

Daniel Arndt, Wolfgang Bangerth, Bruno Blais, Marc Fehling, Rene Gassmüller, Timo Heister*, Luca Heltai, Vladimir Ivannikov, Martin Kronbichler, Matthias Maier, Peter Munch, Andreas Steger, Dominik Still, Bruno Turcksin and David Wells

The deal.II library, version 9.8

<https://doi.org/10.1515/jnma-2026-0118>

Received August 15, 2026; accepted August 16, 2026; published online September 10, 2026

Abstract: This paper provides an overview of the new features of the finite element library deal.II, version 9.8, with substantially expanded support for mixed meshes containing tetrahedra, wedges, and pyramids, major improvements to the matrix-free GPU functionality, and many other improvements.

Keywords: software; finite elements; deal.II

MSC 2010: 65M60; 65N30; 65Y05

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>).

***Corresponding author: Timo Heister**, School of Mathematical and Statistical Sciences, Clemson University, Clemson, SC, 29634, USA, E-mail: heister@clemson.edu

Daniel Arndt and Bruno Turcksin, Computational Coupled Physics Group, Computational Sciences and Engineering Division, Oak Ridge National Laboratory, 1 Bethel Valley Rd., Oak Ridge, TN 37831, USA, E-mail: arndtd@ornl.gov (D. Arndt), turcksinbr@ornl.gov (B. Turcksin)

Wolfgang Bangerth, Department of Mathematics, Colorado State University, Fort Collins, CO 80523, USA; and Department of Geosciences, Colorado State University, Fort Collins, CO 80523, USA, E-mail: bangerth@colostate.edu

Bruno Blais, Department of Chemical Engineering, Chemical Engineering High-Performance Analysis, Optimization and Simulation (CHAOS) Laboratory, Polytechnique Montréal, P.O. Box 6079, Stn Centre-Ville, H3C 3A7, Montréal, QC, Canada, E-mail: bruno.blais@polymtl.ca

Marc Fehling, Department of Mathematical Analysis, Faculty of Mathematics and Physics, Charles University, Sokolovská 49/83, 186 75 Prague 8, Czech Republic, E-mail: marc.fehling@matfyz.cuni.cz

Rene Gassmüller, GEOMAR Helmholtz Centre for Ocean Research Kiel, 24148 Kiel, Germany, E-mail: rgassmoeller@geomar.de

Luca Heltai, Department of Mathematics, University of Pisa, Via Buonarroti 1/c, 56127 Pisa, Italy, E-mail: luca.heltai@unipi.it

<https://orcid.org/0000-0001-5514-4683>

Vladimir Ivannikov, Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Max-Planck-Straße 1, 21502 Geesthacht, Germany, E-mail: vladimir.ivannikov@hereon.de

Martin Kronbichler, Faculty of Mathematics, Ruhr University Bochum, Universitätsstr. 150, 44780 Bochum, Germany, E-mail: martin.kronbichler@rub.de

Matthias Maier, Department of Mathematics, Texas A&M University, 3368 TAMU, College Station, TX 77845, USA, E-mail: maier@math.tamu.edu

Peter Munch, Institute of Mathematics, Technical University of Berlin, Berlin, Germany, E-mail: muench@math.tu-berlin.de

Andreas Steger, TUM School of Engineering and Design, Technical University of Munich, Munich, Germany, E-mail: andreas.steger@tum.de

Dominik Still, Institute for Computational Mechanics, TUM School of Engineering and Design, Technical University of Munich, Munich, Germany, E-mail: dominik.still@tum.de

David Wells, Department of Mathematics, University of North Carolina, Chapel Hill, NC 27516, USA, E-mail: drwells@email.unc.edu

1 Overview

deal.II version 9.8.0 was released August 8, 2026. 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 9.8. deal.II is an object-oriented finite element library used around the world in the development of finite element solvers. It is available under the terms of the *Apache License 2.0*¹ with *LLVM Exception*² and separately under the terms of the *GNU Lesser General Public License v2.1*.³ Downloads are available at <https://www.dealii.org/> and <https://github.com/dealii/dealii>.

The major changes of this release are:

- Substantially extended support for meshes that contain triangles, tetrahedra, wedges, and pyramids (see Section 2.1).
- Improved support for and performance on GPUs (see Section 2.2).
- Support for newer versions of Trilinos that build on its newer Tpetra, rather than the older Epetra, foundation (see Section 2.3).
- Much extended interfaces to the VTK library (see Section 2.4) and the SUNDIALS library (see Section 2.5).
- A lot of work on reducing memory allocations to improve performance (see Section 2.6).
- Continued evolution alongside newer C++ standards (see Section 2.7).
- A substantial number of new tutorial and code gallery programs (see Section 2.8).

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

- Major parts of the MeshWorker framework are now deprecated. When originally written, more than 20 years ago, MeshWorker was an attempt at automatizing many parts of what a finite element code does. It succeeds at making easy things easier, but makes it very difficult to transition, for example, from a discretization of a simple PDE without coefficients to a real problem with nonlinearities, spatially variable coefficients, and similar things. It was also never well documented or tested, and only `step-39` heavily relied on it. `step-39` has now been rewritten, and all of the classes it used are now deprecated. The only piece within the MeshWorker namespace that will survive in the long run is the `MeshWorker::mesh_loop()` function and its dependencies.
- Our wrappers of PETSc’s linear algebra classes now support multi-threaded assembly into and reading from vectors `ghost`.
- We have extended the matrix-free infrastructure for the evaluation and integration of Nédélec elements.
- The internal representation of imposing Dirichlet boundary conditions in the matrix-free infrastructure was reworked and made more similar between the GPU and CPU implementations, leading to faster execution on most CPU systems.

The changelog – listing more than 140 new features and bugfixes – contains a complete record of all changes; see [1].

2 Major changes to the library

This release of deal.II contains a number of large and significant changes, which we will discuss in the following.

¹ <https://spdx.org/licenses/Apache-2.0.html>.

² <https://spdx.org/licenses/LLVM-exception.html>.

