Recent advances in deal.II: matrix-free, multigrid, non-matching, and simplex support

9th deal.II Users and Developers Workshop

Martin Kronbichler[1,2], Peter Munch[2,3]

in collaboration with Maximilian Bergbauer, Niklas Fehn, Dmytro Sashko, Magdalena Schreter, . . .

[1]Department of Information Technology, Division of Scientific Computing, Uppsala University

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

[3]Kontinuumssimulationen, Institut für Werkstoffsystem-Modellierung, Helmholtz-Zentrum Hereon, Germany

June 18, 2021

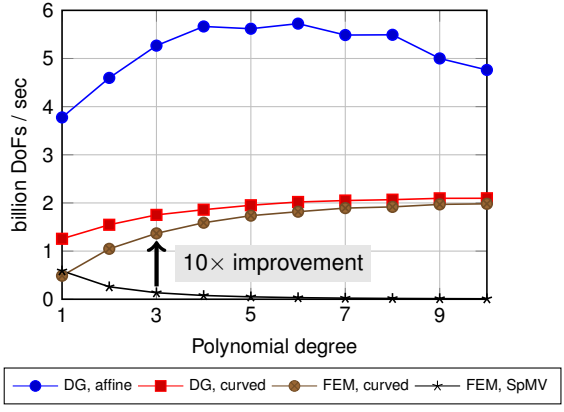
Part 1:

Matrix-free methods

Why matrix-free methods?

- Performance of matrix-vector product essential for iterative solvers
- Classical approach: sparse matrix (CRS, SELL-C)
- Sparse matrices bad fit for modern hardware: **memory-bandwidth** limited
- Inappropriate format for higher polynomial degrees, $p \geq 2$
- Matrix-free algorithm: trade computations for less memory transfer
  - Specify operation at quadrature points
  - Combine with reference cell interpolation matrices
  - Indirect access into vector entries for continuous FEM

Throughput of matrix-vector product (unknowns processed per second) in 3D



System: 1 node of 2×24 cores of Intel Xeon Platinum 8174 (Skylake)
Memory bw: 205 GB/s, arithmetic peak 3.5 TFlop/s

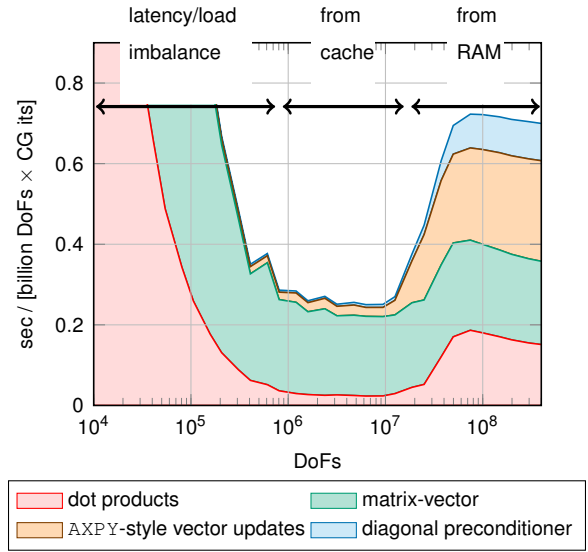
Kronbichler, Wall: A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. SISC 40(5):A3423–48, 2018
Kronbichler, Kormann: Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM TOMS 45(3):29/1–40, 2019

High-order matrix-free capabilities of deal.II

- Matrix-free infrastructure
  - Container of various pre-computed quantities in class `MatrixFree`
  - Evaluator classes `FEEvaluation` and `FEFaceEvaluation`
  - Optimized for **fast integrals through sum factorization techniques**
- Matrix-free solver infrastructure highly attractive for degrees ≥ 2
  - Highly optimized for continuous and discontinuous elements via cell and face integrals
  - Primarily CPU-based optimizations (**SIMD, memory access**), basic support for GPUs
- Mesh adaptivity, native parallel vectors + basic MPI facilities
- Various flavors of **multigrid solvers**, see below
- `step-37`, `step-48`, `step-50`, `step-59`, `step-64`, `step-66`, `step-67`, `step-75`, `step-76`

