The general-purpose finite-element library deal.II

Overview and basic concepts

@peterrum - Peter Munch[1,2,3]

[1]deal.II developer

[2]Institute for Computational Mechanics, Technical University of Munich, Germany

[3]Institute of Materials Research, Materials Mechanics, Helmholtz-Zentrum Geesthacht, Germany

November 18, 2020

Table of contents

... feel free to ask questions at any time you want!

Part 1:

Overview

Introduction

- 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
- more than 1,600 publications (on and with deal.II)
- freely available under LGPL 2.1 license
- yearly releases; current release: 9.2
- 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, ...

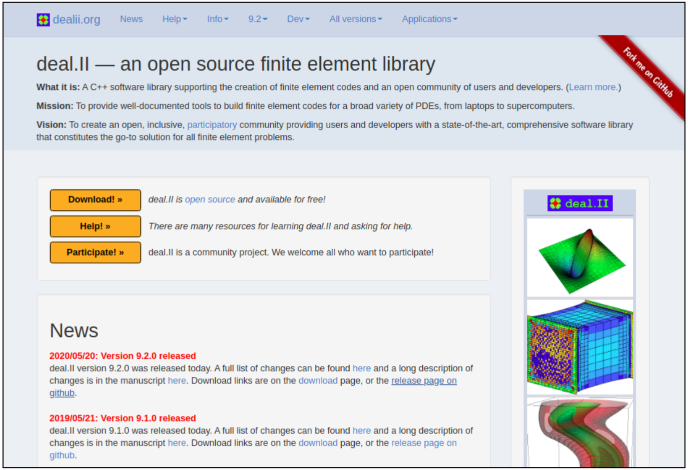
[1] successor of DEAL: **D**ifferential **E**quations **A**nalysis **L**ibrary

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

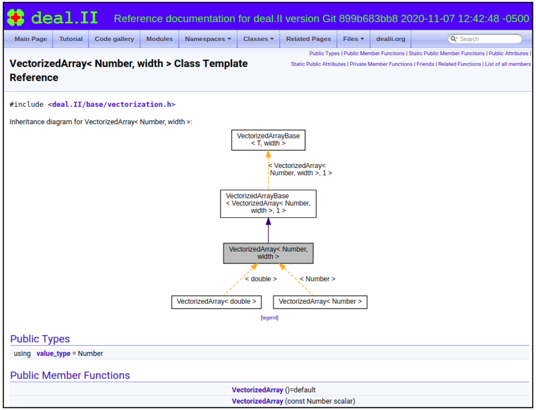
D. Arndt, W. Bangerth, B. Blais, T. C. Clevenger, M. Fehling, A. V. Grayver, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, R. Rastak, I. Thomas, B. Turcksin, Z. Wang, and D. Wells. The deal.II Library, Version 9.2. *Journal of Numerical Mathematics.* 2020.
DOI: https://doi.org/10.1515/jnma-2020-0043

