The deal.II Library, Version 9.1

Daniel Arndt*[1], Wolfgang Bangerth[2], Thomas C. Clevenger[3], Denis Davydov[4], Marc Fehling[5], Daniel Garcia-Sanchez[6], Graham Harper[2], Timo Heister[3,7], Luca Heltai[8], Martin Kronbichler[9], Ross Maguire Kynch[10], Matthias Maier[11], Jean-Paul Pelteret[4], Bruno Turcksin*[1], and David Wells[12]

[1]Computational Engineering and Energy Sciences Group, Computional 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@colostate.edu, harper@math.colostate.edu
[3]School of Mathematical and Statistical Sciences, Clemson University, Clemson, SC, 29634, USA {tcleven, heister}@clemson.edu
[4]Chair of Applied Mechanics, Friedrich-Alexander-Universität Erlangen-Nürnberg, Egerlandstr. 5, 91058 Erlangen, Germany. {denis.davydov,jean-paul.pelteret}@fau.de
[5]Institute for Advanced Simulation, Forschungszentrum Jülich GmbH, 52425 Jülich, Germany. m.fehling@fz-juelich.de
[6]Sorbonne Universités, UPMC Univ. Paris 06, CNRS-UMR 7588, Institut des NanoSciences de Paris, F-75005, Paris, France daniel.garcia-sanchez@insp.upmc.fr
[7]Scientific Computing and Imaging Institute, 72 S Central Campus Drive, Room 3750 Salt Lake City, UT 84112. heister@sci.utah.edu
[8]SISSA, International School for Advanced Studies, Via Bonomea 265, 34136, Trieste, Italy. luca.heltai@sissa.it
[9]Institute for Computational Mechanics, Technical University of Munich, Boltzmannstr. 15, 85748 Garching, Germany. kronbichler@lnm.mw.tum.de
[10]Zienkiewicz Centre for Computational Engineering, College of Engineering, Swansea University, Bay Campus, Fabian Way, Swansea SA1 8EN, Wales, UK. rkynch@gmail.com
[11]Department of Mathematics, Texas A&M University, 3368 TAMU, College Station, TX 77845, USA. maier@math.tamu.edu
[12]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 9.1.

1 Overview

deal.II version 9.1.0 was released May 21, 2019. 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 9.1. deal.II is an object-oriented finite element library used around the world in the

* 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).

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:

– Improved support for automatic differentiation (see Section 2.1),

– Dedicated support for symbolic algebra (see Section 2.2),

– Full support for *hp* adaptivity in parallel computations (see Section 2.3),

– An interface to the HDF5 file format and libraries (see Section 2.4),

– Significantly extended GPU support (see Section 2.5),

– Parallel geometric multigrid (GMG) improvements (see [17] and Section 2.6),

– Four new tutorial programs (step-61, step-62, step-63, step-64), as well as one new code gallery program (see Section 2.7).

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

– The release contains a number of performance improvements and bug fixes for the matrix-free framework. One notable improvement is the support for renumbering of degrees of freedom within the cells for discontinuous elements, avoiding some reshuffling operations across the SIMD lanes with vectorization over several cells and faces, which is especially useful on processors with AVX-512 vectorization (8 doubles), speeding up operations by up to 10%. Secondly, the strategy for the most efficient tensor product evaluators according to the performance analysis of [37] in the context of more quadrature points than shape functions has been revised for better performance.

– A new class `ParsedConvergenceTable` has been introduced that greatly simplifies the construction of convergence tables, reading the options for the generation of the table from a parameter file, and providing methods that, combined with a parameter file, allow one to generate convergence tables using one-liners in user codes.

– The `FE_BernardiRaugel` class implements the non-standard Bernardi-Raugel (BR) element that can be used to construct a stable velocity-pressure pair for the Stokes equation [12]. The BR element is an enriched version of the Q_1^d element with added bubble functions on each edge (in 2d) or face (in 3d). It addresses the fact that the $Q_1^d \times Q_0$ combination is not inf-sup stable (requiring a larger velocity space), and that the $Q_2^d \times Q_0$ combination is stable but converges with only first-order at the cost of the large number of velocity unknowns. The BR space is thus intermediate between the Q_1^d and Q_2^d spaces.

