The deal.II Library, Version 9.4

Daniel Arndt[1*], Wolfgang Bangerth[2,3], Marco Feder[4], Marc Fehling[2], Rene Gassmöller[5], Timo Heister[6], Luca Heltai[4], Martin Kronbichler[7,8], Matthias Maier[9], Peter Munch[8,10], Jean-Paul Pelteret[11], Simon Sticko[7,12], Bruno Turcksin[1*], and David Wells[13]

[1]Scalable Algorithms and Coupled Physics Group, Computational Sciences and Engineering Division, Oak Ridge National Laboratory, 1 Bethel Valley Rd., TN 37831, USA. arndtd/turcksinbr@ornl.gov
[2]Department of Mathematics, Colorado State University, Fort Collins, CO 80523-1874, USA. bangerth/marc.fehling@colostate.edu
[3]Department of Geosciences, Colorado State University, Fort Collins, CO 80523, USA.
[4]SISSA, International School for Advanced Studies, Via Bonomea 265, 34136, Trieste, Italy. marco.feder/luca.heltai@sissa.it
[5]Department of Geological Sciences, University of Florida, 1843 Stadium Road, Gainesville, FL, 32611, USA. rene.gassmoeller@ufl.edu
[6]School of Mathematical and Statistical Sciences, Clemson University, Clemson, SC, 29634, USA heister@clemson.edu
[7]Department of Information Technology, Uppsala University, Box 337, 751 05 Uppsala, Sweden. martin.kronbichler/simon.sticko@it.uu.se
[8]Institute of Mathematics, University of Augsburg, Universitätsstr. 12a, 86159 Augsburg, Germany. martin.kronbichler@uni-a.de
[9]Department of Mathematics, Texas A&M University, 3368 TAMU, College Station, TX 77845, USA. maier@math.tamu.edu
[10]Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Max-Planck-Str. 1, 21502 Geesthacht, Germany. peter.muench@hereon.de
[11]Independent researcher. jppelteret@gmail.com
[12]Department of Mathematics and Mathematical Statistics, Umeå University, SE-90187 Umeå, Sweden
[13]Department of Mathematics, University of North Carolina, Chapel Hill, NC 27516, USA. drwells@email.unc.edu

Abstract: This paper provides an overview of the new features of the finite element library deal.II, version 9.4.

1 Overview

deal.II version 9.4.0 was released June 24, 2022. 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 9.4. 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:

– Advances in simplex- and mixed-mesh support (see Section 2.1);

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

- Repartitioning of distributed meshes (see Section 2.2);
- Advances in matrix-free infrastructure (see Section 2.3);
- Advances in multigrid infrastructure (see Section 2.4);
- CutFEM support (see Section 2.5);
- Experimental integration of the Computational Geometry Algorithms Library (CGAL) (see Section 2.6);
- Performance improvements in the particle infrastructure (see Section 2.7);
- Support for large MPI buffers and parallel I/O (see Section 2.8);
- Improvements to unstructured communication (see Section 2.9);
- Three new tutorial programs and one new code gallery program (see Section 2.10).

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 `DataOutResample` class interpolates values defined on one triangulation onto a second, potentially unrelated triangulation. By using this class, one can output the result obtained on an unstructured mesh on a structured one (which might facilitate a more memory-efficient storage format, for example, if this second triangulation is a uniformly refined rectangle or box), or one can create a slice in 3D.
- The new member function `find_point_owner_rank()` of `parallel::distributed::Triangulation` allows one to find the MPI ranks of cells containing specified points. It is communication-free and leverages the functionality of `p4est` (>v.2.2). Its algorithm is described in [19]. This information will enable efficient construction of the communication pattern used in the class `Utilities::MPI::RemotePointEvaluation`. Furthermore, this function could be used in the future to allow particle simulations in which particle movement is not limited by CFL conditions, as done in [58].
- The new function `GridGenerator::pipe_junction()` generates a triangulation of three cone-shaped pipes that cross at a bifurcation point in any possible configuration. A manifold description is applied to the boundary, which can be extended into the volume via transfinite interpolation [33] using the `TransfiniteInterpolationManifold` class [1].
- A new DoF renumbering function `DoFRenumbering::support_point_wise()` which groups together shape functions by their support point. This functionality is useful in developing nodal schemes since, e.g., the x, y, and z components at a point will be consecutive in the solution vector. It also improves interoperability with external libraries which expect data in this format.
- The `FEInterfaceValues` class, which computes common quantities at the interface of two cells, has been overhauled to make it more consistent with the rest of the library and use more intuitive names for functions. For example, `FEInterfaceValues::jump_gradient()` is now `FEInterfaceValues::jump_in_shape_gradients()`. Several new functions, such as `FEInterfaceValues::get_jump_in_function_values()`, have also been added.
- The `MeshWorker::ScratchData` and `MeshWorker::CopyData` have been made *hp*-compatible, and support face integration where the integration rule and mapping differs on either side of an interface. The `MeshWorker::CopyData` class has also been made compatible with complex numbers.

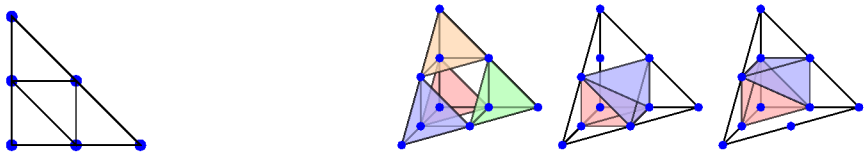


Figure 1: *The previous release, 9.3 [8], added support for adaptive mesh refinement with triangles. This release adds support for global tetrahedral refinement by subdividing each tetrahedron into eight children, as shown on the right.*

– Vectors attached to `DataOut` do not need to be in ghosted state anymore. Internally, we create a copy of the vector with appropriate ghost elements.

The changelog lists more than 100 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 [53].

2.1 Advances in simplex- and mixed-mesh support

`deal.II` has supported simplex and mixed meshes since the previous release, 9.3, on an experimental basis. We have continued to work on this support; the current state is sufficient to support several larger-scale programs, although not all functionality in `deal.II` works with such meshes yet. Specifically, we have fixed many bugs and generalized existing functions that previously only worked for hypercube-shaped cells. The most notable new functionalities are:

– Experimental support for locally refined meshes: For finite element computations on locally refined meshes, one needs (i) the possibility to locally refine the mesh (see Figure 1), and (ii) appropriate hanging-node constraints. Both are now available in 2D; for 3D, the implementation of the constraint definitions is still in progress.