Official webpage

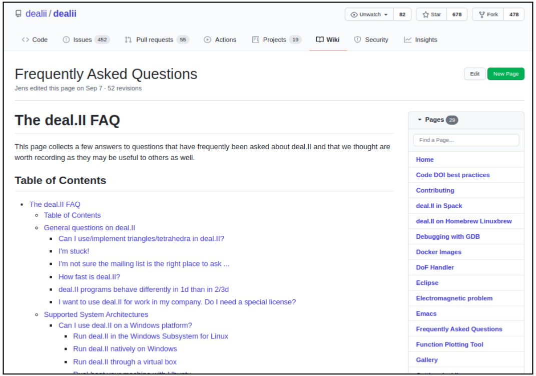


... www.dealii.org

Documentation



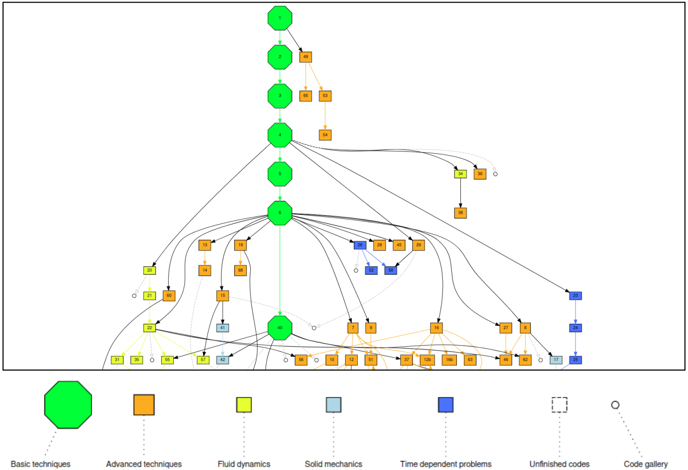
Extensive Doxygen documentation



GitHub Wiki

Documentation (cont.)

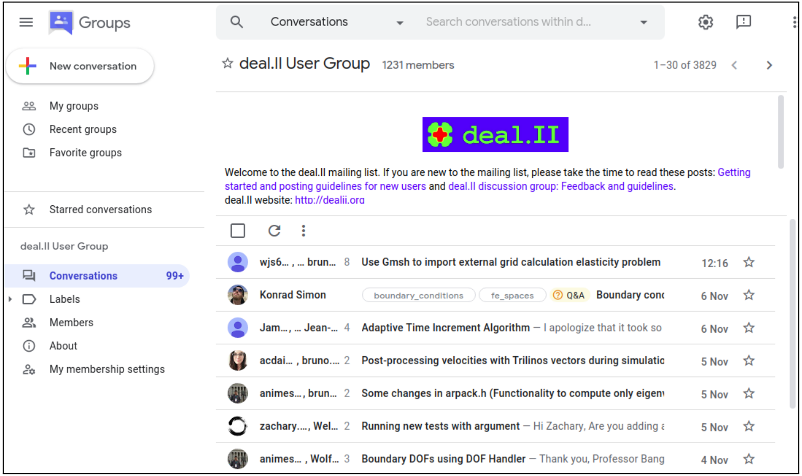
69 tutorials and code gallery:



... further 6 tutorials: work in progress

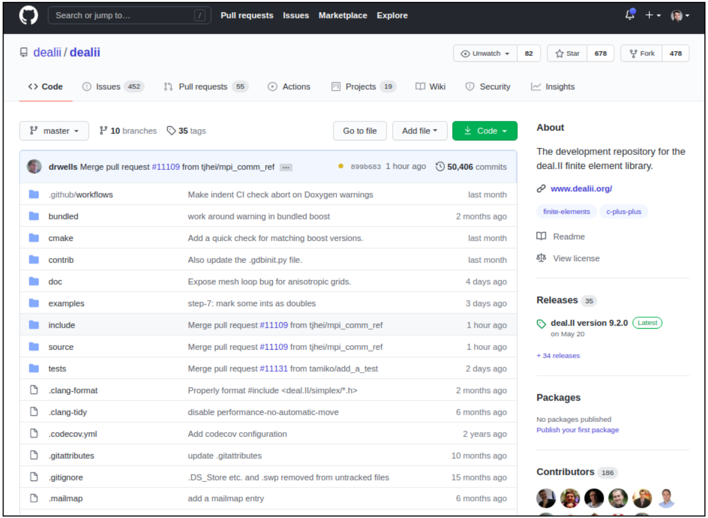
Forum

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

Login All Dashboards Saturday, November 07 2020 18:48:28

Deal.II

Dashboard Calendar Previous Current Project

3 hours ago: 13 tests failed on GNU-9.3.0-master-debian-11
4 hours ago: 12 tests failed on GNU-9.3.0-master-ubuntu-lts-20
7 hours ago: 28 tests failed on GNU-9.3.0-master
7 hours ago: 1 warning introduced on GNU-9.3.0-master
8 hours ago: 38 tests failed on GNU-10.1.0-master-tets

