Original

Arndt, D.; Bangerth, W.; Blais, B.; Fehling, M.; Gassmöller, R.; Heister, T.; Heltai, L.; Kronbichler, M.; Köcher, U.; Maier, M.; Munch, P.; Pelteret, J.; Proell, S.; Simon, K.; Turcksin, B.; Wells, D.; Zhang, J.:
The deal.II library, Version 9.3.
In: Journal of Numerical Mathematics. Vol. 29 (2021) 3, 171 - 186.

First published online by de Gruyter: 25.09.2021

https://dx.doi.org/10.1515/jnma-2021-0081

Daniel Arndt, Wolfgang Bangerth, Bruno Blais, Marc Fehling, Rene Gassmöller, Timo Heister*, Luca Heltai, Uwe Köcher, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Sebastian Proell, Konrad Simon, Bruno Turcksin, David Wells, and Jiaqi Zhang

The deal.II library, Version 9.3

https://doi.org/10.1515/jnma-2021-0081
Received June 25, 2021; accepted June 30, 2021

Abstract: This paper provides an overview of the new features of the finite element library deal.II, version 9.3.

Keywords: software, finite elements, deal.II

Classification: 65M60, 65N30, 65Y05

1 Overview

deal.II version 9.3.0 was released June 17, 2021. This paper provides an overview of the new features of this release and serves as a citable reference for the deal.II software library version 9.3. deal.II is an object-oriented finite element library used around the world in the development of finite element solvers. It is available for free under the GNU Lesser General Public License (LGPL). Downloads are available at https://www.dealii.org/ and https://github.com/dealii/dealii.

The major changes of this release are:

- Experimental support for simplex and mixed meshes (see Section 2.1);
- Improved flexibility of the particle infrastructure (see Section 2.2);
- Support for global-coarsening multigrid algorithms (see Section 2.3);
- Advances in the matrix-free infrastructure (see Section 2.4);
- Usage of MPI-3.0 shared-memory features to reduce memory footprint (see Section 2.5);

Daniel Arndt, Bruno Turcksin, Scalable Algorithms and Coupled Physics Group, Computational Sciences and Engineering Division, Oak Ridge National Laboratory, 1 Bethel Valley Rd., TN 37831, USA.
Wolfgang Bangerth, Marc Fehling, Department of Mathematics, Colorado State University, Fort Collins, CO 80523-1874, USA.
Wolfgang Bangerth, Department of Geosciences, Colorado State University, Fort Collins, CO 80523, USA.
Bruno Blais, Research Unit for Industrial Flows Processes (URPEI), Department of Chemical Engineering, Polytechnique Montréal, PO Box 6079, Stn Centre-Ville, Montréal, Québec, Canada, H3C 3A7.
Rene Gassmöller, Department of Geological Sciences, University of Florida, 1843 Stadium Road, Gainesville, FL 32611, USA.
***Corresponding author: Timo Heister,** School of Mathematical and Statistical Sciences, Clemson University, Clemson, SC 29634, USA. Email: heister@clemson.edu
Luca Heltai, SISSA, International School for Advanced Studies, Via Bonomea 265, 34136 Trieste, Italy.
Uwe Köcher, Chair of Numerical Mathematics, Helmut-Schmidt-University, University of the Federal Armed Forces Hamburg, Holstenhofweg 85, 22043 Hamburg, Germany.
Martin Kronbichler, Peter Munch, Sebastian Proell, Institute for Computational Mechanics, Technical University of Munich, Boltzmannstr. 15, 85748 Garching, Germany.
Martin Kronbichler, Department of Information Technology, Uppsala University, Box 337, 751 05 Uppsala, Sweden.
Matthias Maier, Department of Mathematics, Texas A&M University, 3368 TAMU, College Station, TX 77845, USA.
Peter Munch, Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Max-Planck-Str. 1, 21502 Geesthacht, Germany.
Jean-Paul Pelteret, Independent researcher.
Konrad Simon, Department of Mathematics/Center for Earth System Research and Sustainability (CEN), University of Hamburg, Grindelberg 5, 20144 Hamburg, Germany.
David Wells, Department of Mathematics, University of North Carolina, Chapel Hill, NC 27516, USA.
Jiaqi Zhang, School of Mathematical and Statistical Sciences, Clemson University, Clemson, SC 29634, USA.

– Improved support for evaluation and integration at arbitrary points (see Section 2.6);
– Simplified implementation for face integrals (see Section 2.7);
– Nine new tutorial programs and a new code gallery program (see Section 2.9).

In addition, we discuss the `candi` installation program in Section 2.8.

While all of these major changes are discussed in detail in Section 2, there are a number of other noteworthy changes in the current `deal.II` release that we briefly outline in the remainder of this section:

– Each non-artificial cell now has a globally unique index that can be queried for active cells via

```
CellAccessor::global_active_cell_index()
```

and for level cells via `::global_level_cell_index()`. The information can be used to efficiently index into global vectors storing data for each cell, rather than for each degree of freedom corresponding to a finite element field.

– Previously, functions and classes were marked using the macro `DEAL_II_DEPRECATED` and then typically removed in the release after the one in which these deprecation notices were available to users. We have now extended this policy by introducing the `DEAL_II_DEPRECATED_EARLY` macro, which indicates that a feature will be deprecated in the next release. In contrast to the first macro, it will only give warnings if `deal.II` has been configured with `-D DEAL_II_EARLY_DEPRECATIONS=ON`.
– After each update of the master branch of `deal.II`, we build a new Docker image with all features enabled. It is accessible on `Docker Hub` via `dealii/dealii:master-focal`. This image is particular useful when used in the continuous-integration processes of user codes.
– A long-standing orientation issue for the vector valued finite element `FE_RaviartThomas<3>` was resolved. This issue prevented the user from using this class on meshes with cells that do not have standard orientation, i.e., cells that have faces that are reflected and/or rotated relative to their neighbors. In all eight possible neighboring configurations, we can now guarantee that the necessary $H(\mathrm{div})$-conformity condition is met for all polynomial orders. This is verified through a new conformity test on non-standard meshes.
– `deal.II` now requires compilers to support the C++14 standard [38].

The changelog lists more than 200 other features and bugfixes.

2 Major changes to the library

This release of `deal.II` contains a number of large and significant changes that will be discussed in this section. It of course also contains a vast number of smaller changes and added functionality; the details of these can be found in the file that lists all changes for this release; see [47].