³ <https://spdx.org/licenses/LGPL-2.1-or-later.html>.

2.1 Support for meshes with non-hypercube cells

We have continued deal.II's transition from a library that historically only supported hypercube cells to one that also supports triangular (in 2d), tetrahedral, pyramidal, and wedge-shaped cells (in 3d) meshes, as well as mixed meshes. In particular, we have put substantial effort into supporting more finite element types on these meshes.

The pyramid finite elements FE_PyramidP and FE_PyramidDGP now support quadratic and cubic shape functions, extending the mixed-mesh capabilities introduced in release 9.3 [2] to higher polynomial orders. The underlying approximation space is constructed by transforming the orthogonal basis of Bergot et al. [3] into a nodal Lagrange basis. Analogous constructions are implemented for simplices and wedges. The orthogonal basis functions are expressed in terms of (homogenized) Jacobi polynomials and evaluated using recurrence relations rather than explicit polynomial expressions. Derivatives are evaluated using the same recurrence-based approach, avoiding computations at singularities while improving numerical stability. The pyramid basis functions form an exception, as their approximation space is spanned by rational rather than polynomial functions.

The introduction of second- and third-order nodal pyramid elements requires support for degrees of freedom located on edges and faces for mixed element types. To ensure a consistent distribution of the DoFs across interfaces, the *hp*-infrastructure has been extended to account for the different face topologies on transitional elements, thereby preserving continuity on conforming meshes.

Increasing the polynomial degree of the shape functions also requires quadrature rules of sufficiently high order to consistently evaluate weak-form integrals, particularly in the presence of non-linear terms or curvilinear mappings. To this end, mixed-mesh computations are supported by Stroud quadrature rules. These rules provide integration formulas of arbitrary order and serve as a fallback whenever no alternative quadrature rule of sufficient accuracy is available.

Additionally, isotropic refinement is now supported for pyramids and wedges. The underlying algorithms employ a midpoint refinement strategy similar to the one used for hexahedra and tetrahedra. It recursively subdivides a cell's constituent lower-dimensional entities – from lines to faces, and finally to the element itself. This hierarchical approach ensures mesh conformity across neighboring elements. Under this scheme, an isotropic refinement step partitions a pyramid into ten and a wedge into eight children.

As an alternative to working directly with simplex meshes, deal.II now provides the function `GridGenerator::convert_simplex_to_hypercube_mesh()`, which converts a simplex mesh into a pure hypercube mesh. In two dimensions, each triangle is subdivided into three quadrilaterals by inserting a vertex at the barycenter and connecting it to the edge midpoints. In three dimensions, each tetrahedron is decomposed into four hexahedra by applying the same subdivision to every face and adding a vertex at the tetrahedron's barycenter.

To enable the evaluation of finite element fields across interfaces in mixed meshes, the class `FEInterfaceValues` now supports heterogeneous element types. The two sides of a face or subface may use different finite elements, mappings, and quadrature rules, even allowing for adaptively refined mixed meshes in two dimensions.

2.2 Performance Portable GPU support

This release comes with numerous new features, fixes, and performance improvements for running on GPUs, making GPUs usable in practice to speed up computations.

We build on Kokkos [4] for performance portability, meaning a code written based on the `Portable::MatrixFree` classes can run on current GPUs from NVIDIA, AMD, and Intel without any vendor-specific code visible in the user code. Importantly, all functionality is also available on the CPU if deal.II is configured without GPU support.

A representative use case is the new `step-104` tutorial program (see also Section 2.8), that solves the Stokes equation using a block preconditioner and an inner geometric multigrid V-cycle. The operator evaluation and

Table 1: Stokes operator throughput and time to solve the system for different devices. PMF denotes the new `Portable::MatrixFree` implementation optimized for GPUs, and MF denotes the CPU-based `MatrixFree` framework leveraging SIMD. MF_C exploits the fact that meshes are Cartesian. Results from step-104, on a problem with 53 million degrees of freedom, in 3D.

Device	Type		FP64 [TFLOPS]	VRAM [GB]	BW [TB/s]	Operator [MDoFs/s]	Solve time [s]
AMD W7800	GPU	PMF	1.41	32	0.6	469	31
NVIDIA RTX 6000	GPU	PMF	1.97	96	1.8	1,504	10
NVIDIA H100	GPU	PMF	25.6	80	3.4	1,457	9
AMD 5975WX	CPU	PMF	2.0	–	0.2	113	135
AMD 5975WX	CPU	MF	2.0	–	0.2	995	21
AMD 5975WX	CPU	MF_C	2.0	–	0.2	1,873	13
2× AMD EPYC 9754	CPU	PMF	12.7	–	0.92	654	26
2× AMD EPYC 9754	CPU	MF	12.7	–	0.92	4,885	5
2× AMD EPYC 9754	CPU	MF_C	12.7	–	0.92	12,129	3

multigrid cycle is running without assembling any matrices (i.e., “matrix-free”). Performance results for a variety of GPUs and CPUs are presented in Table 1. Here, “PMF” denotes the new portable matrix-free framework optimized for GPUs. The performance on CPUs is significantly lower than the original matrix-free framework optimized for CPUs (denoted by “MF”). This difference is partly explained by the fact that the PMF implementation is not using SIMD vectorization, and the MF implementation has been optimized specifically for usage on CPUs over the years. The portable framework can currently also not take advantage of structured grids (like assuming the determinant of the Jacobian of the mapping being constant within a cell; “ MF_C ” denotes the use of these optimizations). The comparison with a typical workstation CPU (AMD 5975WX) shows competitive performance of the new PMF framework on GPUs. A comparison with a server-grade dual-CPU AMD EPYC machine, which formally has a lower peak FP64 rate and lower memory bandwidth than the H100, indicates further potential for the GPU implementation. Future work will therefore consider further performance optimization for the portable framework and exploiting Cartesian mesh structure.