See full feed

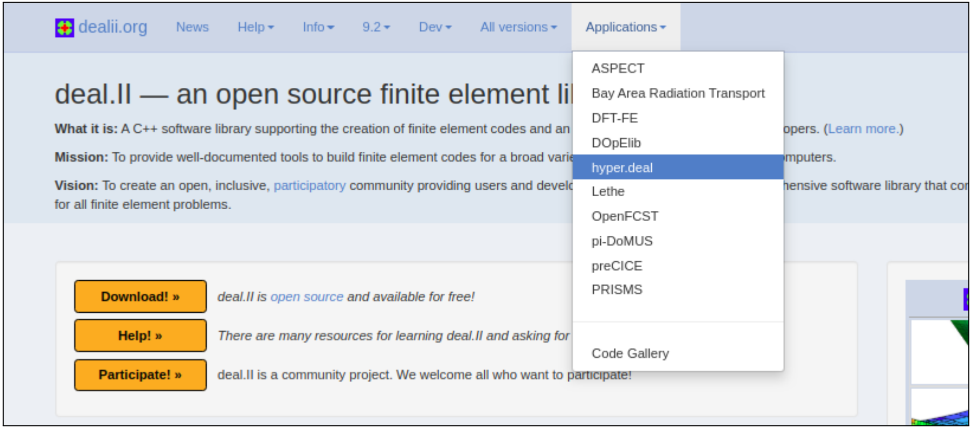
Regression Tests 150 builds

Site	Build Name	Update	Configure		Build		Test			
		Revision	Error	Warn	Error	Warn	Not Run	Fail	Pass	Start Time
tester	Clang-10.0.0-master-avx2-Ofast	10ceee	0	0	0	0	0	50 +2	12417 +2 -2	20 hours ago
tester	Clang-10.0.0-master-avx2-Ofast	414559	0	0	0	0	0	49 +1	12404 -1	Nov 03, 2020 - 07:54 UTC
tester	Clang-10.0.0-master-avx2-Ofast	2bdb45	0	0	0	0	0	48	12417 +2	Nov 06, 2020 - 01:00 UTC
tester	Clang-10.0.0-master-avx2-Ofast	cd2846	0	0	0	0	0	48	12415 +2 -2	Nov 05, 2020 - 03:17 UTC
tester	Clang-10.0.0-master-avx2-Ofast	220f05	0	0	0	0	0	48 -1	12415 +11	Nov 04, 2020 - 05:34 UTC
tester	GNU-10.1.0-master-tets	10ceee	0	0	0	0	0	38	11866 +2	10 hours ago
tester	GNU-10.1.0-master-tets	2bdb45	0	0	0	0	0	38	11864 +2	Nov 06, 2020 - 10:56 UTC

... more than 5,000 tests run for different compilers/hardware/configurations

Applications

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



... motivation for further development

Part 2:

Poisson problem/introduction into the main modules

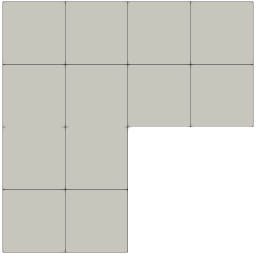
Poisson problem

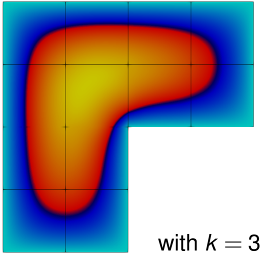
- strong form:

$$\nabla^2 u = f \text{ on } \Omega, \quad u|_\Gamma = 0$$

- weak form:

$$(\nabla v, \nabla u)_\Omega = (v, f)_\Omega, \quad u|_\Gamma = 0$$





with $k = 3$, $f = 1$

- introduce cells & use scalar Lagrange finite element $\mathcal{Q}_k$:

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

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

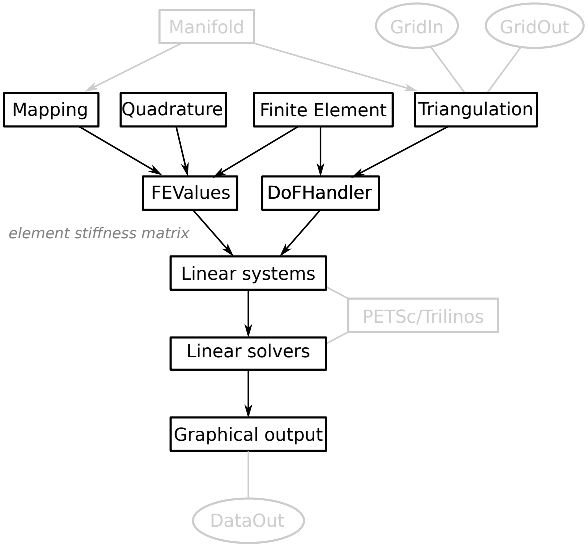
- loop over all cells, assemble system matrix and right-hand-side vector, and solve system

