Daniel Arndt, Wolfgang Bangerth, Maximilian Bergbauer, Marco Feder, Marc Fehling,
Johannes Heinz, Timo Heister*, Luca Heltai, Martin Kronbichler, Matthias Maier,
Peter Munch, Jean-Paul Pelteret, Bruno Turcksin, David Wells, and Stefano Zampini

The deal.II library, Version 9.5

https://doi.org/10.1515/jnma-2023-0089
Received July 29, 2023; accepted August 01, 2023

Abstract: This paper provides an overview of the new features of the finite element library deal.II, version 9.5.

Keywords: software, finite elements, deal.II

Classification: 65M60, 65N30, 65Y05

1 Overview

deal.II version 9.5.0 was released July 7, 2023. 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 9.5. 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:

- Substantial updates and extensions to deal.II's interfaces to other libraries (see Section 2.1). This includes, in particular, the integration of Kokkos (Section 2.1.1); additions and updates to the PETSc and Trilinos interfaces (Sections 2.1.3 and 2.1.4, respectively).
- Uniform handling of nonlinear solver packages (Section 2.1.2) and a uniform way of defining callbacks used by external libraries (Section 2.1.5).
- Advances in matrix-free infrastructure (see Section 2.2).
- Advances in non-matching support (see Section 2.3).
- New features related to linear algebra (see Section 2.4).
- C++ language modernization (see Section 2.5).
- Build-system modernization (see Section 2.6).

Daniel Arndt, Bruno Turcksin, Computational 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.
Maximilian Bergbauer, Institute for Computational Mechanics, Technical University of Munich, Boltzmannstr. 15, 85748 Garching, Germany.
Marco Feder, Luca Heltai, SISSA, International School for Advanced Studies, Via Bonomea 265, 34136, Trieste, Italy.
Johannes Heinz, Institute of Mechanics and Mechatronics, TU Wien, Getreidemarkt 9, 1060 Vienna, Austria.
***Corresponding author: Timo Heister,** School of Mathematical and Statistical Sciences, Clemson University, Clemson, SC, 29634, USA. Email: heister@clemson.edu
Martin Kronbichler, Peter Munch, Institute of Mathematics, University of Augsburg, Universitätsstr. 12a, 86159 Augsburg, Germany.
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.
David Wells, Department of Mathematics, University of North Carolina, Chapel Hill, NC 27516, USA.
Stefano Zampini, Extreme Computing Research Center, King Abdullah University of Science and Technology, 23955-6900 Thuwal, Saudi Arabia.

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, which we briefly outline in the remainder of this section:

– The new function CellAccessor::as_dof_handler_iterator() simplifies the conversion from a CellAccessor to a DoFCellAccessor. The old way,

C++ code

```
const auto cell_dof = typename DoFHandler<dim, spacedim>::
  active_cell_iterator(&dof_handler.get_triangulation(),
    cell->level(), cell->index(), &dof_handler);
```

was lengthy and error-prone. In contrast, the following function is substantially clearer and more concise:

C++ code

```
const auto cell_dof = cell->as_dof_handler_iterator(dof_handler);
```

– Several functions used intensively during initialization of deal.II-based programs, such as the refinement of triangulations, the enumeration of degrees of freedom, the setup of global-coarsening multigrid algorithms, and several evaluation functions of the MappingQ class representing a polynomial mapping of quadrilateral and hexahedral, have been overhauled to run more quickly and sometimes also consume less memory. These and related improvements are guided by several performance tests that are used to monitor the performance of the library over time.

The changelog lists more than 120 other features and bugfixes.

2 Major changes to the library

This release of deal.II contains a number of large and significant changes, which will be discussed in this section. It of course also includes 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 [54]).

2.1 Updates to interfaces to other packages

For many operations, deal.II relies on external libraries – some of these are optional, others are mandatory (such as Boost and, now, Kokkos); a complete list of external dependencies is provided in Section 3. A substantial amount of work has gone into overhauling and extending these interfaces for the current release, as detailed in the following subsections.

2.1.1 Integration of Kokkos

Kokkos [73] is a C++ library that enables the creation of performance portable applications for all major high-performance computing platforms. It implements a programming model that allows developers to write code that can efficiently run on diverse architectures. Kokkos provides abstractions for both parallel execution of code and data management. It supports a wide range of backend programming models, including CUDA, HIP, SYCL, HPX, OpenMP, and C++ threads, and continues to evolve with the development of new hardware and corresponding backend options.

deal.II has, for several releases already, used CUDA to offload some operations onto GPUs. It has also had interfaces to CUDA-based linear algebra libraries. Yet, the diversification of GPU platforms away from a single vendor (Nvidia) has made it clear that we need a different strategy to support what users want. As a

consequence, Kokkos has become a mandatory dependency of deal.II as part of the current release; if it is not found on a given system during configuration time, then the library will fall back on a copy of Kokkos stored in the bundled/ directory in the same way as we already interface with Boost.

In the current release, LinearAlgebra::distributed::Vector and the CUDAWrappers::MatrixFree framework are using Kokkos. This allows them to work on all the architectures supported by Kokkos. In particular, the step-64 example can now also be run on the CPU and doesn't require a GPU anymore. The Kokkos backend used by deal.II is Kokkos::DefaultExecutionSpace which corresponds to the highest available backend in the hierarchy device, host-parallel, and host-serial at the time Kokkos was configured.

2.1.2 Uniform interface for nonlinear solvers

With the current release, deal.II now supports solvers for nonlinear problems provided by several different external packages: (i) KINSOL (part of SUNDIALS); (ii) SNES (part of PETSc, see Section 2.1.3); and (iii) NOX (part of Trilinos, see Section 2.1.4). The wrappers are provided by the classes PETScWrappers::NonlinearSolver<VectorType>, SUNDIALS::KINSOL<VectorType>, and TrilinosWrappers::NOXSolver<VectorType>, respectively.