Similar to the `MatrixFree` framework designed for CPUs [5], [6], throughput for operator evaluation increases with polynomial degree. This is in contrast to matrix-vector products using assembled matrices that get slower the higher the finite element degree, as more entries per row are nonzero. As GPUs are highly parallel devices, a significant number of unknowns per GPU are required to reach peak performance, as is clear from Figure 1.

The new tutorial showcases various new features of `deal.II`: First, `Portable::MatrixFree` now supports systems of equations by allowing for the use of more than one `Portable::FEEvaluation` object within an operator. In step-104, this is used to define the Stokes operator with a velocity and pressure evaluation object using the same quadrature rule. Second, we overhauled the interface by providing `DeviceVector` and `DeviceBlockVector` classes within the matrix-free GPU kernels instead of exposing raw C-style pointers. Third,

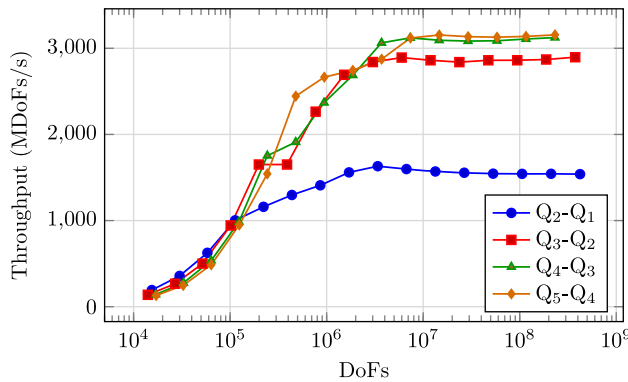


Figure 1: Throughput of a Stokes operator application for different problem sizes and finite element degrees on an NVIDIA RTX 6000 Blackwell Max-Q GPU. The step-104 program provides a detailed description and further performance data.

the complete geometric multigrid cycle including multigrid transfer for h and p coarsening runs on device without copying solution data between host and GPU, even when running with multiple GPUs. This is achieved using device-aware MPI support. Fourth, performance has been significantly improved through code optimizations, reducing the number of GPU fences, kernel fusion where possible, and by using multithreading for setup steps that run on the CPU.

Finally, usability improved with various additional error checks throughout the interface, improved API documentation, and other new features. Taken together, the work over the last twelve months makes the portable matrix-free framework a competitive solution for performance critical deal.II applications.

2.3 Trilinos and Tpetra interfaces

The Trilinos project is an important dependency for deal.II, providing one way to parallel linear algebra classes as well as numerous other pieces of functionality like solvers and preconditioners. Historically, Trilinos has provided parallel linear algebra functionality through its Epetra sub-packages. However, the Trilinos project has long intended to replace their Epetra-based solver stack in favor of Tpetra-based solvers. Support for different number types and Kokkos execution spaces have been the main drivers of this transition, the latter also enabling support for GPUs or other acceleration devices. The Epetra-based stack was finally removed in the recent Trilinos 17 release. Because deal.II's Trilinos wrappers were all based on Epetra, this has required us to complete the Tpetra wrappers we had started to write in recent years; for example, the `TpetraWrappers::Vector` class has been available since deal.II 9.1, and a basic implementation for `TpetraWrappers::SparsityPattern` and `TpetraWrappers::SparseMatrix` was included in the 9.6 deal.II release. However, much functionality was still missing, and in particular deal.II would not configure and compile with Trilinos if the Epetra stack was not present.

With deal.II 9.8 Tpetra support was expanded significantly, while simultaneously improving the compatibility between Tpetra and Epetra based wrapper classes. We now support Trilinos versions that include the Epetra-based stack, the Tpetra-based stack, or both; in particular, this includes Trilinos' latest release 17 that no longer has Epetra support. In case Epetra is available, the `TrilinosWrappers` namespace continues to use Epetra but will use Tpetra if only Tpetra is available. The `TpetraWrappers` namespace will always use Tpetra and is only available if Trilinos was configured with Tpetra.

While we generally expect the Tpetra stack to work, not all functionality from namespace `TrilinosWrappers` is available in the `TpetraWrappers` namespace at this time. However, in practice we built very large packages with the Tpetra wrappers, indicating that all *major* functionality is available. On the other hand, a substantial number of our Trilinos wrapper tests and several of deal.II's example programs fail when run against the Tpetra wrappers, suggesting that some functionality available in our Epetra wrappers are not yet implemented in the Tpetra wrappers.

2.4 VTK interfaces

The interfaces in the `VTKWrappers` namespace have been substantially extended. The new functionality provides a bidirectional bridge between VTK data sets and the mesh, finite-element, and vector abstractions used by deal.II. In particular, VTK files can now be read in stages: `read_tria()` constructs a `Triangulation`, while `read_cell_data()` and `read_vertex_data()` expose data associated with cells and vertices, respectively. The corresponding `read_all_data()` utility collects all data arrays into a single serial vector, and `read_vtk()` provides a combined interface for loading a VTK file and its associated finite-element data.

The description of the data in a VTK file can also be translated into a compatible deal.II finite-element object with `vtk_to_finite_element()`. This makes it possible to preserve the number and location of the degrees of freedom represented in the file without imposing an interpretation on multi-component arrays. Once a `DoFHandler` has been constructed, `data_to_dealii_vector()` maps the VTK data onto a deal.II vector, including distributed vectors on a parallel triangulation. The supporting

`GridTools::parallel_to_serial_vertex_indices()` utility supplies the vertex-numbering correspondence required when data are read on a serial mesh and then transferred to a distributed one.

Finally, the VTK interface now supports two-way conversion between a `Triangulation` and a `vtkUnstructuredGrid`. This allows meshes to be exchanged with VTK-based visualization and meshing tools in both directions, including meshes with non-hypercube reference-cell types supported by the library. The conversion optionally transfers manifold, boundary, and material identifiers associated with the `deal.II` mesh. Together, these additions make it possible to use VTK files as input for restarting or preprocessing computations while retaining the topology and metadata needed by a finite-element program.