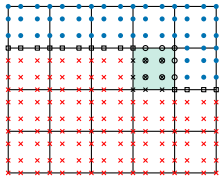
Performance of matrix-free solver: 2×64 core AMD Epyc

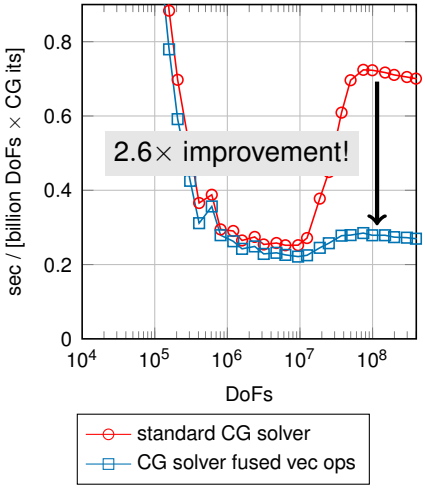
- CEED benchmark problem BP4
  - 3D Poisson, deformed geometry, continuous elements
  - Degree $p = 5$, quadrature with $n_q = 7$ per direction
  - Conjugate gradient + diagonal preconditioner
  - Metric terms computed on the fly from tri-quadratic geometry
- 1 node of dual-socket AMD Epyc 7742
  - bandwidth from RAM: 400 GB/s theory, $\sim$ 300 GB/s achievable
  - 4.6 TFlop/s
- Matrix-vector product no longer dominant operation for large sizes
- Vector operations take significant share of time when from RAM



Ongoing work: Overlap mat-vec with vector operations

- Performance opportunities at leading edge: Combine arithmetic intensive matrix-vector product with memory-heavy vector operations
- `MatrixFree::cell_loop` can run vector updates before Ap_k first touches vector entries, dot products after Ap_k last touches vector entries
  - 87% of vector entries accessed 1× per CG it
  - **2–3× performance improvement**
- Algorithms to be improved: `SolverCG`, `PreconditionChebyshev` work-in-progresss, maybe also `SolverBiCGStab`?



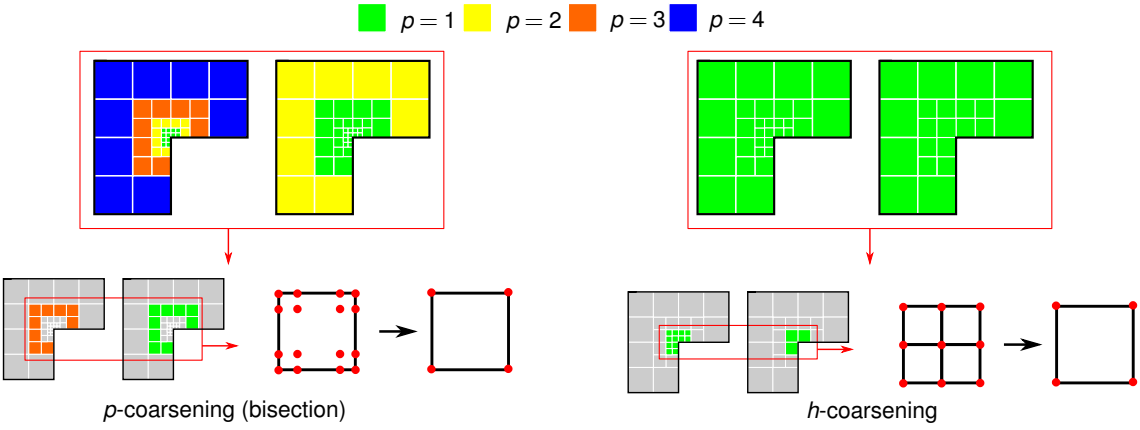


Part 2:

Global coarsening multigrid support

Global coarsening

Global coarsening builds multigrid levels for entire domain, coarsening *all* cells of a triangulation regardless of how many times they have been refined.