Main modules

needed from a FEM library:

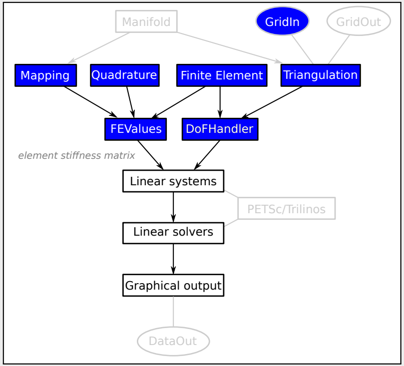
- mesh handling
- finite elements
- quadrature rules
- mapping rules
- assembly procedure
- linear solver

deal.II main modules $\rightarrow$



Example

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

parallel::distributed::Triangulation<dim> tria(MPI_COMM_WORLD);
util::create_reentrant_corner(tria);

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

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
LinearAlgebra::distributed::Vector<double> x, b;
TrilinosWrappers::SparseMatrix             A;
util::initialize_dof_vector(dof_handler, x); util::initialize_dof_vector(dof_handler, b);
util::initialize_system_matrix(dof_handler, constraints, A);
```




```
// 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_values | update_gradients | update_JxW_values);
```

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

Example (cont.)

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

    const auto dofs_per_cell = cell->get_fe().n_dofs_per_cell();  // allocate memory for element matrix/vector
    cell_matrix.reinit(dofs_per_cell, dofs_per_cell);             //
    cell_rhs.reinit(dofs_per_cell);                               //

    for (const auto q : fe_values.quadrature_point_indices())  // compute element matrix/vector
      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) *    //
                                fe_values.shape_grad(j, q) *    //
                                fe_values.JxW(q));              //
                                                                //
    for (const auto q : fe_values.quadrature_point_indices())  //
      for (const auto i : fe_values.dof_indices())              //
        cell_rhs(i) += (fe_values.shape_value(i, q) *           //
                        1. *                                    //
                        fe_values.JxW(q));                      //

    local_dof_indices.resize(cell->get_fe().dofs_per_cell);                        // assembly
    cell->get_dof_indices(local_dof_indices);                                      //
    constraints.distribute_local_to_global(cell_matrix, cell_rhs,  local_dof_indices, A, b); //
  }                                                                                //
                                                                                   //
b.compress(VectorOperation::add); A.compress(VectorOperation::add);                //
```

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

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

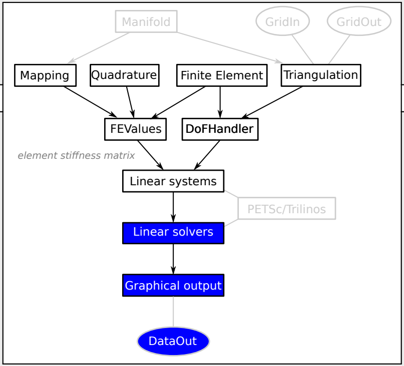
$$\underset{e}{\mathsf{A}} \mathbf{K}^{(e)}, \underset{e}{\mathsf{A}} \mathbf{f}^{(e)}$$

Example (cont.)