2.5 Interfaces to the SUNDIALS ARKODE package

The interfaces to the SUNDIALS ARKODE package have been substantially reorganized, now matching the internal organization of ARKODE. The `SUNDIALS::ARKode` class now acts as a common driver for time integration, while stepper-specific setup and callbacks have been moved into a new `ARKodeStepper` hierarchy. This mirrors the modular structure of ARKODE itself and removes the previous assumption that all time integration is performed through ARKODE's `ARKStep` module. The existing implicit-explicit functionality is now provided by `ARKStepper`. To smooth the migration to the new class hierarchy, deprecated compatibility wrappers have been introduced that preserve the old interface.

This restructuring made it possible to add wrappers for further ARKODE modules. The new `ERKStepper` exposes the explicit `ERKStep` module for non-stiff problems. The `LSRKStepperSTS` and `LSRKStepperSSP` classes expose the low-storage Runge–Kutta `LSRKStep` module for stabilized explicit integration of mildly stiff parabolic problems, and strong-stability-preserving integration of hyperbolic or advection-dominated problems, respectively.

The `LSRKStepperSTS` stepper requires an estimator for the dominant (largest-magnitude) eigenvalue of the Jacobian of the ODE right-hand side to determine the number of polynomial stages to be used. By default, the power iteration method implemented in the `SUNDomEigEstimator_Power` module of ARKODE is employed to compute an estimate of the dominant eigenvalue. Users can replace this estimator with one of their choice.

2.6 Reducing dynamic memory allocations

We made a significant effort to reduce both the memory usage and number of memory allocations done by `deal.II`. We used heaptrack [7] to detect and measure the number of dynamic allocations. Significant improvements include:

- lowered memory consumption of purely tetrahedral `Triangulations` by 20 % by using `ReferenceCell`-dependent strides for its internal data structures,
- lowered run time to construct a `DynamicSparsityPattern` for `DoFHandlers` with a single component by 15 %,
- movement of all internal allocation functions in `Triangulation` to internal helper classes, which also eliminated some duplicated loops over allocated cells.

We achieved these reductions by preferring flat data structures such as `Table` over `std::map`, moving temporary arrays into higher scopes, using `boost::container::small_vector` instead of `std::vector`, and eliminating temporaries by combining intermediate steps. A short quantitative analysis of various example steps shows a reduction in the number of dynamic allocations between 56 % and 78 %; see Table 2.

2.7 Work towards more modern C++ standards

As in most previous releases, we have spent substantial work on evolving our code base to keep track with developments in C++ and the standards that define C++.

Table 2: Total number of allocations performed (with one MPI process and one thread) by several tutorial programs in 9.7.1 and 9.8.0. For step-32 the program was run to 1/100th of its default final time; all other time-dependent problems were run to 1/10th of their default final times.

Tutorial	Description	Allocations		
		deal.II 9.7.1	deal.II 9.8.0	
step-23	Adaptive wave equation solver	2,871 K	623 K	−78 %
step-32	Adaptive time-dependent Stokes solver	26,847 K	11,791 K	−56 %
step-35	Projection solver for Navier–Stokes	4,408 K	975 K	−78 %
step-40	Adaptive Laplace solver	5,265 K	2,028 K	−61 %

Specifically, we have continued our work on building C++20-style modules (see the discussion in the previous release paper [8] and the preprint [9]). Recent compiler versions have become stricter at enforcing requirements spelled out in the standard for how the source files for modules have to be structured, and the scripts we use to convert the deal.II source files to a module structure have correspondingly evolved. We have also substantially expanded the module wrappers for the external projects we interface with. Note that deal.II continues to only require C++17, but can build C++20-style modules if so instructed and if the compiler supports it.

The C++26 standard was released in March 2026 [10]. Among its many additions is a class `std::inplace_vector<T,N>` that has a size that can vary at runtime, but can not exceed N , allowing for efficient allocation of such objects either on the stack or inside another class. In contrast, the widely used class `std::vector<T>` has to allocate memory on the heap dynamically, whereas `std::array<T,N>` has a fixed size N . The new class is useful in contexts where the actual size of the object is not known at compile time, but where it is known that the size can not exceed a certain number. For example, during construction of a `Triangulation`, deal.II creates a `CellData` object for every cell on the coarsest level. The primary information these objects store is the vertices which define the cell. In 3d, the number of vertices must be between 4 (for tetrahedra) and 8 (for hexahedra), so by storing the vertices in a `std::inplace_vector<unsigned int, 8>` we avoid separate heap allocations for each cell and instead store all data contiguously in memory.

We have written a fallback version of this class which is compatible with C++17 and only omits the small number of features which require newer language support (such as support for ranges and more `constexpr` support). We have started to use it in many places where the maximal size of an array is known at compile time, and this has dramatically reduced the number of dynamic memory allocations in many cases – see also the discussion in Section 2.6.

2.8 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: Nearly 100 of the more than 1,500 (non-merge) commits that went into this release touched the tutorial. In addition, there are a number of new tutorial programs:

- step-98 shows how to solve two-dimensional magnetostatic curl-curl problems. It complements step-97, which considers similar problems in 3d, but step-98 puts specific emphasis on the interpretation vectors, the curl, and other operations in 2d settings. step-98 was written by Siarhei Uzunbajakau.
- step-100 introduces the Discontinuous Petrov–Galerkin (DPG) method, a finite element framework in which the test space differs from the trial space. Using broken (discontinuous) test spaces, optimal test functions are computed element-by-element so that the discrete problem inherits the inf-sup stability of the continuous one (see [11]–[13] for a full description of the method). We apply the method to the time-harmonic Helmholtz equation, a problem for which standard Galerkin discretizations lose stability at high wavenumbers, whereas DPG remains robust independently of the wavenumber-mesh regime. Because DPG is equivalent to a residual minimization in a dual norm, the resulting linear system is Hermitian positive definite despite the indefiniteness of the Helmholtz operator, and can therefore be solved with a conjugate

gradient solver. Furthermore, the residual provides a built-in a posteriori error estimator that can be used to drive adaptive mesh refinement. This comes at the cost of additional interface unknowns (traces and face elements); the program uses local condensation at the element level to obtain a global system in which only the trace degrees of freedom are present. `step-100` was written by Oreste Marquis, with help and guidance from Bruno Blais and Matthias Maier.