  The element is currently only implemented for parallelogram meshes due to difficulties associated with the mapping of shape functions: The shape functions of the Q_1^d part of the element need to be mapped as scalars, as is common for the vector components of the Q_1^d element; on the other hand, the vector-valued edge bubble functions need to be mapped using the Piola transform as is common for the Raviart-Thomas element. `deal.II` does not currently have the ability to use different mappings for individual shape functions, though this functionality is planned for the next release.

– The `FE_NedelecSZ` class is a new implementation of the Nédélec element on quadrilaterals and hexahedra. It is based on the work of Zaglmayr [58] and overcomes the sign conflict issues present in traditional Nédélec elements that arise from the edge and face parameterizations used in the basis functions. Therefore, this element should provide consistent results for general quadrilateral and hexahedral elements for which the relative orientations of edges and faces (as seen from all adjacent cells) are often difficult to establish. The `FE_NedelecSZ` element addresses the sign conflict problem by assigning a globally defined orientation to local edges and faces. A detailed overview of the implementation of the `FE_NedelecSZ` element in `deal.II` can be found in [40].

– All of the elementary geometrical objects of the library (namely `Point<dim>`, `Segment<dim>`, and `BoundingBox<dim>`) have been augmented with the traits needed to comply with `boost::geometry` concepts. A new interface to `boost::geometry::index::rtree` has been added that simplifies the construction of spatial indices based on points, bounding boxes, or segments.

In addition to these changes, the changelog lists more than 200 other features and bugfixes.

2 Major changes to the library

This release of `deal.II` contains a number of large and significant changes that will be discussed in this section.

It of course also contains a vast number of smaller changes and added functionality; the details of these can be found in the file that lists all changes for this release, see [42].

2.1 Improved support for automatic differentiation

In the previous release, numerous classes that are used to assemble linear systems and right hand sides, as well those used to define constitutive laws, were given full support for "white-listed" automatically differentiable (AD) number types from the ADOL-C and Sacado libraries. This included the classes that represent the local contributions of one cell to the global linear system (i.e., `FullMatrix` and `Vector`) as well as the classes commonly used for the computations at individual quadrature points (primarily the `Tensor` and `SymmetricTensor` classes of various ranks). In the current release we have provided a unified interface to these AD libraries, focusing on two specific use contexts:

1. The construction and linearization of finite element residuals; and
2. The construction and linearization of constitutive model kinetic variables.

In the first context, the finite element degrees of freedom are considered the independent variables. From these primitives, the `EnergyFunctional` helper class in the namespace `Differentiation::AD` may be used to compute both the residual and its linearization by directly defining the contribution to the (twice differentiated) scalar total energy functional from each cell. Similarly, the `ResidualLinearization` class requires the (once-differentiated) finite element residual to be defined on a per-cell basis, and this contribution is automatically linearized.

The second context aims directly at constitutive model formulations, and serves to compute the directional derivatives of components of (multi-field) constitutive laws with respect to the scalar, vector, tensor, and symmetric tensor fields in terms of which they are parameterized. The `ScalarFunction` class may be used to define a scalar function (such as strain energy function) that may be twice differentiated, while the `VectorFunction` may be used to define a vector function (such as a set of kinematic fields) that may be differentiated once. Since the total derivatives of all components are computed at once, these two helper classes provide an interface to retrieve each sub-component of the gradient and Hessian (for a `ScalarFunction`) or values and Jacobian (for a `VectorFunction`).

Although these aforementioned helper classes have been documented with a specific use in mind, they remain generic and may (with a reinterpretation of the meaning of the independent and dependent variables) be used for other purposes as well. Furthermore, through the implementation of `TapedDrivers` and `TapelessDrivers` classes that interface with the active AD library, the generic helper classes hide library-dependent implementation details and facilitate switching between the supported libraries and AD number types based on the user's requirements.