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

if (Utilities::MPI::this_mpi_process(util::get_mpi_comm(tria)) == 0)
  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 (e.g. VTK, VTU, Tecplot, HDF5, svg, gnuplot, ...)
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);
x.update_ghost_values();
data_out.add_data_vector(dof_handler, x, "solution");
data_out.build_patches(mapping, degree + 1);
data_out.write_vtu_with_pvtu_record("./", "result", 0, MPI_COMM_WORLD);
```



Full code: https://github.com/peterrum/dealii-examples/blob/master/poisson.cc

Example (cont.)

Some general remarks:

- deal.II provides classes that can be used - parameters have to be handled by user code
- time integration is not a first-class citizen of deal.II: users need to implement their own time-integration schemes or rely on external packages[2]

[2] deal.II has a wrapper to SUNDIALS.

Part 3:

Elasticity

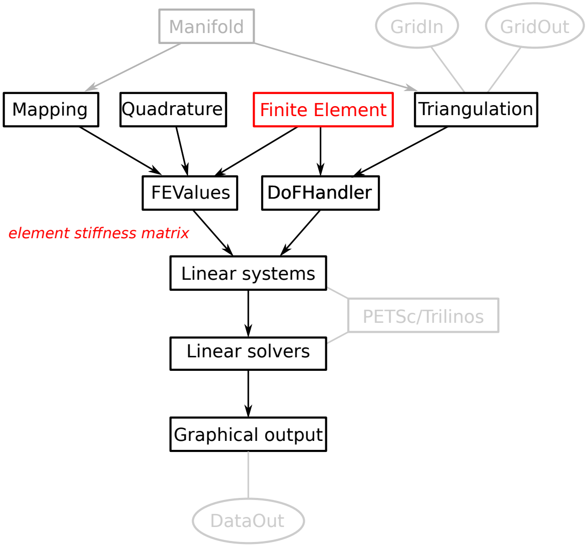
Main modules

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

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

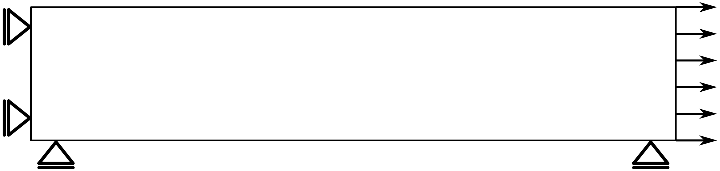
definition of **element stiffness matrix $\mathbf{K}^{(e)}$**:

$$\mathbf{K}^{(e)} = \int_{\Omega^{(e)}} \mathbf{B}^T \mathbf{C} \, \mathbf{B} \, \mathrm{d}\Omega$$



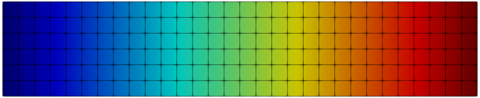
Example

Tension rod:

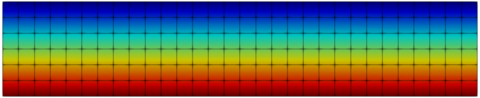


... with the result:

x displacement:



y displacement:



... full code: https://github.com/peterrum/dealii-examples/blob/master/elasticity.cc

Only relevant differences are shown:

- system of scalar finite elements:

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

Example (cont.)

- computation of $\mathbf{K}^{(e)} = \int_{\Omega^{(e)}} \mathbf{B}^T \mathbf{C}\,\mathbf{B}\,\mathrm{d}\Omega$: ▷ *according to step-18*

```
for (const auto i : fe_values.dof_indices())                      // with SymmetricTensor<4, dim> C;
  for (const auto j : fe_values.dof_indices())
    for (const auto q : fe_values.quadrature_point_indices())
      {
        SymmetricTensor<2, dim> eps_phi_i = get_strain(fe_values, i, q), eps_phi_j = get_strain(fe_values, j, q);

        cell_matrix(i, j) += (eps_phi_i * C * eps_phi_j ) * fe_values.JxW(q); // (B^T C B)\cdot|J|xW
      }
```

```
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;
  }
```

Part 4:

Volume and surface coupling

Volume coupling: a monolithic view

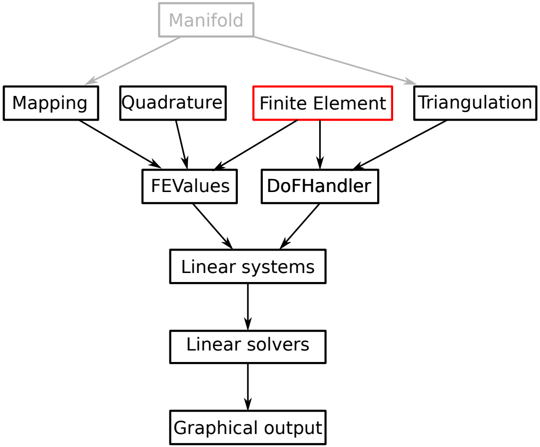
Example: TSI

- thermo-structural element:

$$\underbrace{[\mathcal{Q}_p^d, \dots, \mathcal{Q}_p^d]}_{\times d+1}$$

- alternatively with $p_t \neq p_u$:

$$\left[\mathcal{Q}_{p_t}^d, \underbrace{[\mathcal{Q}_{p_u}^d, \dots, \mathcal{Q}_{p_u}^d]}_{\times d}\right]$$




