellAccessor or FEEvaluation
- example: step-87 (sharp interface method)
- major challenge: preconditioning

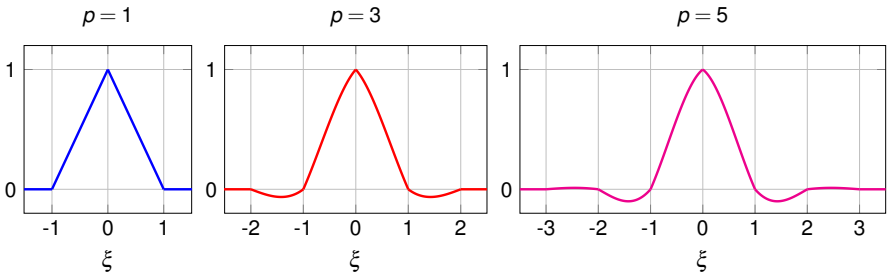
[1] Bergbauer, Munch, Wall, Kronbichler, 2024, High-performance matrix-free unfitted finite element operator evaluation, arxiv

Part 3:

Galerkin difference methods (GDM)

Galerkin difference methods in a nutshell

Galerkin difference methods: a type of FEM with shape functions spanning over more cells:



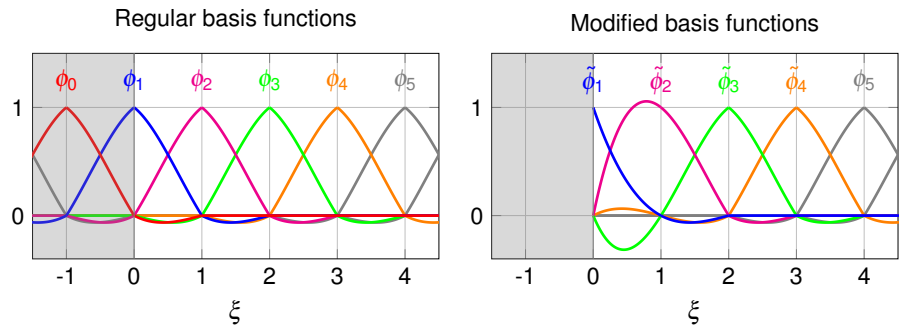
▷ *Lagrange functions associated with continuous piecewise polynomials*

Banks and Hagstrom (2016) showed “for first-order systems no significant CFL penalty”.

Advantages: no additional DoFs, same stencil for each internal point (similar to FDM)

Galerkin difference methods in a nutshell (cont.)

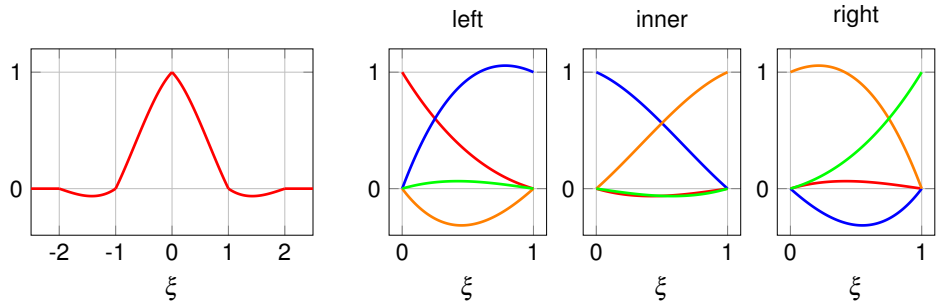
- at boundary: ① express basis functions that are outside of the computational domain as linear combinations of internal basis functions, ② eliminate from system.



- extension to higher dimensions: tensor product

Implementation details in deal.II

① consider all non-zero basis functions in a cell → "elements"



basis-function-centric ⟷ cell-centric view (p unique "elements" in 1D)

② custom element:
hp::FECollection with p^d entries → FE_Q_Base → AnisotropicPolynomials → Polynomial

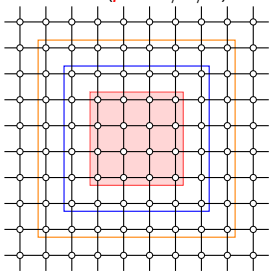
Implementation details in deal.II (cont.)

```
template <int dim>
class FE_GDM : public FE_Q_Base<dim>
{
public:
  FE_GDM(const ScalarPolynomialsBase<dim> &poly)
    : FE_Q_Base<dim>(poly, create_data(poly.n()), std::vector<bool>(1, false))
  {}

  std::string get_name() const override {/*...*/}

  std::unique_ptr<FiniteElement<dim>> clone() const override {/*...*/}

private:
  static FiniteElementData<dim>
  create_data(const unsigned int n)
  {
    std::vector<unsigned int> dofs_per_object(dim + 1);
    dofs_per_object[dim] = n;
    FiniteElementData<dim> fe_data(dofs_per_object, 1, 0 /*not relevant*/);
    return fe_data;
  }
};
```

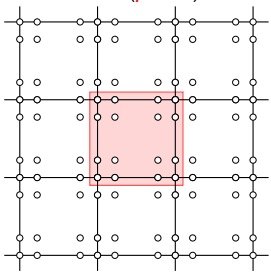
Implementation details in deal.II (cont.)

③ custom gathering

GDM ($p = 3/5/7$)



FEM ($p = 3$)



all deal.II functions that access global matrices/vectors had to be rewritten

Implementation details in deal.II (cont.)

4. distributed Cartesian mesh
   - lexicographic ordering of cells and DoFs allows the conversion

$$f(i,j) = i + j * N_0 \;\leftrightarrow\; g(i) = \begin{pmatrix} i \% N_0 \\ i / N_0 \end{pmatrix}$$

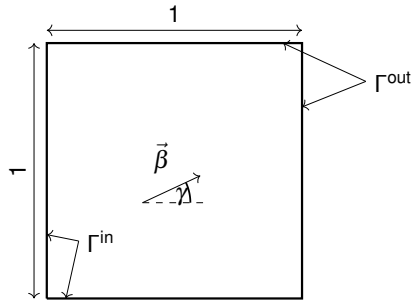
     and, as a consequence, to easily determine neighbors and patch indices
   - layer-wise partitioning with arbitrary number of ghost layers

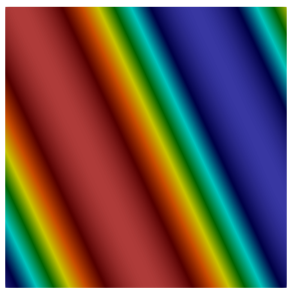
Experiment

Solve advection equation with RK4 (CFL=$||\beta||\Delta t/h$=0.4):

$$\dot{u} + \beta \cdot \nabla u = 0 \qquad \text{in } \Omega \times (0, T)$$

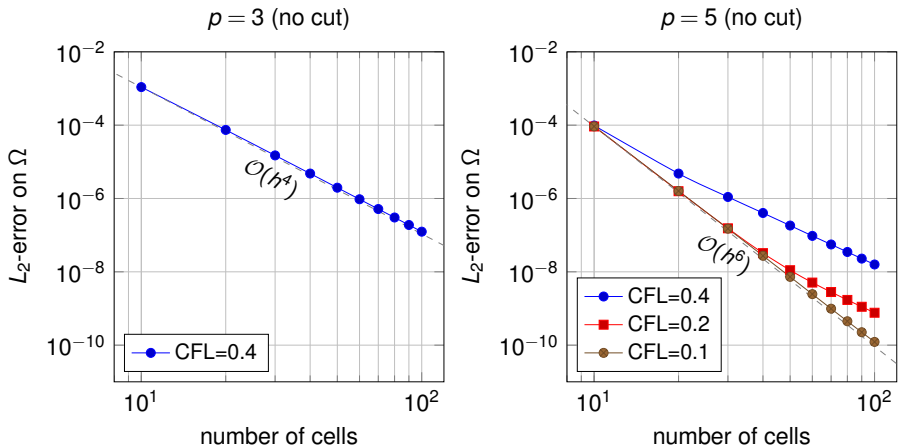
with the setup:





... with prescribed boundary and initial condition.

Experiment (cont.)



Part 4:

CutGDM

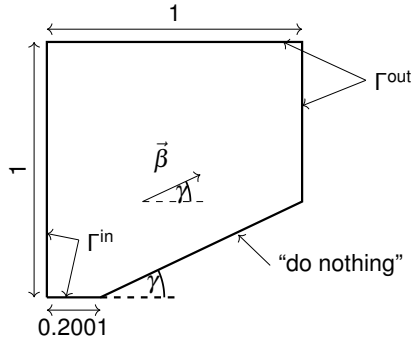
Extension to CutGDM

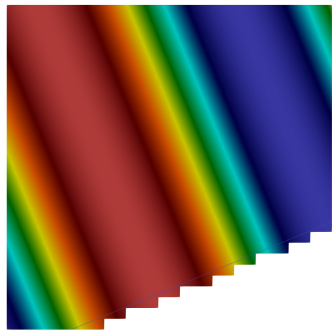
- conceptually easy since we evaluate the shape functions cell by cell
- however, missing features:
  - several classes in NonMatching namespace were not working for hp
  - several methods were only working for DoFCellAccessor but not for CellAccessor
- we use ghost penalty, but only consider gradients:

$$j(v, u_h) = \sum_{F \in \mathcal{F}_\Gamma} h^{2k+1} \langle [\partial_n v], [\partial_n u_h] \rangle$$

Experiment

Same setup as before, but with ramp[2] (and small cuts):

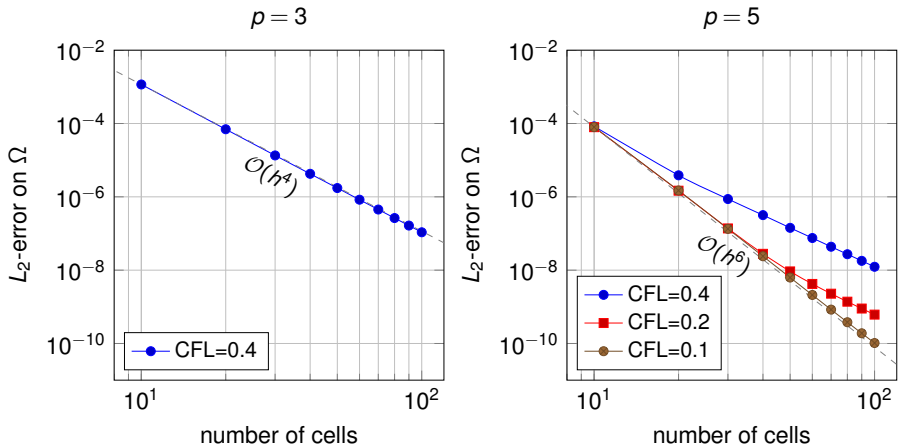




- note: similar results for non-parallel ramp

[2] C Engwer, S May, A Nüßing, F Streitbürger, 2020, A stabilized DG cut cell method for discretizing the linear transport equation, SISC.

Experiment (cont.)



- Future work: stability analysis, A-priori error estimation, ...