2.2 Dedicated support for symbolic algebra, including algebra differentiation

To complement the automatic differentiation features in `deal.II`, this release sees the first step towards integrating and supporting a highly performant computer algebra system (CAS) via the SymEngine library [54]. This allows the development of exact algebraic expressions using variables that are manipulated symbolically and may represent any value (or supported data structure). In the context of finite element simulations, typical applications include (but, due to the generality of the CAS, are not limited to) the development of constitutive models and the implementation of finite element assembly operations through the construction and linearization of finite element residuals.

The `Expression` class in the namespace `Differentiation::SD` interfaces to SymEngine and forms the basis of symbolic computations, offering a full set of overloaded operators and a C++ style interface. This class offers the following basic functionality:

– symbolic variable definition,

– symbolic function definition,

– expression creation using standard C++ syntax,

– expression parsing,

– comparison operations,

– logical operations,

– conditional expression construction,

– differentiation,

– substitution (partial and complete), and

– serialization.

`deal.II` now also provides an extensive set of math operations, with a syntax mimicking that used in the C++ standard library. Using the `Expression` class as a basis, we have developed a set of functions that can be used to create `deal.II` `Tensor`s and `SymmetricTensor`s of symbolic variables and symbolic functions. This gives full symbolic tensor algebra support using the pre-existing `Tensor` and `SymmetricTensor` classes and associated functions. We have also implemented a set of utility functions with the following features:

– `differentiate` scalar expressions with respect to other scalar expressions, as well as tensors and symmetric tensors of expressions;

– `differentiate` tensors and symmetric tensors of expressions with respect to scalar expressions, as well as other tensors and symmetric tensors of expressions;

– create symbolic substitution maps;

– resolve explicit dependencies between expressions; and

– perform scalar and tensor valued substitution (including conversion from symbolic to real-valued scalars and tensors).

In the next release we expect to implement classes to assist in performing assembly operations in the same spirit as it is already possible using automatic differentiation using the `Differentiation::AD` namespace, although in a fully symbolic manner. We will also address performance issues of the `Expression` class by leveraging the optimization capabilities of SymEngine, including common subexpression elimination (CSE), as well as by generating high performance code-paths to evaluate these expressions through the use of a custom-generated `std::function` (so-called "lambda" optimization) or by compiling expressions using the LLVM JIT compiler.

2.3 Full support for *hp* adaptivity in parallel computations

`deal.II` has had support for *hp* adaptive methods since around 2005 (documented in [11]) and for parallel computations on distributed meshes since around 2010 (see [9]), but not for both at the same time. The challenges to combine these are related to a number of areas:

1. Data structures: The data structures necessary to store the indices of degrees of freedom are substantially more complicated for *hp* algorithms than for the *h* adaptive schemes that were already implemented. This is because the number of degrees of freedom per cell is now no longer constant. Furthermore, faces and edges may need to store more than one set of indices if the adjacent cells use different polynomial degrees; in the case of edges, the number of sets of indices may also be of variable size.

   All of this poses challenges in the parallel context because some of the information may not be known, or not be known right away, for cells that are not locally owned (i.e., for ghost and "artificial" cells), and for which the data structures stored on different processors have to be reconciled.

2. Algorithms: Already for *h* adaptive meshes, enumerating all degrees of freedom uniquely on the global mesh is difficult, as evidenced by the complications of the algorithms shown in Section 3.1 of [9], which requires more than one page of text and is implemented in many hundreds of lines of code.

   These difficulties are even more pronounced when using *hp* adaptivity. The main obstacle is the desire to unify the indices of matching degrees of freedom on adjacent cells whenever elements with continuous polynomials are used. For example, the edge degree of freedom of a Q_2 element has to be merged with the middle one of the three edge degrees of freedom of a Q_4 element on a neighboring cell. Section 4.2 of [11] discusses a sequential algorithm that eliminates one of these degrees of freedom in favor of another, but it introduces a "master" and a "slave" side of the interface. This is of no major consequence in sequential computations, but is inconvenient in parallel computations if the "master" side is a ghost cell whose degree of freedom indices are not (yet) available while enumerating local degrees of freedom, or if the master is an artificial cell whose information will never be available on a processor.