- `step-104` solves the Stokes problem on GPUs using a block preconditioner and a matrix-free geometric multigrid preconditioner; see Section 2.2 for more details. `step-104` was written by Quang Hoang and Timo Heister.

There are also several 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):

- “*Parallel implementation of heat equation*” is a transient heat equation solver with adaptive mesh refinement and a parallel solver. This program was written by Wasim Niyaz Munshi, Chandrasekhar Annavarapu, and Wolfgang Bangerth.
- “*L-BFGS phasefield solver*” is a limited-memory BFGS monolithic solver for phase-field crack simulations. This program was written by Tao Jin.
- “*Agglomeration-based solver for the Poisson problem*” is a discontinuous Galerkin solver for the Poisson problem on general polytopal meshes generated through mesh agglomeration. This program was written by Marco Feder, Pasquale Claudio Africa, Xinping Gui, and Andrea Cangiani.
- “*An ALE approach for large-deformation thermoplasticity*” is an implementation of a large-deformation arbitrary Lagrangian-Eulerian finite-strain thermoplasticity solver. This program was written by Maien Hamed.
- “*Parallel Vibroacoustic Solver*” is a parallel frequency-domain vibroacoustic solver calculating the sound transmission loss of a concrete wall. This program was written by Andreas Hegendörfer.
- “*Multilevel Monte Carlo for random Darcy flow*” is an implementation of a multilevel Monte Carlo method for random Darcy flow. This program was written by Jonas Plank.
- “*Parallel flow routing*” solves the problem of determining how much water is where in a landscape given by a digital elevation model, using massively parallel algorithms. This program was written by Wolfgang Bangerth.

2.9 Incompatible changes

The 9.8 release includes around 20 incompatible changes; see [1]. Many of these incompatibilities remove previously deprecated functions or classes, or require now widely available but newer versions of external dependencies than previous deal.II versions. Others change internal interfaces that are not usually used in external applications. The following are worth mentioning since they are more broadly visible:

- Most of the parallel vector classes (in particular those wrapping the functionality in the PETSc, Trilinos Epetra, and Trilinos Tpetra packages) observe semantics whereby vectors that have ghost elements are treated as unchangeable. This is because whenever you change a locally-owned element of such a vector, this change is not propagated to those processes that store this element among their ghost elements; these processes’ view of the vector is now no longer in sync with the view on that process that owns the element, and that is a common source of very hard to find bugs.

As a consequence, many vector class member functions that modify the vector check that the vector has no ghost elements. But some failed to do so, such as `operator+=()`, `operator-=()`, `operator*=()`, and `operator/=()` functions of the `TrilinosWrappers::MPI::Vector` class, and the corresponding functions in the Epetra and Tpetra wrappers, along with an assortment of other non-const functions in these classes that include assigning a scalar to all elements of a vector, and writing into (or adding, or multiplying/dividing) a single vector entry.

This oversight has now been fixed. We know that some of these functions are used in parallel contexts in application codes that will now no longer work and will need to be fixed. For this, you will want to perform the modifying operations on fully-distributed vectors (i.e., vectors of the same type that store no ghost elements) and then copy the result into the ghosted vector.

- Several of the classes used in the construction of triangulations, specifically `CellData` and `TriangulationDescription::CellData` use the new `std_cxx26::inplace_vector` class (see Section 2.7) for member variables to reduce the number of memory allocations. In practice, because both this and the previously used `std::vector` class use array syntax, few codes will notice the difference in type.
- The `FESubfaceValues` class – used to integrate over the children of a face of a cell – used to do just that: Consider a part of a face that corresponds to a subdivision of that face due to refinement. It did not actually check consistently whether the face *was* indeed refined (i.e., whether the neighboring *cell* was refined). It will now generate an error if one tries to integrate over a face that is not actually refined. In practice, there should not be any reasons why one would integrate over sub-faces that are not in fact refined.
- In three space dimensions, the curl of a vector field is unambiguously a three-dimensional vector. In two space dimensions, it is typically defined as a scalar, interpreted as the magnitude of a vector aligned with the z-axis that results from taking the curl of a vector field that lives in the x–y plane. For historical reasons, deal.II used to represent this scalar as a `Tensor<1, 1>` (i.e., a tensor of rank one – a tensor with one index – for which the only possible value of the index is zero). This is, of course, entirely equivalent to it being a scalar, but it is semantically wrong: The data type should have been a scalar, not a vector of length one. This is now fixed: The data type defined by `FEValuesViews::Vector::curl_type` used for curls is now simply a scalar value for `dim==2`. For `dim==3`, it is unchanged from what it was before.

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 [15]. If you rely on specific features of the library, please consider citing any of the following:

-
- For geometric multigrid: [16]–[19];
 - For distributed parallel computing: [20];
 - For *hp*-adaptivity: [21], [22];
 - For partition-of-unity (PUM) and finite element enrichment methods: [23];
 - For matrix-free and fast assembly techniques: [5], [6];
 - For computations on lower-dimensional manifolds: [24];
 - For curved geometry representations and manifolds: [25];
 - For integration with CAD files and tools: [26];
 - For boundary element computations: [27];
 - For the `LinearOperator` and `PackagedOperation` facilities: [28], [29];
 - For uses of the `WorkStream` interface: [30];
 - For uses of the `ParameterAcceptor` concept, the `MeshWorker::ScratchData` base class, and the `ParsedConvergenceTable` class: [31];
 - For uses of the particle functionality in deal.II: [32].
 - For the design of the video lectures and how they can be used in teaching: [33].
-

deal.II can interface with many other libraries:

–ADOL-C [34]	–GSL [42]	–OpenCASCADE [51]	–SUNDIALS [59]
–ArborX [35]	–Ginkgo [43], [44]	–p4est [52]	–SymEngine [60]
–ARPACK [36]	–HDF5 [45]	–PETSc [53], [54]	–Taskflow [61]
–Assimp [37]	–Kokkos [4]	–PSBLAS [55]	–TBB [62]
–BLAS, LAPACK [38]	–Magic Enum [46]	–ROL [56]	–Trilinos [63], [64]
–Boost [39]	–METIS [47]	–ScaLAPACK [57]	–UMFPACK [65]
–CGAL [40]	–MUMPS [48], [49]	–SLEPc [58]	–VTK [66]
–Gmsh [41]	–muparser [50]		–zlib [67]

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], [68].