All three of these classes have a very similar interface, except that they vary in the kind of algorithms and parameters offered. The new class NonlinearSolverSelector<VectorType> provides a wrapper on top of the three external solvers with a unified interface. The user can either let deal.II decide which of the packages to use (depending on the current availability of the external packages) or specify it manually.

The following code snippet shows a complete example using the new class for applying a Newton solver by automatically choosing one of the available packages:

C++ code

```
using NLS = NonlinearSolverSelector<VectorType>;

NLS::AdditionalData additional_data(
  NLS::automatic, // other options: kinsol, nox, petsc_snes
  NLS::newton);

NLS nonlinear_solver(additional_data, mpi_communicator);

solver.reinit_vector       = [&](VectorType &x) {/*...*/};
solver.residual            = [&](const VectorType &src,
                                 VectorType       &dst) {/*...*/};
solver.setup_jacobian      = [&](const VectorType &src) {/*...*/};
solver.apply_jacobian      = [&](const VectorType &src,
                                 VectorType       &dst) {/*...*/};
solver.solve_with_jacobian = [&](const VectorType &src,
                                 VectorType       &dst,
                                 const double      tol) {/*...*/};

solver.solve(current_solution);
```

Note that while the selector class provides a unified interface and therefore allows to easily switch between backends, there are cases when users need to use the underlying wrapper classes, if, e.g., the functionality is only provided by one implementation. Details of each of the implementations are discussed in subsequent sections.

2.1.3 Updates and additions to the PETSc wrappers

The deal.II classes wrapping PETSc objects have been rewritten in substantial ways to support PETSc's System of Nonlinear Equations Solver SNES and the Ordinary Differential Equations (ODE) solver TS [1]. First, we

briefly describe the most important improvements to the existing classes and then outline the newly designed interfaces to the nonlinear solvers, together with a new interface to the communication module SF in PETSc [76].

Vector and matrix classes of the PETSc wrappers have been extended with an additional constructor that can wrap an already existing PETSc vector or matrix, respectively. The BlockVector and BlockSparseMatrix classes now internally use PETSc nested objects, i.e., VECNEST and MATNEST respectively. We have added a new class PETScWrappers::PreconditionShell to support user-defined preconditioning that can be simply customized as the following code snippet shows:

C++ code

```
PETScWrappers::PreconditionShell preconditioner(/*...*/);
preconditioner.vmult = [&](const VectorType &src,
                           VectorType       &dst) {/*...*/};
```

The resulting object can be passed to PETSc and used within the nonlinear solver hierarchy. See Section 2.1.5 for additional information on such kind of callbacks.

The PETSc SNES subpackage solves systems of nonlinear equations of the form $F(x) = 0$. The interface to SNES in the class PETScWrappers::NonlinearSolver has been modeled on the already existing interface to the KINSOL solver from the SUNDIALS package: the nonlinear problem is specified via a set of callbacks as shown above in Section 2.1.2. The default configuration is set up to use a Jacobian-free Newton–Krylov (JFNK) approach [46], using solve_with_jacobian as linear solver. Numerous other solver configurations are possible and can be selected programmatically using the constructor arguments, or via the powerful command line customization of PETSc. This includes, for example, quasi-Newton methods, Anderson's acceleration, and nonlinear preconditioning [20]. The step-77 tutorial program has gained a section that discusses the use of these new SNES interfaces.

The PETSc TS subpackage solves ODEs in explicit or implicit form [1], i.e.,

$$\dot{u} = G(t, u) \qquad \text{(explicit)}$$
$$F(t, u, \dot{u}) = 0 \qquad \text{(implicit)}.$$

The interface to TS has been modeled on the already existing interfaces to the IDA and ARKODE solvers from the SUNDIALS package. Specifically:

C++ code

```
PETScWrappers::TimeDependentSolver<VectorType> solver(/*...*/);

// If solving udot = G(t,u)
solver.explicit_function = [&](const double     t,
                               const VectorType &u,
                               VectorType       &G) {/*...*/};
// If solving F(t,u,udot) = 0
solver.implicit_function = [&](const double     t,
                               const VectorType &u,
                               const VectorType &udot,
                               VectorType       &F) {/*...*/};
```

In addition to specifying the function callbacks, users can further customize the solution of the linearized equations $\alpha\, \partial F/\partial \dot{u} + \partial F/\partial u$ via the following callbacks:

C++ code

```
solver.setup_jacobian = [&](const double     t,
                            const VectorType &u,
                            const VectorType &udot,
                            const double     alpha) {/*...*/};
solver.solve_with_jacobian = [&](const VectorType &rhs,
                                 VectorType       &sol) {/*...*/};
```

As with SNES, the default configuration of an implicit solver is set up to use a JFNK approach, and the entire suite of solvers offered by PETSc is available programmatically or via the command line interface, including adaptive time-stepping and Implicit–Explicit schemes.

We close this section by introducing the interface to the SF subpackage, the abstract communication model of PETSc. The deal.II interface to SF has been modeled on the existing Utilities::MPI::Partitioner and Utilities::MPI::NoncontiguousPartitioner classes, with minimal changes for the communication routines API; the equivalent classes based on SF are PETScWrappers::Partitioner and PETScWrappers::CommunicationPattern. Future developments will add support for GPU buffers.

We refer interested readers to our documentation for more advanced functions of the SNES, TS, and SF wrappers, and to the tests/petsc/ folder for examples on how to use them.

2.1.4 Interfaces to Trilinos' Belos and NOX packages

Trilinos is a large collection of individual subpackages [41, 72]. deal.II has long had interfaces to the Trilinos packages that provide parallel vector and matrix classes (both Epetra and Tpetra), as well as a small number of linear algebra packages for iterative solvers and preconditions. In the current release, there are now also interfaces to two additional packages: Belos and NOX.

Belos is the successor of the Trilinos package AztecOO and provides basic and advanced iterative solvers that heavily rely on multivector operations. The interface of the wrapper is similar to deal.II's own iterative solvers:

C++ code

```
TrilinosWrappers::SolverBelos<VectorType> solver(/*...*/);
solver.solve(matrix, x, r, preconditioner);
```

The omitted constructor arguments allow selecting the actual iterative solver to be used, along with other configuration options.