   An earlier implementation of the algorithm enumerating degrees of freedom, already available in `deal.II` 9.0, simply did not unify indices on processor boundaries. However, this makes the total number of degrees of freedom dependent on both the partition of the mesh and the number of processors available. We have therefore re-implemented the algorithm so that the unification does happen also on processor boundaries, and will report on the details elsewhere.

3. Data transfer patterns: An important algorithm in parallel finite element methods is the exchange of information stored on cells during mesh repartitioning. This happens, for example, when interpolating the solution from one mesh to the next adaptively refined mesh; or when adapting the polynomial degrees associated with each cell and repartitioning in order to balance the computational cost of each processor's partition. When using *h* adaptive methods, the amount of data associated with each cell is fixed and the algorithms that implement the data transfer are consequently relatively simple. On the other hand, in *hp* contexts, each cell may have a different number of unknowns associated with it, and the algorithms that transfer the data are substantially more complicated. In order to implement those, we rely on recent enhancements of the `p4est` library (documented in [15]) to transfer data of variable size across processors. Furthermore, the amount of data associated with each cell may be large on cells with higher polynomial degrees, and might profit from compression before sending.

4. Balancing computational cost: For *h* adaptive algorithms, the amount of work associated with each cell is essentially the same, both during the assembly of linear systems as well as during the solver phase. For *hp* adaptive methods, this is no longer the case. Consequently, balancing the cost of work between different processors' partitions is no longer as easy as ensuring that every processor owns a roughly equal number of cells. Rather, one needs to introduce a weighting factor for each cell that describes its relative cost compared to some reference. To make things worse, the relative cost of assembly on a cell might not match the relative cost of the linear solver associated with this cell, leading to difficult trade-offs in defining optimal weighting factors. In this release, we supplied the basic functionality to attach any amount of weighting factors to cells, but users still have to find reasonable weights for themselves.

All of these issues have been addressed in the current release and are available to users. We will report on the details of the algorithms and their performance in a separate publication.

2.4 Interface to the HDF5 file format and libraries

`HDF5` is an open-source library and file format designed to store large amounts of data. The `HDF5` format is specially tailored for high volume parallel I/O operations using MPI. `HDF5` files are self-describing and allow complex data relationships and a large variety of datatypes. In addition, the `HDF5` format is designed for long-term preservation of data; a `HDF5` file created by an HPC system can be easily read by a commodity laptop.

`deal.II`'s new `HDF5` interface allows to write `HDF5` files and manipulate datasets, groups, and attributes in serial and in parallel using MPI. The `HDF5` MPI library calls that modify the structure of the file are always collective, whereas writing and reading raw data in a dataset can be done independently or collectively. In the `deal.II`'s `HDF5` interface all the calls are set to collective in order to maximize performance. This means that all the MPI processes have to contribute to every single call, even if they don't have data to write. We have added the following classes to the `deal.II`'s `HDF5` namespace.

– The new `HDF5::File` class can be used to open and create `HDF5` files in serial or in parallel.

– The new `HDF5::Group` class can be used to open and create groups. `HDF5` files have a tree structure. The root contains groups and the groups can contain other groups.

– The new `HDF5::DataSet` class can be used to open and create datasets which can be placed inside groups or at the root of the `HDF5` file. A dataset can be a vector, a matrix, or a tensor. It is possible to read and write in the dataset using hyperslabs or unordered data. Hyperslabs are portions of datasets which can be a contiguous collection of points in a dataset, or a regular pattern of points or blocks in a dataset.

- It is possible to define attributes in the `HDF5::File`, `HDF5::Group` and `HDF5::DataSet` classes. An attribute is a small metadata such as a number or a string. Attributes are commonly used to store simulation parameters.

As we have shown in `step-62`, the `deal.II`'s `HDF5` interface can be easily used to exchange data with Python and Jupyter notebooks.