Part 5:

Computational plasma physics[5]

[5] P. Munch, High-dimensional finite-element computations with deal.II and hyper.deal, Chair of Optimal Control, Technical University of Munich, Munich, Germany, March 30, 2023.

Part 1:

Introduction

Computational plasma physics

Goal

Describe the evolution of a plasma and its interaction in magnetic fields. A field of application is fusion energy research, in which the plasma in fusion reactors (e.g., tokamak and stellarator) is investigated.

Mathematical descriptions:

1. particle model: description of the motion of each particle ▷ n-body problem; PIC

$$\frac{\partial^2 x_i}{\partial t^2} = \frac{q_i}{m_i}\left(\vec{E}(t,\vec{x}) + \vec{v} \times \vec{B}(t,\vec{x})\right)$$

2. kinetic model: described by a distribution function $f(t,\vec{x},\vec{v})$, which evolves according to the Vlasov equation coupled to a system of Maxwell's equations
3. fluid model: e.g., magnetohydrodynamics, treats the plasma as a single fluid (combination of Maxwell's equations and the Navier–Stokes equations; MHD)

Motivation

on 5.12/13.12.2022

Figure: Announcement of Department of Energy (DOE) about “first controlled fusion experiment”

Vlasov–Maxwell equations

Vlasov equation: with a single particle species with charge q and mass m

$$\boxed{\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} f + \vec{a}(t, f, \vec{x}, \vec{v}) \cdot \nabla_{\vec{v}} f = 0} \qquad \vec{a}(t, f, \vec{x}, \vec{v}) = \frac{q}{m} \left(\vec{E}(t, \vec{x}) + \vec{v} \times \vec{B}(t, \vec{x}) \right)$$

which is coupled to the Maxwell's equations for the self-consistent fields.

Vlasov–Maxwell equations

Vlasov equation: with a single particle species with charge q and mass m

$$\boxed{\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} f + \vec{a}(t, f, \vec{x}, \vec{v}) \cdot \nabla_{\vec{v}} f = 0} \qquad \vec{a}(t, f, \vec{x}, \vec{v}) = \frac{q}{m} \left(\vec{E}(t, \vec{x}) + \vec{v} \times \vec{B}(t, \vec{x}) \right)$$

which is coupled to the Maxwell's equations for the self-consistent fields.

Alternative notation:

$$\boxed{\frac{\partial f}{\partial t} + \begin{pmatrix} \vec{v} \\ \vec{a}(t, f, \vec{x}, \vec{v}) \end{pmatrix} \cdot \begin{pmatrix} \nabla_{\vec{x}} \\ \nabla_{\vec{v}} \end{pmatrix} f = 0}$$

... nonlinear, high-dimensional, hyperbolic PDE: $\boxed{f(t, \vec{x}, \vec{v}) : \mathbb{R}^{d_{\vec{x}} + d_{\vec{v}} + 1} \to \mathbb{R}}$

Vlasov–Maxwell equations

Vlasov equation: with a single particle species with charge q and mass m

$$\boxed{\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} f + \vec{a}(t, f, \vec{x}, \vec{v}) \cdot \nabla_{\vec{v}} f = 0} \qquad \vec{a}(t, f, \vec{x}, \vec{v}) = \frac{q}{m} \left(\vec{E}(t, \vec{x}) + \vec{v} \times \vec{B}(t, \vec{x}) \right)$$

which is coupled to the Maxwell's equations for the self-consistent fields.

Solution strategies:

- finite element method
- sparse-grid methods
- semi-Lagrangian methods
- Monte-Carlo approaches
- low-rank solvers
- (gyrokinetic equations)

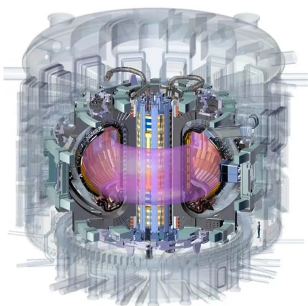
Pollinger, et. al., 2022. A mass-conserving sparse grid combination technique with biorthogonal hierarchical basis functions for kinetic simulations. arXiv.
Kormann, K., Reuter, K. and Rampp, M., 2019. A massively parallel semi-Lagrangian solver for the six-dimensional Vlasov–Poisson equation. IJHPCA.

Vlasov–Maxwell equations

Vlasov equation: with a single particle species with charge q and mass m

$$\boxed{\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} f + \vec{a}(t, f, \vec{x}, \vec{v}) \cdot \nabla_{\vec{v}} f = 0} \qquad \vec{a}(t, f, \vec{x}, \vec{v}) = \frac{q}{m} \left(\vec{E}(t, \vec{x}) + \vec{v} \times \vec{B}(t, \vec{x}) \right)$$

which is coupled to the Maxwell's equations for the self-consistent fields.



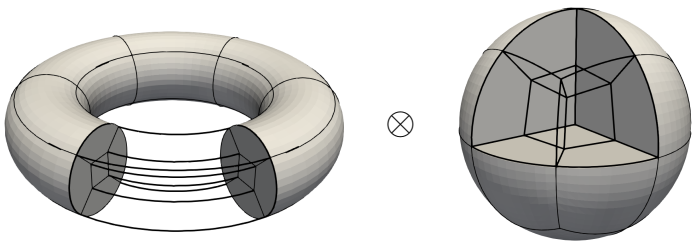
schematic of ITER (source: https://scitechdaily.com/)

Vlasov–Maxwell equations

Vlasov equation: with a single particle species with charge q and mass m

$$\boxed{\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} f + \vec{a}(t, f, \vec{x}, \vec{v}) \cdot \nabla_{\vec{v}} f = 0} \qquad \vec{a}(t, f, \vec{x}, \vec{v}) = \frac{q}{m} \left(\vec{E}(t, \vec{x}) + \vec{v} \times \vec{B}(t, \vec{x}) \right)$$

which is coupled to the Maxwell's equations for the self-consistent fields.



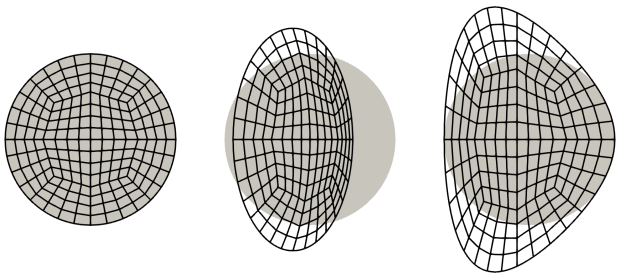
$\vec{x}$-space $\qquad$ $\vec{v}$-space

Vlasov–Maxwell equations

Vlasov equation: with a single particle species with charge q and mass m

$$\boxed{\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} f + \vec{a}(t, f, \vec{x}, \vec{v}) \cdot \nabla_{\vec{v}} f = 0} \qquad \vec{a}(t, f, \vec{x}, \vec{v}) = \frac{q}{m} \left(\vec{E}(t, \vec{x}) + \vec{v} \times \vec{B}(t, \vec{x}) \right)$$

which is coupled to the Maxwell's equations for the self-consistent fields.



Shafranov shift Czarny et. al.

Part 2:

Model problem

Model problem

High-dimensional transport equation with constant velocity:

- short-hand notation:

$$\frac{\partial f}{\partial t} + \vec{a} \cdot \nabla f = 0 \quad \text{with} \quad \vec{a} := \begin{pmatrix} \vec{v}_{\vec{x}} \\ \vec{v}_{\vec{v}} \end{pmatrix} \quad \text{and} \quad \nabla := \begin{pmatrix} \nabla_{\vec{x}} \\ \nabla_{\vec{v}} \end{pmatrix}$$

Model problem

High-dimensional transport equation with constant velocity:

- short-hand notation:

$$\frac{\partial f}{\partial t} + \vec{a} \cdot \nabla f = 0 \quad \text{with} \quad \vec{a} := \begin{pmatrix} \vec{v}_{\vec{x}} \\ \vec{v}_{\vec{v}} \end{pmatrix} \quad \text{and} \quad \nabla := \begin{pmatrix} \nabla_{\vec{x}} \\ \nabla_{\vec{v}} \end{pmatrix}$$

- skew-symmetric discontinuous Galerkin discretization:

$$\left(g, \frac{\partial f}{\partial t} \right)_{\Omega^{(e)}} = \left(g, -\beta \vec{a} \nabla f \right)_{\Omega^{(e)}} + \left(\nabla g, \vec{a}(1-\beta) f \right)_{\Omega^{(e)}} + \left\langle g, \vec{n} \cdot (\vec{a} f)^* - \beta f^- (\vec{n} \cdot \vec{a}) \right\rangle_{\Gamma^{(e)}}$$

with test function g, numerical flux α ($\alpha = 0$: central; $\alpha = 1$: upwind flux)

$$(\vec{a} f)^* = \frac{1}{2} \left((f^- + f^+)(\vec{n} \cdot \vec{a}) + (f^- - f^+) |\vec{n} \cdot \vec{a}| \right) \cdot \alpha \Big) .$$

and $\beta \rightarrow$ flux formulation ($\beta = \frac{1}{2}$: skew-symmetric; $\beta = 0$: conservative).

Model problem

High-dimensional transport equation with constant velocity:

- short-hand notation:

$$\frac{\partial f}{\partial t} + \vec{a} \cdot \nabla f = 0 \quad \text{with} \quad \vec{a} := \begin{pmatrix} \vec{v}_{\vec{x}} \\ \vec{v}_{\vec{v}} \end{pmatrix} \quad \text{and} \quad \nabla := \begin{pmatrix} \nabla_{\vec{x}} \\ \nabla_{\vec{v}} \end{pmatrix}$$

- semi-discrete system:

$$\mathcal{M} \frac{\partial \vec{f}}{\partial t} = \mathcal{A}(\vec{f}, t) \qquad \leftrightarrow \qquad \frac{\partial \vec{f}}{\partial t} = \mathcal{M}^{-1} \mathcal{A}(\vec{f}, t).$$

- time integration: low-storage Runge–Kutta methods $\rightarrow$ efficient $\mathcal{M}^{-1}$?
- observation: for nodal DG with nodes in the Gauss–Lobatto points:
  - Gauss–Legendre quadrature $\rightarrow$ block diagonal
  - Gauss–Lobatto quadrature (collocation) $\rightarrow$ diagonal

Part 3:

Software

deal.II

- deal.II[1]: mathematical software for finite-element analysis, written in C++
- origin in Heidelberg 1998: Wolfgang Bangerth, Ralf Hartmann, Guido Kanschat
- 275 contributors + principal developer team with 11 active members
- approx. 1,800 publications (on and with deal.II)
- freely available under LGPL 2.1 license
- annual releases; current release: 9.4; next release: May/June '23
- features comprise: matrix-free implementations, parallelization (MPI, threading via TBB & Taskflow, SIMD, GPU support), discontinuous Galerkin methods, AMR via p4est, wrappers for PETSc and Trilinos, particles, *hp*-adaptivity, simplex and mixed meshes, preCICE adapter, ...
- only: 1D–3D