2.1 Experimental simplex and mixed mesh support

The current release of `deal.II` adds experimental support for simplex meshes (consisting of triangles in 2D; tetrahedra in 3D) and mixed meshes (consisting of triangles and/or quadrilaterals in 2D; tetrahedra, pyramids, wedges, and/or hexahedra in 3D). Many freely available mesh-generation tools produce such kind of meshes and they are widely used in industry and applications, but were previously unsupported by `deal.II`. As a consequence, users of `deal.II` had to pre-process such meshes and convert them to pure quadrilateral or hexahedral meshes.

Support for simplex and mixed meshes is not universal in `deal.II` at this point. While `deal.II` can read such meshes, write output for them, and solve partial differential equations with certain finite elements, there are also many areas that have not been fully adapted to the new functionality. In particular, `deal.II` currently

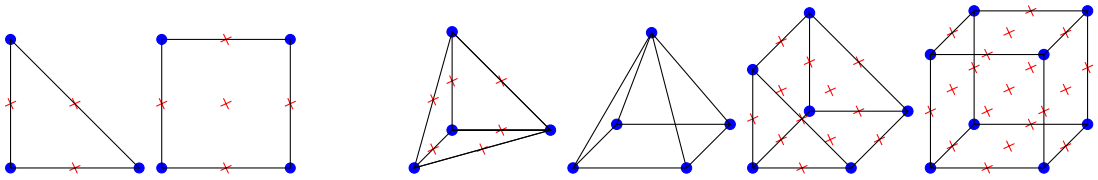

Fig. 1: Reference cells in 2D (triangle, quadrilateral) and 3D (tetrahedron, pyramid, wedge, hexahedron) with the support points indicated both for linear (•) and quadratic (×) shape functions.

Tab. 1: List of new scalar `FiniteElement` classes for the new reference-cell types. Vectorial elements can be constructed based on these classes via `FE_Systems`.

reference cell	finite element	dimension	degree
simplex (line, triangle, tetrahedron)	FE_SimplexP, FE_SimplexDGP	1–3	1–2
	FE_SimplexP_Bubbles	1–3	1–2
pyramid	FE_PyramidP, FE_PyramidDGP	3	1
wedge	FE_WedgeP, FE_WedgeDGP	3	1–2

only offers low-order finite elements on such meshes, and many utility functions might throw exceptions when used with such meshes.

A user-focused summary of information around simplex and mixed mesh support is also available on the new module page at https://www.dealii.org/current/doxygen/deal.II/group__simplex.html. In particular, it shows how to solve a simple Poisson problem like in the `step-3` tutorial program on simplex and mixed meshes, with a focus on the necessary changes to the workflow. At the time of this release, there are also 92 tests (in the folder `tests/simplex`) targeting the new simplex and mixed mesh support. In particular, the folder also contains ported variants of a number of existing tutorials: 1, 2, 3, 4, 6, 7, 8, 12, 17, 18, 20, 23, 31, 38, 40, 55, 67, 68, and 74.

2.1.1 Refactoring of internal data structures

To enable simplex and mixed mesh support, we performed a major refactoring of the internal data structures of `deal.II`. In particular, the `Triangulation` class and the `DoFHandler` class have undergone large changes and now support meshes composed of all of the cells shown in Fig. 1.

The template parameters of the internal data structures of `Triangulation` have been removed and the type of each cell and of each face (in 3D) is stored. The function `Triangulation::create_triangulation()`, which converts a given list of cells and vertices to the internal data structures, has been rewritten inspired by [48] and this has had the side effect of a speed-up of up to a factor of 5. Minor adjustments have also been made to the `parallel::shared::Triangulation` and `parallel::fullydistributed::Triangulation` classes such that the new mesh types can also be used in parallel.

The internal data structures of `DoFHandler` used to be hard-coded for pure hypercube meshes, while the `hp::DoFHandler` used to be built around CRS-like data structures. Due to the need of CRS data structures in the `DoFHandler` in the context of more general meshes, we have merged `hp::DoFHandler` into the `DoFHandler`. The class `hp::DoFHandler`, which is now a dummy derivation of `DoFHandler`, currently only exists for compatibility reasons and has been deprecated, see Section 2.10. It will be removed in the next release.

2.1.2 Generating meshes

The most obvious way to generate a simplex or a mixed mesh is to read the mesh from a file generated by an external mesh generator. Currently, we support the `VTK`, `MSH` (generated by the `GMSH` program [27]), and `EXODUS II` file formats.

Alternatively, one can create a pure hypercube mesh with the functions in the `GridGenerator` namespace and convert it to a pure simplex mesh with the function `GridGenerator::convert_hypercube_to_simplex_mesh()`.

2.1.3 Using simplex meshes

After creating a triangulation, one can proceed in the same way as for hypercube meshes. In particular, one selects an appropriate finite element, mapping, and quadrature class as follows:

C++ code

```
FE_SimplexP<dim, spacedim> fe(degree);
MappingFE<dim, spacedim> mapping(FE_SimplexP<dim, spacedim>(1));
QGaussSimplex<dim> quadrature(degree +1);

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

FEValues<dim, spacedim> fe_values(mapping, fe, quadrature, flags);
```

The list of currently supported finite element classes is provided in Table 1. Currently, only linear and quadratic mappings via the `MappingFE` and `MappingFEField` classes built around the listed nodal elements are available. For quadrature, the classes `QDuffy`, `QGaussSimplex`, `QWitherdenVincentSimplex` [58, 65], `QGaussPyamid`, and `QGaussWedge` are available.

2.1.4 Using mixed meshes

For mixed meshes, concepts known from the *hp*-context have been applied: different finite element classes are assigned to different cells based on their respective kind of reference cell. In the case of a 2D mixed mesh, which can only consist of triangles and quadrilaterals, the finite element defined on a triangle (e.g., `FE_SimplexP`) and on a quadrilateral (e.g., `FE_Q`) can be collected in a `hp::FECollection` as follows:

C++ code

```
hp::FECollection<dim, spacedim> fe
  {FE_SimplexP<dim, spacedim>(degree), FE_Q<dim, spacedim>(degree)};
```

Similarly, `hp::QCollection` and `hp::MappingCollection` can be used to construct appropriate collections. Furthermore, the correct active finite element index, which points to the correct finite element of that cell, has to be assigned to each cell. The following piece of code will then correctly enumerate all degrees of freedom on the mesh:

C++ code

```
DoFHandler<dim> dof_handler(tria);

for (const auto &cell :dof_handler.active_cell_iterators())
  switch (cell->reference_cell_type())
    {
      case ReferenceCell::Type::Tri: cell->set_active_fe_index(0); break;
```

```
      case ReferenceCell::Type::Quad: cell->set_active_fe_index(1); break;
      // 3D (Tet, Pyramid, Wedge, Hex) not shown
      default: Assert(false, ExcNotImplemented());
    }

dof_handler.distribute_dofs(fe);
```