R. Becker and M. Braack. Multigrid techniques for finite elements on locally refined meshes. Numerical linear algebra with applications, 7(6):363379, 2000

Relevant functions & classes in deal.II

- set up each two-level transfer operator:

```
MGLevelObject<MGTwoLevelTransfer<dim, VectorType>> transfers;

// loop over all levels and set up geometric transfer or ...
transfers[level + 1].reinit_geometric_transfer(dof_handlers[level + 1],
                                               dof_handlers[level],
                                               constraints[level + 1],
                                               constraints[level]);

// ... polynomial transfer
transfers[level + 1].reinit_polynomial_transfer(dof_handlers[level + 1],
                                                dof_handlers[level],
                                                constraints[level + 1],
                                                constraints[level]);
```

- set up final global-coarsening transfer operator:

```
MGTransferGlobalCoarsening<dim, VectorType> transfer(
    transfers,
    [&](const auto l, auto &vec) {operators[l]->initialize_dof_vector(vec);});
```

Relevant functions & classes in deal.II (cont.)

- create polynomial coarsening sequence:

```
std::vector<unsigned int>
MGTransferGlobalCoarseningTools::create_polynomial_coarsening_sequence (
  const unsigned int max_degree,
  const PolynomialCoarseningSequenceType & p_sequence);
```

... “go to one”, “bisect”, “decrease by one”

- create geometric coarsening sequence:

```
template<int dim, int spacedim>
std::vector<std::shared_ptr<const Triangulation<dim, spacedim> > >
MGTransferGlobalCoarseningTools::create_geometric_coarsening_sequence (
  const Triangulation< dim, spacedim > & tria);
```

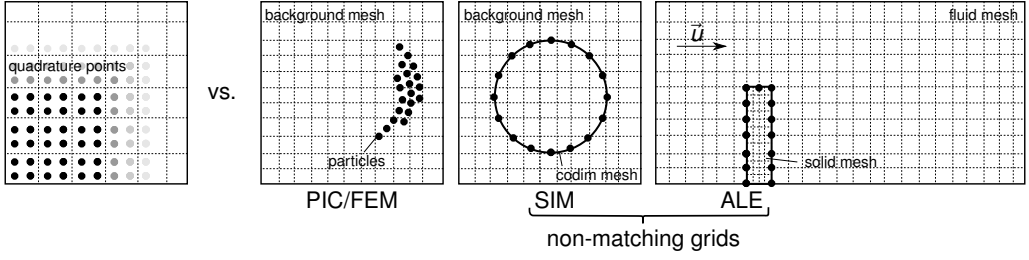
... result: sequence of triangulations

Part 3:

Non-matching support

FEPointEvaluation

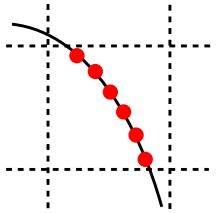
- new class: **`FEPointEvaluation`**
- interface is similar to `FEEvaluation` (with `evalute()`/`integrate()`)
- specialized for evaluation in points at arbitrary locations within a cell



Example: evaluate surface tension

sharp-interface method:

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



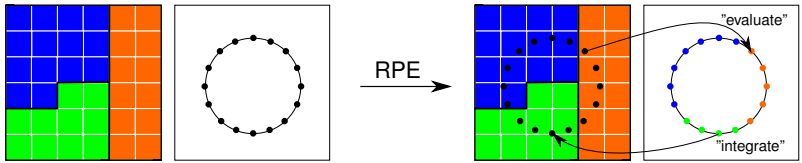
... with $\mathbf{n} = \nabla\phi/|\nabla\phi|$ and $\kappa = \Delta\phi$, based on level-set field ϕ

in code:

```
phi_normal.reinit(cell, reference_points);                    // same for phi_curv/phi_force

phi_normal.evaluate(normal_values, EvaluationFlags::values);
phi_curv.evaluate(curvature_values, EvaluationFlags::values);

for (unsigned int q = 0; q < n_points; ++q)
   phi_force.submit_value(phi_curv.get_value(q) * phi_normal.get_value(q) * JxW[q], q);

phi_force.integrate(force_values, EvaluationFlags::values);
```

Finding cells and reference positions

- `ParticleA::get_surrounding_cell()` and `get_reference_location()`
- `GridTools::find_(all_)active_cell(s)_around_point()`
- **`RemotePointEvaluation`** → for surface and volume coupling of non-matching grids



find cells + set up communication pattern (with the help of bounding boxes)