2.5 GPU support via CUDA

GPU support was significantly extended for the current release:

- The `CUDAWrappers::PreconditionILU` and `CUDAWrappers::PreconditionIC` classes can now be used for preconditioning `CUDAWrappers::SparseMatrix` objects.
- `LinearAlgebra::distributed::Vector`: the MPI-parallel vector class has gained a second template argument `MemorySpace` which can either be `Host` or `CUDA`. In the latter case, the data resides in GPU memory. By default, the template parameter is `Host` and the behavior is unchanged compared to previous versions. When using CUDA, the ghost exchange can be performed either by first copying the relevant data to the host, performing MPI communication, and finally moving the data to the device or, if CUDA-aware MPI is available, by performing MPI communication directly between GPUs.
- Constrained degrees of freedom: the matrix-free framework now supports constrained degrees of freedom. The implementation is based on the algorithms described in [43]. With this addition, both Dirichlet boundary conditions and the constraints arising from adaptively refined meshes can be imposed within the matrix-free framework.
- MPI matrix-free computations: using `LinearAlgebra::distributed::Vector`, the matrix-free framework can scale to multiple GPUs by taking advantage of MPI. Each MPI process can only use one GPU and therefore, if multiple GPUs are available in one node, it is necessary to have as many ranks as there are GPUs. Using Nvidia Multi-Process Service (MPS), it is also possible for multiple processes to use the same GPU. This can be advantageous if the amount of work on one rank is not sufficient to fully utilize a GPU.

The matrix-free GPU components integrated in `deal.II` have been compared against CPUs in [38], where the application to geometric multigrid solvers is discussed.

2.6 Parallel geometric multigrid improvements

For the 9.1 release, the geometric multigrid facilities have been extended and revised for performance. The geometric multigrid algorithms for uniform and adaptively refined meshes in `deal.II` are based on so-called local coarsening, i.e., smoothening is done level-per-level, skipping parts of the domain where the mesh is not as refined. The algorithm for the assignment of the owner on level cells and the implications on load balancing have been analyzed in detail in [17]. While most of the functionality has already been available since the 8.5 release of `deal.II` presented in [6], several components have been finalized, such as the support for certain renumbering algorithms that are beneficial for matrix-free execution, and interfaces that allow the combination with matrix-free GPU computations as showcased in [38].

A number of data structures and implementations in `deal.II` have been adapted to ensure scalability of the matrix-free algorithms and geometric multigrid infrastructure on more than 100,000 MPI ranks. A geometric multigrid solver for the Poisson equation as described in [39] has been used as a performance test during the acceptance phase of the SuperMUC-NG supercomputer in Garching, Germany. Scaling tests have been performed on up to the full machine with 304,128 cores of the Intel Xeon Skylake architecture and an arithmetic performance of around 5 PFlop/s for a geometric multigrid solver with polynomials of degree 4 has been reached. Compared to the official LINPACK performance of the machine of 19.5 PFlop/s (the machine is listed on position 8 of the top-500 list of November 2018), this can be considered an extremely good value for PDE solvers which have classically only reached a few percent of the LINPACK performance. More importantly, this is achieved within a flexible framework supporting arbitrary polynomial order on adaptively refined, unstructured meshes and with algorithms in the matrix-free module of `deal.II` designed to minimize time to solution and scalability, rather than maximizing the number of floating point operations. The largest Poisson problem that has been solved on 304k cores contained 2.15 trillion unknowns (or 7.1 million unknowns per MPI rank) and was solved in 3.5 seconds. Also, CFD production runs with 10^{11} unknowns and 10^5 time steps have been completed in less than seven hours, demonstrating the capabilities of `deal.II` for large-scale parallel computations. The scaling tests also revealed several relatively expensive operations in the setup of the multigrid unknowns in `deal.II`'s `DoFHandler` and `MGTransfer` classes. While a few bottlenecks have already been resolved for the present release, we plan several further improvements of the setup stage for the next release.