2.1.5 Practical implications

The introduction of simplex and mixed meshes leads to some implications for the user if these features are to be used. For instance, each cell might have a different type with different number of vertices, lines, and faces so that these quantities can no longer be compile-time constants. This information used to be queried from the `GeometryInfo` class. Instead, we have extended relevant classes, e.g., `TriaAccessor` or `TriaCellAccessor`, with useful new functions like `n_vertices()`, `n_lines()`, or `n_faces()` so that users can simply write:

C++ code

```
for (const auto &cell :tria.active_cell_iterators())
  for (unsigned int f =0; f <cell->n_faces(); ++f)
    // do something with cell->face(f);
```

Alternatively, one can use an iterator-based approach to loop over all faces of a cell, introduced in the previous release. The relevant functions have been adjusted to be able to deal with the now variable number of faces per cell:

C++ code

```
for (const auto &cell :tria.active_cell_iterators())
  for (const auto &face :cell->face_iterators())
    // do something with face
```

Furthermore, for mixed meshes, the number of degrees of freedom will differ between cells so that cell-local arrays need to be resized for each cell (as has previously already been the case in the *hp*-context):

C++ code

```
Vector<double> local_rhs;
for (const auto &cell :dof_handler.active_cell_iterators())
 {
   hp_fe_values.reinit(cell);
   local_rhs.reinit(cell->get_fe().n_dofs_per_cell());
   // ...
 }
```

What is true for cells is also true for faces in 3D: faces can be either triangles or quadrilaterals. This is the case even if no mixed mesh is used if the mesh consists exclusively of pyramids or wedges. As a consequence, some functions, e.g., `FiniteElementData::n_dofs_per_face()`, have been extended with a new optional argument for the face number.

Geometric information about cells and faces — previously provided by the `GeometryInfo` class for hypercube-type cells — is now available via the `ReferenceCell` class that is defined for each of the seven possible reference cells. The correct `ReferenceCell` for a cell or face can be obtained, respectively, using `cell->reference_cell()`, and either `cell->face(f)->reference_cell()` or `cell->reference_cell().face_reference_cell(f)`.

Furthermore, many functions in `deal.II` used mappings that, when not given explicitly, defaulted to (bi-/tri-)linear ones. These no longer work for simplex or mixed meshes, so users will need to explicitly provide the correct mapping for the mesh to be used.

2.1.6 Matrix-free support

deal.II's matrix-free support has also been extended to simplex and mixed meshes, for both continuous and discontinuous elements. From a user perspective, the main changes are to pass a d-dimensional quadrature object (rather than one for 1D), and the fact that FEEvaluation and FEFaceEvaluation must not specify the polynomial degree via template arguments, determining all information at runtime. More details can be found in Section 2.4.

For the current release, no advanced algorithms for evaluating values and gradients at quadrature points, such as sum factorization, are used. The use of full interpolation matrices is for now acceptable since only low-order elements are supported.

2.2 Advances in the particle infrastructure

deal.II's particle infrastructure has been modernized to store nearly all particle data as a collection of continuous data arrays instead of a container of objects. This reorganization improves cache efficiency when iterating over particles and reduces the amount of data that needs to be moved when a particle moves to a different cell. Keeping track of unused data slots allows to reuse them for new particles, significantly reducing memory allocations when particles are created after other particles have left the local domain. The data arrays are rebuilt and sorted during every mesh refinement cycle.

Additionally, the particle infrastructure now supports a faster search algorithm for particles that moved farther than one cell width between particle sorting operations, and support for updating ghost particles (particles that live in ghost cells around the local domain) by updating their properties instead of destroying and rebuilding their container has been added. The latter step improves the efficiency of ghost particle exchange significantly. With this feature, deal.II can be used for scalable parallel Lagrangian models such as the Discrete Element Method (DEM) [30] or Molecular Dynamics.

Finally, the Particles::DataOut class now supports writing particle properties as vectors or tensors instead of a collection of scalars if the properties are marked as such.

2.3 Advances in the multigrid infrastructure

Until now, deal.II has only supported 'local smoothing' multigrid algorithms [18] wherein smoothers only act on the cells of a given refinement level, skipping those parts of the mesh that have not been adaptively refined to that level. This approach guarantees that the work done summed up over all levels is proportional to the number of unknowns, and is consequently necessary so that the overall multigrid preconditioner can have a complexity of $\mathcal{O}(N)$.

In contrast, the current release now also supports 'global coarsening' algorithms [14, 60] when using continuous (FE_Q, FE_SimplexP) and discontinuous (FE_DGQ, FE_SimplexDGP) elements. Global coarsening builds multigrid levels for the entire domain, coarsening *all* cells of a triangulation regardless of how many times they have been refined. In addition, the framework now available in deal.II is not only applicable to geometric coarsening, but can also perform coarsening by reducing the polynomial degree (p-multigrid, see [24]), for example to support hp-adaptive meshes. Finally, the implementation also supports transfer between continuous and discontinuous elements as a further way to create multigrid levels.

These new multigrid variants promise fewer solver iterations and better parallel scalability than the existing local smoothing algorithms, but have to deal with hanging nodes within each level and generally require more computational work per iteration overall.

The transfer operators between two levels have been implemented in the new class MGTwoLevelTransfer, which can be set up via the functions MGTwoLevelTransfer::reinit_geometric_transfer() or MGTwoLevelTransfer::reinit_polynomial_transfer() for given DoFHandler and AffineConstraints objects corresponding to the two levels. The resulting transfer operators can then be collected in a single MGTransfer

`GlobalCoarsening` object that can be used just as the previous workhorse `MGTransferMatrixFree` within the `Multigrid` algorithm. To facilitate the construction of matrix diagonals with matrix-free methods as well as a matrix representation of the coarse level matrix, new utility functions `create_diagonal()` and `create_matrix()` have been added to the `MatrixFreeTools` namespace (see also Subsection 2.4).

The usage of the new transfer operators (and of some of the utility functions) in the context of a hybrid multigrid algorithm (*hp*-multigrid with algebraic multigrid as coarse-grid solver) for *hp*-problems is demonstrated in the new tutorial `step-75`, see also Section 2.9.

2.4 Advances in the matrix-free infrastructure

`deal.II`'s matrix-free framework enables high-throughput operations for applications in which only the availability of the *action* of a matrix, but not the entries of the matrix, are necessary. This framework has been substantially extended in the current release.