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:

Daniel Abele, Mathias Anselmann, Kyle Arnold, Zdeněk Bonaventura, Julian Brotz, Michele Bucelli, Praveen Chandrashekar, Jerett Cherry, Vishnu Datha, Raksha Devi, Vivienne Ehlert, Marco Feder, Federico Fernandez, Mohamad Ghadban, Quang Hoang, Tyree Huang, Yimin Jin, Jalil Khan, Felix Kimmerle, Sebastian Kinnewig, Katharina Kormann, Damien Lebrun-Grandie, Lukáš Lejdar, Anna Long, Oreste Marquis, Harsha Matta, Ryan Moulday, Natalia Nebulishvili, Manikandan Nishanth, Jonas Plank, Davide Polverino, Ivan Prusak, Ce Qin, Alexander Reinhold, Sam Scheuerman, Torsten Schmid, Magdalena Schreter-Fleischhacker, Florian Schulze, Richard Schussnig, Gordan Segon, Qingyuan Shi, Wyatt Smith, Simon Sticko, Jan Philipp Thiele, Pasquale Tricarico, Siarhei Uzunbajakau, Quentin Viville, Yiliang Wang, Zhaokun Wang, Ivy Weber, Michał Wichrowski, Jiaqi Zhang, Ran Zhang.

That is, 67 people in total contributed to this release. Their contributions are much appreciated – this project lives by its community of users and contributors!

Research ethics: Not applicable.

Informed consent: Not applicable.

Author contributions: All authors have accepted responsibility for the entire content of this manuscript and approved its submission.

Use of Large Language Models, AI and Machine Learning Tools: None declared.

Conflict of interest: The authors state no conflict of interest.

Research 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 and supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Next-Generation Scientific Software Technologies program, under contract number DE-AC05-00OR22725. W. Bangerth was partially supported by the National Science Foundation under awards EAR-1925595 and OAC-2410847. W. Bangerth, T. Heister, and R. Gassmüller were partially supported by the Computational Infrastructure for Geodynamics initiative (CIG), through the National Science Foundation (NSF) under Award No. EAR-2149126 via The University of California – Davis. B. Blais was supported by the National Science and Engineering Research Council of Canada (NSERC) through the RGPIN-2020-04510 Discovery Grant and the MMIAOW Canada Research Chair Level 2 in Computer-Assisted Design and Scale-up of Alternative Energy Vectors for Sustainable Chemical Processes (CRC-2022-00340). M. Fehling was supported by the ERC-CZ grant LL2105 CONTACT and the P JAC grant CZ.02.01.01/00/22_008/0004591 FerrMion, both funded by the Czech Ministry of Education, Youth and Sports, as well as by the Czech Science Foundation project No. 26-21877S and the Charles University Research Centre Program No. UNCE/24/SCI/005. R. Gassmüller was partially supported by the Helmholtz Association's Impulse and Networking Fund as part of the IDS_29 ASPECT-NEXT project in Science Serve Call 2025. T. Heister

was also partially supported by NSF Awards OAC-2607668, OAC-2410848, EAR-1925575. L. Heltai acknowledges the MIUR Excellence Department Project awarded to the Department of Mathematics, University of Pisa, CUP I57G22000700001, and partial support by King Abdullah University of Science and Technology Research Funding (KRF) under Award No. ORFS-2025-CRG13-6911.3. L. Heltai is also member of Gruppo Nazionale per il Calcolo Scientifico (GNCS) of Istituto Nazionale di Alta Matematica (INdAM). M. Kronbichler was partially supported by the German Federal Ministry of Research, Technology and Space, project “PDExa: Optimized software methods for solving partial differential equations on exascale supercomputers”, grant agreement No. 16ME0637K. M. Kronbichler and L. Heltai were partially supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (call HORIZON-EUROHPC-JU-2023-COE-03, grant agreement No. 101172493 “dealii-X: an Exascale Framework for Digital Twins of the Human Body”). M. Maier was partially supported by NSF Award DMS-2045636 and by the Air Force Office of Scientific Research under grant/contract number FA9550-23-1-0007. D. Still was partially supported by the German Federal Ministry of Research, Technology and Space, project “PDExa: Optimized software methods for solving partial differential equations on exascale supercomputers”, grant agreement No. 16ME0638 and the European Union – NextGenerationEU and by BREATHE, an ERC-2020-ADG Project, Grant Agreement ID 101021526. D. Wells was supported by NSF Award OAC-1931516. This research used in part resources on the Palmetto Cluster at Clemson University under National Science Foundation awards MRI 1228312, II NEW 1405767, MRI 1725573, and MRI 2018069. The views expressed in this article do not necessarily represent the views of NSF or the United States government. This research was supported by grants from NVIDIA and utilized NVIDIA products.

Data availability: Not applicable.

Software availability: See <https://github.com/dealii/dealii/>.