[1] successor of DEAL: **D**ifferential **E**quations **A**nalysis **L**ibrary

Matrix-free operator evaluation

Matrix-free operator evaluation as a loop over all cells and faces:

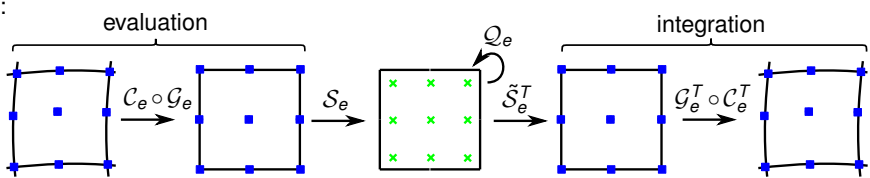
$$\boldsymbol{v} = \mathcal{A}(\boldsymbol{u}) \quad \leftrightarrow \quad \boxed{\boldsymbol{v} = \sum_{e \in \{cells\}} \mathcal{G}_e^T \circ \mathcal{C}_e^T \circ \mathcal{S}_e^T \circ \mathcal{Q}_e \circ \mathcal{S}_e \circ \mathcal{C}_e \circ \mathcal{G}_e \circ \boldsymbol{u} + \sum_{f_1, f_2 \in \{faces\}} \ldots + \sum_{f \in \{faces\}} \ldots}$$

Matrix-free operator evaluation

Matrix-free operator evaluation as a loop over all cells and faces:

$$\boldsymbol{v} = \mathcal{A}(\boldsymbol{u}) \quad \leftrightarrow \quad \boxed{\boldsymbol{v} = \sum_{e \in \{cells\}} \mathcal{G}_e^T \circ \mathcal{C}_e^T \circ \mathcal{S}_e^T \circ \mathcal{Q}_e \circ \mathcal{S}_e \circ \mathcal{C}_e \circ \mathcal{G}_e \circ \boldsymbol{u} + \sum_{f_1, f_2 \in \{faces\}} \ldots + \sum_{f \in \{faces\}} \ldots}$$

with:



- $\mathcal{G}_e$ gathering; $\mathcal{C}_e$ constraints
- $\mathcal{S}_e$ evaluation (computation: $\mathcal{O}(dk^{d+1})$; memory: $\mathcal{O}(dk^d)$)
- $\mathcal{Q}_e$ operation on quadrature points

Kronbichler, M. and Kormann, K., 2012. A generic interface for parallel cell-based finite element operator application. Computers & Fluids, 63, pp.135-147.
Kronbichler, M. and Kormann, K., 2019. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM TOMS, 45(3), pp.1-40.

Matrix-free operator evaluation

Matrix-free operator evaluation as a loop over all cells and faces:

$$\boldsymbol{v} = \mathcal{A}(\boldsymbol{u}) \quad \leftrightarrow \quad \boxed{\boldsymbol{v} = \sum_{e \in \{cells\}} \mathcal{G}_e^T \circ \mathcal{C}_e^T \circ \mathcal{S}_e^T \circ \mathcal{Q}_e \circ \mathcal{S}_e \circ \mathcal{C}_e \circ \mathcal{G}_e \circ \boldsymbol{u} + \sum_{f_1, f_2 \in \{faces\}} \ldots + \sum_{f \in \{faces\}} \ldots}$$

Algorithm 1: Face-centric loop

```
/* loop over all cells                                   */
foreach c ∈ C do
    process_cell(c)
/* loop over all faces (face-cell pairs)                 */
foreach f ∈ I do
    process_face(f)
/* loop over all faces (face-cell pairs)                 */
foreach f ∈ B do
    process_boundary_face(f)
```

Matrix-free operator evaluation

Matrix-free operator evaluation as a loop over all cells and faces:

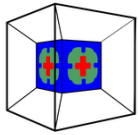
$$\boldsymbol{v} = \mathcal{A}(\boldsymbol{u}) \quad \leftrightarrow \quad \boxed{\boldsymbol{v} = \sum_{e \in \{cells\}} \mathcal{G}_e^T \circ \mathcal{C}_e^T \circ \mathcal{S}_e^T \circ \mathcal{Q}_e \circ \mathcal{S}_e \circ \mathcal{C}_e \circ \mathcal{G}_e \circ \boldsymbol{u} + \sum_{f_1, f_2 \in \{faces\}} \ldots + \sum_{f \in \{faces\}} \ldots}$$

Example: transport equation (arbitrary d)

$$\left(g, \frac{\partial f}{\partial t}\right)_{\Omega^{(e)}} = \left(g, -\beta \vec{a} \nabla f\right)_{\Omega^{(e)}} + \left(\nabla g, \vec{a}(1-\beta) f\right)_{\Omega^{(e)}} + \left\langle g, \vec{n} \cdot (\vec{a} f)^* - \beta f^- (\vec{n} \cdot \vec{a}) \right\rangle_{\Gamma^{(e)}}$$

$$\rightarrow \left(g, -|\mathcal{J}| \beta \vec{a} \mathcal{J}^{-1} \nabla_\xi f\right)_{\Omega_\xi^{(e)}} + \left(\nabla_\xi g, \mathcal{J}^{-T} |\mathcal{J}| \vec{a}(1-\beta) f\right)_{\Omega_\xi^{(e)}}$$

hyper.deal



hyper.deal

- an efficient, matrix-free FEM library for high-dimensional PDEs (2D–6D)
- focus: computational plasma physics
- based on deal.II (1D–3D) and extends it via a tensor-product ansatz
- node-level optimization (SIMD, SHMEM); scaling up to 150k CPUs on SuperMUC-NG
- developed by K. Kormann, K. Kronbichler, P. Munch
- funded by DFG SPPEXA, part of the ExaDG project[2]

Munch, P., Kormann, K. and Kronbichler, M., 2021. hyper. deal: An efficient, matrix-free finite-element library for high-dimensional partial differential equations. ACM Transactions on Mathematical Software (TOMS), 47(4), pp.1-34.

[2] Arndt, D., et. al., 2020. ExaDG: High-order discontinuous Galerkin for the exa-scale. SIP.

Part 4:

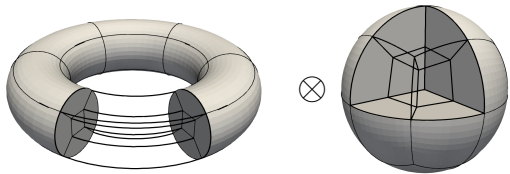
Tensor-product ansatz

Idea

Tensor-product ansatz

$$\boxed{\Omega = \Omega_{\vec{x}} \otimes \Omega_{\vec{v}} \quad \text{and} \quad \mathcal{T} = \mathcal{T}_{\vec{x}} \otimes \mathcal{T}_{\vec{v}}}$$

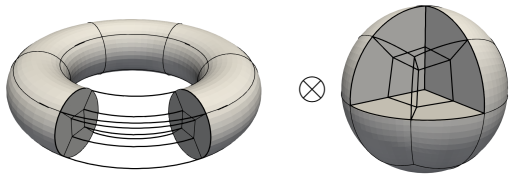
implies:



Idea

Tensor-product ansatz

$$\boxed{\Omega = \Omega_{\vec{x}} \otimes \Omega_{\vec{v}} \quad \text{and} \quad \mathcal{T} = \mathcal{T}_{\vec{x}} \otimes \mathcal{T}_{\vec{v}}}$$



implies:

- mapping

$$\mathcal{J} = \begin{pmatrix} \mathcal{J}_{\vec{x}} & 0 \\ 0 & \mathcal{J}_{\vec{v}} \end{pmatrix}, \quad |\mathcal{J}| = |\mathcal{J}_{\vec{x}}| \cdot |\mathcal{J}_{\vec{v}}|$$

- shape functions (for tensor-product elements):

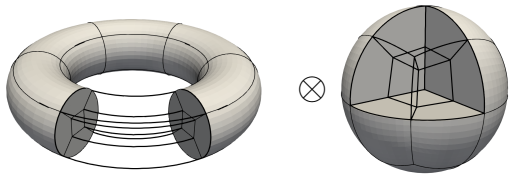
$$\mathcal{P}_k^{d_{\vec{x}}+d_{\vec{v}}} = \mathcal{P}_k^{d_{\vec{x}}} \otimes \mathcal{P}_k^{d_{\vec{v}}} = \underbrace{\mathcal{P}_k^1 \otimes \cdots \otimes \mathcal{P}_k^1}_{\times d_{\vec{x}}} \otimes \underbrace{\mathcal{P}_k^1 \otimes \cdots \otimes \mathcal{P}_k^1}_{\times d_{\vec{v}}} = \underbrace{\mathcal{P}_k^1 \otimes \cdots \otimes \mathcal{P}_k^1}_{\times (d_{\vec{x}}+d_{\vec{v}})}$$

- quadrature: similar to shape functions

Idea

Tensor-product ansatz

$$\boxed{\Omega = \Omega_{\vec{x}} \otimes \Omega_{\vec{v}} \quad \text{and} \quad \mathcal{T} = \mathcal{T}_{\vec{x}} \otimes \mathcal{T}_{\vec{v}}}$$



implies:

- cells

$$\mathcal{C} := \mathcal{C}_{\vec{x}} \otimes \mathcal{C}_{\vec{v}}$$

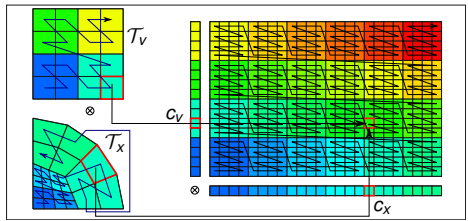
- inner faces

$$\mathcal{I} := (\mathcal{I}_{\vec{x}} \otimes \mathcal{C}_{\vec{v}}) \cup (\mathcal{C}_{\vec{x}} \otimes \mathcal{I}_{\vec{v}})$$

- boundary faces

$$\mathcal{B} := (\mathcal{B}_{\vec{x}} \otimes \mathcal{C}_{\vec{v}}) \cup (\mathcal{C}_{\vec{x}} \otimes \mathcal{B}_{\vec{v}})$$

- Cartesian partitioning $\rightarrow \int \mathrm{d}\Omega_{\vec{x}} / \int \mathrm{d}\Omega_{\vec{v}}$