Secondly, we have added a wrapper to NOX, a nonlinear solver library that is similar to both the KINSOL solver from the SUNDIALS package to which we already had interfaces, and also to the SNES collections of functions from PETSc (see also Section 2.1.3). As for the SNES interface above, the NOX interface is drive by the callbacks shown in Section 2.1.2, and implemented in the TrilinosWrappers::NOXSolver class.

2.1.5 Uniform error reporting in callbacks

With the current release, deal.II has gained interfaces to several external packages that are largely driven via *callbacks* – see for example the code examples for nonlinear or ODE solvers in the previous sections.

This substantially enlarged use of callbacks used by different backend libraries raises the issue that each underlying package has its own convention on how success or error codes of these callbacks should be encoded. In the case of PETSc's SNES and TS, and for Trilinos's NOX, success is indicated by a zero integer return value, whereas failure is indicated by a nonzero return value. On the other hand, the SUNDIALS packages indicate success by a zero integer return value, a recoverable failure with a positive value, and an irrecoverable failure with a negative value. Newer PETSc versions can deal with recoverable failures as well, in some cases by calling back into PETSc from a callback to set an error flag, in others by setting the elements of a returned vector to NaN. None of these conventions mesh well with C++ where error codes are generally indicated via exceptions. It is conceivable that libraries we want to interface with in the future use yet other conventions.

In order to shield those who write these callbacks from having to learn the intricacies of the underlying libraries, we have adopted a convention whereby user-provided callbacks are just regular functions that return errors via exceptions as is common in C++. Internally, the interfaces to different underlying libraries then translate these exceptions into the appropriate error codes, saving the thrown exception for possible later use;

if an underlying library supports recoverable errors, then a callback should indicate such an error by throwing an object of the special type `RecoverableUserCallbackError`. If one of the underlying packages returns with an error caused by a user-provided callback throwing an exception, then the wrapper code rethrows the previously saved exception, allowing the calling site to catch it to obtain information about what might have gone wrong.

This convention for user callbacks is documented in a glossary entry that is also linked to from the documentation of all variables storing callbacks.

2.2 Updates to matrix-free algorithms

The current release includes numerous updates to the matrix-free infrastructure, including:

- In release 9.3, we enabled parallel *hp*-operations in the matrix-free infrastructure. The infrastructure did not work properly for cells that do not have any degrees of freedom because they use `FE_Nothing`. This has been fixed now. Furthermore, `FE_Nothing` now also works together with discontinuous Lagrange elements (i.e., with the `FE_DGQ` class). Due to the popularity of `FE_Nothing` as a means to enable or disable cells, we have introduced the new class `ElementActivationAndDeactivationMatrixFree`, which wraps a `MatrixFree` object, only loops over all active cells, and optionally interprets faces between active and deactivated cells as boundary faces. This functionality has enabled simulations in powder-bed-fusion additive manufacturing in [64].
- The matrix-free infrastructure allows interleaving cell loops with vector updates by providing `pre`/`post` functions that are run on index ranges. The `deal.II` library uses this feature, e.g., to improve the performance of (preconditioned) conjugate gradient solvers [50] as well as of relaxation and Chebyshev iterations (see Subsection 2.4). Up to release 9.3, the `pre`/`post` infrastructure was only supported for continuous elements (cell loop); now, it also works for discontinuous elements which also require face loops to assemble jump and penalty terms.
- The operator `CellwiseInverseMassMatrix` now also efficiently evaluates the inverse for coupling (dyadic) coefficients in the case of multiple components:

  $$\left(v_i, D_{ij} u_j\right)_{\Omega^{(K)}}, \quad 1 \leqslant i,j \leqslant c$$

  with c being the number of components and $D \in \mathbb{R}^{c\times c}$ a tensorial coefficient. The algorithm relies on the construction of the element mass matrix,

  $$M = \left(I_1 \otimes N^T\right)\left(D \otimes I_2\right)\left(I_1 \otimes N\right)$$

  with N being the tabulated values of shape functions at quadrature and I_1, I_2 identity matrices associated to c vector components and the quadrature points, respectively. The algorithms assumes a square N:

  $$M^{-1} = \left(I_1 \otimes N^{-1}\right)\left(D^{-1} \otimes I_2\right)\left(I_1 \otimes N^{-T}\right).$$

  For hypercube-shaped cells, N^{-1} has an explicit representation again in terms of tensor products; for example, in 3D it can be expressed as $N^{-1} = N_{1D}^{-1} \otimes N_{1D}^{-1} \otimes N_{1D}^{-1}$, allowing the use sum factorization [51].

2.3 Advances in non-matching support

The (matrix-free) non-matching support of `deal.II` heavily relies on the classes `FEPointEvaluation` and `RemotePointEvaluation`, which were introduced in release 9.3. While `FEPointEvaluation` is responsible for efficient evaluation/integration at arbitrary (reference) points within a cell, `RemotePointEvaluation` is responsible for sorting points with regards to the cells they reside in, and for the communication necessary to evaluate solutions on cells owned by other MPI processes.

In the current release, we have considerably optimized `FEPointEvaluation`, e.g, by caching the evaluated shape functions, templating loop bounds, and exploiting the tensor-product structure of the shape functions if all points are positioned on a face, a common use case in the context of fluid–structure interaction. Furthermore, the extended class `NonMatching::MappingInfo` allows for precomputing and storing metric terms, like the Jacobian, its determinant, or the unit outer normal vectors. This is useful in cases in which these metric terms do not change and can be reused, e.g., in the context of iterative solvers. This development is part of an effort to make the interfaces of the (matrix-free) non-matching support more similar to the ones of the established matrix-free infrastructure of `deal.II` for fixed quadrature formulas.

In addition to these node-level performance optimizations, we added experimental support for (i) generating intersections of distributed non-matching grids and working on them, and (ii) multigrid with non-nested levels. In the following, we describe these in more detail.