2.4.1 Precompilation of evaluation kernels

Both the `FEEvaluation` and `FEFaceEvaluation` classes use template parameters for the polynomial degree of the finite element k and the number of the 1D quadrature points q to generate near-optimal code for these operations. For application codes that rely on operators of many different degrees (e.g., because they use *p*-multigrid or *hp*-algorithms), creating all instantiations can be overly complex and incur long compile times.

In the current release, specializations of these classes that do not rely on the template parameters k and q (expressed in the code using special values '−1' and '0') have been added. For example:

C++ code

```
FEEvaluation<dim, -1, 0, n_components, Number, VectorizedArrayType>
  phi(range, dofhandler_index, quadrature_index, first_selected_component);
```

These classes select at runtime — for common low and medium polynomial degree/quadrature combinations ($k \leqslant 6$ and $q \in \{k+1, k+2, \lfloor (3k)/2 \rfloor\}$) — efficient precompiled implementations and default to non-templated evaluation kernels otherwise (see also `FEEvaluation::fast_evaluation_supported()`).

In the case that even higher polynomial degrees are needed (e.g., $k \leqslant 12$), one can precompile the relevant internal classes (`FEEvaluationFactory`, `FEFaceEvaluationFactory`, `CellwiseInverseMassFactory`) in the user code for needed `degrees` and `VectorizedArrayTypes` in the following way:

C++ code

```
#define FE_EVAL_FACTORY_DEGREE_MAX 12

#include <deal.II/matrix_free/evaluation_template_factory.templates.h>

DEAL_II_NAMESPACE_OPEN
template struct dealii::internal::FEEvaluationFactory
  <dim, VectorizedArrayType::value_type, VectorizedArrayType>;
// same for FEFaceEvaluationFactory and CellwiseInverseMassFactory (skipped)
DEAL_II_NAMESPACE_CLOSE
```

2.4.2 Parallel matrix-free *hp*-implementation

In release 9.1, large parts of the *hp*-algorithms in `deal.II` were ported to a model that allows for parallel matrix-based simulations [6]. In the present release, the parallel *hp* support was extended to `MatrixFree`.

Until now, the `FEEvaluation` classes used the template parameters k and q to select the correct active FE and quadrature index and users were responsible for the cumbersome detection of subranges of the same k and q within the cell ranges returned by the matrix-free loops. The creation of subranges is now performed internally, and the non-templated versions of the `FEEvaluation` classes have been extended for the *hp*-case. To determine the desired FE and quadrature index of a subrange, the current cell/face range has to be provided to the constructors of the `FEEvaluation` classes. These changes enable users to write matrix-free code independently of whether *hp*-capabilities are used or not.

The new tutorial `step-75` presents how to use the new *hp*-related features in `MatrixFree` in the context of a hybrid-multigrid solver.

2.5 MPI-3.0 shared-memory support

In many large computations, certain pieces of data are computed (or read from disk) once and then treated as read-only. If this information is needed by more than one MPI process, it is more efficient to store this information only once in shared memory among all processes located on a multicore node. MPI supports the creation of such shared memory windows since version 3.0, and `deal.II` can now use this in the `AlignedVector` and `Table` classes that are often used for large lookup tables for classes such as `InterpolatedTensorProductGridData`.

Shared memory storage is also used in the `MatrixFree` and `LinearAlgebra::distributed::Vector` classes. If `MatrixFree` has been configured by setting `MatrixFree::AdditionalData::communicator_sm` appropriately, then `MatrixFree::create_dof_vector()` creates vectors that share information among all processes on one node. As a consequence, the `FEEvaluation` classes can access vector elements owned by other processes and node-local communication can be skipped in certain cases. To prevent race conditions, `MatrixFree` uses local barriers at the beginning and the end of loops (`loop()`, `cell_loop()`, and `loop_cell_centric()`).

The new `step-76` tutorial program illustrates this case in the context of the solution of the Euler equations. It reaches a speed-up of 27% compared to the original version, `step-67`, by using the new feature. For more details and the usage of the feature in the library `hyper.deal`, see [52].

2.6 Evaluation and integration at arbitrary points

In a number of circumstances, finite element solutions need to be evaluated at arbitrary reference points that change from one element to the next. Two important examples are particle simulations coupled to a finite element solution, or algorithms on non-matching grids. The existing `FEValues` class is a poor fit for this task, as it is based on the assumption that evaluation of shape functions and their derivatives happens at the same quadrature points on every cell, and that consequently expensive computations can be done once and results re-used for many subsequent cells. The new class `FEPointEvaluation` provides a more convenient interface for cases where the evaluation on different cells does not happen at the same points mapped from the reference cell. For tensor product finite elements (`FE_Q`, `FE_DGQ`) and tensor product mappings (`MappingQGeneric` and derived classes), the new approach is also very fast, as it can use some of the matrix-free infrastructure and vectorization facilities.

As an example, let us consider the evaluation of a surface tension force in the context of sharp-interface methods, whose contribution is added to a fluid solver by multiplication with test function and addition over quadrature points located at the interface section $\Gamma_K = \Gamma \cap K$ of the current cell K and that is, in general, positioned differently within K than for any other cell:

$$(\mathbf{v}, \varkappa\,\mathbf{n})_\Gamma \approx \sum_q \mathbf{v}(\mathbf{x}_q) \cdot \left(\varkappa(\mathbf{x}_q)\,\mathbf{n}(\mathbf{x}_q)\right) (JxW)_q .$$

In `deal.II`, this can now be conveniently written as

C++ code

```
phi_curvature.reinit(cell, reference_points);
phi_normal.reinit(cell, reference_points);
phi_force.reinit(cell, reference_points);

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

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

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

The quadrature points (at reference positions `reference_points`) and the related `JxW` value can, for example, come from a mesh of lower dimension. Determining to which `cell` a quadrature point belongs to on the background mesh, including the reference cell coordinates `reference_points`, is aided by functions like `find_active_cell_around_point()` and `find_all_active_cells_around_point()` from the `GridTools` namespace. While these functions have been available in `deal.II` previously, their performance has been considerably enhanced with the aforementioned more optimized code paths for selected mappings.

While `FEPointEvaluation` assumes that evaluation points are already sorted according to the owning cells and thus can focus on cell-local operations, the new class `RemotePointEvaluation` is responsible for determining the owning cells in a distributed context and for providing efficient communication patterns for the data exchange. In `deal.II`, the class has been successfully applied together with `FEPointEvaluation` to evaluate a distributed solution vector at arbitrary points (see also the `VectorTools::point_values()` function).

2.7 Simplified implementation for face integrals

Discontinuous Galerkin (DG) methods — and other methods with penalty terms defined on faces — require the evaluation of averages and jumps across cell faces, involving values and derivatives of the shape functions and solutions from two adjacent cells. The `FEInterfaceValues` class, first introduced in `deal.II` 9.2, is designed to provide the necessary interface.

For example, the interface terms of the SIPG formulation for a Laplace problem

$$\sum_F - \langle [\![v_h]\!] , \{\!\{ \nabla u_h \}\!\} \cdot \mathbf{n} \rangle_F - \langle \{\!\{ \nabla v_h \}\!\} \cdot \mathbf{n}, [\![u_h]\!] \rangle_F + \langle [\![v_h]\!] , \sigma [\![u_h]\!] \rangle_F$$

with face jump $[\![\cdot]\!]$ and average $\{\!\{\cdot\}\!\}$ can be implemented as (also see `step-74`):

C++ code