Example: Transport equation

Given

$$\left(g, \frac{\partial f}{\partial t}\right)_{\Omega^{(e)}} = \left(g, -\beta \vec{a} \nabla f\right)_{\Omega^{(e)}} + \left(\nabla g, \vec{a}(1-\beta) f\right)_{\Omega^{(e)}} + \left\langle g, \vec{n} \cdot (\vec{a} f)^* - \beta f^- (\vec{n} \cdot \vec{a}) \right\rangle_{\Gamma^{(e)}}$$

and exploiting the structure of $\mathcal{J}$, one gets

$$\left(g, \frac{\partial f}{\partial t}\right)_{\Omega_{\vec{x}}^{(e)} \otimes \Omega_{\vec{v}}^{(e)}} = \left(g, -\beta \begin{pmatrix} \vec{a}_{\vec{x}} \mathcal{J}_{\vec{x}}^{-1} \\ \vec{a}_{\vec{v}} \mathcal{J}_{\vec{v}}^{-1} \end{pmatrix} \nabla_\xi f\right)_{\Omega_{\vec{x}}^{(e)} \otimes \Omega_{\vec{v}}^{(e)}} + \left(\nabla_\xi g, \begin{pmatrix} \mathcal{J}_{\vec{x}}^{-1} \vec{a}_{\vec{x}} \\ \mathcal{J}_{\vec{v}}^{-1} \vec{a}_{\vec{v}} \end{pmatrix} (1-\beta) f\right)_{\Omega_{\vec{x}}^{(e)} \otimes \Omega_{\vec{v}}^{(e)}}$$
$$+ \underbrace{\left(g, \vec{n}_{\vec{x}} \cdot (\vec{a}_{\vec{x}} f)^* - \beta f^- (\vec{n}_{\vec{x}} \cdot \vec{a}_{\vec{x}})\right)_{\Gamma_{\vec{x}}^{(e)} \otimes \Omega_{\vec{v}}^{(e)}}}_{x\text{-face integrals}} + \underbrace{\left(g, \vec{n}_{\vec{v}} \cdot (\vec{a}_{\vec{v}} f)^* - \beta f^- (\vec{n}_{\vec{v}} \cdot \vec{a}_{\vec{v}})\right)_{\Omega_{\vec{x}}^{(e)} \otimes \Gamma_{\vec{v}}^{(e)}}}_{v\text{-face integrals}}.$$

This implies

- (distributed) loop over cell/cell- and face/cell-pairs and
- query $\mathcal{J}_x$, $\mathcal{J}_v$, $|\mathcal{J}_x|$, $|\mathcal{J}_v|$, $|\vec{n}_x|$, $|\vec{n}_v|$ independently during loop over quadrature-point pairs.

Matrix-free loop in hyper.deal

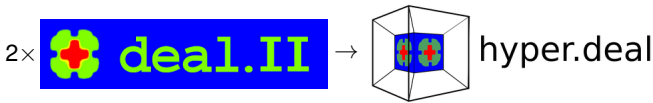
Algorithm 2: Face-centric loop (boundary faces not shown)

```
   /* loop over all cells (cell pairs)                                  */
1  foreach c ∈ C_x̃ × C_ỹ do
2      process_cell(c)
   /* loop over all x-faces (face-cell pairs)                           */
3  foreach f ∈ I_x̃ × C_ỹ do
4      process_face(f)
   /* loop over all y-faces (cell-face pairs)                           */
5  foreach f ∈ C_x̃ × I_ỹ do
6      process_face(f)
```

Advantages & challenges

Modularity & re-usability:

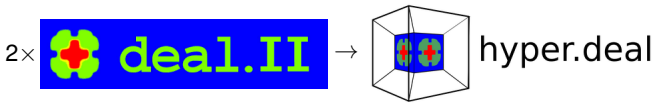
Loop over two low-dimensional meshes and query geometrical information.



Advantages & challenges

Modularity & re-usability:

Loop over two low-dimensional meshes and query geometrical information.



But: Curse of dimensionality d!

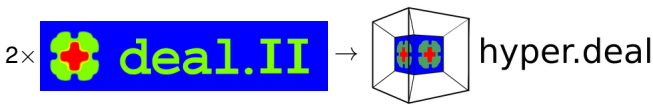
leads to:

- N_{1D}^d global DoFs
- increased ghost-value exchange due to increased surface-to-volume ratio with $\mathcal{O}(k^{d-1})$ DoFs to be communicated per face $\rightarrow$ shared memory
- $\mathcal{O}(k^{d+1})$ DoFs/quadrature points per cell
- increased work $\mathcal{O}(dk^{d+1})$
- increased working-set size $\mathcal{O}(k^d)$ $\rightarrow$ exceeding L1 cache

Advantages & challenges

Modularity & re-usability:

Loop over two low-dimensional meshes and query geometrical information.



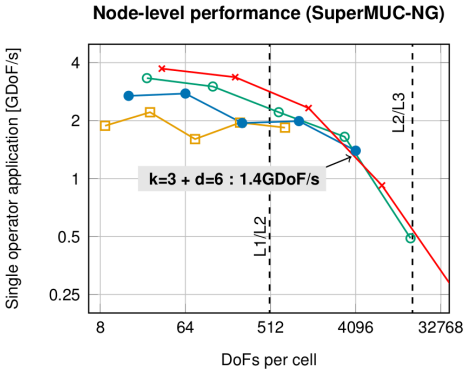
But: Curse of dimensionality d!

leads to:

- N_{1D}^d global DoFs
- increased ghost-value exchange due to increased surface-to-volume ratio with $\mathcal{O}(k^{d-1})$ DoFs to be communicated per face $\rightarrow$ shared memory
- $\mathcal{O}(k^{d+1})$ DoFs/quadrature points per cell
- increased work $\mathcal{O}(dk^{d+1})$
- increased working-set size $\mathcal{O}(k^d)$ $\rightarrow$ exceeding L1 cache

▷ *Example 6D ($N_{1D} = 100$, $k = 3$):*
▷ $N = 10^{12}$
▷ 1024 double
▷ 100 Flops/DoF
▷ 4096 double
▷ reference: 1 million entries in element matrix

hyper.deal: node-level performance



- setup:
  tensor product of two hypercubes
  (see: `examples` → `advection`)
- challenge:
  temporal data size of cell batch $\mathcal{O}(k^d)$
- outlook: vectorization within elements

$k = 2$ $k = 3$ $k = 4$ $k = 5$

for different dimensions

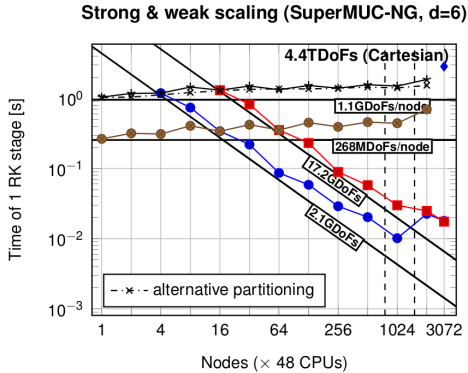
hyper.deal: strong and weak scaling

largest simulation:

- $4.4 \cdot 10^{12} = (2.1 \cdot 10^6)^2 = 128^6$DoFs

challenges:

- communication pattern
- communication data volume



Examples

Further examples for high-dimensional applications in which the tensor-product ansatz is applicable:

- extruded mesh
- space-time finite element methods
- 3D problems that involve a low-dimensional parameter space
- Fokker–Planck-type equations (e.g., Black–Scholes equation for option pricing in mathematical finance)

Part 5:

Vlasov–Poisson problem

Equations

Vlasov equation for electrons in a neutralizing background in the absence of magnetic fields:

$$\frac{\partial f}{\partial t} + \begin{pmatrix} \vec{v} \\ -\vec{E}(t,\vec{x}) \end{pmatrix} \cdot \nabla f = 0,$$

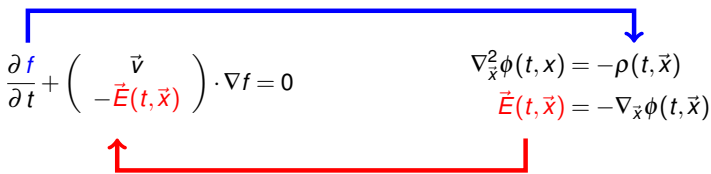
where the electric field can be obtained from the solution of the Poisson problem:

$$\rho(t,\vec{x}) = 1 - \int f(t,\vec{x},\vec{v})\,\mathrm{d}v, \qquad -\nabla^2_{\vec{x}}\phi(t,x) = \rho(t,\vec{x}), \qquad \vec{E}(t,\vec{x}) = -\nabla_{\vec{x}}\phi(t,\vec{x}).$$

Algorithm: coupling deal.II/hyper.deal

In each Runge-Kutta step solve:

$$\rho(t,\vec{x}) = 1 - \int f(t,\vec{x},\vec{v})\,\mathrm{d}v$$

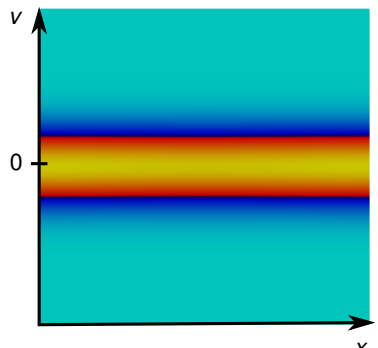


... the same approach is applicable to the Vlasov–Maxwell equations!

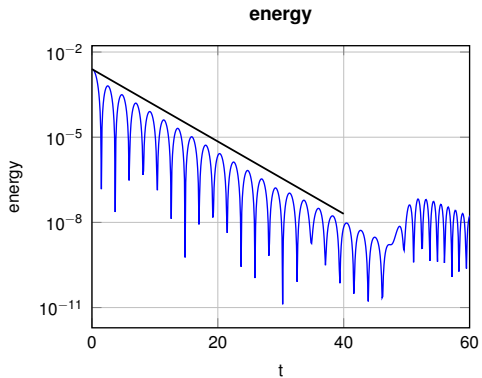
Benchmark 1: Landau damping

Solve VP for single population of electrons with initial condition:

$$f(0,x,v) = \frac{(1+0.01\cos(0.5x))\, e^{-v^2/2}}{\sqrt{2\pi}} \quad \text{on} \quad \Omega = [0,4\pi]\times[-6,6]$$



Benchmark 1: Landau damping (cont.)

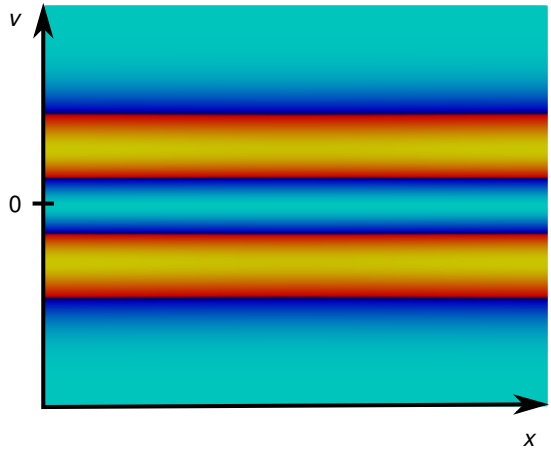


quantities of interest:

$$\int (\nabla\phi)^2 \mathrm{d}\Omega_{\vec{x}} \quad \int f \mathrm{d}\Omega \quad \sqrt{\int f^2 \mathrm{d}\Omega} \quad \int v^2 f \mathrm{d}\Omega \quad \int v_i f \mathrm{d}\Omega$$

Benchmark 2a: two-stream instability

setup: two populations of electrons with opposite velocity ($v_1(x) = -v_2(x)$)



Benchmark 2a: two-stream instability (cont.)