2.3.1 Intersected meshes

In the current release, we added experimental support to compute intersections on `parallel::distributed::Triangulation` objects using `CGAL` [70]. For this purpose, we introduced a free function `distributed_compute_intersection_locations()` that computes intersections and relevant information for communication from `intersection_requests`. The data structure `intersection_requests` is a vector indicating entities of a given triangulation that intersections are computed upon. Each entity (face or cell) is described by a vector of vertices. Currently, the function is placed in `GridTools::internal`, since the location and arguments of the function might still change in the future. To compute intersections between two geometric entities, the function internally uses the new function `CGALWrappers::compute_intersection_of_cells()`.

For the common case of Nitsche-type mortaring, quadrature points must be distributed on the intersections to evaluate the underlying physical coupling terms. The data structure returned by `distributed_compute_intersection_locations()` can convert itself to a data structure that can be used to fill `RemotePointEvaluation`. This conversion is triggered by the member function `convert_to_distributed_compute_point_locations_internal()`, given the number of quadrature points per intersection. The whole procedure is done without communication. This functionality is useful since `RemotePointEvaluation` can now be used to access quantities at quadrature points on intersections without further ado. To reduce the user's effort, we plan to add a wrapper that takes care of the described procedure and provides interfaces like `FEEvaluation` to access quantities easily, e.g., in a matrix-free loop.

The procedure described above and an early version of the wrapper have been used successfully in [37] to perform Nitsche-type mortaring in the context of the conservative formulation of acoustic equations discretized with a discontinuous Galerkin method to suppress artificial modes.

2.3.2 Non-nested multigrid

`deal.II` has provided support for geometric multigrid methods (GMG) for locally refined meshes nearly since its inception. (`deal.II` also supports algebraic multigrid methods by interfacing to the external libraries `PETSc` and `Trilinos`.) Traditionally, these GMGs were based on local-smoothing methods (described many years later in [43, 44], see also [23]), and more recently also global-coarsening algorithms [59]. The global-coarsening infrastructure, furthermore, allows globally coarsening the polynomial degree (*p*-multigrid), with specific applicability to *hp*-adaptive methods. In the current release, we have added support for the case that multigrid levels are given by non-nested meshes [2, 16, 19]. An example for such meshes is presented in Fig. 1.

The current implementation extends the existing global-coarsening infrastructure by introducing, in addition to the (conformal) `MGTwoLevelTransfer`, a new (non-conformal) two-level transfer operator:

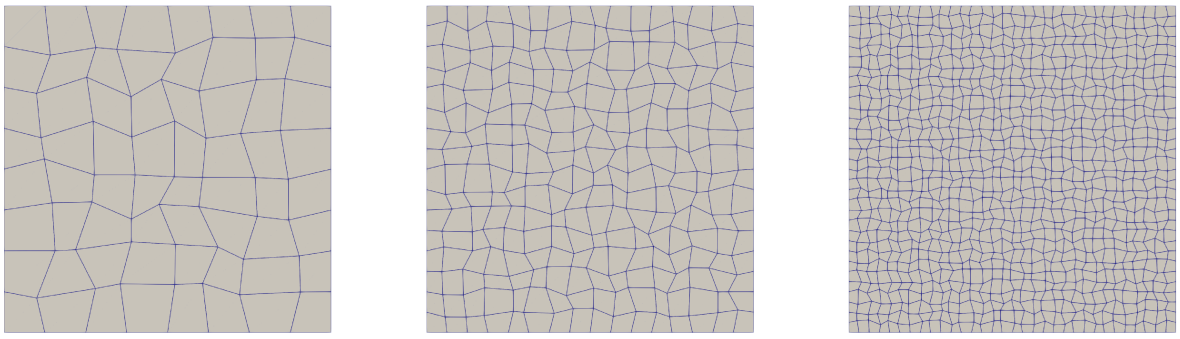

Fig. 1: Example of non-nested multigrid levels.

C++ code

```
MGTwoLevelTransferNonNested two_level_transfer(/*...*/);

two_level_transfer.reinit(dof_handler_fine, dof_handler_coarse,
                          mapping_fine, mapping_coarse,
                          constraints_fine, constraints_coarse)
```

The resulting object can be passed to `MGTransferGlobalCoarsening` (and `MGTransferBlockGlobalCoarsening`) in the usual way. The interface of the new class is purposefully similar to `MGTwoLevelTransfer`, with the difference that the former (i) also takes `Mapping` instances, and (ii) is not limited to the case that the coarser of the two `DoFHandler` instances can be generated by a coarsening step from the fine mesh.

At the time of writing, the `MGTwoLevelTransferNonNested` operator provides a prolongation operator that prolongates a vector from the coarse space to the fine space, by a pointwise interpolation (injection) from the coarse mesh to the support points of the fine mesh. This is done by looping over all coarse cells and interpolating to all (fine support) points that fall into the cell with `FEPointEvaluation`. The local result is scattered into the fine vector, which is finalized by a communication and a weighting step that is necessary as multiple elements could add to the same global entry.

The current implementation works for scalar, continuous (`FE_Q`) and discontinuous (`FE_DGQ`) elements on hypercube-shaped cells. We plan to support vectorial and simplex elements and to add support for projection, in addition to the pointwise interpolation outlined above. Furthermore, we envision providing utility tools to generate coarser meshes, based on a fine mesh. Currently, the users themselves have to provide such meshes, e.g., by generating meshes with different cell sizes in external mesh-generation tools.

Note that the new class `MGTwoLevelTransferNonNested` is not limited to multigrid but can be used to prolongate results and restrict residuals between any two meshes. Indeed, the infrastructure has been successfully applied to perform conservative interpolation between a fine (near) mesh and a coarse (far) mesh in the context of aeroacoustic problems.

2.4 New linear algebra features

There are a number of linear algebra related new features in this release:

- Both `SolverGMRES` and `SolverFGMRES` now support the classical Gram–Schmidt orthonormalization in addition to the modified Gram–Schmidt algorithm. This reduces the cost of vector operations in terms of communication latency and memory transfer significantly.
- Our Chebyshev preconditioner (`PreconditionChebyshev`) now also supports James Lottes's novel fourth-kind Chebyshev polynomial [55, 62].