```
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

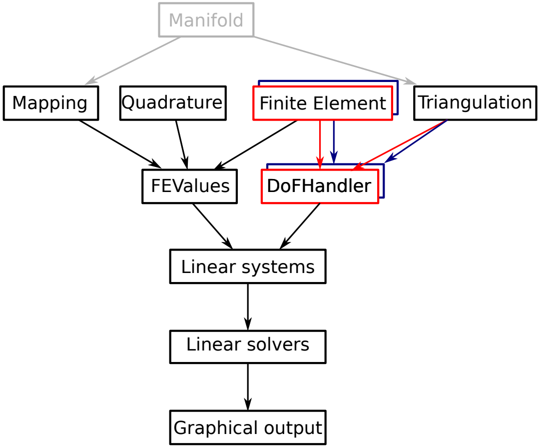
Example: TSI

- 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

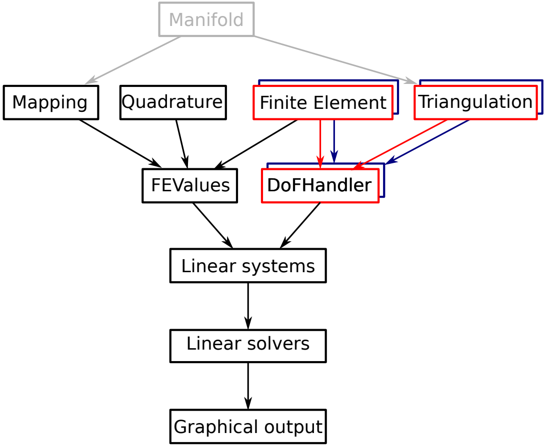
- elasticity:

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

- 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., preCICE adapter can be used



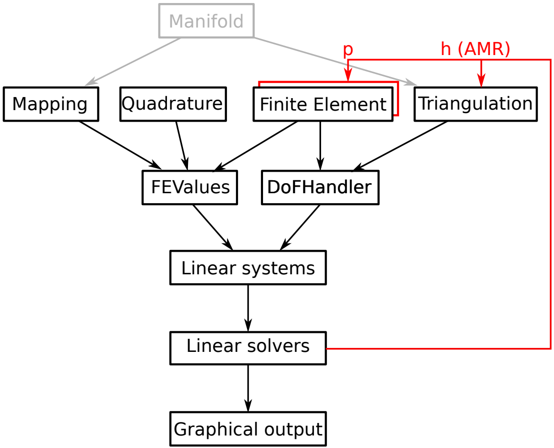
Part 5:

h/p/hp-adaptivity

Main modules

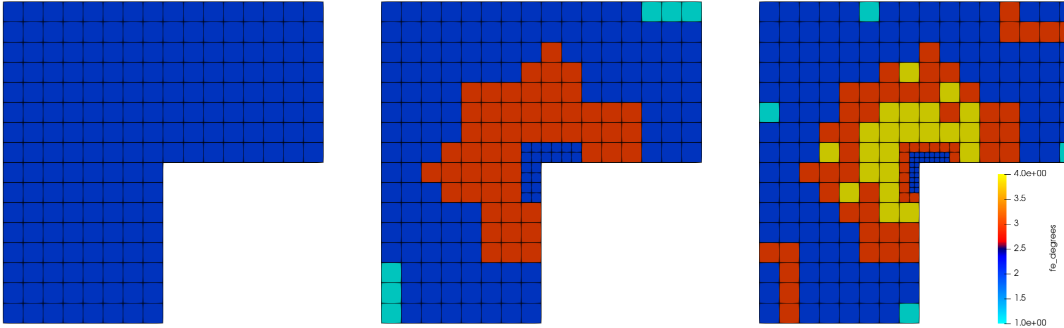
support of:

- 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

Part 6:

Matrix-free operator evaluation

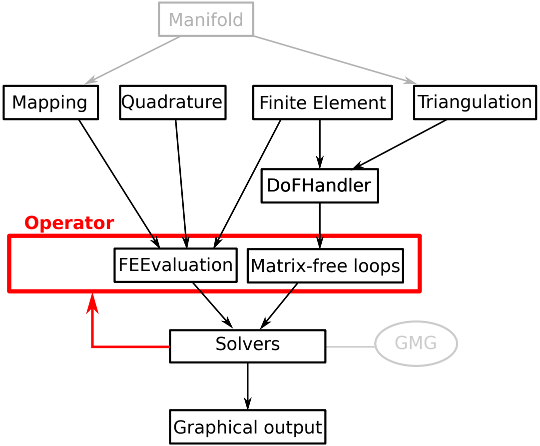
Fast matrix-free operator evaluation

idea: rely on operator evaluations

`vmult(dst, src)`

features:

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



M. Kronbichler, K. Kormann, A generic interface for parallel cell-based finite element operator application. *Comput. Fluids* 63:135–147, 2012

M. Kronbichler, K. Kormann, Fast matrix-free evaluation of discontinuous Galerkin finite element operators. *ACM TOMS 45(3):1-40*, 2019

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);          // on a cell batch
                                                                        //
        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);
```

$$\longrightarrow \mathcal{J}_q^{-1}|\mathcal{J}_q| w_q \mathcal{J}_q^{-T} \nabla \hat{u}_q$$

$$(\nabla v, \nabla u)_{\Omega^{(e)}}$$

$$\downarrow$$

$$\uparrow$$

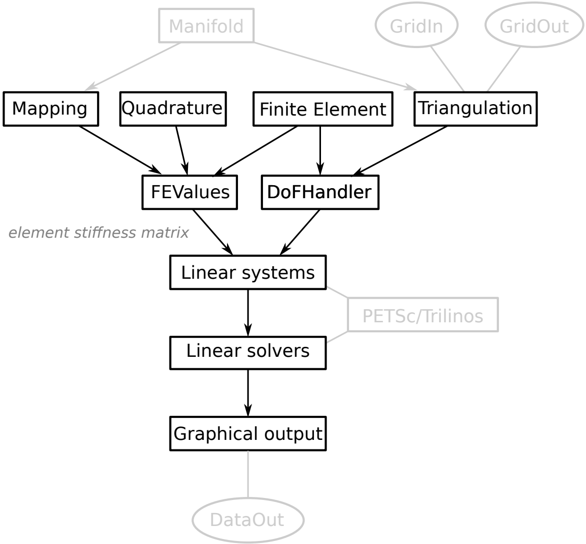
$$\sum_q (\nabla_\xi v, \mathcal{J}_q^{-1}|\mathcal{J}_q| w_q \mathcal{J}_q^{-T} \nabla_\xi u)$$

... full code: https://github.com/peterrum/dealii-examples/blob/master/poisson-mf.cc

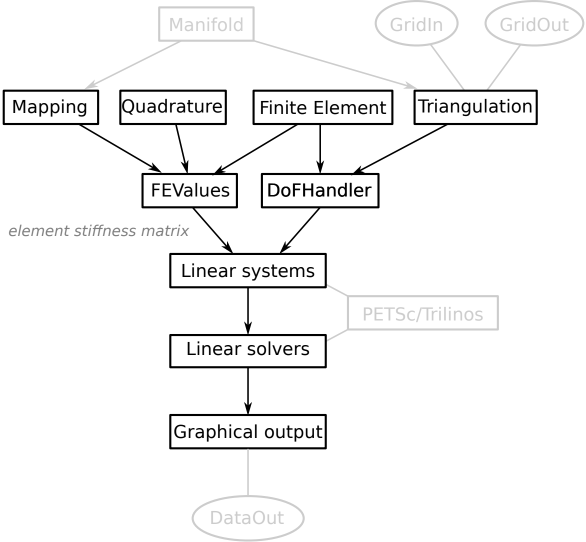
Part 7:

Summary

Triangle and tetrahedron support?

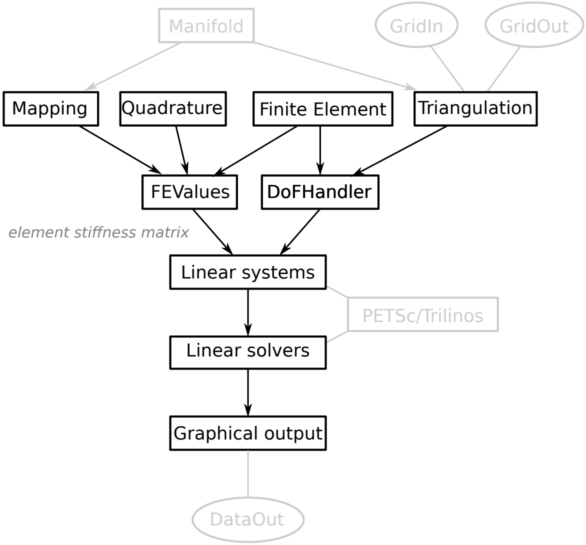


Triangle and tetrahedron support?



Question 1:
How would you proceed to use simplicities (triangles, tetrahedrons)?

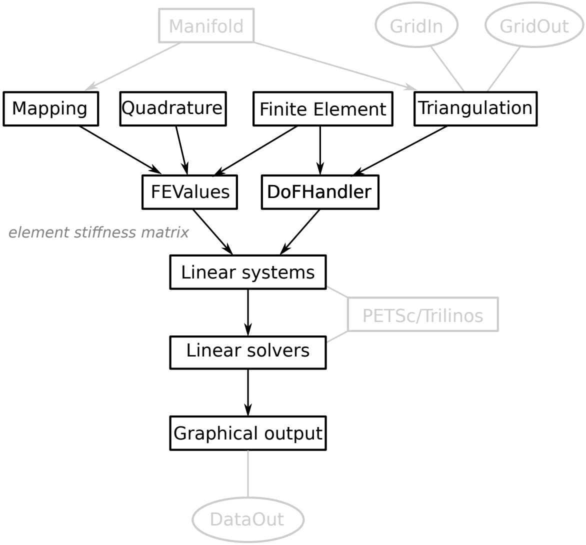
Triangle and tetrahedron support?



Question 1:
How would you proceed to use simplicities (triangles, tetrahedrons)?