– `QIteratedSimplex` allows to build composite simplex quadrature rules.

– The new wrappers to the `CGAL` library allow for the easy creation of simplex meshes. For more details, see Section 2.6.

Furthermore, we have continued to remove uses of the `GeometryInfo` class (which is specific to hypercube cells) from the library, and to replace them with more general equivalent functionality based on the `ReferenceCell` class. Once all instances of `GeometryInfo` are removed, we will deprecate the class.

2.2 Repartitioning of distributed meshes

`deal.II` has two classes that support distributed storage of meshes: `parallel::distributed::Triangulation` (which in the following we will abbreviate as `p::d::T`) and `parallel::fully distributed::Triangulation` (or, in short, `p::f::T`). The former is partitioned based on the space-filling Morton (or Z-order) curve implemented by the `p4est` backend. The latter, more recent class is currently statically partitioned at creation time.

There are now new utility functions in the `RepartitioningPolicyTools` namespace that can be used to create a new `p::f::T` instance, given a distributed triangulation (`p::f::T` or `p::d::T`) and

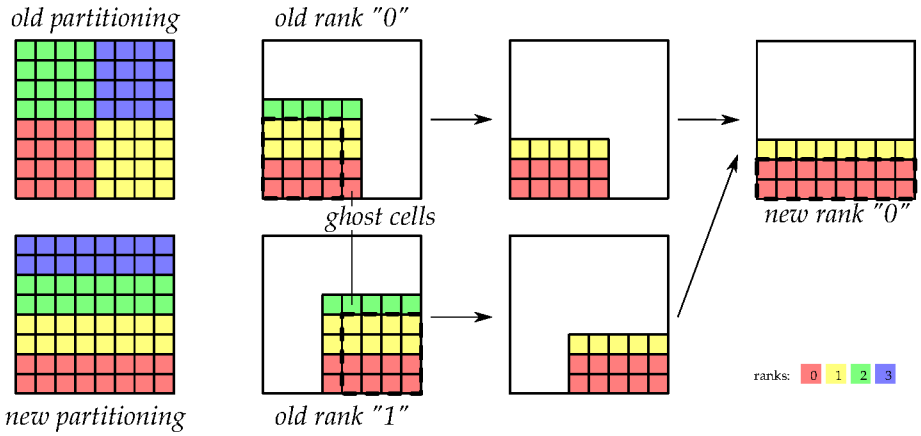


Figure 2: *Visualization of the repartitioning process and the construction of the new mesh of process 0. Top left: Existing ownership of the cells of the mesh, distributed on four processes. Bottom left: Requested new ownership. Second column: Processes 0 and 1 collect information about the cells they own or that are ghost cells. Processes 2 and 3 do not contribute to process 0. Third column: What processes 0 and 1 would send to process 0. Fourth column: The combined knowledge on process 0.*

a vector that indicates the designated owner processes of locally owned cells. This approach allows for partitioning `p::f::T` based on arbitrary criteria. The workflow is shown in the following listing:

C++ code

```
// Original triangulation:
parallel::distributed::Triangulation<dim> tria(communicator);

// Select a partitioning policy and use it:
const RepartitioningPolicyTools::CellWeightPolicy<dim> policy(tria, fu);
const auto construction_data = TriangulationDescription::Utilities::
  create_description_from_triangulation(tria, policy.partition(tria));

// Create the new triangulation:
parallel::fullydistributed::Triangulation<dim> tria_pft(comm);
tria_pft.create_triangulation(construction_data);
```

The setup process (of `construction_data`) is visualized in Figure 2. At first, locally owned cells and their surrounding (ghost) cells are collected on each process and sent to the new owner. On the receiving side, the sets of all cells are combined and possible duplicates are removed. This information is enough to construct a new triangulation. For sending/receiving, we apply consensus-based algorithms [42], which we introduced into the library in release 9.2 [7] – see also Section 2.9. This kind of consensus-based algorithm is used also in [43] for repartitioning.

In addition to the predefined partitioning policies, users can write their own by implementing the `RepartitioningPolicyTools::Base` interface. Like the "active" level of finest cells, the scheme also allows for arbitrarily repartitioned multigrid levels.

In future releases, we plan to add support for repartitioning based on distributed graph partitioning libraries, e.g., `ParMETIS` or `Zoltan`. Furthermore, we intend to extend `p::f::T` to support adaptive mesh refinement, completing `p::f::T` as a complement to `p::d::T`.

2.3 Advances in matrix-free infrastructure

The matrix-free infrastructure of `deal.II` is used to solve problems without assembling matrices, providing only the action of the matrix instead. It has seen a number of new features in this release, the most notable ones being:

- Improved support for computations with Hessians of shape functions: Just like values and gradients, Hessians can be now evaluated and integrated during matrix-free loops both for cells and faces. This could enable, for example, writing a matrix-free version of step-47, which solves the biharmonic equation with the discontinuous Galerkin method.

- Cell-centric loops now also allow access to gradients and Hessians of neighboring cells on faces. The major difficulty here was the relative orientation of cells on unstructured meshes.

- Users can now create their own cell batches, by providing `FEEvaluation::reinit()` a list of cell IDs. `FEEvaluation` accesses the appropriate data and reshuffles mapping data accordingly on the fly in order to enable vectorization over cells. The new feature is useful in several contexts, for example for sharp interfaces (e.g., two-phase flow or shock capturing), where one needs to treat cells that are "cut" by the interface in a special way. A challenge is that cell batches might contain cut or non-cut cells, effectively counteracting the cross-cell vectorization. Previous functionality has provided the option of masking certain cells in cell batches, which works well if the code paths do not diverge too much, or to categorize cells during `MatrixFree::reinit()` in such a way that mixed cell batches do not occur. However, `MatrixFree::reinit()` might be too expensive if recategorization needs to happen very frequently to follow the dynamics of a system, e.g., in each time step. Despite some overhead compared to static matrix-free loops, the new feature can be the best option in such dynamic scenarios.