– Our relaxation preconditioner (`PreconditionRelaxation`) now also allows interleaving cell loops and vector updates related to relaxation. The relaxation iteration reads as

$$x^{(i+1)} \leftarrow x^{(i)} + \omega P^{-1}(b - Ax^{(i)}).$$

In the case that the preconditioner P is a diagonal matrix, the zeroing of the destination vector x_{i+1} can be performed during a `pre`-operation of the application of A and the vector update $x_j^{(i+1)} \leftarrow x_j^{(i)} + \omega P_{j,j}^{-1}(b_j - (Ax^{(i)})_j)$ during a `post` operation, allowing for a reduction in the number of read and write accesses from 8 and 4, to 1 and 3, respectively. The existing `pre`/`post` support in our Chebyshev-preconditioner implementation (`PreconditionChebyshev`) has been improved. In addition, both `PreconditionRelaxation` and `PreconditionChebyshev` support `pre`/`post` optimizations now not only for diagonal preconditioners but also for preconditioners that are built around cell loops and, as a consequence, support interleaving. Examples of such preconditioners are patch-based additive Schwarz preconditioners.
– In the context of additive Schwarz methods, the preconditioner application is defined as

$$v = P^{-1}u = \sum R_i^T A_i^{-1} R_i u$$

with $A_i = R_i A R_i^T$ being a block of the assembled system matrix A restricted to an index set described by R_i. In the special case when R_i denotes the unknowns of cells, the expression $R_i A R_i^T$ resembles the reverse of matrix assembly. During the restriction step, rows of the system matrix that are potentially owned by other processes are needed. In `deal.II`, it is not possible to access remote entries of sparse matrices. Two new functions query this information. First, `restrict_to_serial_sparse_matrix()` creates, based on a given index set, a serial sparse matrix from a distributed matrix:

C++ code

```
SparseMatrixTools::restrict_to_serial_sparse_matrix (
  sparse_matrix_in, sparsity_pattern, requested_index_set,
  system_matrix_out, sparsity_pattern_out)
```

This function can be used, e.g., if the granularity of the additive Schwarz preconditioner is a complete subdomain, potentially, with a fixed overlap.
In contrast, `restrict_to_full_matrices()` performs the restriction for arbitrary number of patches/blocks:

C++ code

```
SparseMatrixTools::restrict_to_full_matrices (
  sparse_matrix_in, sparsity_pattern, indices_of_blocks, blocks)
```

The data is stored in dense matrices, since the typical granularity is a (rather small) cell-centric or vertex-star patch.
– For certain types of configurations, there are computationally more efficient approaches than extracting submatrices from an assembled matrix. For example, for the Laplace operator on 2D Cartesian meshes, the (element/patch) matrix is given as

$$A_i^{\text{cart}} = K_1 \otimes M_0 + M_1 \otimes K_0$$

i.e., as the tensor product of 1D mass and stiffness matrices. The inverse is explicitly available via the fast diagonalization method [56] as

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

with T_i and Λ_i being the (orthonormal) eigenvectors and the diagonal matrix of eigenvalues, obtained from a generalized eigendecomposition $K_i T_i = M_i T_i \Lambda_i$, as also used by `deal.II`'s `step-59` tutorial program. Since $A_i \approx A_i^{\text{cart}}$ might be a good approximation also in the case of non-Cartesian meshes and A_i^{cart} has an explicit inverse, it is considered in the literature as a patch preconditioner in the context of additive Schwarz [24, 63,

75] and block-Jacobi methods [49]. In `deal.II`, the new function `TensorProductMatrixCreator::create_laplace_tensor_product_matrix()` computes K_i and M_i for cell-centric patches with a specified overlap and given boundary conditions. A set of T_i and Λ_i is applied to a cell via `TensorProductMatrixSymmetricSum` or to a collection of cells via the new class `TensorProductMatrixSymmetricSumCollection`, which tries to reuse the eigenvalues and eigenvectors between cells.

2.5 C++ language modernization

This version of `deal.II` uses C++14 as the language standard to which its code base is written. It can also use the classes `std::optional` and `std::variant` if the compiler supports C++17, but falls back to implementations obtained via the `Boost` library otherwise, and this is true also for a number of individual functions that were introduced in C++17 or C++20.

As part of the current release, `deal.II` now also uses some C++20 features to annotate classes and functions with regard to properties template arguments need to satisfy. This aids in situations such as with the following function:

C++ code

```
template <class MeshType>
std::vector<typename MeshType::active_cell_iterator>
find_cells_adjacent_to_vertex(const MeshType &   mesh,
                              const unsigned int vertex);
```

This function is intended to be called with either a `Triangulation` or `DoFHandler` object as first argument (and documents this requirement), but the compiler can not check this requirement at the call site and will gladly call it with any other kind of object as well. Because the implementation of the function is in a `.cc` file, and the template is only instantiated for the two classes mentioned above, calling the function (erroneously) with anything else as first argument is not detected at compile time, but only later when the linker reports an undefined symbol.

We have started to address this by annotating functions using C++20-style '`requires`' clauses:

C++ code

```
template <class MeshType>
  requires (concepts::is_triangulation_or_dof_handler<MeshType>)
std::vector<typename MeshType::active_cell_iterator>
find_cells_adjacent_to_vertex(const MeshType &   mesh,
                              const unsigned int vertex)
```

If the compiler supports C++20, then the added clause will cause the compiler to reject any call to the function for which the first argument `MeshType` does not satisfy the named concept, which is defined as the type being either a `Triangulation` or `DoFHandler` object, as intended. (The `requires` annotation is suppressed if the compiler does not support C++20.)

Given the heavy dependence of `deal.II` on templates, there are likely hundreds or thousands of locations that should be annotated with requirements on template arguments over time; for the moment, the library contains some 300 of these `requires` clauses.