... also used in `VectorTools::point_values()`

see also application codes: `adaflo`, `ASPECT`, `ExaDG`, and `MeltPoolDG`

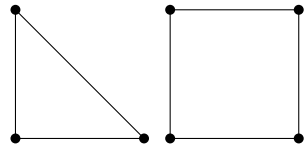
Part 4:

Experimental simplex and mixed mesh support[1]

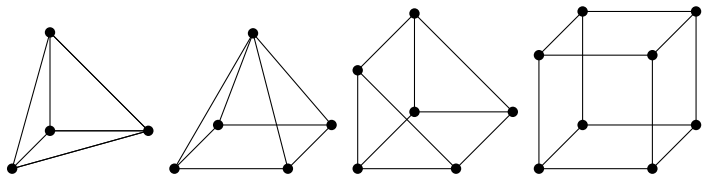
[1] with contributions by Pasquale Africa, Wolfgang Bangerth, Nicolas Barnafi, Maximilian Bergbauer, Bruno Blais, Michele Bucelli, Elias Dejene, Marco Fedele, Marc Fehling, Niklas Fehn, Ivan Fumagalli, Nicola Giuliani, Luca Heltai, Vladimir Ivannikov, Katharina Kormann, Martin Kronbichler, Olivier Guevremont, Timo Heister, Wenyu Lei, Nils Much, Peter Munch, Daniel Pauckner, Laura Prieto, Sebastian Proell, Magdalena Schreter, David Wells, Jiaqi Zhang, ...

Reference-cell types

- reference-cell types in 0D: vertex
- reference-cell types in 1D: line
- reference-cell types in 2D: triangle, quadrilateral

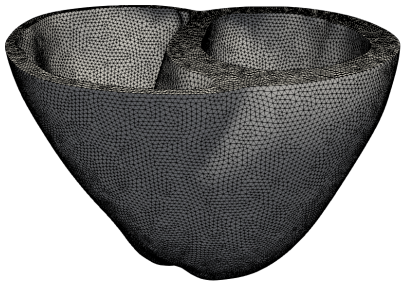


- reference-cell types in 3D: tetrahedron, pyramid, wedge/prism, hexahedron

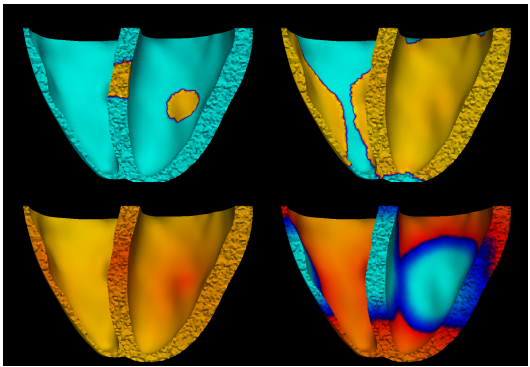


Application examples

Monodomain model for Cardiac Electrophysiology in a left+right ventricle geometry:



... submitted by Pasquale Africa (life^x)[2]

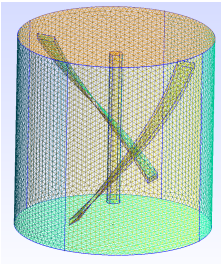


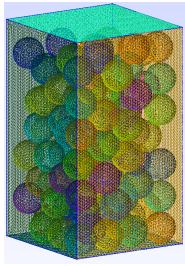
transmembrane potential

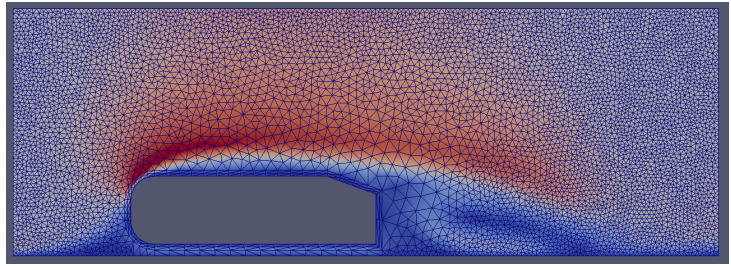
[2] `https://lifex.gitlab.io/lifex/`; MOX - Department of Mathematics - Politecnico di Milano; founded by "iHEART, ERC Advanced Grant, Project ID: 740132 (PI: Prof. A. Quarteroni)"

Application examples (cont.)