- Initial support for H(div)-conforming elements with Piola transform, based on Raviart–Thomas finite element spaces with the updated class `FE_RaviartThomasNodal`, has been added. This feature is currently limited to meshes in standard orientation and affine geometries. Full support and performance optimizations will be provided in a future release.

- Selected matrix-free algorithms can now exploit additional data locality between the matrix-vector product and vector operations happening nearby in an algorithm. Interfaces have been added to both the `PreconditionChebyshev` class (a frequently used smoother in multigrid methods) and the conjugate gradient implementation in `SolverCG`. From the user's perspective, an operator needs to define a `vmult` operation, taking two additional `std::function` arguments. The first function defines the operation to be scheduled on the vector entries before the matrix-vector product touches them, and the second what happens afterward. The new features also include a renumbering to maximize data locality. The theory is described in the contribution [50].

Besides these new features, we improved the performance of hanging-node-constraint evaluation on the CPU. Instead of performing quasi-dense matrix-vector multiplications [48], we now use an approach based on in-place interpolation and sum factorization, similar to what was already done in the GPU code [54]. In [60], the algorithm is described and performance numbers are shown, indicating a reduction of overhead of cells with hanging nodes by a factor of ten.

Finally, we have performed a major restructuring of the internals of the `FEEvaluation` classes. This reduces some overhead for low polynomial degrees and will enable us to add support for new element types in the future.

We would like to remind users that we transitioned from the use of Booleans to flags to configure the evaluation and integration process of `FEEvaluation` and `FEFaceEvaluation`:

C++ code

```
fe_eval.evaluate(false, true, false)          // old (deprecated)
fe_eval.evaluate(EvaluationFlags::gradients) // new
```

2.4 Advances in multigrid infrastructure

In release 9.3 [8], we added support for global-coarsening multigrid in addition to the established local-smoothing infrastructure. Global coarsening algorithms smoothen over the whole computational domain on each multigrid level, which is obtained by coarsening the finest cells of the next finer multigrid level. For this purpose, we use a sequence of triangulations, and we perform the smoothing only on their active levels. To create the sequence of triangulations, one can use the functions `MGTransferGlobalCoarseningTools::create_geometric_coarsening_sequence()`. A new version takes an instance of `RepartitioningPolicyTools::Base` (see Subsection 2.2) as argument, which allows specifying the parallel distribution of each multigrid level (in contrast to the fixed first-child policy in the case of local smoothing). These features have been developed and tuned for running on a supercomputer scale with complicated coarse meshes as presented in [47]. Furthermore, we added support for block vectors, fixed a number of limitations, and performed performance optimizations of the transfer operator; particularly, the redundant copy from/to temporary vectors has been eliminated. Furthermore, hanging-node constraints are applied efficiently in the same way as in the matrix-free loops (see Subsection 2.3).

In [59], the performance of the local-smoothing and global-coarsening infrastructure of `deal.II` was compared for locally refined meshes. The results indicate that the local definition of multigrid levels might introduce load imbalances in the case of local smoothing so that global coarsening is favorable despite potentially more expensive intergrid transfers. In order to assess the benefits of one approach against the other, `deal.II` provides new functions `workload_imbalance()` and `vertical_communication_efficiency()` in the `MGTools` namespace for the estimation of the imbalance during, e.g., smoothing or the communication efficiency during intergrid transfer, purely based on the given mesh.

2.5 CutFEM support

Several classes have been added to the `NonMatching` namespace to enable the use of cut finite element methods [18]. In the literature, these types of methods are also referred to as immersed, extended, or fictitious finite element methods. Here, the domain, Ω, is immersed in the background mesh, as illustrated in Fig. 3a. Often, one solves for the degrees of freedom of the smallest submesh which completely covers the domain, i.e., the blue and yellow cells of Fig. 3b. The bilinear form in the weak form would then, for example, look like

$$a(u,v) = (\nabla u, \nabla v)_\Omega - (\partial_n u, v)_\Gamma + \dots \tag{1}$$

Thus, when assembling on a cut cell K, we are required to integrate over the part of the domain and the part of the boundary, $\Gamma = \partial\Omega$, that falls inside the cell: $K \cap \Omega$ and $K \cap \Gamma$. Many of the new classes that support these operations assume that the domain is described by a level set function, $\psi : \mathbb{R}^d \to \mathbb{R}$, such that

$$\Omega = \{x \in \mathbb{R}^d : \psi(x) < 0\}, \qquad \Gamma = \{x \in \mathbb{R}^d : \psi(x) = 0\}. \tag{2}$$

Specifically, the following are the key new classes and functions:

- The `MeshClassifier` class identifies how the active cells and faces are located relative to the zero contour of the level set function, as illustrated in Figure 3b. Its member function `location_to_level_set()` takes a cell or face and returns an enum, `LocationToLevelSet`,

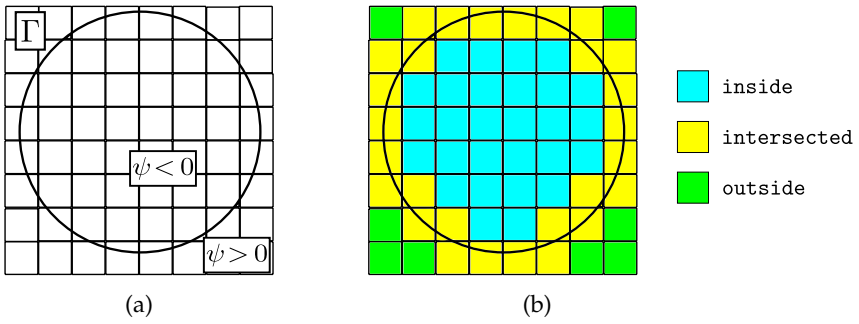


Figure 3: *(a) Domain immersed in a background mesh. (b) Value of* `LocationToLevelSet` *for each cell.*

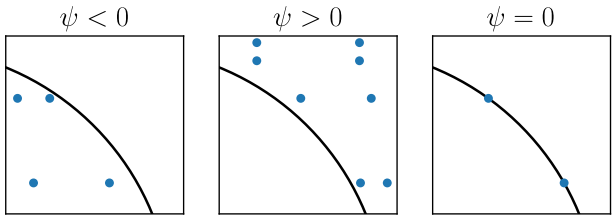


Figure 4: *Quadrature points for integrating over the three different regions of a cell cut by the zero contour of the level set function, ψ.*

with values {`inside`, `outside`, `intersected`}. This information is typically needed when choosing what element (e.g., `FE_Q` or `FE_Nothing`) and/or what quadrature (defined over the complete cell or over a part—see below) a cell should use.