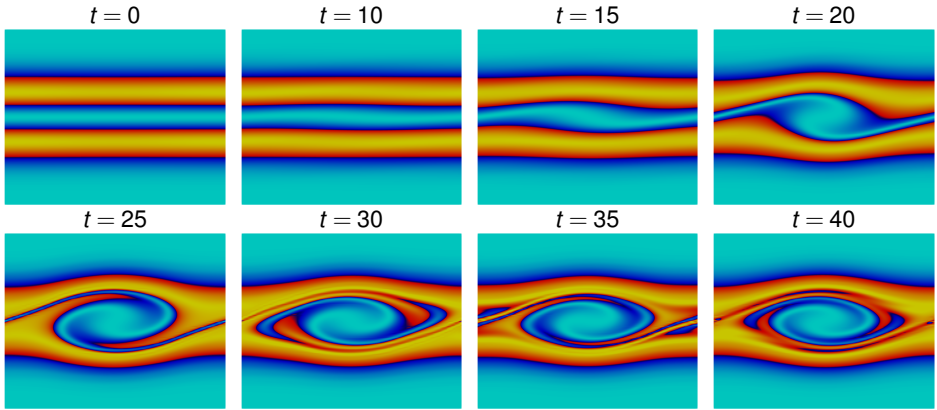
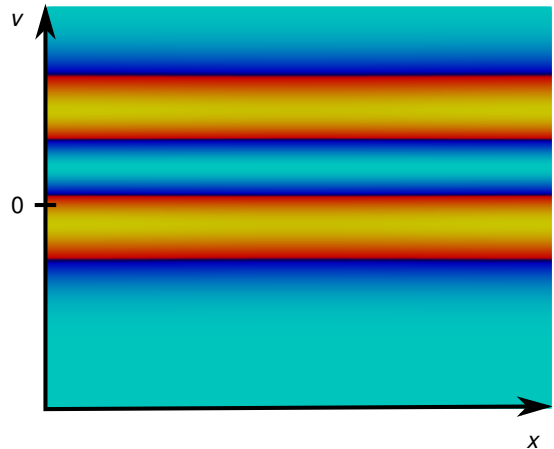


Figure: Solution of the Vlasov–Poisson equation for a two-stream instability.

Benchmark 2b: two-stream instability

setup: two populations of electrons with one being at rest ($v_1(x) > 0; v_2(x) = 0$)



Benchmark 2b: two-stream instability (cont.)

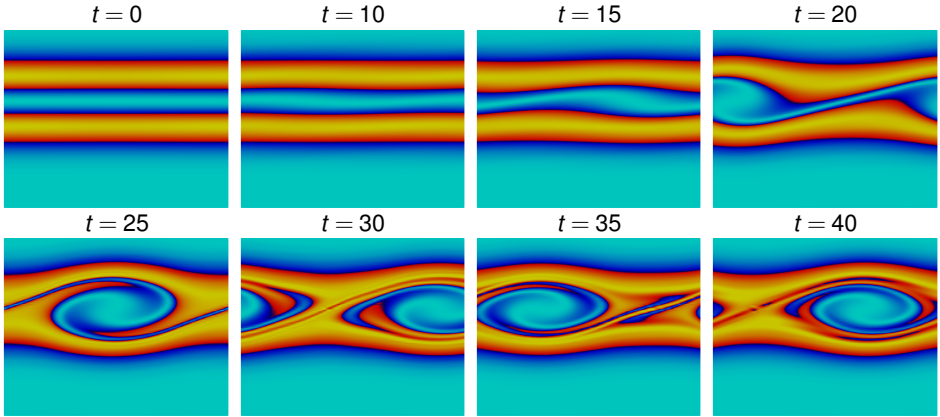
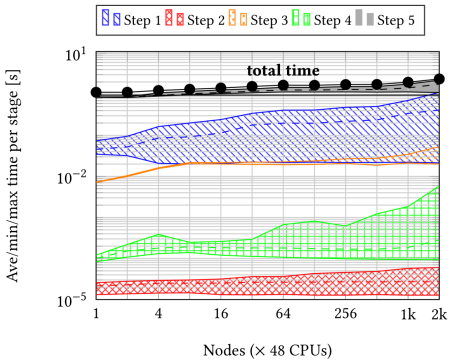


Figure: Solution of the Vlasov–Poisson equation for a two-stream instability.

Parallel scaling (6D)



1. $\rho(t,\vec{x}) \leftarrow 1 - \int f(t,\vec{x},\vec{v})\,\mathrm{d}v$
2. $t \leftarrow (\xi,\rho)_{\vec{x}}$
3. $(\nabla\xi,\nabla\phi)_{\vec{x}} = t$
4. $\vec{E}(t,\vec{x}) \leftarrow -\nabla_{\vec{x}}\phi(t,\vec{x})$
5. advection equation

Part 6:

Space-time finite-element computations[6]

[6] P. Munch, N. Margenberg, A space-time multigrid method for space-time finite-element discretizations, SIAM Conference on Computational Science and Engineering (CSE25), Fort Worth, Texas, USA, March 3-7, 2025.

Part 1:

Introduction

(Tensor-product) space-time FEM

Idea: time-dependent PDE

- space: standard continuous Lagrange finite element
- time: use DG (dG(k)) or FEM (cGP(k))

Advantages:

- variational time discretization $\rightarrow$ natural integration with the variational space discretization and natural capture of coupled problems and nonlinearities
- advantageous for duality and goal-oriented adaptivity in space and time [Schmich and Vexler '08, Bause et al. '21, Besier & Rannacher 2012, Roth et al. 2023]
- unified approach to stability and error analysis [Matthies and Schieweck '11]
- solves multiple time steps at once $\rightarrow$ relation to "parallel in time" algorithms [Gander '15, Ong and Schroder '20, Falgout et al. '14, '17]

Considered equations

- heat equation

$$\partial_t u - \nabla \cdot (\rho \nabla u) = f$$

- wave equations

$$\partial_t u - v = 0, \quad \partial_t v - \nabla \cdot (\rho \nabla u) = f$$

- convection-diffusion-reaction equation

$$\partial_t u - \nabla \cdot (\varepsilon \nabla u) + b \cdot \nabla u + \alpha u = f$$

- Stokes equations

$$\partial_t \boldsymbol{v} - \nu \Delta \boldsymbol{v} + \nabla p = \boldsymbol{f}, \quad \nabla \cdot \boldsymbol{v} = 0$$

- Navier–Stokes equations (WIP)

$$\partial_t \boldsymbol{u} - \nu \Delta \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \nabla p = \boldsymbol{f}, \quad \nabla \cdot \boldsymbol{u} = 0$$

Considered equations

- heat equation

$$\partial_t u - \nabla \cdot (\rho \nabla u) = f$$

- wave equations

$$\partial_t u - v = 0, \quad \partial_t v - \nabla \cdot (\rho \nabla u) = f$$

- convection-diffusion-reaction equation

$$\partial_t u - \nabla \cdot (\varepsilon \nabla u) + b \cdot \nabla u + \alpha u = f$$

- Stokes equations

$$\partial_t \boldsymbol{v} - \nu \Delta \boldsymbol{v} + \nabla p = \boldsymbol{f}, \quad \nabla \cdot \boldsymbol{v} = 0$$

- Navier–Stokes equations (WIP)

$$\partial_t \boldsymbol{u} - \nu \Delta \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla) \boldsymbol{u} + \nabla p = \boldsymbol{f}, \quad \nabla \cdot \boldsymbol{u} = 0$$

Considered equations

- heat equation

$$\partial_t u - \nabla \cdot (\rho \nabla u) = f$$

- convection-diffusion-reaction equation

$$\partial_t u - \nabla \cdot (\varepsilon \nabla u) + b \cdot \nabla u + \alpha u = f$$

- Stokes equations

$$\partial_t \boldsymbol{v} - \nu \Delta \boldsymbol{v} + \nabla p = \boldsymbol{f}, \quad \nabla \cdot \boldsymbol{v} = 0$$

- Navier–Stokes equations (WIP)

$$\partial_t \boldsymbol{u} - \nu \Delta \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \nabla p = \boldsymbol{f}, \quad \nabla \cdot \boldsymbol{u} = 0$$

- wave equations

$$\partial_t u - v = 0, \quad \partial_t v - \nabla \cdot (\rho \nabla u) = f$$

Table of content:

Considered equations

- heat equation

$$\partial_t u - \nabla \cdot (\rho \nabla u) = f$$

- convection-diffusion-reaction equation

$$\partial_t u - \nabla \cdot (\varepsilon \nabla u) + b \cdot \nabla u + \alpha u = f$$

- Stokes equations

$$\partial_t \boldsymbol{v} - \nu \Delta \boldsymbol{v} + \nabla p = \boldsymbol{f}, \quad \nabla \cdot \boldsymbol{v} = 0$$

- Navier–Stokes equations (WIP)

$$\partial_t \boldsymbol{u} - \nu \Delta \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \nabla p = \boldsymbol{f}, \quad \nabla \cdot \boldsymbol{u} =$$

- wave equations

$$\partial_t u - v = 0, \quad \partial_t v - \nabla \cdot (\rho \nabla u) = f$$

Table of content:

N. Margenberg and PM, “A space-time multigrid method for space-time finite element discretizations of parabolic and hyperbolic PDEs”, submitted, 2024.

N. Margenberg, M. Bause, and PM, “An *hp* multigrid approach for tensor-product space-time finite element discretizations of the Stokes equations”, submitted, 2025.

Part 2:

Solution procedures

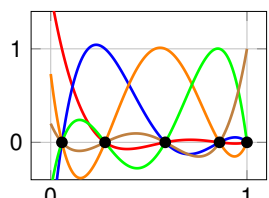
Tensor-product space-time FEM

Idea: time-dependent PDE

- space: standard continuous Lagrange finite element
- time: use DG (dG(k)) or FEM (cGP(k))

dG(k)

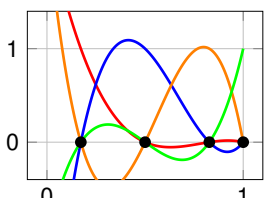
test/ansatz



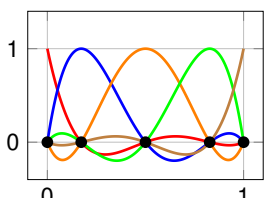
jump: $[w_{\tau,h}] = w_n^+ - w_n^-$

cGP(k)

test



ansatz



continuity condition: $w_n^+ = w_n^-$

Algebraic system for dG(k) discretization of the heat equation

Local algebraic system at n-th time step

$$\underbrace{(\boldsymbol{M}_\tau \otimes \boldsymbol{A}_h + \boldsymbol{A}_\tau \otimes \boldsymbol{M}_h)}_{:=\boldsymbol{S}} \boldsymbol{u}_n = \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_n + \alpha \otimes \boldsymbol{M}_h \boldsymbol{u}_{n-1}^{N_t}$$

with $(\boldsymbol{M}_\tau)_{i,j} := \tau \int_{\hat{I}} \hat{\xi}_j(\hat{t})\hat{\xi}_i(\hat{t})\,\mathrm{d}\hat{t}$, $\quad (\boldsymbol{A}_\tau)_{i,j} := \int_{\hat{I}} \hat{\xi}_j'(\hat{t})\hat{\xi}_i(\hat{t})\,\mathrm{d}\hat{t} + \hat{\xi}_j(0)\hat{\xi}_i(0)$, $\quad \alpha_i := \hat{\xi}_i(0)$.