Simulation of an impeller, of a porous domain, and of an Ahmed body:



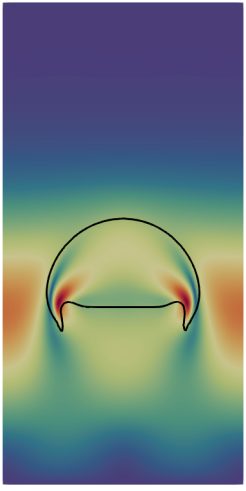


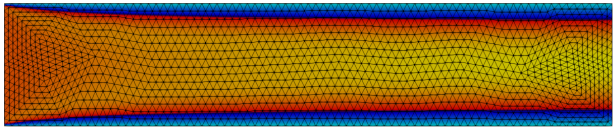


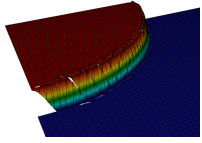
... submitted by Bruno Blais and Olivier Guevremont (Lethe-CFD)[3]

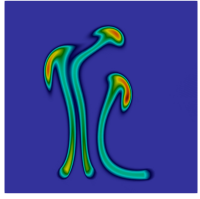
[3] `https://github.com/lethe-cfd/lethe`; Polytechnique Montreal

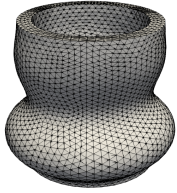
Application examples (cont.)



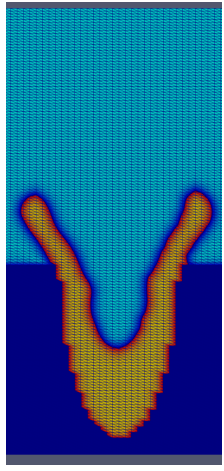








by Magdalena Schreter

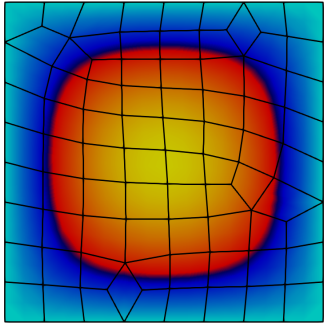


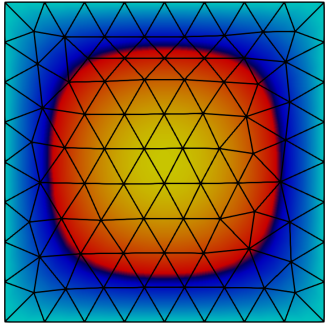
see also: adaflo and MeltPoolDG[4]

[4] https://github.com/kronbichler/adaflo and https://github.com/MeltPoolDG/MeltPoolDG

Example: simplex mesh

Solution of the Poisson problem on a pure quadrilateral and a pure triangle “Gmsh” mesh:





Gmsh journals:

```
Rectangle(1) =
  {0, 0, 0, 1, 1, 0};
Recombine Surface{1};
Mesh 2;
Save "box_2D_quad.msh";
```

```
Rectangle(1) =
  {0, 0, 0, 1, 1, 0};
Mesh 2;
Save "box_2D_tri.msh";
```

... online module page “Simplex support” → “Example: simplex mesh”

Example: simplex mesh (cont.)