– The `QuadratureGenerator` class, which implements the algorithm in [67], generates high-order quadrature rules for the three different regions of a `BoundingBox`, B, defined by the sign of the level set function:

$$B \cap \Omega = \{x \in B : \psi(x) < 0\}, \quad B \cap \Gamma = \{x \in B : \psi(x) = 0\}, \quad \{x \in B : \psi(x) > 0\}. \tag{3}$$

An example of these quadratures is shown in Figure 4. The `FaceQuadratureGenerator` class does the same for faces. Furthermore, the new classes `DiscreteQuadratureGenerator` and `DiscreteFaceQuadratureGenerator` can be used to generate these quadrature rules over a cell or face when the level set function lies in a finite element space: $\psi_h \in V_h$, and when the reference cell of the cell or face is a hypercube.

– `ImmersedSurfaceQuadrature` is a class representing a quadrature rule over a $(d-1)$-dimensional surface embedded in $\mathbb{R}^d$ ($\psi = 0$ in Figure 4). In addition to the weight, it stores the unit normal to the surface, for each quadrature point. This is needed to transform the quadrature rule from reference space to real space.

– `FEImmersedSurfaceValues` is an `FEFaceValues`-like class for evaluating real space values based on an `ImmersedSurfaceQuadrature`.

– `NonMatching::FEValues` combines the functionality of several of the above classes to simplify assembly of linear systems. It works similarly to `hp::FEValues`: When calling the `reinit()` function, immersed quadrature rules are generated in the background and `FEValues` objects for the inside/outside region and a `FEImmersedSurfaceValues` object for the surface regions are set up internally. These can then be obtained using getter-functions (`get_inside/outside/surface_fe_values()`) and used for the assembly. Since the generation of immersed quadrature rules is not cheap, `NonMatching::FEValues` calls `QuadratureGenerator` only if needed, i.e., if the cell is intersected. If not, already cached

`FEValues` objects will be returned by the getter functions. Correspondingly, the class `NonMatching::FEInterfaceValues` generates `FEInterfaceValues` objects for assembling face terms over $F \cap \{x : \psi(x) < 0\}$ or $F \cap \{x : \psi(x) > 0\}$.

The new `step-85` tutorial illustrates how many of these classes work together.

2.6 Experimental integration of the Computational Geometry Algorithms Library (CGAL)

The Computational Geometry Algorithms Library (CGAL, https://www.cgal.org/) is a widely used library to describe geometries and meshes [70]. `deal.II` now has wrappers for CGAL classes and functions, provided in the new namespace `CGALWrappers`: they implement functionality spanning from mesh generation to boolean operations between triangulations and cells. These wrappers are enabled only if `deal.II` is compiled with `C++17`. *This feature is still experimental and interfaces might change during the next release cycle.*

The main mesh generation function is `GridGenerator::implicit_function()`, which creates a `Triangulation<dim,3>` out of the zero level set of an implicit function ψ similar to (2). For `dim==3`, the mesh consists of tetrahedra. A prototypical use case is the following, where the surface is the zero level set of Taubin's heart function $f = \left(x^2 + \frac{9y^2}{4} + z^2 - 1\right) - x^2z^3 - \frac{9y^2z^3}{80}$. The resulting `Triangulation<3>` is shown in Figure 5a and the required steps are:

C++ code

```
// 1) An implicit function, e.g., Taubin's heart surface (not shown)
ImplicitFunction f;

// 2) Configure output mesh (optional)
CGALWrappers::AdditionalData<3> data; data.cell_size = .05;

// 3) Create mesh
Triangulation<3> tria;
GridGenerator::implicit_function(tria, f, data, /* more arguments */...);
```

A related function is `GridGenerator::surface_mesh_to_volumetric_mesh()`, which computes a tetrahedral volume mesh `Triangulation<3>`, based on a given surface mesh `Triangulation<2,3>` that bounds the three dimensional shape.

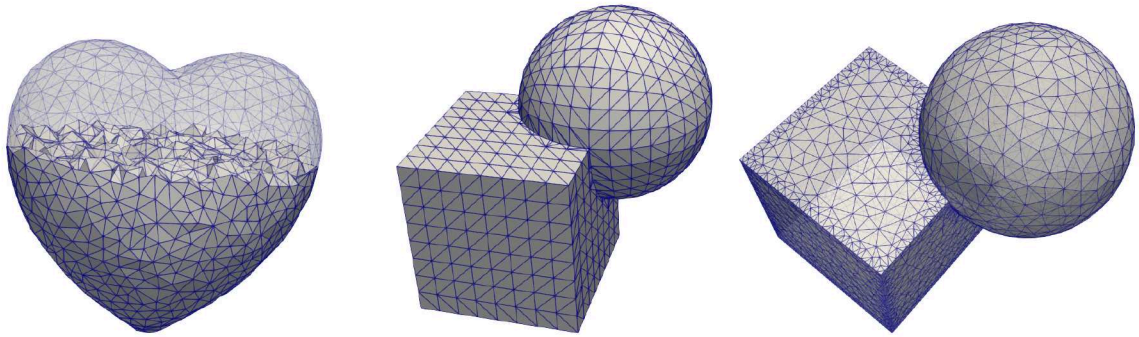
CGAL also provides Boolean operations on meshes: they can be accessed via the utility function `CGALWrappers::compute_boolean_operation()`. The available operations are *co-refinement*, *intersection*, *union*, and *difference*. Oftentimes, Boolean operations and co-refinement around the intersection produces badly shaped mesh cells. To overcome this issue, one can use `CGALWrappers::remesh_surface()`. A possible workflow to create a high-quality mesh of the union of a cube and a sphere mesh is given in the following listing:

C++ code

```
// 1) Create deal.II triangulations, e.g., cube and sphere (not shown)
Triangulation<spacedim> tria0, tria1;

// 2) Convert to CGAL surface meshes (assuming Kernel is already defined)
CGAL::Surface_mesh<Kernel> surface_mesh0, surface_mesh1;
CGALWrappers::dealii_tria_to_cgal_surface_mesh(tria0, surface_mesh0);
CGALWrappers::dealii_tria_to_cgal_surface_mesh(tria1, surface_mesh1);

// 3) Compute the union of the two meshes
CGALWrappers::compute_boolean_operation(surface_mesh0, surface_mesh1,
  BooleanOperation::compute_union, out_mesh);
```



(a) (b) (c)

Figure 5: *(a) Triangulation created by filling a heart-shaped surface implicitly described by a function f. (b) Union of a cube with a sphere with badly shaped cells at the intersection. (c) Remeshed version of the same triangulation.*

```
// 4) Convert CGAL surface mesh to deal.II surface mesh
Triangulation<2, 3> tria_out;
CGALWrappers::cgal_surface_mesh_to_dealii_triangulation(out_mesh, tria_out);

// 5) Convert surface to volume mesh via surface_mesh_to_volumetric_mesh()
```

The output of the Boolean operation is shown in Fig. 5b; Fig. 5c shows the mesh after remeshing.

The function `CGALWrappers::compute_quadrature_on_boolean_operation()` returns a quadrature that allows exact integration of polynomials on polyhedral elements created by a Boolean operation between `deal.II` cells. The quadrature rule is built by (i) subdividing the polyhedral region in tetrahedra, (ii) computing on each a `QGaussSimplex<3>` quadrature rule via `QSimplex<3>::compute_affine_transformation()`, and (iii) collecting all in a single quadrature object on the *physical* cell.

The new utility functions will be the building blocks for functions in the `NonMatching` namespace in future releases. While our current CutFEM implementation (see Section 2.5) relies on a level set description of the domain to determine the boundary and the quadrature rules, one could do that also based on non-matching overlapping grids [57], for which the newly introduced, CGAL-based Boolean operations and quadrature-generation functionality might be helpful. The functionalities are also applicable in a broad context of immersed problems, such as for weakly imposing boundary conditions on the interface of an immersed boundary using Nitsche's method and for coupling terms using Lagrange multipliers.

2.7 Performance improvement of particle infrastructure

In the release described herein, we have reorganized the data structures and optimized the algorithms that support using particles in `deal.II`. In our previously reported implementation [8] we stored all particle data as a separate contiguous array for each particle property, but particle identifiers (IDs) were still stored in a multimap tree-like structure.

Our new particle containers are organized as follows: Particle IDs are stored in a list of dynamic arrays, each array containing the particle IDs of all particles in a unique cell. Each ID is a handle that determines the location of the data of this unique particle in the property arrays. The list of ID arrays only contains entries for cells that contain particles. We keep a separate cache structure that contains pointers to the particular list entries for each cell. If a cell has no particles, this pointer is invalid.

Table 1: *Timing of various particle operations for tutorial program* `step-68` *(particle advection in a 2D, Cartesian box) using 400,000 particles on a single process.*

Particle Operation	`deal.II` 9.3	`deal.II` 9.4	Speedup
Generation	444 ms	235 ms	1.9×
Iteration	4.18 ms	0.638 ms	6.6×
Advection	37.8 ms	33.9 ms	1.15×
Sorting	21.9 ms	9.27 ms	2.4×

This structure allows for the following significant performance improvements:

- All particle data (both their identifiers and their actual data) are now stored as separate and contiguous arrays in memory, which improves spatial locality for better prefetching of data and makes iterating over particles extremely efficient.
- The choice of a list container that only includes entries for cells that contain particles means iteration is efficient, even if many cells in the domain do not contain any particles (as can be the case for discrete element methods [32]).
- Creating separate arrays for each cell allows us to easily move particle IDs from one cell to another as a local operation, affecting only the two cell containers in question. We take care to reuse allocated memory to minimize the number of memory reallocations.
- The separate cache structure that contains entries for each cell allows quick random access to the particles of a particular cell, and also allows quickly determining if a particular cell has particles at all.

In addition to the new storage structure, we have made the following algorithmic improvements: Determining if a particle is inside a cell after changing its position involves inverting the mapping for this cell. We have reorganized our algorithms to perform these inversions on a batch of particles in the same cell instead of particle-by-particle, which allows us to make use of vectorized instructions during the inversion using the generic scheme of [48]. In addition, after sorting all particles into their new cells, the arrays that store particle properties are now sorted in the same order as the particle IDs in the list of arrays, which allows for cache efficient iteration over particle properties. Because the particle IDs are already sorted in the intended order at this time, we can reorder the particle properties as part of a copy operation into a new data container that replaces the existing container. This approach avoids a costly in-place sort of the particle properties.

We illustrate the combined effect of these performance improvements in Table 1. We measure the averaged compute time for four particle operations in a slightly modified version of the `deal.II` tutorial program `step-68` when advecting 400,000 particles on a single process (we have not observed any influence of the described changes on the parallel scalability of the algorithms). The four operations we have measured are:

- Generation of a set of 400,000 particles at positions that are not aligned with the background mesh, i.e., the containing cell of each particle has to be identified with a search algorithm.
- Iteration over the whole set of created particles, without significant computation and in particular without accessing particle data.
- Advection of all particles, which involves iteration over all particles, evaluation of the finite element solution at the location of the particles, read and write access to the position of all particles to modify their location, and write access to the particle properties (to store their velocity for visualization purposes).

Table 2: *Performance comparison of writing large VTU graphical output using MPI I/O on TACC Frontera `/scratch1` file system with 16 file servers ("OSTs") and a theoretical peak performance of 50 GB/s. The ROMIO version used enforces sequential writes when using `MPI_File_write_ordered()`.*

Version	MPI routine used	1 file, striping 16	16 files, no striping
`deal.II` 9.3	`MPI_File_write_ordered()`	3 GB/s	15 GB/s
`deal.II` 9.4	`MPI_File_write_at_all()`	17 GB/s	21 GB/s

– Sorting, i.e., the inversion of the mapping of each cell to find the new particle locations relative to this cell, and moving all particles that have left their original cell into new cells (both `deal.II` 9.3 and `deal.II` 9.4). In `deal.II` 9.4, this operation also includes reordering the particle properties for optimal iteration.