Furthermore, the implementation of the Chebyshev iteration, `deal.II`'s most popular smoother in the matrix-free context, has been revised to reduce the number of vector accesses. The vector operations have become an increasing bottleneck due to the level of optimization available for the operator evaluation with our matrix-free framework [37], especially on newer CPUs which can perform a large number of arithmetic operations per byte loaded from main memory. This speeds up matrix-free multigrid solvers by up to 10–15% on geometries with affine (parallelogram and parallelpiped) cells, and up to 5% on deformed cells.

2.7 New and improved tutorial and code gallery programs

Many of the `deal.II` tutorial programs were substantially revised as part of this release. In particular, we have converted many places that now allow for simpler code through the use of C++11 features such as range-based for loops and lambda functions.

In addition, there are four new tutorial programs and one new code gallery program:

– `step-61` is a program that implements the "weak Galerkin" method to solve the Laplace equation. Weak Galerkin methods are related to the Hybridized Discontinuous Galerkin method in that they introduce degrees of freedom on the interfaces between cells, but they do not require the reformulation of the problem as a first-order system and instead re-define what the gradient of a discontinuous function is.

– `step-62` demonstrates the solution of problems related to phononic or photonic crystals. Among the techniques shown in this program is the solution of complex-valued linear systems, and the use of absorbing boundary conditions through the Perfectly Matched Layer technique.

– `step-63` implements a geometric multigrid preconditioner and solver for the advection-diffusion equation, yielding optimal complexity. The tutorial compares point-based smoothers to cell-based smoothers and demonstrates the effect of downstream ordering on smoother performance.

– `step-64` demonstrates the usage of matrix-free methods on Nvidia GPUs. GPUs are shown to be advantageous for these kind of operations because of their superior hardware characteristics, in particular a higher memory bandwidth than server CPUs within a given power envelope.

– The `MCMC-Laplace` code gallery program is a code useful for the forward solution used as a building block in Bayesian inverse problems, and for sampling the parameter space through a Metropolis–Hastings sampler (a kind of Monte Carlo Markov Chain method).

2.8 Incompatible changes

The 9.1 release includes around 15 incompatible changes; see [42]. The majority of these changes should not be visible to typical user codes; some remove previously deprecated classes and functions; and the majority change internal interfaces that are not usually used in external applications. However, some are worth mentioning:

– The `VectorView` class was removed. We recommend either copying the vector subset into a `Vector` or using a `BlockVector`.

– The function `Subscriptor::subscribe()`, used through the `SmartPointer` class, now requires a pointer to a `std::atomic<bool>` that tracks whether or not the pointer to the subscribed-to object is still valid.

– The `ConstraintMatrix` class gained a template parameter for the scalar type and was been renamed `AffineConstraints`. Several methods that take vectors or matrices as arguments, such as `AffineConstraints::distribute_local_to_global()`, now require that all matrix and vector arguments have matching number types.

– Similarly, the functions `create_mass_matrix` and `create_boundary_mass_matrix` in the `MatrixCreator` namespace no longer support matrix and vector objects of different types.

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 it.

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 for this document see:

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

The original `deal.II` paper containing an overview of its architecture is [10]. If you rely on specific features of the library, please consider citing any of the following:

– For geometric multigrid: [34, 33, 17];

– For distributed parallel computing: [9];

– For *hp* adaptivity: [11];

– For partition-of-unity (PUM) and enrichment methods of the finite element space: [21];

– For matrix-free and fast assembly techniques: [36, 37];

– For computations on lower-dimensional manifolds: [22];

– For integration with CAD files and tools: [28];

– For Boundary Elements Computations: [26];

– For `LinearOperator` and `PackagedOperation` facilities: [44, 45].

– For uses of the `WorkStream` interface: [56];

– For uses of the `ParameterAcceptor` concept, the `MeshWorker::ScratchData` base class, and the `ParsedConvergenceTable` class: [52].