Multiple-time-steps system

Let $\boldsymbol{B} := \mathbf{1}_{k+1} \otimes \alpha \otimes \boldsymbol{M}_h$, then we collect consecutive time steps $n_1, \ldots, n_c$

$$\begin{pmatrix} \boldsymbol{S} & & & & \\ -\boldsymbol{B} & \boldsymbol{S} & & & \\ & \ddots & \ddots & & \\ & & -\boldsymbol{B} & \boldsymbol{S} & \\ & & & -\boldsymbol{B} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_{n_1} \\ \boldsymbol{u}_{n_2} \\ \vdots \\ \boldsymbol{u}_{n_c-1} \\ \boldsymbol{u}_{n_c} \end{pmatrix} = \begin{pmatrix} \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_1} + \alpha \otimes \boldsymbol{M}_h \boldsymbol{u}_{n_1-1}^{N_t} \\ \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_2} \\ \vdots \\ \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_c-1} \\ \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_c} \end{pmatrix}.$$

Algebraic system for cGP(k) discretization of the heat equation

Local algebraic system at n-th time step

$$\underbrace{(\boldsymbol{M}_\tau \otimes \boldsymbol{A}_h + \boldsymbol{A}_\tau \otimes \boldsymbol{M}_h)}_{:=\boldsymbol{S}} \boldsymbol{u}_n = \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_n - \beta \otimes \boldsymbol{M}_h \boldsymbol{f}_{n-1}^{N_t} + \underbrace{(\beta \otimes \boldsymbol{A}_h + \alpha \otimes \boldsymbol{M}_h)}_{:=\boldsymbol{b}} \boldsymbol{u}_{n-1}^{N_t}$$

with $(\boldsymbol{M}_\tau)_{i,j-1} := \tau \int_I \hat{\xi}_j(\hat{t})\,\hat{\psi}_i(\hat{t})\,\mathrm{d}\hat{t}$, $\quad (\boldsymbol{A}_\tau)_{i,j-1} := \int_I \hat{\xi}_j'(\hat{t})\,\hat{\psi}_i(\hat{t})\,\mathrm{d}\hat{t}$,

$\beta_i := \tau \int_I \hat{\xi}_1(\hat{t})\,\hat{\psi}_i(\hat{t})\,\mathrm{d}\hat{t}$, $\quad \alpha_i := \int_I \hat{\xi}_1'(\hat{t})\,\hat{\psi}_i(\hat{t})\,\mathrm{d}\hat{t}$, $\quad i = 1, \ldots, k,\ j = 2, \ldots, k+1$

Multiple-time-steps system

Let $\boldsymbol{B} := \mathbf{1}_k \otimes \boldsymbol{b}$, then we collect consecutive time steps $n_1, \ldots, n_c$

$$\begin{pmatrix} \boldsymbol{S} & & & & \\ -\boldsymbol{B} & \boldsymbol{S} & & & \\ & \ddots & \ddots & & \\ & & -\boldsymbol{B} & \boldsymbol{S} & \\ & & & -\boldsymbol{B} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_{n_1} \\ \boldsymbol{u}_{n_2} \\ \vdots \\ \boldsymbol{u}_{n_c-1} \\ \boldsymbol{u}_{n_c} \end{pmatrix} = \begin{pmatrix} \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_1} - \beta \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_1-1}^{N_t} + \boldsymbol{b} \otimes u_{n_1-1}^{N_t} \\ \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_2} - \beta \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_1}^{N_t} \\ \vdots \\ \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_c-1} - \beta \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_c-2}^{N_t} \\ \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_c} - \beta \otimes \boldsymbol{M}_h \boldsymbol{f}_{n_c-1}^{N_t} \end{pmatrix}.$$

Space-time multigrid

To solve

$$\begin{pmatrix} \boldsymbol{S} & & & & \\ -\boldsymbol{B} & \boldsymbol{S} & & & \\ & \ddots & \ddots & & \\ & & -\boldsymbol{B} & \boldsymbol{S} & \\ & & & -\boldsymbol{B} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_{n_1} \\ \boldsymbol{u}_{n_2} \\ \vdots \\ \boldsymbol{u}_{n_c-1} \\ \boldsymbol{u}_{n_c} \end{pmatrix} = \begin{pmatrix} \cdots \\ \cdots \\ \vdots \\ \cdots \\ \cdots \end{pmatrix},$$

we use GMRES with space-time multigrid [Hackbusch '85, Gander and Neumüller '16]:

- h- and p-multigrid both in space and time
- first coarsen p and then h
- simultaneously coarsen in space and time
- smoother: additive Schwarz (element-centric patches)
  $\rightarrow$ full matrices with $O(kp^d)$ rows/columns

Space-time multigrid

Software:

To solve

$$\begin{pmatrix} \boldsymbol{S} & & & & \\ -\boldsymbol{B} & \boldsymbol{S} & & & \\ & \ddots & \ddots & & \\ & & -\boldsymbol{B} & \boldsymbol{S} & \\ & & & -\boldsymbol{B} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_{n_1} \\ \boldsymbol{u}_{n_2} \\ \vdots \\ \boldsymbol{u}_{n_c-1} \\ \boldsymbol{u}_{n_c} \end{pmatrix} = \begin{pmatrix} \dots \\ \dots \\ \vdots \\ \dots \\ \dots \end{pmatrix},$$

we use GMRES with space-time multigrid [Hackbusch '85, Gander and Neumüller '16]:

- h- and p-multigrid both in space and time
- first coarsen p and then h
- simultaneously coarsen in space and time
- smoother: additive Schwarz (element-centric patches)
  $\rightarrow$ full matrices with $O(kp^d)$ rows/columns

Space-time multigrid

Software:

To solve

$$\begin{pmatrix} \boldsymbol{S} & & & & \\ -\boldsymbol{B} & \boldsymbol{S} & & & \\ & \ddots & \ddots & & \\ & & -\boldsymbol{B} & \boldsymbol{S} & \\ & & & -\boldsymbol{B} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_{n_1} \\ \boldsymbol{u}_{n_2} \\ \vdots \\ \boldsymbol{u}_{n_c-1} \\ \boldsymbol{u}_{n_c} \end{pmatrix} = \begin{pmatrix} \cdots \\ \cdots \\ \vdots \\ \cdots \\ \cdots \end{pmatrix},$$

we use GMRES with space-time multigrid [Hackbusch '85, Gander and Neumüller '16]:

- h- and p-multigrid both in space and time
- first coarsen p and then h
- simultaneously coarsen in space and time
- smoother: additive Schwarz (element-centric patches)
  $\rightarrow$ full matrices with $O(kp^d)$ rows/columns

Next:

1. evaluation of $\boldsymbol{S}$
2. transfer operator

Space-time multigrid: Matrix-free operator evaluation

Operator $\boldsymbol{S} = (\boldsymbol{M}_\tau \otimes \boldsymbol{A}_h + \boldsymbol{A}_\tau \otimes \boldsymbol{M}_h)$ is never assembled but directly applied to $\boldsymbol{u}_n$:

$$\boldsymbol{v} = \boldsymbol{S}\boldsymbol{u} = (\boldsymbol{M}_\tau \otimes \boldsymbol{I}_h)(\boldsymbol{I}_\tau \otimes \boldsymbol{A}_h)\boldsymbol{u} + (\boldsymbol{A}_\tau \otimes \boldsymbol{I}_h)(\boldsymbol{I}_\tau \otimes \boldsymbol{M}_h)\boldsymbol{u}$$

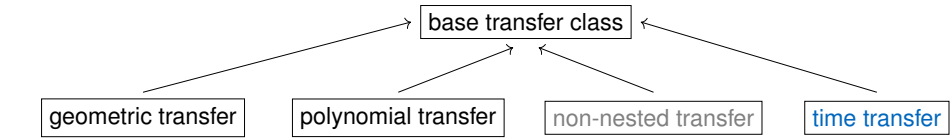
implying two steps:

1. apply $\boldsymbol{A}_h/\boldsymbol{M}_h$ to each block ▷ $(\boldsymbol{I}_\tau \otimes \boldsymbol{A}_h)$, $(\boldsymbol{I}_\tau \otimes \boldsymbol{M}_h)$
2. compute linear combination using $\boldsymbol{A}_\tau/\boldsymbol{M}_\tau$. ▷ $(\boldsymbol{M}_\tau \otimes \boldsymbol{I}_h)$, $(\boldsymbol{A}_\tau \otimes \boldsymbol{I}_h)$

Furthermore: application of $\boldsymbol{A}_h/\boldsymbol{M}_h$ is efficiently implemented in a matrix-free way [Kronbichler & Kormann, '12].

Space-time multigrid: transfer operators

Heart of deal.II's multigrid infrastructure: transfer operators



also working for multivectors: $\boldsymbol{u}^{(f)} = (\boldsymbol{I}_\tau \otimes \boldsymbol{P}_h)\boldsymbol{u}^{(c)}$

Time transfer:

- prolongation as operation on multivectors

  $$\boldsymbol{u}^{(f)} = (\boldsymbol{P}_\tau \otimes \boldsymbol{I}_h)\boldsymbol{u}^{(c)}$$

  ... L^2 projection; $\boldsymbol{P}_\tau$: different for geometric/polynomial coarsening
- restriction as adjoint of prolongation operator

Part 3:

Application: heat equation

Numerical experiments

Test setup

- cG(p)-cGP(k) and cG(p)-dG(k) methods, $p = k$, $k \in \{2, 3, 4, 5\}$
- heat equation with thermal diffusivity $\rho = 1$
- prescribed solution with $f = 2$

$$u(\boldsymbol{x}, t) = \sin(2\pi f t)\sin(2\pi f x)\sin(2\pi f y)\sin(2\pi f z)$$

Study the errors $e_u = u(\boldsymbol{x}, t) - u_{\tau,h}(\boldsymbol{x}, t)$ in the norms given by

$$\|e_u\|_{L^\infty(L^\infty)} = \max_{t\in I}\left(\sup_{\Omega}\|e_u\|_\infty\right), \quad \|e_u\|_{L^2(L^2)} = \left(\int_I\int_\Omega |e_u|^2 \,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t\right)^{\frac{1}{2}}.$$

Numerical experiments (cont.)