New classes:

- finite elements: `FE_SimplexP`, `FE_SimplexDGP`, ... with $p \leq 2$
- quadrature rules: `QDuffy`, `QGaussSimplex`, `QWitherdenVincentSimplex`, ...
- mapping: `MappingFE`, `MappingFEFields`

Generalized classes:

- `Triangulation`[5] ... parallel: `p::s::T` and `p::f::T`
- `GridIn`/`GridOut`/`DataOut`: vtk, vtu, Gmsh, hdf5, ExodusII

[5] limited support for adaptivity

Example: simplex mesh (cont.)

step-3

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

```
const FE_Q<2>          fe;
const QGauss<2> quadrature_formula;
```

```
FEValues<2> fe_values(fe, ...);
```

```
VectorTools::interpolate_boundary_values(
  dof_handler, 0, ...);
```

```
data_out.build_patches();
```

step-3 with simplex mesh

```
#include <deal.II/fe/fe_simplex_p.h>
#include <deal.II/fe/mapping_fe.h>
```

```
const MappingFE<2>      mapping;
const FE_SimplexP<2>    fe;
const QGaussSimplex<2> quadrature_formula;
```

```
FEValues<2> fe_values(mapping, fe, ...);
```

```
VectorTools::interpolate_boundary_values(
  mapping, dof_handler, 0, ...);
```

```
data_out.build_patches(mapping, 2);
```

Implications for usage

If you need flexibility to switch between cell types, use smart pointers!

```
std::unique_ptr<Mapping<dim, spacedim>>       mapping;
std::unique_ptr<FiniteElement<dim, spacedim>> fe;
std::unique_ptr<Quadrature<dim>>              quadrature;
```

... which are set up centrally and are passed around via a ScratchData!

Implications for usage (cont.)

RECOMMENDATION: do not use `GeometryInfo`!

The class `TriaAccessor` has been extended with new methods. E.g.:

```
for(const auto & cell : tria)
  for(const auto v :
      GeometryInfo<dim>::face_indices())
    // do something
```

old

```
for(const auto & cell : tria)
  for(const auto v :
      cell->face_indices())
    // do something
```

new

... to query the number of subentities and to iterate over them.

Last resort: to get the type of cells (`ReferenceCell`) via:

```
if(cell->reference_cell() == ReferenceCells::Hexahedron)
  // do something
```

Implications for usage (cont.)

WARNING: you cannot rely on default mappings! Be explicit! E.g.:

```
FEValues::FEValues(fe, quad, flags);
```

```
VectorTools::interpolate(
  dof, function, vec);
```

```
DataOut::build_patches(subdivisions);
```

implicit (MappingQGeneric(1))

```
FEValues::FEValues(mapping, fe, quad, flags);
```

```
VectorTools::interpolate(
  mapping, dof, function, vec);
```

```
DataOut::build_patches(mapping, subdivisions);
```

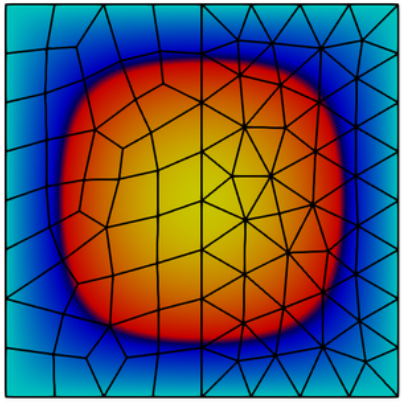
explicit

... there are many functions with default argument mapping!!!

Example: mixed mesh

Solution of the Poisson problem on a mixed “Gmsh” mesh:

▷ using *hp* infrastructure



Gmsh journal:

```
Rectangle(1) = {0, 0, 0, 0.5, 1, 0};
Rectangle(2) = {0.5, 0, 0, 0.5, 1, 0};
Recombine Surface{1};
Physical Surface("All") = {1, 2};
Mesh 2;
Coherence Mesh;
Save "box_2D_mixed.msh";
```

... online module page “Simplex support” → “Example: mixed mesh”

Resources & limitations

- “Simplex support” module page
- upcoming release paper
- test folder “tests/simplex” with ported versions of steps:
  1-4, 6-8, 12, 17, 18, 20, 23, 31, 38, 40, 55, 67, 68, and 74
- application codes: `adaflo`, `Lethe-CFD`, and `MeltPoolDG`

EXPERIMENTAL simplex and mixed mesh support ... ***help wanted!***

- limitations: no adaptivity, limited number of low-order element & quadrature rules ...
- possible bugs
- possible changing interfaces (in particular for mixed meshes)

Recent advances in deal.II: matrix-free, multigrid, non-matching, and simplex support

Martin Kronbichler, Peter Munch

Questions?