References

- [1] The deal.II authors. “List of changes for deal.II release 9.8.” Available at: https://dealii.org/developer/doxygen/deal.II/changes_between_9_7_1_and_9_8_0.html [Accessed: Aug. 22, 2026].
- [2] D. Arndt et al., “The deal.II library, version 9.3,” *J. Numer. Math.*, vol. 29, no. 3, pp. 171–186, 2021.
- [3] M. Bergot, G. Cohen, and M. Durufle, “Higher-order finite elements for hybrid meshes using new nodal pyramidal elements,” *J. Sci. Comput.*, vol. 42, pp. 345–381, 2010.
- [4] C. R. Trott et al., “Kokkos 3: Programming model extensions for the exascale era,” *IEEE Trans. Parallel Distr. Syst.*, vol. 33, no. 4, pp. 805–817, 2022.
- [5] M. Kronbichler and K. Kormann, “A generic interface for parallel cell-based finite element operator application,” *Comput. Fluids*, vol. 63, pp. 135–147, 2012.
- [6] M. Kronbichler and K. Kormann, “Fast matrix-free evaluation of discontinuous Galerkin finite element operators,” *ACM Trans. Math. Software*, vol. 45, no. 3, 2019, Art. no. 29.
- [7] M. Wolff et al., “Heaptrack — a heap memory profiler for Linux.” Available at: <https://github.com/KDE/heaptrack> [Accessed: Jul. 06, 2026].
- [8] D. Arndt, et al., “The deal.II library, version 9.7,” *J. Numer. Math.*, vol. 33, no. 4, pp. 403–415, 2025.
- [9] W. Bangerth, “Experience converting a large mathematical software package written in C++ to C++20 modules,” *arXiv preprint*, 2025, <https://arxiv.org/abs/2506.21654>.
- [10] International Standards Organization, “ISO/IEC 14882:2026: The C++26 programming language standard, working draft,” 2026. Available at: <https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2026/n5046.pdf>.
- [11] L. Demkowicz and J. Gopalakrishnan, “A class of discontinuous Petrov–Galerkin methods. Part I: The transport equation,” *Comput. Methods Appl. Mech. Eng.*, vol. 199, nos. 23–24, pp. 1558–1572, 2010.
- [12] L. Demkowicz and J. Gopalakrishnan, “A class of discontinuous Petrov–Galerkin methods. II. Optimal test functions,” *Numer. Methods Part. Differ. Equ.*, vol. 27, no. 1, pp. 70–105, 2011.
- [13] L. Demkowicz, J. Gopalakrishnan, and A. H. Niemi, “A class of discontinuous Petrov–Galerkin methods. Part III: Adaptivity,” *Appl. Numer. Math.*, vol. 62, no. 4, pp. 396–427, 2012.
- [14] W. Bangerth, R. Hartmann, and G. Kanschat, “deal.II — A general purpose object oriented finite element library,” *ACM Trans. Math. Software*, vol. 33, no. 4, 2007, Art. no. 24.
- [15] D. Arndt et al., “The deal.II finite element library: Design, features, and insights,” *Comput. Math. Appl.*, vol. 81, pp. 407–422, 2021.
- [16] G. Kanschat, “Multi-level methods for discontinuous Galerkin FEM on locally refined meshes,” *Comput. Struct.*, vol. 82, no. 28, pp. 2437–2445, 2004.

- [17] 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.*, vol. 33, no. 4, pp. 2095–2114, 2011.
- [18] T. C. Clevenger, T. Heister, G. Kanschat, and M. Kronbichler, “A flexible, parallel, adaptive geometric multigrid method for FEM,” *ACM Trans. Math. Software*, vol. 47, no. 1, 2021, Art. no. 7.
- [19] P. Munch, T. Heister, L. Prieto Saavedra, and M. Kronbichler, “Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations,” *ACM Trans. Parallel Comput.*, vol. 10, no. 1, 2023, Art. no. 3.
- [20] 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*, vol. 38, no. 2, pp. 14/1–28, 2012.
- [21] W. Bangerth and O. Kayser-Herold, “Data structures and requirements for *hp* finite element software,” *ACM Trans. Math. Software*, vol. 36, no. 1, 2009, Art. no. 4.
- [22] M. Fehling and W. Bangerth, “Algorithms for parallel generic *hp*-adaptive finite element software,” *ACM Trans. Math. Software*, vol. 49, no. 3, 2023, Art. no. 25.
- [23] D. Davydov, T. Gerasimov, J.-P. Pelteret, and P. Steinmann, “Convergence study of the *h*-adaptive PUM and the *hp*-adaptive FEM applied to eigenvalue problems in quantum mechanics,” *Adv. Model. Simul. Eng. Sci.*, vol. 4, no. 1, 2017, Art. no. 7.
- [24] A. DeSimone, L. Heltai, and C. Manigrasso, “Tools for the solution of PDEs defined on curved manifolds with deal.II,” Tech. Rep. 42/2009/M, SISSA, 2009.
- [25] L. Heltai, W. Bangerth, M. Kronbichler, and A. Mola, “Propagating geometry information to finite element computations,” *ACM Trans. Math. Software*, vol. 47, no. 4, 2021, Art. no. 32.
- [26] 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,” Tech. Rep., SISSA, 2015.
- [27] N. Giuliani, A. Mola, and L. Heltai, “ π -BEM: A flexible parallel implementation for adaptive, geometry aware, and high order boundary element methods,” *Adv. Eng. Software*, vol. 121, pp. 39–58, 2018.
- [28] M. Maier, M. Bardelloni, and L. Heltai, “LinearOperator — A generic, high-level expression syntax for linear algebra,” *Comput. Math. Appl.*, vol. 72, no. 1, pp. 1–24, 2016.
- [29] M. Maier, M. Bardelloni, and L. Heltai, *LinearOperator Benchmarks*, Version 1.0.0, Geneva, Switzerland, Zenodo, 2016.
- [30] B. Turcksin, M. Kronbichler, and W. Bangerth, “WorkStream — a design pattern for multicore-enabled finite element computations,” *ACM Trans. Math. Software*, vol. 43, no. 1, 2016, Art. no. 2.
- [31] A. Sartori, N. Giuliani, M. Bardelloni, and L. Heltai, “deal2lkit: A toolkit library for high performance programming in deal.II,” *SoftwareX*, vol. 7, pp. 318–327, 2018.
- [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,” *Geochem. Geophys. Geosyst.*, vol. 19, no. 9, pp. 3596–3604, 2018.
- [33] J. Zarestky, M. Bigler, M. Brazile, T. Lopes, and W. Bangerth, “Reflective writing supports metacognition and self-regulation in graduate computational science and engineering,” *Comput. Educ. Open*, vol. 3, pp. 100085/1–12, 2022.
- [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 Trans. Math. Software*, vol. 22, no. 2, pp. 131–167, 1996.
- [35] A. Prokopenko, D. Arndt, D. Lebrun-Grandié, and B. Turcksin, “The ArborX library: Version 2.0,” *ACM Trans. Math. Software*, vol. 51, no. 4, 2025, <https://doi.org/10.1145/3772288>.
- [36] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users’ Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, Philadelphia, SIAM, 1998.
- [37] T. Schulze et al., “Open asset import library (assimp),” 2021. Available at: <https://github.com/assimp/assimp>.
- [38] E. Anderson et al., *LAPACK Users’ Guide*, 3rd ed. Philadelphia, PA, Society for Industrial and Applied Mathematics, 1999.
- [39] Boost C++ Libraries. “Boost C++ libraries.” Available at: <https://www.boost.org/> [Accessed: Aug. 22, 2026].
- [40] The CGAL Project, *CGAL User and Reference Manual*, CGAL Editorial Board, 2024. 6.0.1 edition.
- [41] 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 Eng.*, vol. 79, no. 11, pp. 1309–1331, 2009.
- [42] M. Galassi et al., *GNU Scientific Library Reference Manual*, 3rd ed. Network Theory Ltd., 2009. Available at: <https://www.gnu.org/software/gsl>.
- [43] H. Anzt et al., “Ginkgo: A high performance numerical linear algebra library,” *J. Open Source Softw.*, vol. 5, no. 52, 2020, Art. no. 2260.
- [44] H. Anzt et al., “Ginkgo: A modern linear operator algebra framework for high performance computing,” *ACM Trans. Math. Software*, vol. 48, no. 1, 2022, Art. no. 2.
- [45] The HDF Group, “Hierarchical data format,” version 5, 2025. Available at: <https://www.hdfgroup.org/solutions/hdf5/>.
- [46] Neargye. “Magic Enum C++.” Available at: https://github.com/Neargye/magic_enum [Accessed: Aug. 22, 2026].
- [47] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.
- [48] P. R. Amestoy, I. S. Duff, J.-Y. L’Excellent, and J. Koster, “A fully asynchronous multifrontal solver using distributed dynamic scheduling,” *SIAM J. Matrix Anal. Appl.*, vol. 23, no. 1, pp. 15–41, 2001.
- [49] 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*, vol. 45, no. 1, 2019, Art. no. 2.