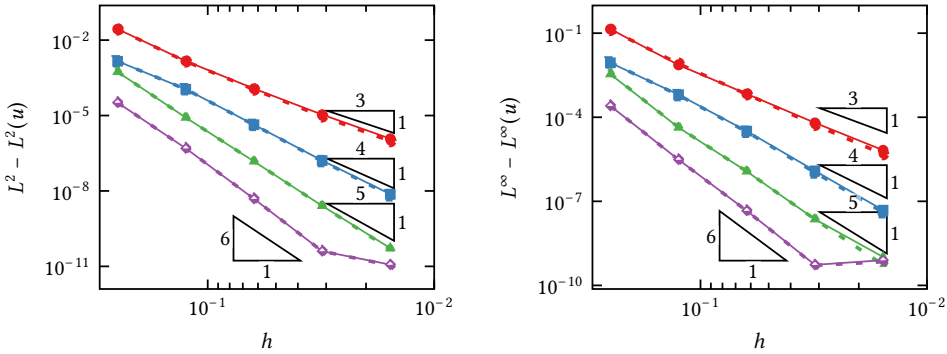


Figure: Computed errors for the displacement u for different polynomial orders $p = k$ for $CG(p) - DG(k)$ discretizations of the heat equation. The expected orders of convergence $k+1$, represented by the triangles, match with the experimental orders.

Numerical experiments (cont.)

$cG(p) - dG(k)$ single time step

$k \setminus r$	2	3	4	5	6
2	9.0	9.75	9.00	8.875	8.656
3	12.0	11.75	10.88	10.188	10.563
4	14.5	14.00	12.88	11.813	11.781

$cG(p) - cGP(k)$ single time step

$k \setminus r$	2	3	4	5	6
2	9.0	9.75	9.25	8.875	8.688
3	12.0	12.00	10.88	10.188	10.594
4	14.5	14.00	12.88	11.875	11.781

Numerical experiments (cont.)

$cG(p) - dG(k)$ single time step

$k \setminus r$	2	3	4	5	6
2	9.0	9.75	9.00	8.875	8.656
3	12.0	11.75	10.88	10.188	10.563
4	14.5	14.00	12.88	11.813	11.781

$cG(p) - dG(k)$ 2 time steps at once

$k \setminus r$	2	3	4	5	6
2	10.0	10.0	10.0	9.60	9.234
3	12.0	12.38	11.75	10.88	11.484
4	15.0	15.0	13.75	12.88	12.75

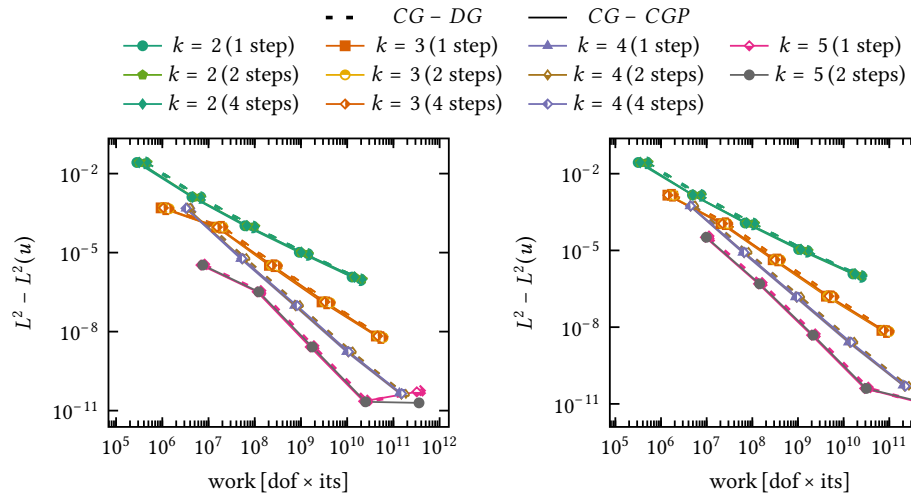
$cG(p) - cGP(k)$ single time step

$k \setminus r$	2	3	4	5	6
2	9.0	9.75	9.25	8.875	8.688
3	12.0	12.00	10.88	10.188	10.594
4	14.5	14.00	12.88	11.875	11.781

$cG(p) - dGP(k)$ 2 time steps at once

$k \setminus r$	2	3	4	5	6
2	10.0	10.0	10.0	9.75	9.484
3	12.8	13.00	11.75	10.875	11.484
4	15.0	15.00	13.75	12.875	12.75

Numerical experiments (cont.)



Outlook

N. Margenberg and PM, “A space-time multigrid method for space-time finite element discretizations of parabolic and hyperbolic PDEs”, submitted, 2024.

- deformed meshes and heterogeneous coefficients
- wave equation:

$$\begin{aligned}
\boldsymbol{v}_n &= \boldsymbol{M}_\tau^{-1}\boldsymbol{A}_\tau \boldsymbol{u}_n - \boldsymbol{M}_\tau^{-1}\alpha \boldsymbol{u}_{n-1}^{N_t}, \\
\underbrace{(\boldsymbol{M}_\tau \otimes \boldsymbol{A}_h + \boldsymbol{A}_\tau \boldsymbol{M}_\tau^{-1}\boldsymbol{A}_\tau \otimes \boldsymbol{M}_h)}_{:=\boldsymbol{S}} \boldsymbol{u}_n &= \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f} + \alpha \otimes \boldsymbol{M}_h \boldsymbol{v}_{n-1}^{N_t} + \underbrace{\boldsymbol{A}_\tau \boldsymbol{M}_\tau^{-1}\alpha \otimes \boldsymbol{M}_h}_{:=\boldsymbol{D}} \boldsymbol{u}_{n-1}^{N_t}.
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{v}_n &= \boldsymbol{M}_\tau^{-1}\boldsymbol{A}_\tau \boldsymbol{u}_n - \boldsymbol{M}_\tau^{-1}\alpha \boldsymbol{u}_{n-1}^{N_t} + \boldsymbol{M}_\tau^{-1}\beta \boldsymbol{v}_{n-1}^{N_t} \\
\underbrace{(\boldsymbol{M}_\tau \otimes \boldsymbol{A}_h + \boldsymbol{A}_\tau \boldsymbol{M}_\tau^{-1}\boldsymbol{A}_\tau \otimes \boldsymbol{M}_h)}_{:=\boldsymbol{S}} \boldsymbol{u}_n &= \boldsymbol{M}_\tau \otimes \boldsymbol{M}_h \boldsymbol{f} - \beta \otimes \boldsymbol{M}_h \boldsymbol{f}_{n-1}^{N_t} \\
&\quad + (\beta \otimes \boldsymbol{A}_h + \boldsymbol{A}_\tau \boldsymbol{M}_\tau^{-1}\alpha \otimes \boldsymbol{M}_h)\boldsymbol{u}_{n-1}^{N_t} + (\alpha - \boldsymbol{A}_\tau \boldsymbol{M}_\tau^{-1}\beta) \otimes \boldsymbol{M}_h \boldsymbol{v}_{n-1}^{N_t}
\end{aligned}$$

- scaling studies with 20,556 MPI ranks

Part 4:

Application: Stokes equations

Solution procedure

dG(k) space-time formulation: Find $(\boldsymbol{V}_n, \boldsymbol{P}_n) \in \mathbf{R}^{(k+1)(M^{\boldsymbol{v}}+M^p)}$ such that

$$\begin{pmatrix} \boldsymbol{K}_n^\tau \otimes \boldsymbol{M}_h + \boldsymbol{M}_n^\tau \otimes \boldsymbol{A}_h & \boldsymbol{M}_n^\tau \otimes \boldsymbol{B}_h^\top \\ \boldsymbol{M}_n^\tau \otimes \boldsymbol{B}_h & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{V}_n \\ \boldsymbol{P}_n \end{pmatrix} = \begin{pmatrix} \boldsymbol{F}_n \\ \mathbf{0} \end{pmatrix} + \boldsymbol{C}_n^\tau \otimes \begin{pmatrix} \boldsymbol{M}_h \\ \mathbf{0} \end{pmatrix} \boldsymbol{V}_{n-1}\,.$$

The global discrete solution spaces are defined by the tensor products

$$\boldsymbol{H}_{\tau,h}^{\boldsymbol{v}} = Y_\tau^k(I) \otimes \boldsymbol{V}_h^{r+1}(\Omega)\,, \quad H_{\tau,h}^p = Y_\tau^k(I) \otimes Q_h^r(\Omega)\,,$$

with

$$\begin{aligned} \boldsymbol{V}_h^{r+1}(\Omega) &:= \{\boldsymbol{v}_h \in \boldsymbol{V} \,:\, \boldsymbol{v}_{h|K} \in \mathbb{Q}_{k+1}^d(K) \text{ for all } K \in T_h\} \cap \boldsymbol{H}_0^1(\Omega)\,, \\ Q_h^k(\Omega) &:= \{q_h \in Q \,:\, q_{h|K} \in \mathbb{P}_r^{\mathrm{disc}}(K) \text{ for all } K \in T_h\}\,. \end{aligned}$$

Preconditioner: space-time multigrid with additive Vanka smoother (element-centric patches consisting of v and p)

Numerical experiments

Model problem on the space-time domain $\Omega \times I = [0,1]^2 \times [0,1]$ with prescribed solution given for velocity $\boldsymbol{v}\colon \Omega \times I \to \mathbb{R}^2$ and pressure $p\colon \Omega \times I \to \mathbb{R}$ by

$$\boldsymbol{v}(\mathbf{x}, t) = \sin(t) \begin{pmatrix} \sin^2(\pi x)\sin(\pi y)\cos(\pi y) \\ \sin(\pi x)\cos(\pi x)\sin^2(\pi y) \end{pmatrix},$$

$$p(\mathbf{x}, t) = \sin(t)\sin(\pi x)\cos(\pi x)\sin(\pi y)\cos(\pi y).$$

We set the kinematic viscosity to $\nu = 0.1$ and choose the external force $\mathbf{f}$ appropriately.

Numerical experiments (cont.)

Table: Number of GMRES iterations until convergence for different polynomial degrees r and numbers of refinements r with $\mathbb{Q}_{k+1}^2/\mathbb{P}_k^{\text{disc}}$ discretization of the Stokes system.

h-multigrid in space

$r\backslash k$	1	2	3	4	5	6
2	14.0	15.0	15.0	14.0	13.0	10.6
3	19.0	17.9	18.9	18.3	16.4	14.0
4	24.0	26.8	24.7	24.6	21.4	18.4
5	26.0	26.4	28.8	27.7	24.7	21.9
6	35.0	33.9	34.6	30.9	29.6	26.9
7	40.0	38.8	39.6	36.7	34.5	31.9

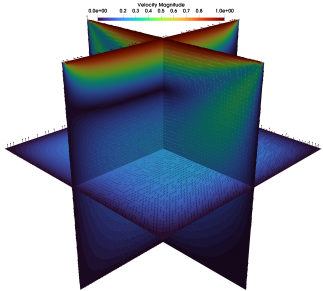
hp STMG

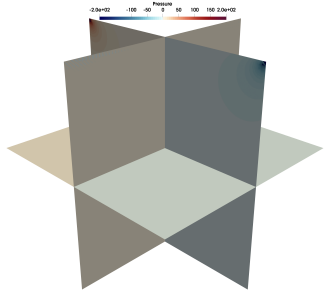
$r\backslash k$	1	2	3	4	5	6
2	14.0	15.0	15.0	14.0	13.0	10.6
3	19.8	15.9	16.0	15.0	13.7	11.0
4	27.8	23.0	22.9	21.9	19.0	15.5
5	31.0	26.4	26.6	22.8	18.7	14.9
6	45.0	36.1	36.7	29.0	23.1	17.2
7	50.8	43.8	42.8	32.8	25.6	19.6

Outlook

N. Margenberg, M. Bause, and PM, “An hp multigrid approach for tensor-product space-time finite element discretizations of the Stokes equations”, submitted, 2025.