Table 1 shows that all particle operations are much faster in `deal.II` 9.4 than in version 9.3. In particular, operations that depend strongly on particle storage structure and require few fixed computations (like iteration and sorting) benefit massively from the above-mentioned optimizations. We note that the exact gains will depend strongly on the exact combination of geometry, mapping, dimensionality, and the number of particles per cell in any specific model, and can be smaller or larger than the measurements provided here.

2.8 Support for large MPI buffers and parallel I/O

Sending large messages over MPI or reading and writing large buffers using MPI I/O, especially when dealing with small datatypes like `MPI_CHAR`, can often require processing more than 2^{31} objects at once. Prior to the recent MPI-4 standard, which introduced a new set of `MPI_*_c()` functions, all MPI functions expected the "count" to be a signed integer. This limits the number of objects to 2^{31}, i.e., writing of up to 2 GB at once if the datatype is `MPI_CHAR`. Several places in the library could exceed this limit, for example when producing large graphical output, large checkpoint files, or when broadcasting large datasets such as lookup tables.

With this release we introduce a new module `Utilities::MPI::LargeCount` which enables sending and receiving MPI messages and I/O containing more than 2^{31} objects by implementing the before-mentioned `MPI_*_c()` functions, e.g., `MPI_Bcast_c()`, using MPI-3 features if necessary. The solution is based on custom datatypes as described in [36]. This functionality is now used in all places in the library where large counts might be necessary (`DataOut::write_vtu_in_parallel()` for graphical output, checkpointing of `Triangulation` objects, etc.).

While implementing these changes we also worked on the following:

– Testing of large I/O with large chunks per MPI rank (>2 GB) and large total sizes (>4 GB). Several instances of 32-bit data types for offsets were changed to 64-bit to correctly support files larger than 4 GB (HDF5 output, VTU output, checkpointing).

– Performance testing of MPI I/O routines used for parallel VTU output with large performance improvements by switching from a shared file pointer to individual file pointers, see Table 2.

2.9 Improvements to unstructured communication

Many problems in parallel computations can be stated in the following way: Each process in a parallel universe has a number of queries to send to other processes that do not know that they will be asked, and who will then have to respond with replies. This problem is solved by "consensus algorithms" [42]. An example of where this problem appears is given in Section 2.2.

`deal.II` has had an implementation of these algorithms for some time, but the current release substantially expands on it. Specifically, the updated interfaces – now based on function objects such as lambda functions to formulate and process queries and replies – can deal with arbitrary data types for queries and replies, rather than only arrays of data types natively supported by MPI. To make this possible, the implementation packs and unpacks these objects into character arrays via the `Utilities::pack()` and `Utilities::unpack()` functions. We have also worked on making these functions efficient: if the object to be packed is an array (or array of arrays) of elements that satisfy the `std::is_trivially_copyable` type trait, then the data is copied into the character array via `std::memcpy`; only for other objects do the packing and unpacking functions rely on BOOST's serialization library.

A special case of consensus algorithms is where the sender does not actually require an answer. This happens, for example, during repartitioning of meshes (Section 2.2), where a process sends parts of the new mesh to the new owners: the new owner does not know how many processes will send it mesh parts and the sender only needs an acknowledgment that the data has been received. Previously, such a case was implemented through a consensus algorithm where the reply message is simply empty. The rewritten interfaces now support this case more explicitly: Code using these interfaces no longer has to provide functions that formulate and read the (empty) replies, though internally these functions still send around an empty reply; this case will be implemented in the future, using the interfaces now already in place.

By the time of writing, `deal.II` uses consensus-based algorithms to determine the owners of distributed index sets (`Utilities::MPI::Partitioner`, `Utilities::MPI::Noncontiguous Partitioner`, `internal::MatrixFreeFunctions::VectorDataExchange`; see [9]), to set up the global-coarsening transfer operators (see [7] and Section 2.4), to repartition distributed meshes (see Section 2.2), and basis coupling algorithms between non-matching meshes, based on the communication patters in `RemotePointeEvaluation` (see [7]).

Finally, in the spirit of optimizing communication, the `Utilities::MPI::broadcast()` function has been optimized for objects that are arrays of data types natively supported by MPI and which consequently can be sent without packing and unpacking.

2.10 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-81` was contributed by Manaswinee Bezbaruah (Texas A&M University) and Matthias Maier (Texas A&M University). It explains how to solve the complex-valued time-harmonic Maxwell equations for an optical scattering problem.
- `step-82` was contributed by Andrea Bonito (Texas A&M University) and Diane Guignard (University of Ottawa). It shows how `deal.II` can be used to implement the local discontinuous Galerkin (LDG) method for approximating the solution to the bi-Laplacian problem. The method is an alternative to the C0IP method used in `step-47`.
- `step-85` was contributed by Simon Sticko (Uppsala University). It shows how to use the new CutFEM infrastructure to solve a Poisson problem on a circular domain that is embedded in a Cartesian background mesh.

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 that are intended as starting points for further research rather than as teaching tools):

– "TRBDF2-DG projection solver for the incompressible Navier–Stokes equations" was contributed by Giuseppe Orlando (Politecnico di Milano). It shows how to solve the incompressible Navier–Stokes equations efficiently with `deal.II`'s matrix-free DG infrastructure, multigrid, and adaptive-mesh refinement. Interested readers are referred to [63].

Finally, the "MCMC for the Laplace equation" code gallery program has been updated by providing MATLAB and Python versions of the benchmark that is implemented in this code.

2.11 Incompatible changes

The 9.4 release includes around 40 incompatible changes; see [53]. The majority of these changes should not be visible to typical user codes; some remove previously deprecated classes and functions; and the majority changes 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:

– In continuation of our attempt to merge the classes `DoFHandler` and `hp::DoFHandler`, we have removed the template parameter `DoFHandlerType` from a number of classes and functions, instead using `dim`/`spacedim` template arguments. Affected classes include `SolutionTransfer` and `DataOut`.

– `FE_RaviartThomasNodal` now uses a different polynomial space to allow for a simpler use on faces in non-standard orientation. The new polynomials are anisotropic tensor products of Lagrange polynomials on the points of the Gauss–Lobatto quadrature formula. This change leads to different entries, for example, in the matrices and constraints, but no change in accuracy should be expected as the resulting basis spans the same polynomial space.

– The class `MappingQ` now applies a high-order mapping to all cells, not just the cells near the boundary, functionality that was previously provided by the `MappingQGeneric` class. The latter has been marked as deprecated.