- [50] Berg, Ingo. “muparser: Fast math parser library.” Available at: <https://beltoforion.de/en/muparser0> [Accessed: Aug. 22, 2026].
- [51] OpenCASCADE: Open CASCADE Technology, “3D modeling & numerical simulation.” Available at: <https://www.opencascade.org/> [Accessed: Aug. 22, 2026].
- [52] C. Burstedde, L. C. Wilcox, and O. Ghattas, “p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees,” *SIAM J. Sci. Comput.*, vol. 33, no. 3, pp. 1103–1133, 2011.
- [53] S. Balay et al., “PETSc/TAO users manual,” Tech. Rep. ANL-21/39 — Revision 3.23, Argonne National Laboratory, 2025.
- [54] S. Balay et al., “PETSc web page,” 2025. Available at: <https://petsc.org/>.
- [55] S. Filippone and M. Colajanni, “PSBLAS: A library for parallel linear algebra computation on sparse matrices,” *ACM Trans. Math. Software*, vol. 26, no. 4, pp. 527–550, 2000.
- [56] A. Javeed, D. P. Kouri, D. Ridzal, and G. Von Winckel, “Get ROL-ing: An introduction to Sandia’s rapid optimization library,” in *7th International Conference on Continuous Optimization*, 2022.
- [57] L. S. Blackford et al., *ScaLAPACK Users’ Guide*, Philadelphia, PA, Society for Industrial and Applied Mathematics, 1997.
- [58] J. E. Roman, F. Alvarruiz, C. Campos, L. Dalcin, P. Jolivet, and A. Lamas Daviña, “Improvements to SLEPc in releases 3.14–3.18,” *ACM Trans. Math. Software*, vol. 49, no. 3, 2023, Art. no. 29.
- [59] D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos, “Enabling new flexibility in the sundials suite of nonlinear and differential/algebraic equation solvers,” *ACM Trans. Math. Software*, vol. 48, no. 3, 2022, Art. no. 31.
- [60] SymEngine Contributors. “SymEngine: Fast symbolic manipulation library, written in C++.” Available at: <https://symengine.org/> [Accessed: Aug. 22, 2026].
- [61] T.-W. Huang, D.-L. Lin, C.-X. Lin, and Y. Lin, “Taskflow: A lightweight parallel and heterogeneous task graph computing system,” *IEEE Trans. Parallel Distr. Syst.*, vol. 33, no. 6, pp. 1303–1320, 2021.
- [62] J. Reinders, *Intel Threading Building Blocks*, Sebastopol, CA, O’Reilly Media, 2007.
- [63] M. Mayr et al., “Enabling scientific computing across diverse hardware architectures at scale,” *ACM Trans. Math. Software*, vol. 52, no. 2, 2026, Art. no. 12.
- [64] The Trilinos Project Team, *The Trilinos Project, Linux Foundation/High Performance Software Foundation*. Available at: <https://trilinos.github.io/> [Accessed: Aug. 22, 2026].
- [65] T. A. Davis, “Algorithm 832: UMFPACK V4.3 — an unsymmetric-pattern multifrontal method,” *ACM Trans. Math. Software*, vol. 30, no. 2, pp. 196–199, 2004.
- [66] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit*, 4th ed. Kitware, 2006. Available at: <https://vtk.org/>.
- [67] G. Jean-loup and A. Mark. “zlib.” Available at: <https://zlib.net> [Accessed: Aug. 22, 2026].
- [68] P. C. Africa et al., “The deal.II library, version 9.6,” *J. Numer. Math.*, vol. 32, no. 4, pp. 369–380, 2024.