The next release of `deal.II` will build upon C++17.

2.6 Build-system modernization

`deal.II`'s original CMake build system, merged for the 8.0.0 release in 2013, was written for CMake 2.8.8 and thus lacks many of the modern features introduced into CMake over the last several years. This created a particular issue with an increasing number of dependencies switching their CMake configuration to *import targets*, where

all necessary information for using an external resource, such as include directories and library link interfaces, are associated with an imported CMake target rather than being provided with individual (and inconsistently) named CMake variables. With the deal.II 9.5 release the handling of external dependencies has been rewritten entirely to support imported targets. The Trilinos and Kokkos interfaces have been modernized. We plan to migrate all remaining dependencies to import targets wherever possible for the next release.

deal.II's CMake project configuration itself now exports three targets, dealii::dealii_debug, dealii::dealii_release, and dealii::dealii. Linking against one of the first two variants via target_link_libraries() populates the target with all necessary include directories and the full link interface necessary for deal.II. The third variant, dealii::dealii, automatically switches between debug and release interface depending on the build type set via CMAKE_BUILD_TYPE. The dealii::dealii third variant also populates compiler and linker options, whereas for the first two variants a client project has to ensure that the compiler and linker are properly set up. The new import targets now make it possible to configure a dependent project without the use of any deal.II specific macros:

Code

```
cmake_minimum_required(VERSION 3.13.4)
set(CMAKE_BUILD_TYPE Debug CACHE STRING "")
project(step CXX)

find_package(deal.II 9.5.0 REQUIRED)

add_executable(step step.cc)
target_link_libraries(step dealii::dealii)
```

2.7 New and improved tutorials and code gallery programs

While there are no new deal.II tutorial programs in this release, many were extensively revised: Around 145 of the more than 2200 (non-merge) commits that went into this release touched the tutorial, in some cases adding substantial amounts of text.

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

- *'A posteriori error estimator for first order hyperbolic problems'*, contributed by Marco Feder;
- *'Distributed moving laser heating'*, contributed by Hongfeng Ma and Tatiana E. Itina;
- *'Generalized Swift–Hohenberg equation solver'*, contributed by Sam Scheuerman.
- *'Information density-based mesh refinement'*, contributed by Wolfgang Bangerth.

2.8 Incompatible changes

The 9.5 release includes around 35 incompatible changes (see [54]). Many of these incompatibilities 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 class hp::DoFHandler has been removed. The DoFHandler class now implements all *hp*-related functionality, as it has for the past two releases.
- We improved the type safety for active and future FE indices in the DoFHandler implementation by introducing a new data type types::fe_index. Corresponding functions like DoFCellAccessor::active_fe_index() and DoFHandler::get_active_fe_indices() have changed their interface.

This rework also affects the serialization process of active FE indices. You will need to recreate your serialized data *if and only if* you work in *hp*-mode, but can continue to use previously generated data otherwise. For `parallel::distributed::Triangulation` when used with non-*hp* data, it is sufficient to increase the version in the metadata file from '4' to '5'.
– Several old interfaces to `MatrixFree` have been removed, e.g., initialization functions without `Mapping` argument and some queries to the number of cell batches as well as `DoFHandler` objects. In each case, new interfaces are available.

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 [14], and a more recent publication documenting `deal.II`'s design decisions is available as [10]. If you rely on specific features of the library, please consider citing any of the following:

– For geometric multigrid: [23, 43, 44, 59];
– For distributed parallel computing: [13];
– For *hp*-adaptivity: [15, 30];
– For partition-of-unity method (PUM) and finite element enrichment method: [28];
– For matrix-free and fast assembly techniques: [47, 48];
– For computations on lower-dimensional manifolds: [29];
– For curved geometry representations and manifolds: [39];
– For integration with CAD files and tools: [38];
– For boundary element computations: [34];
– For the `LinearOperator` and `PackagedOperation` facilities: [57, 58];
– For uses of the `WorkStream` interface: [74];
– For uses of the `ParameterAcceptor` concept, the `MeshWorker::ScratchData` base class, and the `ParsedConvergenceTable` class: [67];
– For uses of the particle functionality in `deal.II`: [32].

`deal.II` can interface with many other libraries:

– ADOL-C [35]
– ArborX [52]
– ARPACK [53]
– Assimp [68]
– BLAS and LAPACK [5]
– Boost [18]
– CGAL [70]
– cuSOLVER [25]
– cuSPARSE [26]
– Gmsh [33]
– GSL [31, 36]
– Ginkgo [6, 7]
– HDF5 [71]
– METIS [45]
– MUMPS [3, 4]
– muparser [60]
– OpenCASCADE [61]
– p4est [21, 22]
– PETSc [11, 12]
– ROL [66]
– ScaLAPACK [17]
– SLEPc [40]
– SUNDIALS [42]
– SymEngine [69]
– TBB [65]
– Trilinos [41, 72]
– UMFPACK [27]

Please consider citing the appropriate references if you use interfaces to these libraries.
The two previous releases of `deal.II` can be cited as [8, 9].

4 Acknowledgments

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

`deal.II` is a worldwide 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, Nicolas Barnafi, Nistha Bhawsinka, Bruno Blais, Fabian Castelli, Terry Cojean, Niklas Fehn, Emmanuel Ferdman, Vadim Gallyamov, Daniel Garcia-Sanchez, Rene Gassmoeller, Robin Goermer, Graham Harper, Quang Hoang, Sean Ingimarson, Vladimir Ivannikov, Pengfei Jia, Tao Jin, Nils Margenberg, Luz Paz, Laura Prieto Saavedra, Sebastian Proell, Ce Qin, Oleg Rogozin, Andrew Salmon, Michael Schlottke-Lakemper, Christoph Schmidt, Magdalena Schreter-Fleischhacker, Richard Schussnig, Nils Schween, Ahmad Shahba, Simon Sticko, Buğrahan Temür, Ivy Weber, Niklas Wik, Vladimir Yushutin, Jiaqi Zhang.