```
using namspace Simplex;
FE_P<dim>       fe(degree);
QGauss<dim>     quad(degree + 1);
MappingFE<dim>  mapping(FE_P<dim>(1));
```

Triangle and tetrahedron support?



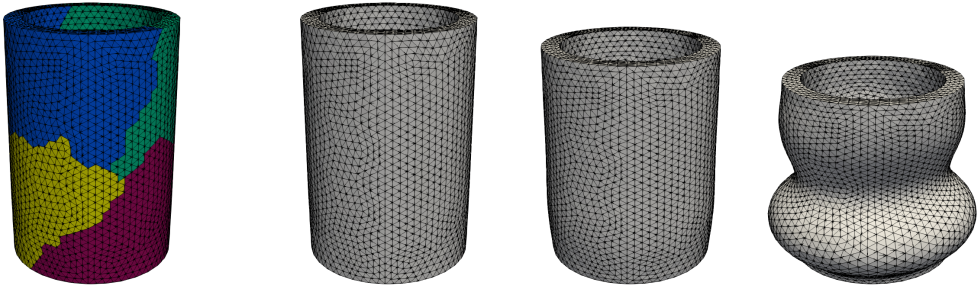
Question 1:
How would you proceed to use simplicities (triangles, tetrahedrons)?

```
using namespace Simplex;
FE_P<dim>       fe(degree);
QGauss<dim>     quad(degree + 1);
MappingFE<dim>  mapping(FE_P<dim>(1));
```

Question 2:
How would you proceed to add simplex support to a library with pure hexahedral mesh support?

Triangle and tetrahedron: yes, with dealii/master!

step-18 with p::s::T (5 procs):
55,168 cells, 251,028 DoFs



Questions/comments?