```
cell_matrix(i, j) +=
  ( -fe_iv.jump(i, q) *(fe_iv.average_gradient(j, q) *n)
    -(fe_iv.average_gradient(i, q) *n) *fe_iv.jump(j, q)
    +penalty *fe_iv.jump(i, q) *fe_iv.jump(j, q)
  ) *JxW[q];
```

Internally, this class provides an abstraction for two `FEFaceValues` objects (or `FESubfaceValues` when using adaptive refinement). The class introduces new interface degrees of freedom indices that are the union of the degrees of freedom indices of the two `FEFaceValues` objects. The interface degrees of freedom indices can be converted to the corresponding local degrees of freedom indices of the two cells using a helper function. New in the current release is better support for vector-valued problems: scalar or vector components of shape functions can now be extracted by providing an `FEValuesExtractors` object.

See step-12, step-47, step-50, and step-74 for more details.

2.8 The source-based toolchain installer candi

The requirement to download, compile, and install deal.II and its dependencies from source is a major obstacle to many deal.II users. Compiling all dependencies from source can be difficult but is a necessity on operating systems for which binary packages aren't available or on compute clusters and other machines without root privileges for the user to install system dependencies.

The source based installation of deal.II and its many dependent libraries can be done with the candi script tool for various Linux operating systems, within the Windows Subsystem Linux (WSL), and on OS X (experimental). The general assumption is that a C, C++, and Fortran compiler and suitable MPI-compilers for the base compilers as well as the corresponding development system packages are available.

candi is a bash-script based tool, is an abbreviation of 'compile and install' and is released under the GNU LGPL v3.0. The origin of candi is a fork made in 2013 from dorsal, a now-retired source based installer for the FEniCS library. The candi tool can be found via the download page of deal.II since version 8.5 or from github.com/dealii/candi, and is under active development. To install older releases of deal.II toolchains, one can check out the candi branch of the git repository that corresponds to the desired deal.II version.

candi downloads, unpacks, compiles, and installs an individual library (called a 'package' in candi), a list of libraries, or a complete toolchain. The toolchain installation is the default behavior and the default configuration ensures that most of the deal.II step tutorials can be used directly. The package installation mode is also useful to generate docker containers. In toolchain mode, candi checks for and deals with dependencies between libraries appropriately.

Each package for a library is defined by variables for its name and version, a (remote or local) download location, its packaging format (e.g., tar, gz, or zip), a checksum, its general build chain (e.g., using autotools, cmake, or others), as well as configuration options. Moreover, one can give specific instructions for each of the steps being necessary for the unpacking, configuration, or building of a package. A new feature allows candi to skip user prompts, allowing its use in a batch mode.

candi does all this by either downloading packages from the internet, including through mirrors, or reusing previously downloaded or checked out from a repository. If temporary files of candi are not removed, one can use the developer mode to prepare patches or developments for any of the packages.

candi is organized in the following way:

- candi.sh: the bash script which controls the overall process. Available command line options are listed by calling 'candi.sh -h' and explained in the README.md file.
- candi.cfg (or local.cfg): the default toolchain configuration file. Here one can easily switch on/off features, give additional configuration, and list the packages for a toolchain.
- deal.II-toolchain folder: the central project folder for deal.II-based toolchains. It contains subfolders for packages (libraries, dependencies, general tools in a specific version), platforms (operating systems) and for patches to be applied after unpacking a package.

The installer is developed in the dealii/candi github repository, and uses continuous integration (CI) to ensure that old and new features are working as expected.

2.9 New and improved tutorials and code gallery programs

Many of the deal.II tutorial programs were revised in a variety of ways as part of this release. In addition, there are a number of new tutorial programs:

- step-19 is an introductory demonstration of deal.II's particle functionality. It solves the coupled problem of charged particles and an electric field, using a cathode tube as an example.

- step-66, a program written by Fabian Castelli at Karlsruhe Institute of Technology, shows how to solve a nonlinear problem using Newton's method in the matrix-free framework in parallel. The PDE considered is the Gelfand problem $-\triangle u = \exp(u)$.
- step-68 is a demonstration of how to embed and distribute particles in a parallel triangulation. It tracks the movement of a set of particles in a fluid flow and illustrates how to distribute a parallel domain according to the number of locally owned particles instead of the number of locally owned cells.
- step-71 focuses on automatic and symbolic differentiation (AD and SD, in short) as a tool to make solvers for complex, nonlinear problems possible. To this end, deal.II can interface to a number of AD and SD libraries, specifically Trilinos' Sacado package [13], ADOL-C [31], and SymEngine [61]. The tutorial illustrates how these techniques can be used to compute derivatives first of a rather simple function, and then of the much more complex energy functions of two magnetoelastic and magneto-viscoelastic material formulations in which just the scalar energy functional takes up the better part of a page, and even first derivatives can only be computed with heroic effort.
- step-72 illustrates the use of automatic differentiation to simplify the computation of derivatives in the context of nonlinear partial differential equations where one needs to compute the Jacobian from the residual operator for efficient Newton iterations. step-72 builds on the minimal surface solver step-15 and replaces the hand-construction of the Jacobian by either computing it as the derivative of the residual, or alternatively as the second derivative of an energy functional that then also yields the residual itself.
- step-74 implements the symmetric interior penalty Galerkin (SIPG) method for Poisson's equation using the FEInterfaceValues class within the MeshWorker::mesh_loop() framework. This tutorial demonstrates a simple way to assemble face integrals.