- lid-driven cavity





- scaling studies with 13,824 MPI ranks

Part 5:

Block preconditioners

Motivation

- space-time multigrid is a monolithic and robust approach, however, needs expensive smoothers (here: element-centric additive patch smoothers)
- efficient implementation of patch smoothers: still open research; examples:
  - Pazner and Persson '17 $\rightarrow$ SVD-based tensor-product preconditioner
  - Brubeck and Farrell '21 $\rightarrow$ vertex-star relaxation
- alternative: block preconditioning $\rightarrow$ use cheaper smoothers on blocks; examples:
  - for space-time FEM: Danieli et al. '22
  - for IRK: Southworth et al. '22, Axelsson et al. '20, '24, Dravis et al.'24, PM et al.'24
    (stage-parallel IRK)

Stage-parallel implicit Runge–Kutta preconditioning (cont.)

PM, I. Dravins, M. Kronbichler, and M. Neytcheva, "Stage-parallel fully implicit Runge-Kutta implementations with optimal multilevel preconditioners at the scaling limit", in SISC, 2022.

For a linear system of equations, IRK has the form: ... Butcher tableau: $\begin{array}{c|c} \boldsymbol{c}_Q & A_Q \\ \hline & \boldsymbol{b}_Q^\top \end{array}$

$$\mathbf{u}_{m+1} = \mathbf{u}_m + \tau \sum_{q=1}^{Q} b_q \mathbf{k}_q \quad \text{w.} \quad \underbrace{(A_Q^{-1} \otimes M + \tau \mathbb{I}_Q \otimes K)}_{A} \mathbf{k} = (A_Q^{-1} \otimes \mathbb{I}_n)\overline{\mathbf{g}} - (A_Q^{-1} \otimes K)(\mathbf{e}_Q \otimes \mathbf{u}_0)$$

Following Butcher [1976], A can be factorized, using $A_Q^{-1} = S\Lambda S^{-1}$, and explicitly inverted:

$$A = (S \otimes \mathbb{I}_n)(\Lambda \otimes M + \tau \mathbb{I}_Q \otimes K)(S^{-1} \otimes \mathbb{I}_n) \quad A^{-1} = (S \otimes \mathbb{I}_n)(\Lambda \otimes M + \tau \mathbb{I}_Q \otimes K)^{-1}(S^{-1} \otimes \mathbb{I}_n).$$

Axelsson, Neytcheva [2020] proposed real-value preconditioner ($LU = A_Q^{-1} \to L = \tilde{S}\tilde{\Lambda}\tilde{S}^{-1}$):

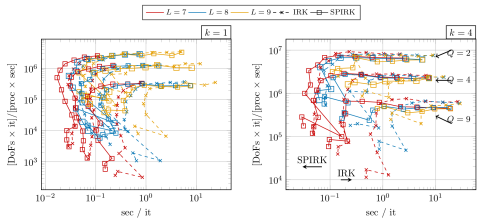
$$P^{-1} = (\tilde{S} \otimes \mathbb{I}_n)(\tilde{\Lambda} \otimes M + \tau \mathbb{I}_Q \otimes K)^{-1}(\tilde{S}^{-1} \otimes \mathbb{I}_n).$$

... Q stages can be solved in parallel! Helmholtz operator $\to$ multigrid

Stage-parallel implicit Runge–Kutta preconditioning (cont.)

Main results:

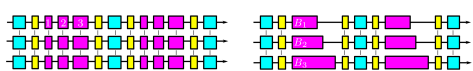
- for the first time shown: stage parallelism shifts the scaling limit



... clear speedup for $\leq$ 10k DoFs per process!

- performance model: minimize lin. iterations performed in serial, speedup limited by Q

$$\sum_{1 \leq q \leq Q} N_Q^{\mathrm{IT}} \quad \text{vs.} \quad \max_{1 \leq q \leq Q} N_Q^{\mathrm{IT}}$$



(b) parallel IRK with 3 processes (c) stage-parallel IRK with 3 processes

- application also to advection/diffusion; extension to nonlinear equations?

Implicit Runge–Kutta methods vs. space-time FEM

- implicit Runge–Kutta method: $\mathbf{u}_{m+1} = \mathbf{u}_m + \tau \sum_{q=1}^{Q} b_q \mathbf{k}_q$ with

$$(A_Q^{-1} \otimes M + \tau \mathbb{I}_Q \otimes K)\mathbf{k} = (A_Q^{-1} \otimes I_n)(\overline{\mathbf{g}} - \mathbf{e}_Q \otimes (K\mathbf{u}_0))$$

- space-time FEM with dG(k): $\mathbf{u}_{m+1} = \mathbf{k}_Q$ with

$$\boxed{(\tilde{A}_Q^{-1} \otimes M + \tau \mathbb{I}_Q \otimes K)\mathbf{k} = \tau \overline{\mathbf{g}} + \tilde{\alpha} \otimes (M\mathbf{u}_0)}$$

and $\tilde{A}_Q^{-1} = M_\tau^{-1} A_\tau, \tilde{\alpha} = M_\tau^{-1} \alpha$

Observations:

- system matrix: same structure
- different coefficients
- different rhs

- space-time FEM with cGP(k): $\mathbf{u}_{m+1} = \mathbf{k}_Q$ with

$$\boxed{(\tilde{A}_Q^{-1} \otimes M + \tau \mathbb{I}_Q \otimes K)\mathbf{k} = \tau \overline{\mathbf{g}} - \tau \tilde{\beta} \otimes \overline{\mathbf{g}}_0 + (\tau \tilde{\beta} \otimes K + \tilde{\alpha} \otimes M)\mathbf{u}_0}$$

and $\tilde{A}_Q^{-1} = M_\tau^{-1} A_\tau$, $\tilde{\alpha} = M_\tau^{-1} \alpha$, and $\tilde{\beta} = M_\tau^{-1} \beta$

Numerical results

2D setup (similar as above):

$$u(\boldsymbol{x}, t) = \sin(2\pi ft)\sin(2\pi fx)\sin(2\pi fy)$$

We set $f = 1$, $\tau = 0.1$ and run 10 time steps. Preconditioner on block: 1 V-cycle of geometric multigrid with Chebyshev smoother around point Jacobi.

Preliminary results:

r	IRK(Q=2)	dG(k=2)	cGP(k=3)
4	4	4	4
5	4	4	4
6	4	4	4

r	IRK(Q=5)	dG(k=5)	cGP(k=6)
3	7.9	7.9	7.9
4	8	8	8
5	8	8	8

Same number of iterations!

For complex variant, see: Werder et al. ['01], Banks et al. ['14]

Part 6:

Conclusions & Outlook

Conclusions

- space-time multigrid for space-time finite-element computations
  - matrix-free implementation
  - simple implementation with deal.II
  - smoother: additive Schwarz (element-centric patches)
  - robust but expensive
  - application: heat equation and Stokes equation
- alternative: block preconditioners $\rightarrow$ preliminary results

Outlook

- efficient patch smoothers
- block preconditioning
- application: Navier–Stokes equations [Anselmann and Bause '23]

deal.II Workshop @ Durham University

Lecture 4: multiphysics and coupled problems

Peter Munch[1]

[1] Institute of Mathematics, Technical University of Berlin, Germany

April 4, 2025

Volume coupling: a monolithic view

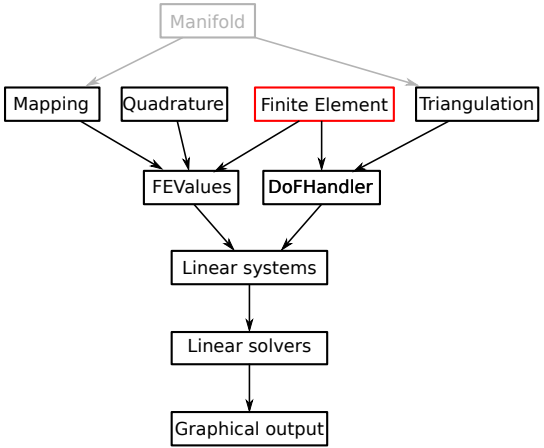
Example: TSI

- thermo-structural element:

$$\underbrace{[\mathcal{Q}_p^d, \ldots, \mathcal{Q}_p^d]}_{\times d+1}$$

- alternatively with $p_t \neq p_u$:

$$\big[\mathcal{Q}_{p_t}^d, \underbrace{[\mathcal{Q}_{p_u}^d, \ldots, \mathcal{Q}_{p_u}^d]}_{\times d}\big]$$




```
FESystem<dim> fe(FE_Q<dim>(degree), dim + 1);
// vs.
FESystem<dim> fe(FE_Q<dim>(degree), 1, FESystem<dim> fe(FE_Q<dim>(degree), dim));
```

Volume coupling: a partitioned view

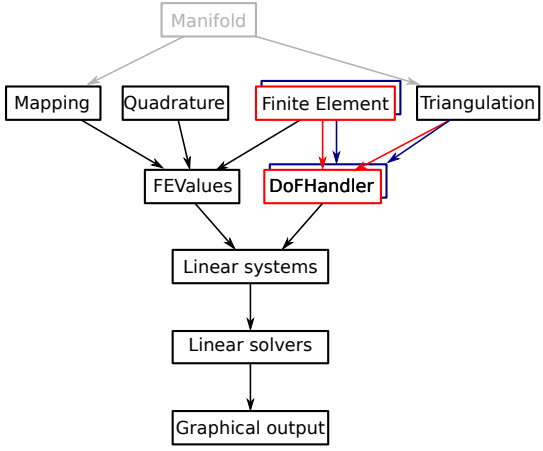
Example: TSI

- thermoelement:

$$\mathcal{Q}^d_{p_t}$$

- structural element:

$$\underbrace{[\mathcal{Q}^d_{p_u}, \ldots, \mathcal{Q}^d_{p_u}]}_{\times d}$$




```
FE_Q<dim>      fe_t(degree);
FESystem<dim> fe_s(FE_Q<dim>(degree), dim);
```

Surface coupling: non-matching grids

Example: FSI with 2 grids
- elasticity:

$$\underbrace{[\mathcal{Q}_p^d, \ldots, \mathcal{Q}_p^d]}_{\times d}$$

- incompressible Navier-Stokes eq.:

$$\underbrace{[\mathcal{Q}_{p_v}^d, \ldots, \mathcal{Q}_{p_v}^d]}_{\times d} \quad \text{and} \quad \mathcal{Q}_{p_p}^d$$

for surface coupling, e.g., `RemotePointEvaluation` or preCICE adapter can be used