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 and T. Heister were partially supported by the Computational Infrastructure for Geodynamics initiative (CIG), through the National Science Foundation (NSF) under Award No. EAR-1550901 and EAR-2149126 via 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.

M. Bergbauer was supported by the German Research Foundation (DFG) under the project 'High-Performance Cut Discontinuous Galerkin Methods for Flow Problems and Surface-Coupled Multiphysics Problems' Grant Agreement No. 456365667.

J. Heinz was supported by the European Union's Framework Programme for Research and Innovation Horizon 2020 (2014-2020) under the Marie Skłodowska-Curie Grant Agreement No. 812719.

T. Heister was also partially supported by NSF Awards OAC-2015848, DMS-2028346, and EAR-1925575.

L. Heltai and M .Feder were partially supported by the Italian Ministry of University and Research (MUR), under the grant MUR PRIN 2022 No. 2022WKWZA8 'Immersed methods for multiscale and multiphysics problems (IMMEDIATE)'.

M. Kronbichler and P. Munch were partially supported by the German Ministry of Education and Research, project 'PDExa: Optimized software methods for solving partial differential equations on exascale supercomputers' and the Bayerisches Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen (KONWIHR), projects 'High-order matrix-free finite element implementations with hybrid parallelization and improved data locality' and 'Fast and scalable finite element algorithms for coupled multiphysics problems and non-matching grids'.

M. Maier was partially supported by NSF Award DMS-2045636 and and by the Air Force Office of Scientific Research under grant/contract number FA9550-23-1-0007.

D. Wells was supported by the NSF Award OAC-1931516.

S. Zampini was supported by the KAUST Extreme Computing Research Center.

Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster.

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper (see http://www.tacc.utexas.edu).

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant No. ACI-1053575 access through the CIG Science Gateway and Community Codes for the Geodynamics Community MCA08X011 allocation.

References

[1] S. Abhyankar, J. Brown, E. M. Constantinescu, D. Ghosh, B. F. Smith, and H. Zhang, PETSc/TS: A modern scalable ODE/DAE solver library, *arXiv:1806.01437*, 2018.

[2] M. Adams, Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics, *Int. J. Numer. Meth. Engrg.* **55** (2002), No. 5, 519–534.

[3] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, Performance and scalability of the block low-rank multifrontal factorization on multicore architectures, *ACM Trans. Math. Software* **45** (2019), No. 1, 2/1–26.

[4] 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 Anal. Appl.* **23** (2001), No. 1, 15–41.

[5] 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.

[6] H. Anzt, T. Cojean, Y.-C. Chen, G. Flegar, F. Göbel, T. Grützmacher, P. Nayak, T. Ribizel, and Y.-H. Tsai, Ginkgo: A high performance numerical linear algebra library, *J. Open Source Software* **5** (2020), No. 52, 2260.

[7] H. Anzt, T. Cojean, G. Flegar, F. Göbel, T. Grützmacher, P. Nayak, T. Ribizel, Y. M. Tsai, and E. S. Quintana-Ortí, Ginkgo: A modern linear operator algebra framework for high performance computing, *ACM Trans. Math. Software* **48** (2022), No. 1, 2/1–33.

[8] 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.

[9] D. Arndt, W. Bangerth, B. Blais, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, U. Köcher, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, S. Proell, K. Simon, B. Turcksin, D. Wells, and J. Zhang, The `deal.II` Library, Version 9.3, *J. Numer. Math.* **29** (2021), No. 3, 171–186.

[10] 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 & Mathematics with Applications* **81** (2021), 407–422.

[11] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, and J. Zhang, *PETSc/TAO Users Manual*, Argonne National Laboratory, Report No. ANL-21/39 – Revision 3.17, 2022.

[12] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, and J. Zhang, *PETSc Web Page*, https://petsc.org/, 2023.

[13] 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** (2012), No. 2, 14/1–28.

[14] 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–es.

[15] W. Bangerth and O. K.-Herold, Data structures and requirements for *hp* finite element software, *ACM Trans. Math. Software* **36** (2009), No. 1, 4/1–31.

[16] M. L. Bittencourt, C. C. Douglas, and R. A. Feijóo, Nonnested multigrid methods for linear problems, *Numerical Methods for Partial Differential Equations: An Int. J.* **17** (2001), No. 4, 313–331.

[17] 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.

[18] *Boost C++ Libraries*, http://www.boost.org/.

[19] J. H. Bramble, J. E. Pasciak, and J. Xu, The analysis of multigrid algorithms with nonnested spaces or noninherited quadratic forms, *Mathematics of Computation* **56** (1991), No. 193, 1–34.

[20] P. R. Brune, M. G. Knepley, B. F. Smith, and X. Tu, Composing scalable nonlinear algebraic solvers, *SIAM Review* **57** (2015), No. 4, 535–565.

[21] 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.