- step-75 demonstrates a state-of-the-art way of solving a simple Laplace problem using *hp*-adaptation and hybrid multigrid methods on machines with distributed memory. This tutorial points out particularities in porting serial *hp*-adaptive code for parallelization. Furthermore, it guides through the process of writing an efficient matrix-free preconditioner for *hp*-adaptive applications.
- step-76 is an explicit time integrator for the compressible Euler equations discretized with a high-order discontinuous Galerkin (DG) scheme, using the matrix-free infrastructure just as step-67 does. The tutorial presents advanced topics, like the usage of cell-centric loops and the new MPI-3.0 shared-memory capabilities of MatrixFree to reach high throughput. Furthermore, the utilization of the template parameter VectorizedArrayType and the application of lambda functions to describe cell and face integrals are discussed.
- step-77 is a program that illustrate deal.II's interfaces to the SUNDIALS library [37], and specifically the KINSOL nonlinear solver. Like the step-72 program mentioned above, it is a variation of the minimal surface solver step-15: Instead of implementing the nonlinear Newton and line search loop ourselves, step-77 relies on KINSOL for decisions such as when to rebuild the Jacobian matrix, when to actually solve linear systems with it, and how to form updates that drive the residual to convergence. The program illustrates the substantial savings that can be obtained by not re-inventing the wheel but instead building on an existing and well-tuned software such as KINSOL. deal.II's interfaces to the various SUNDIALS sub-packages were also updated to the latest SUNDIALS release, 5.7.
- step-78, a program written by Tyler Anderson at Colorado State University, solves the one-dimensional Black-Scholes equations to model the price of stock options.
- step-79 is a program for topology optimization. Written by Justin O'Connor at Colorado State University, the program asks what the optimal distribution of a finite amount of material in a domain is to maximize its strength (or optimize some other objective functional). It uses advanced optimization techniques to deal with the nonlinearity of the problem as well as with the equality and inequality constraints that characterize the application.

There is also a new program in the code gallery (a collection of user-contributed programs that often solve more complicated problems than tutorial programs, and intended as starting points for further research rather than as teaching tools):

– 'Laplace equation coupled to an external simulation program' was contributed by David Schneider and Benjamin Uekermann at Technical University of Munich. It extends step-4 to a time-dependent Poisson problem and couples it with a simple external surface-based application, using the preCICE library [16, 46]. preCICE, also allows to couple deal.II to external simulation packages, such as OpenFOAM, SU2, CalculiX, or FEniCS, using adapter classes. The webpage https://precice.org/ provides several more deal.II-based tutorials, including interesting fluid-structure interaction applications.

2.10 Incompatible changes

The 9.3 release includes around 45 incompatible changes; see [47]. The majority of these changes should not be visible to typical user codes; some remove previously deprecated classes and functions; and the majority change internal interfaces that are not usually used in external applications. That said, the following are worth mentioning since they may have been more widely used:

– The hp::DoFHandler class has been deprecated. The standard DoFHandler class is now capable of all *hp*-functionalities.
– Consequently, all template arguments DoFHandlerType are now obsolete, and classes like DataOut or SolutionTransfer for example no longer require them. If you rely on these template arguments, an interim namespace Legacy has been introduced that provides all affected classes with the old interface for a transition period.
– GridTools::find_active_cell_around_point() no longer throws an exception when no cell is found, but returns an invalid iterator. User codes previously catching an exception will need to be changed.

3 How to cite deal.II

In order to justify the work the developers of deal.II put into this software, we ask that papers using the library reference one of the deal.II papers. This helps us justify the effort we put into this library.

There are various ways to reference deal.II. To acknowledge the use of the current version of the library, *please reference the present document*. For up to date information and a bibtex entry see:

https://www.dealii.org/publications.html

The original deal.II paper containing an overview of its architecture is [11], and a more recent publication documenting deal.II's design decisions is available as [7]. If you rely on specific features of the library, please consider citing any of the following:

– For geometric multigrid: [18, 39, 40];
– For distributed parallel computing: [10];
– For *hp*-adaptivity: [12];
– For partition-of-unity (PUM) and finite element enrichment methods: [22];
– For matrix-free and fast assembly techniques: [42, 43];
– For computations on lower-dimensional manifolds: [23];
– For curved geometry representations and manifolds: [32];
– For integration with CAD files and tools: [33];
– For boundary element computations: [29];
– For the LinearOperator and PackagedOperation facilities: [49, 50];
– For uses of the WorkStream interface: [63];
– For uses of the ParameterAcceptor concept, the MeshWorker::ScratchData base class, and the ParsedConvergenceTable class: [57];
– For uses of the particle functionality in deal.II: [26].

deal.II can interface with many other libraries:

- ADOL-C [31, 64];
- ArborX [44];
- ARPACK [45];
- Assimp [59];
- BLAS and LAPACK [4];
- cuSOLVER [19];
- cuSPARSE [20];
- Gmsh [27];
- GSL [25];
- Ginkgo [28];
- HDF5 [62];
- METIS [41];
- MUMPS [1–3, 51];
- muparser [53];
- OpenCASCADE [54];
- p4est [17];
- PETSc [8, 9];
- ROL [56];
- ScaLAPACK [15];
- SLEPc [34];
- SUNDIALS [37];
- SymEngine [61];
- TBB [55];
- Trilinos [35, 36];
- UMFPACK [21].

Please consider citing the appropriate references if you use interfaces to these libraries.
The two previous releases of deal.II can be cited as [5, 6].

4 Acknowledgments

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy.

deal.II is a world-wide project with dozens of contributors around the globe. Other than the authors of this paper, the following people contributed code to this release:

Pasquale Africa, Tyler Anderson, Mathias Anselmann, Arpit Babbar, Maximilian Bergbauer, Nicolas Barnafi, Michele Bucelli, Marcus Calhoun-Lopez, David F. Castellanos, Fabian Castelli, Praveen Chandrashekar, Conrad Clevenger, Andrew Davis, Elias Dejene, Niklas Fehn, Menno Fraters, Ivan Fumagalli, Daniel Garcia-Sanchez, Nicola Giuliani, Krishnakumar Gopalakrishnan, Alexander Grayver, Olivier Guevremont, Jake Harmon, Graham Harper, Katharina Kormann, Christoph Kammer, Wenyu Lei, Zhou Lei, Phillip Mobley, Nils Much, Pratik Nayak, Toni El Geitani Nehme, Justin O'Connor, Daniel Paukner, Lei Qiao, Ce Qin, Reza Rastak, Julian Roth, Laura Prieto Saavedra, Doug Shi-Dong, David Schneider, Magdalena Schreter, Richard Schussnig, Dominic Soldner, Simon Sticko, Malhar Tidke, Ignacio Tomas, Benjamin Uekermann, Peter Westerbaan.

Their contributions are much appreciated!

Funding: deal.II and its developers are financially supported through a variety of funding sources:
- D. Arndt and B. Turcksin: Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy;
- W. Bangerth, T. Heister, R. Gassmöller, and J. Zhang were partially supported by the Computational Infrastructure for Geodynamics initiative (CIG), through the National Science Foundation (NSF) under Award No. EAR-1550901 and The University of California – Davis;
- W. Bangerth and M. Fehling were partially supported by Award OAC-1835673 as part of the Cyberinfrastructure for Sustained Scientific Innovation (CSSI) program;
- W. Bangerth was also partially supported by Awards DMS-1821210 and EAR-1925595;
- B. Blais was partially supported by the National Science and Engineering Research Council of Canada (NSERC) through the RGPIN-2020-04510 Discovery Grant;
- T. Heister and J. Zhang were also partially supported by NSF Award OAC-2015848;
- T. Heister was also partially supported by the NSF Awards DMS-2028346, EAR-1925575, and by Technical Data Analysis, Inc. through US Navy STTR Contract N68335-18-C-0011;
- R. Gassmöller was also partially supported by the NSF Award EAR-1925595;
- L. Heltai was partially supported by the Italian Ministry of Instruction, University and Research (MIUR), under the 2017 PRIN project NA-FROM-PDEs MIUR PE1, 'Numerical Analysis for Full and Reduced Or-

der Methods for the efficient and accurate solution of complex systems governed by Partial Differential Equations';
– M. Kronbichler and P. Munch were partially supported by the Bayerisches Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen (KONWIHR) in the context of the projects 'Performance tuning of high-order discontinuous Galerkin solvers for SuperMUC-NG' and 'High-order matrix-free finite element implementations with hybrid parallelization and improved data locality';
– M. Maier was partially supported by NSF Awards DMS-1912847 and DMS-2045636;
– D. Wells was supported by NSF through Grant DMS-1344962.

The Interdisciplinary Center for Scientific Computing (IWR) at Heidelberg University has provided hosting services for the `deal.II` web page.

References

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, A fully asynchronous multifrontal solver using distributed dynamic scheduling, *SIAM J. Matrix Analysis Appl.* **23** (2001), No. 1, 15–41.
[2] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, Hybrid scheduling for the parallel solution of linear systems, *Parallel Computing* **32** (2006), No. 2, 136–156.
[3] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, *Comput. Methods Appl. Mech. Engrg.* **184** (2000), 501–520.
[4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.
[5] 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, *J. Numer. Math.* **28** (2020), No. 3, 131–146.
[6] D. Arndt, W. Bangerth, T. C. Clevenger, D. Davydov, M. Fehling, D. Garcia-Sanchez, G. Harper, T. Heister, L. Heltai, M. Kronbichler, R. M. Kynch, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells, The deal.II library, Version 9.1, *J. Numer. Math.* **27** (2019), No. 4, 203–213.
[7] 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, *Comput. & Math. Appl.* **81** (2021), 407–422.
[8] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. May, L. Curfman McInnes, R. Mills, T. Munson, K. Rupp, P. Sanan B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc Users Manual*, Argonne National Laboratory, Report No. ANL-95/11 – Revision 3.15, 2021.
[9] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. May, L. C. McInnes, R. Mills, T. Munson, K. Rupp, P. Sanan B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc Web page*, 2021, https://www.mcs.anl.gov/petsc.
[10] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler, Algorithms and data structures for massively parallel generic adaptive finite element codes, *ACM Trans. Math. Software* **38** (2011), No. 2, 14/1–14/28.
[11] W. Bangerth, R. Hartmann, and G. Kanschat, deal.II – a general purpose object oriented finite element library, *ACM Trans. Math. Software* **33** (2007), No. 4, 24/1–24/27.
[12] W. Bangerth and O. Kayser-Herold, Data structures and requirements for *hp* finite element software, *ACM Trans. Math. Software* **36** (2009), No. 1, 4/1–4/31.
[13] R. A. Bartlett, D. M. Gay, and E. T. Phipps, *Automatic Differentiation of C++ Codes for Large-Scale Scientific Computing*, International Conference on Computational Science – ICCS 2006 (Eds. V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, and J. Dongarra), Springer, Berlin–Heidelberg, 2006, pp. 525–532.
[14] R. Becker and M. Braack, Multigrid techniques for finite elements on locally refined meshes, *Numer. Linear Alg. Appl.* **7** (2000), No. 6, 363–379.
[15] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.
[16] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, and B. Uekermann, preCICE – a fully parallel library for multi-physics surface coupling, *Computers & Fluids* **141** (2016), 250–258.
[17] C. Burstedde, L. C. Wilcox, and O. Ghattas, p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, *SIAM J. Sci. Comput.* **33** (2011), No. 3, 1103–1133.

[18] T. C. Clevenger, T. Heister, G. Kanschat, and M. Kronbichler, A flexible, parallel, adaptive geometric multigrid method for FEM, *ACM Trans. Math. Software* **47** (2021), No. 1, 7/1–7/27.
[19] *cuSOLVER Library*, https://docs.nvidia.com/cuda/cusolver/index.html.
[20] *cuSPARSE Library*, https://docs.nvidia.com/cuda/cusparse/index.html.
[21] T. A. Davis, Algorithm 832: UMFPACK v4.3 – an unsymmetric-pattern multifrontal method, *ACM Trans. Math. Software* **30** (2004), 196–199.
[22] D. Davydov, T. Gerasimov, J.-P. Pelteret, and P. Steinmann, Convergence study of the *h*-adaptive PUM and the *hp*-adaptive FEM applied to eigenvalue problems in quantum mechanics, *Adv. Modeling Simul. Engrg. Sci.* **4** (2017), No. 1, 7.