– Changes to weighted repartitioning of `parallel::distributed::Triangulation` objects include the removal of the default weight of each cell in order to ensure consistency with the rest of the library and to improve flexibility.

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: [45, 44, 21, 59];
- For distributed parallel computing: [13];
- For *hp*-adaptivity: [15, 27];
- For partition-of-unity (PUM) and finite element enrichment methods: [25];
- For matrix-free and fast assembly techniques: [48, 49];
- For computations on lower-dimensional manifolds: [26];
- For curved geometry representations and manifolds: [37];
- For integration with CAD files and tools: [38];
- For boundary element computations: [31];
- For the `LinearOperator` and `PackagedOperation` facilities: [55, 56];
- For uses of the `WorkStream` interface: [73];
- For uses of the `ParameterAcceptor` concept, the `MeshWorker::ScratchData` base class, and the `ParsedConvergenceTable` class: [66];
- For uses of the particle functionality in `deal.II`: [29].

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

- ADOL-C [34]
- ArborX [51]
- ARPACK [52]
- Assimp [68]
- BLAS and LAPACK [4]
- Boost [17]
- CGAL [70]
- cuSOLVER [22]
- cuSPARSE [23]
- Gmsh [30]
- GSL [28, 35]
- Ginkgo [5, 6]
- HDF5 [71]
- METIS [46]
- MUMPS [3, 2]
- muparser [61]
- OpenCASCADE [62]
- p4est [20, 19]
- PETSc [11, 12]
- ROL [65]
- ScaLAPACK [16]
- SLEPc [39]
- SUNDIALS [41]
- SymEngine [69]
- TBB [64]
- Trilinos [40, 72]
- UMFPACK [24]

Please consider citing the appropriate references if you use interfaces to these libraries.

The two previous releases of `deal.II` can be cited as [7, 8].

4 Acknowledgments

`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, Tyler Anderson, Francesco Andreuzzi, Mathias Anselmann, Maximilian Bergbauer, Manaswinee Bezbaruah, Bruno Blais, Till Budde, Fabian Castelli, Praveen Chandrashekar, Huimin Chen, Cu Cui, Ivo Dravins, Niklas Fehn, Corbin Foucart, Johannes Friedlein, Sebastian Fuchs, Daniel Garcia-Sanchez, Nicola Giuliani, Alexander Grayver, Diane Guignard, Jake Harmon, Sean Ingimarson, Pengfei Jia, Sebastian Kinnewig, Uwe Köcher, Katharina Kormann, Paras Kumar, Wenyu Lei, Alberto F. Martin, Nils Much, Lucas Myers, Justin O'Connor, Judith Pauen, Vachan Potluri, Raghunandan Pratoori, Sebastian Proell, Ce Qin, Reza Rastak, Jose E.

Roman, Raphael Schoof, Magdalena Schreter, Konrad Simon, Daniel Sun, Kuljit S. Virk, Michał Wichrowski, Niklas Wik, Jiaqi Zhang.

Their contributions are much appreciated!

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, and R. Gassmöller 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.

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

R. Gassmöller was also partially supported by the NSF Awards EAR-1925677 and EAR-2054605.

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 Order 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 "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 Awards DMS-1912847 and DMS-2045636.

S. Sticko was partially supported by eSSENCE of e-Science.

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

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

References

[1] G. Alzetta, D. Arndt, W. Bangerth, V. Boddu, B. Brands, D. Davydov, R. Gassmoeller, T. Heister, L. Heltai, K. Kormann, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. The deal.II library, version 9.0. *J. Numer. Math.*, 26(4):173–184, 2018.

[2] 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 Transactions on Mathematical Software*, 45:2:1–2:26, 2019.

[3] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[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*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[5] 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. *Journal of Open Source Software*, 5(52):2260, 2020.

[6] 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 Transactions on Mathematical Software*, 48(1):1–33, Mar. 2022.

[7] D. Arndt, W. Bangerth, B. Blais, T. C. Clevenger, M. Fehling, A. V. Grayver, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, R. Rastak, I. Thomas, B. Turcksin, Z. Wang, and D. Wells. The `deal.II` library, version 9.2. *Journal of Numerical Mathematics*, 28(3):131–146, 2020.

[8] 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. *Journal of Numerical Mathematics*, 29(3):171–186, 2021.

[9] 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. *Journal of Numerical Mathematics*, 27(4):203–213, Dec. 2019.

[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*, 81:407–422, 2021.

[11] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. 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, H. Zhang, and J. Zhang. PETSc/TAO users manual. Technical Report ANL-21/39 - Revision 3.17, Argonne National Laboratory, 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, 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, H. Zhang, and J. Zhang. PETSc Web page. https://petsc.org/, 2022.

[13] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Transactions on Mathematical Software*, 38(2):14/1–28, 2012.

[14] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II — a general purpose object oriented finite element library. *ACM Trans. Math. Softw.*, 33(4), 2007.

[15] W. Bangerth and O. Kayser-Herold. Data structures and requirements for *hp* finite element software. *ACM Transactions on Mathematical Software*, 36(1):4/1–4/31, 2009.

[16] 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.

[17] Boost C++ Libraries. http://www.boost.org/.

[18] E. Burman, S. Claus, P. Hansbo, M. G. Larson, and A. Massing. CutFEM: Discretizing geometry and partial differential equations. *International Journal for Numerical Methods in Engineering*, 104(7):472–501, Nov. 2015.

[19] C. Burstedde. Parallel tree algorithms for AMR and non-standard data access. *ACM Transactions on Mathematical Software (TOMS)*, 46(4):1–31, 2020.

[20] 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(3):1103–1133, 2011.

[21] T. C. Clevenger, T. Heister, G. Kanschat, and M. Kronbichler. A flexible, parallel, adaptive geometric multigrid method for FEM. *ACM Transactions on Mathematical Software*, 47(1):7/1–27, 2021.

[22] cuSOLVER Library. https://docs.nvidia.com/cuda/cusolver/index.html.

[23] cuSPARSE Library. https://docs.nvidia.com/cuda/cusparse/index.html.

[24] T. A. Davis. Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30:196–199, 2004.

[25] 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(1):7, Dec 2017.