[22] C. Burstedde, Parallel tree algorithms for AMR and non-standard data access, *ACM Trans. Math. Software* **46** (2020), No. 4, 32/1–31.
[23] 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, 7/1–27.
[24] W. Couzy, Spectral element discretization of the unsteady Navier–Stokes equations and its iterative solution on parallel computers, *EPFL Report*, 1995.
[25] *cuSOLVER Library*, https://docs.nvidia.com/cuda/cusolver/index.html.
[26] *cuSPARSE Library*, https://docs.nvidia.com/cuda/cusparse/index.html.
[27] T. A. Davis, Algorithm 832: UMFPACK V4.3 – an unsymmetric-pattern multifrontal method, *ACM Trans. Math. Software* **30** (2004), 196–199.
[28] 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, *Advanced Modeling and Simulation in Engineering Sciences* **4** (2017), No. 1, 7.
[29] 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.
[30] M. Fehling and W. Bangerth, Algorithms for parallel generic *hp*-adaptive finite element software, *ACM Trans. Math. Software* **48** (2022).
[31] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, *GNU Scientific Library Reference Manual*, 3rd ed., Network Theory Ltd., 2009.
[32] 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.
[33] 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.
[34] N. Giuliani, A. Mola, and L. Heltai, π-BEM: A flexible parallel implementation for adaptive, geometry aware, and high order boundary element methods, *Advances in Engineering Software* **121** (2018), 39–58.
[35] 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.
[36] *GSL: GNU Scientific Library*, http://www.gnu.org/software/gsl.
[37] J. Heinz, P. Munch, and M. Kaltenbacher, High-order non-conforming discontinuous Galerkin methods for the acoustic conservation equations, *Int. J. Numer. Methods Engrg.* **124** (2023), No. 9, 2034–2049.
[38] 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.
[39] L. Heltai, W. Bangerth, M. Kronbichler, and A. Mola, Propagating geometry information to finite element computations, *ACM Trans. Math. Software* **47** (2021), No. 4, 32/1–30.
[40] 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.
[41] 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.
[42] 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.
[43] 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.
[44] G. Kanschat, Multi-level methods for discontinuous Galerkin FEM on locally refined meshes, *Comput. & Struct.* **82** (2004), No. 28, 2437–2445.
[45] 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.
[46] D. A. Knoll and D. E. Keyes, Jacobian-free Newton–Krylov methods: a survey of approaches and applications, *J. Comput. Physics* **193** (2004), No. 2, 357–397.
[47] M. Kronbichler and K. Kormann, A generic interface for parallel cell-based finite element operator application, *Comput. Fluids* **63** (2012), 135–147.
[48] M. Kronbichler and K. Kormann, Fast matrix-free evaluation of discontinuous Galerkin finite element operators, *ACM Trans. Math. Software* **45** (2019), No. 3, 29/1–40.
[49] M. Kronbichler, K. Kormann, N. Fehn, P. Munch, and J. Witte, A Hermite-like basis for faster matrix-free evaluation of interior penalty discontinuous Galerkin operators, *arXiv:1907.08492*, 2019.
[50] M. Kronbichler, D. Sashko, and P. Munch, Enhancing data locality of the conjugate gradient method for high-order matrix-free finite-element implementations, *The Int. J. High Performance Computing Applications* (2022) 10943420221107880.
[51] M. Kronbichler, S. Schoeder, C. Müller, and W. A. Wall, Comparison of implicit and explicit hybridizable discontinuous Galerkin methods for the acoustic wave equation, *Int. J. Numer. Meth. Eng.* **106** (2016), No. 9, 712–739.
[52] D. Lebrun-Grandié, A. Prokopenko, B. Turcksin, and S. R. Slattery, ArborX: a performance portable geometric search library, *ACM Trans. Math. Software* **47** (2020), No. 1, 2/1–15.

[53] 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.
[54] *List of Changes for 9.5*, https://www.dealii.org/developer/doxygen/deal.II/changes_between_9_4_2_and_9_5_0.html.
[55] J. Lottes, Optimal polynomial smoothers for multigrid V-cycles, *arXiv:2202.08830*, 2022.
[56] R. E. Lynch, J. R. Rice, and D. H. Thomas, Direct solution of partial difference equations by tensor product methods, *Numerische Mathematik* **6** (1964), No. 1, 185–199.
[57] M. Maier, M. Bardelloni, and L. Heltai, LinearOperator – a generic, high-level expression syntax for linear algebra, *Computers and Mathematics with Applications* **72** (2016), No. 1, 1–24.
[58] M. Maier, M. Bardelloni, and L. Heltai, *LinearOperator Benchmarks, Version 1.0.0*, March 2016.
[59] P. Munch, T. Heister, L. P. Saavedra, and M. Kronbichler, Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations, *ACM Trans. Parallel Computing* **10** (2023), No. 1, 1–38.
[60] *muparser: Fast Math Parser Library*, https://beltoforion.de/en/muparser.
[61] *OpenCASCADE: Open CASCADE Technology, 3D Modeling & Numerical Simulation*, http://www.opencascade.org/.
[62] M. Phillips and P. Fischer, Optimal Chebyshev smoothers and one-sided V-cycles, *arXiv:2210.03179*, 2022.
[63] M. Phillips, S. Kerkemeier, and P. Fischer, Auto-tuned preconditioners for the spectral element method on GPUs, *arXiv:2110.07663*, 2021.
[64] S. D. Proell, P. Munch, M. Kronbichler, W. A. Wall, and C. Meier, A highly efficient computational framework for fast scan-resolved simulations of metal additive manufacturing processes on the scale of real parts, *arXiv:2302.05164*, 2023.
[65] J. Reinders, *Intel Threading Building Blocks*, O'Reilly, 2007.
[66] D. Ridzal and D. P. Kouri, *Rapid Optimization Library*, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), Report, 2014.
[67] 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.
[68] 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, 2021.
[69] *SymEngine: Fast Symbolic Manipulation Library, Written in C++*, https://symengine.org/.
[70] The CGAL Project, *CGAL User and Reference Manual*, 5.4.1 ed., CGAL Editorial Board, 2022, https://doc.cgal.org/5.4.1/Manual/packages.html.
[71] The HDF Group, *Hierarchical Data Format, version 5*, 2022, http://www.hdfgroup.org/HDF5/.
[72] The Trilinos Project Team, *The Trilinos Project Website*, https://trilinos.github.io/.
[73] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. Wilke, Kokkos 3: Programming Model Extensions for the Exascale Era, *IEEE Trans. Parallel Distributed Systems* **33** (2022), No. 4, 805–817.
[74] 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, 2/1–29.
[75] J. Witte, D. Arndt, and G. Kanschat, Fast tensor product Schwarz smoothers for high-order discontinuous Galerkin methods, *Computational Methods in Applied Mathematics* **21** (2021), No. 3, 709–728.
[76] J. Zhang, J. Brown, S. Balay, J. Faibussowitsch, M. Knepley, O. Marin, R. T. Mills, T. Munson, B. F. Smith, and S. Zampini, The PetscSF scalable communication layer, *IEEE Trans. Parallel Distributed Systems* **33** (2021), No. 4, 842–853.