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

- ADOL-C [27, 57]
- ARPACK [41]
- Assimp [53]
- BLAS and LAPACK [5]
- cuSOLVER [18]
- cuSPARSE [19]
- Gmsh [24]
- GSL [23]
- Ginkgo [25]
- HDF5 [55]
- METIS [35]
- MUMPS [2, 3, 4, 46]
- muparser [47]
- nanoflann [14]
- NetCDF [50]
- OpenCASCADE [48]
- p4est [15, 16]
- PETSc [7, 8]
- ROL [51]
- ScaLAPACK [13]
- SLEPc [29]
- SUNDIALS [32]
- SymEngine [54]
- TBB [49]
- Trilinos [30, 31]
- UMFPACK [20]

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

The two previous releases of `deal.II` can be cited as [6, 1].

4 Acknowledgments

`deal.II` is a world-wide project with dozens of contributors around the globe. Other than the authors of this paper, the following people contributed code to this release:
Giovanni Alzetta, Mathias Anselmann, Daniel Appel, Alexander Blank, Vishal Boddu, Benjamin Brands, Pi-Yueh Chuang, Sambit Das, Stefano Dominici, Nivesh Dommaraju, Niklas Fehn, Isuru Fernando, Andreas Fink, Rene Gassmöller, Alexander Grayver, Joshua Hanophy, Logan Harbour, Daniel Jodlbauer, Stefan Kaessmair, Eldar Khattatov, Alexander Knieps, Uwe Köcher, Kurt Kremitzki, Dustin Kumor, Damien Lebrun-Grandie, Jonathan Matthews, Stefan Meggendorfer, Pratik Nayak, Lei Qiao, Ce Qin, Reza Rastak, Roland Richter, Alberto Sartori, Svenja Schoeder, Sebastian Stark, Antoni Vidal, Jiaxin Wang, Yuxiang Wang, Zhuoran Wang.

Their contributions are much appreciated!

`deal.II` and its developers are financially supported through a variety of funding sources:

D. Arndt and M. Kronbichler were partially supported by the German Research Foundation (DFG) under the project "High-order discontinuous Galerkin for the exa-scale" (ExaDG) within the priority program "Software for Exascale Computing" (SPPEXA).

W. Bangerth, T. C. Clevenger, and T. Heister were partially supported by the National Science Foundation under award OAC-1835673 as part of the Cyberinfrastructure for Sustained Scientific Innovation (CSSI) program and by the Computational Infrastructure in Geodynamics initiative (CIG), through the National Science Foundation under Award No. EAR-1550901 and The University of California – Davis.

W. Bangerth and T. Heister were also partially supported by award DMS-1821210.

D. Davydov was supported by the German Research Foundation (DFG), grant DA 1664/2-1 and the Bayerisches Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen (KONWIHR).

T. Heister was also partially supported by NSF Award DMS-1522191, and by Technical Data Analysis, Inc. through US Navy SBIR N16A-T003.

M. Kronbichler was also supported by the Bayerisches Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen (KONWIHR) in the context of the project "Performance tuning of high-order discontinuous Galerkin solvers for SuperMUC-NG".

R. M. Kynch was supported by the Engineering and Physical Science Research Council (EPSRC) UK through grant EP/K023950/1 while working at Swansea University, UK 2013-2015 where the majority of his contribution was undertaken.

M. Maier was partially supported by ARO MURI Award No. W911NF-14-0247.

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.

D. Wells was supported by the National Science Foundation (NSF) through Grant DMS-1344962.

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

References

[1] 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. Amestoy, I. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods in Appl. Mech. Eng.*, 184:501–520, 2000.

[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] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[6] 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` library, version 8.5. *Journal of Numerical Mathematics*, 25(3):137–146, 2017.

[7] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. May, L. C. McInnes, R. Mills, T. Munson, K. Rupp, P. S. B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.9, Argonne National Laboratory, 2018.

[8] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. May, L. C. McInnes, R. Mills, T. Munson, K. Rupp, P. S. B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2018.

[9] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Trans. Math. Softw.*, 38:14/1–28, 2011.