[26] A. DeSimone, L. Heltai, and C. Manigrasso. Tools for the solution of PDEs defined on curved manifolds with deal.II. Technical Report 42/2009/M, SISSA, 2009.

[27] M. Fehling and W. Bangerth. Algorithms for parallel generic *hp*-adaptive finite element software, 2022.

[28] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual*. Network Theory Ltd., 3rd edition, 2009.

[29] 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(9):3596–3604, Sept. 2018.

[30] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering*, 79(11):1309–1331, 2009.

[31] 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*, 121:39–58, July 2018.

[32] S. Golshan, P. Munch, R. Gassmöller, M. Kronbichler, and B. Blais. Lethe-dem: An open-source parallel discrete element solver with load balancing. *Computational Particle Mechanics*, pages 1–20, 2022.

[33] W. J. Gordon and L. C. Thiel. Transfinite mappings and their application to grid generation. *Appl. Math. Comput.*, 10:171–233, 1982.

[34] A. Griewank, D. Juedes, and J. Utke. Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167, 1996.

[35] GSL: GNU Scientific Library. http://www.gnu.org/software/gsl.

[36] J. R. Hammond, A. Schäfer, and R. Latham. To int_max... and beyond! exploring large-count support in mpi. In *2014 Workshop on Exascale MPI at Supercomputing Conference*, pages 1–8. IEEE, 2014.

[37] L. Heltai, W. Bangerth, M. Kronbichler, and A. Mola. Propagating geometry information to finite element computations. *ACM Transactions on Mathematical Software*, 47(4):1–30, Dec. 2021.

[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. Technical report, SISSA, 2015.

[39] 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(3):351–362, 2005.

[40] 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 Transactions on Mathematical Software*, 31:397–423, 2005.

[41] 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 Transactions on Mathematical Software*, 31(3):363–396, 2005.

[42] T. Hoefler, C. Siebert, and A. Lumsdaine. Scalable communication protocols for dynamic sparse data exchange. *ACM Sigplan Notices*, 45(5):159–168, 2010.

[43] D. A. Ibanez, E. S. Seol, C. W. Smith, and M. S. Shephard. Pumi: Parallel unstructured mesh infrastructure. *ACM Transactions on Mathematical Software (TOMS)*, 42(3):1–28, 2016.

[44] 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(4):2095–2114, 2011.

[45] G. Kanschat. Multi-level methods for discontinuous Galerkin FEM on locally refined meshes. *Comput. & Struct.*, 82(28):2437–2445, 2004.

[46] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[47] M. Kronbichler, N. Fehn, P. Munch, M. Bergbauer, K.-R. Wichmann, C. Geitner, M. Allalen, M. Schulz, and W. A. Wall. A next-generation discontinuous Galerkin fluid dynamics solver with application to high-resolution lung airflow simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'21*, pages 1–15, St. Louis, MO, USA, 2021. Association for Computing Machinery (ACM).

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

[49] M. Kronbichler and K. Kormann. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. *ACM Transactions on Mathematical Software*, 45(3):29:1–29:40, 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 International Journal of High Performance Computing Applications*, in press, 2022.

[51] D. Lebrun-Grandié, A. Prokopenko, B. Turcksin, and S. R. Slattery. ArborX: A performance portable geometric search library. *ACM Transactions on Mathematical Software*, 47(1):1–15, 2020.

[52] 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.

[53] List of changes for 9.4. https://www.dealii.org/developer/doxygen/deal.II/changes_between_9_3_3_and_9_4_0.html.

[54] K. Ljungkvist. Matrix-free finite-element computations on graphics processors with adaptively refined unstructured meshes. In *SpringSim (HPC)*, pages 1–1, 2017.

[55] M. Maier, M. Bardelloni, and L. Heltai. LinearOperator – a generic, high-level expression syntax for linear algebra. *Computers and Mathematics with Applications*, 72(1):1–24, 2016.

[56] M. Maier, M. Bardelloni, and L. Heltai. LinearOperator Benchmarks, Version 1.0.0, Mar. 2016.

[57] A. Massing, M. G. Larson, and A. Logg. Efficient implementation of finite element methods on nonmatching and overlapping meshes in three dimensions. *SIAM Journal on Scientific Computing*, 35(1):C23–C47, 2013.

[58] M. Mirzadeh, A. Guittet, C. Burstedde, and F. Gibou. Parallel level-set methods on adaptive tree-based grids. *Journal of Computational Physics*, 322:345–364, 2016.

[59] P. Munch, T. Heister, L. Prieto Saavedra, and M. Kronbichler. Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations, 2022.

[60] P. Munch, K. Ljungkvist, and M. Kronbichler. Efficient application of hanging-node constraints for matrix-free high-order fem computations on CPU and GPU. In A.-L. Varbanescu, A. Bhatele, P. Luszczek, and B. Marc, editors, *ISC High Performance 2022*, volume 13289 of *LNCS*, pages 133–152. Springer International Publishing, 2022.

[61] muparser: Fast Math Parser Library. https://beltoforion.de/en/muparser.

[62] OpenCASCADE: Open CASCADE Technology, 3D modeling & numerical simulation. http://www.opencascade.org/.

[63] G. Orlando, A. Della Rocca, P. F. Barbante, L. Bonaventura, and N. Parolini. An efficient and accurate implicit dg solver for the incompressible navier-stokes equations. *International Journal for Numerical Methods in Fluids*, 2021.

[64] J. Reinders. *Intel Threading Building Blocks*. O'Reilly, 2007.

[65] D. Ridzal and D. P. Kouri. Rapid optimization library. Technical report, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2014.

[66] A. Sartori, N. Giuliani, M. Bardelloni, and L. Heltai. deal2lkit: A toolkit library for high performance programming in deal.II. *SoftwareX*, 7:318–327, 2018.

[67] R. I. Saye. High-Order Quadrature Methods for Implicitly Defined Surfaces and Volumes in Hyperrectangles. *SIAM Journal on Scientific Computing*, 37(2):A993–A1019, Jan. 2015.

[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*. CGAL Editorial Board, 5.4.1 edition, 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] B. Turcksin, M. Kronbichler, and W. Bangerth. *WorkStream* – a design pattern for multicore-enabled finite element computations. *ACM Transactions on Mathematical Software*, 43(1):2/1–2/29, 2016.