[23] A. DeSimone, L. Heltai, and C. Manigrasso, Tools for the solution of PDEs defined on curved manifolds with deal.II, SISSA, Report No. 42/2009/M, 2009.
[24] N. Fehn, P. Munch, W. A. Wall, and M. Kronbichler, Hybrid multigrid methods for high-order discontinuous Galerkin discretizations, *J. Comput. Phys.* **415** (2020), 109538.
[25] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich, *GNU Scientific Library Reference Manual (Edition 2.3)*, 2016.
[26] R. Gassmöller, H. Lokavarapu, E. Heien, E. G. Puckett, and W. Bangerth, Flexible and scalable particle-in-cell methods with adaptive mesh refinement for geodynamic computations, *Geochemistry, Geophysics, Geosystems* **19** (2018), No. 9, 3596–3604.
[27] C. Geuzaine and J.-F. Remacle, Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities, *Int. J. Numer. Methods Engrg.* **79** (2009), No. 11, 1309–1331.
[28] *Ginkgo: High-Performance Linear Algebra Library for Manycore Systems*, https://github.com/ginkgo-project/ginkgo.
[29] N. Giuliani, A. Mola, and L. Heltai, π-BEM: A flexible parallel implementation for adaptive, geometry aware, and high order boundary element methods, *Adv. Engrg. Software* **121** (2018), 39–58.
[30] S. Golshan, P. Munch, R. Gassmöller, M. Kronbichler, and B. Blais, Lethe-DEM: An open-source parallel discrete element solver with load balancing, *Preprint arXiv:2106.09576*, 2021.
[31] A. Griewank, D. Juedes, and J. Utke, Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++, *ACM Trans. Math. Software* **22** (1996), No. 2, 131–167.
[32] L. Heltai, W. Bangerth, M. Kronbichler, and A. Mola, Propagating geometry information to finite element computations, *ACM Trans. Math. Software*, **47** (2021), No. 4, 32:1–32:30.
[33] L. Heltai and A. Mola, *Towards the Integration of CAD and FEM using open source libraries: A collection of deal.II manifold wrappers for the OpenCASCADE library*, SISSA, Report, 2015.
[34] V. Hernandez, J. E. Roman, and V. Vidal, SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems, *ACM Trans. Math. Software* **31** (2005), No. 3, 351–362.
[35] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, An overview of the Trilinos project, *ACM Trans. Math. Software* **31** (2005), 397–423.
[36] M. A. Heroux et al., *Trilinos Web page*, 2021, https://trilinos.org.
[37] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, SUNDIALS: suite of nonlinear and differential/algebraic equation solvers, *ACM Trans. Math. Software* **31** (2005), No. 3, 363–396.
[38] International Standards Organization, *ISO/IEC 14882:2014: The C++14 Programming Language Standard*, 2014, https://www.iso.org/standard/64029.html.
[39] B. Janssen and G. Kanschat, Adaptive multilevel methods with local smoothing for H^1- and H^{curl}-conforming high order finite element methods, *SIAM J. Sci. Comput.* **33** (2011), No. 4, 2095–2114.
[40] G. Kanschat, Multi-level methods for discontinuous Galerkin FEM on locally refined meshes, *Comput. & Struct.* **82** (2004), No. 28, 2437–2445.
[41] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.* **20** (1998), No. 1, 359–392.
[42] M. Kronbichler and K. Kormann, A generic interface for parallel cell-based finite element operator application, *Comput. Fluids* **63** (2012), 135–147.
[43] M. Kronbichler and K. Kormann, Fast matrix-free evaluation of discontinuous Galerkin finite element operators, *ACM Trans. Math. Software* **45** (2019), No. 3, 29:1–29:40.
[44] D. Lebrun-Grandié, A. Prokopenko, B. Turcksin, and S. R. Slattery, ArborX: a performance portable geometric search library, *ACM Trans. Math. Software* **47** (2021), No. 1, 2/1–2/15.
[45] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, 1998.
[46] F. Lindner, A. Totounferoush, M. Mehl, B. Uekermann, N. E. Pour, V. Krupp, S. Roller, T. Reimann, D. C. Sternel, R. Egawa, et al., ExaFSA: parallel fluid-structure-acoustic simulation, *Software for Exascale Computing-SPPEXA 2016-2019*, **136** (2020), 271.
[47] *List of Changes for 9.3*, https://www.dealii.org/developer/doxygen/deal.II/changes_between_9_2_0_and_9_3_0.html.
[48] A. Logg, Efficient representation of computational meshes, *Int. J. Comp. Sci. Engrg.* **4** (2009), No. 4, 283–295.

[49] M. Maier, M. Bardelloni, and L. Heltai, LinearOperator – a generic, high-level expression syntax for linear algebra, *Comput. Math. Appl.* **72** (2016), No. 1, 1–24.
[50] M. Maier, M. Bardelloni, and L. Heltai, *LinearOperator Benchmarks, Version 1.0.0*, March 2016, Zenodo. https://doi.org/10.5281/zenodo.47202.
[51] *MUMPS: a MUltifrontal Massively Parallel sparse direct Solver*, http://graal.ens-lyon.fr/MUMPS/.
[52] P. Munch, K. Kormann, and M. Kronbichler, hyper.deal: An efficient, matrix-free finite-element library for high-dimensional partial differential equations, *ACM Trans. Math. Software*, **47** (2021), No. 4, 33:1-33:34.
[53] *muparser: Fast Math Parser Library*, http://muparser.beltoforion.de/.
[54] *OpenCASCADE: Open CASCADE Technology, 3D Modeling & Numerical Simulation*, http://www.opencascade.org/.
[55] J. Reinders, *Intel Threading Building Blocks*, O'Reilly, 2007.
[56] D. Ridzal and D. P. Kouri, *Rapid Optimization Library.*, Sandia National Laboratories (SNL-NM), Albuquerque, NM, Report, 2014.
[57] A. Sartori, N. Giuliani, M. Bardelloni, and L. Heltai, deal2lkit: A toolkit library for high performance programming in deal.II, *SoftwareX* **7** (2018), 318–327.
[58] N. Schlömer, *quadpy: Your one-stop shop for numerical integration in python*, 2021, https://github.com/nschloe/quadpy/.
[59] T. Schulze, A. Gessler, K. Kulling, D. Nadlinger, J. Klein, M. Sibly, and M. Gubisch, Open asset import library (assimp), https://github.com/assimp/assimp.
[60] H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler, Parallel geometric-algebraic multigrid on unstructured forests of octrees, In: *SC'12: Proc. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis*, IEEE, 2012, pp. 1–11.
[61] *SymEngine: Fast Symbolic Manipulation Library, Written in C++*, https://symengine.org/.
[62] The HDF Group, *Hierarchical Data Format, Version 5*, 1997-2018, http://www.hdfgroup.org/HDF5/.
[63] B. Turcksin, M. Kronbichler, and W. Bangerth, WorkStream – a design pattern for multicore-enabled finite element computations, *ACM Trans. Math. Software* **43** (2016), No. 1, 2/1–2/29.
[64] A. Walther and A. Griewank, Getting started with ADOL-C, In: *Combinatorial Scientific Computing*, (Eds. U. Naumann and O. Schenk), Chapman-Hall CRC Comput. Sci., 2012, pp. 181–202.
[65] F. D. Witherden and P. E. Vincent, On the identification of symmetric quadrature rules for finite element methods, *Computers & Math. Appl.* **69** (2015), No. 10, 1232–1241.