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

[11] W. Bangerth and O. Kayser-Herold. Data structures and requirements for *hp* finite element software. *ACM Trans. Math. Softw.*, 36(1):4/1–4/31, 2009.

[12] C. Bernardi and G. Raugel. Analysis of some finite elements for the Stokes problem. *Mathematics of Computation*, 44(169):71–79, 1985.

[13] 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.

[14] J. L. Blanco and P. K. Rai. nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) with KD-trees. `https://github.com/jlblancoc/nanoflann`, 2014.

[15] C. Burstedde. Parallel tree algorithms for AMR and non-standard data access. *arXiv e-prints*, page arXiv:1803.08432, Mar 2018.

[16] 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.

[17] T. C. Clevenger, T. Heister, G. Kanschat, and M. Kronbichler. A flexible, parallel, adaptive geometric multigrid method for FEM. Technical report, arXiv:1904.03317, 2019.

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

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

[20] T. A. Davis. Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30:196–199, 2004.

[21] 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.

[22] 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.

[23] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich. Gnu scientific library reference manual (edition 2.3), 2016.

[24] 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.

[25] Ginkgo: high-performance linear algebra library for manycore systems. `https://github.com/ginkgo-project/ginkgo`.

[26] 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(March):39–58, 2018.

[27] 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 (TOMS)*, 22(2):131–167, 1996.

[28] 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. Submitted.

[29] 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.

[30] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31:397–423, 2005.

[31] M. A. Heroux et al. Trilinos web page, 2018. http://trilinos.org.

[32] 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 (TOMS)*, 31(3):363–396, 2005.

[33] 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.

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

[35] 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.

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

[37] M. Kronbichler and K. Kormann. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. *ACM Trans. Math. Soft.*, in press:1–37, 2019.

[38] M. Kronbichler and K. Ljungkvist. Multigrid for matrix-free high-order finite element computations on graphics processors. *ACM Trans. Parallel Comput.*, 6(1):2/1–32, 2019.

[39] M. Kronbichler and W. A. Wall. A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. *SIAM J. Sci. Comput.*, 40(5):A3423–A3448, 2018.

[40] R. M. Kynch and P. D. Ledger. Resolving the sign conflict problem for hp–hexahedral Nédélec elements with application to eddy current problems. *Computers & Structures*, 181:41–54, Mar. 2017.

[41] 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.

[42] List of changes for 9.1. https://www.dealii.org/developer/doxygen/deal.II/changes_between_9_0_1_and_9_1_0.html.

[43] K. Ljungkvist. Matrix-free finite-element computations on graphics processors with adaptively refined unstructured meshes. In *Proceedings of the 25th High Performance Computing Symposium*, HPC '17, pages 1:1–1:12, San Diego, CA, USA, 2017. Society for Computer Simulation International.

[44] 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.

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

[46] MUMPS: a MUltifrontal Massively Parallel sparse direct Solver. http://graal.ens-lyon.fr/MUMPS/.

[47] muparser: Fast Math Parser Library. http://muparser.beltoforion.de/.

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

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

[50] R. Rew and G. Davis. NetCDF: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990.

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

[52] 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.

[53] T. Schulze, A. Gessler, K. Kulling, D. Nadlinger, J. Klein, M. Sibly, and M. Gubisch. Open asset import library (assimp). *Computer Software, URL: https://github. com/assimp/assimp*, 2012.

[54] SymEngine: fast symbolic manipulation library, written in C++. `https://github.com/symengine/symengine`, `http://sympy.org/`.

[55] The HDF Group. Hierarchical Data Format, version 5, 1997-2018. `http://www.hdfgroup.org/HDF5/`.

[56] 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.

[57] A. Walther and A. Griewank. Getting started with ADOL-C. In *Combinatorial Scientific Computing*, Chapman-Hall CRC Computational Science, pages 181–202. U. Naumann and O.Schenk, 2012.

[58] S. Zaglmayr. *High Order Finite Element Methods for Electromagnetic Field Computation*. PhD thesis, Johannes Kepler University, Linz, Austria, 2006.