Wolfgang Bangerth, Denis Davydov, Timo Heister*, Luca Heltai, Guido Kanschat, Martin Kronbichler, Matthias Maier, Bruno Turcksin, and David Wells

The deal.II Library, Version 8.4

Abstract: This paper provides an overview of the new features of the finite element library deal.II version 8.4.

Keywords: software, finite elements, deal.II

MSC 2010: 65M60, 65N30, 65Y05

DOI: 10.1515/jnma-2016-1045
Received May 11, 2016; accepted May 13, 2016

1 Overview

deal.II version 8.4.0 was released March 11, 2016. 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 8.4. deal.II is an object-oriented finite element library used around the world in the development of finite element solvers. It is available for free under the GNU Lesser General Public License (LGPL) from the deal.II homepage at http://www.dealii.org/.

The major changes of this release are:

- Parallel triangulations can now be partitioned in ways that allow weighting cells differently.
- Improved support for mixed-type arithmetic throughout the library.
- A new triangulation type that supports parallel computations but ensures that the entire mesh is available on every processor.
- An implementation of the Rannacher–Turek element, as well as an element that extends the usual $Q(p)$ elements by bubble functions.
- Second and third derivatives of finite element fields are now computed exactly.
- The various *Concepts*, or requirements on template parameters in the library, are now consistently labeled and documented as such.
- The interface between finite elements, quadrature, mapping, and the FEValues class has been rewritten. It is now much better documented.
- Initial support for compiling with Visual C++ 2013 and 2015 under Microsoft Windows has been added.
- More than 140 other features and bugfixes.

The more important ones of these will be detailed in the following section. Information on how to cite deal.II is provided in Section 3.

Wolfgang Bangerth, Bruno Turcksin: Department of Mathematics, Texas A&M University, College Station, TX 77843, USA.
Denis Davydov: Chair of Applied Mechanics, University of Erlangen-Nuremberg, Egerlandstr. 5, 91058 Erlangen, Germany.
***Corresponding Author: Timo Heister:** Mathematical Sciences, O-110 Martin Hall. Clemson University. Clemson, SC 29634, USA. Email: heister@clemson.edu
Luca Heltai: SISSA — International School for Advanced Studies, Via Bonomea 265, 34136 Trieste, Italy.
Guido Kanschat: Interdisciplinary Center for Scientific Computing (IWR), Universität Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany.
Martin Kronbichler: Institute for Computational Mechanics, Technical University of Munich, Boltzmannstr. 15, 85748 Garching, Germany.
Matthias Maier: School of Mathematics, University of Minnesota, 127 Vincent Hall, 206 Church Street SE, Minneapolis, MN 55455, USA.
David Wells: Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, USA.

2 Significant changes to the library

This release of deal.II contains a number of large and significant changes that will be discussed in the following sections. 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 [26]) and that is linked to from the web site of each release as well as the release announcement.

2.1 Parallel triangulations can now be partitioned with weights

Previously, partitioning a parallel mesh represented by objects of class `parallel::distributed::Triangulation` between processors assumed that every cell should be weighted equally. On the other hand, the p4est library which manages the partitioning process, allows for attaching weights to each cell and thereby enables ways in which not the number of cells per MPI process is equilibrated, but the sum of weights on the cells managed by each process. deal.II now also supports this feature.

The implementation of this mechanism is based on a callback mechanism, in the form of the signal-slot design pattern. User codes can register functions that will be called upon mesh refinement and coarsening, returning a weight for each cell. These weights will be added up over all slots (i.e., callback functions) connected to the signal.

The mechanism chosen has the advantage that all parties that use a triangulation, for example multiple `DoFHandler` objects or a scheme that tracks particles that are advected along with a flow field and stores them per-cell, can indicate their computational needs for each cell. In particular, there is no central place in a user code (other than the triangulation itself) that has to collect these needs and forward this information.

2.2 Improved support for mixed-type arithmetic throughout the library

When evaluating finite element fields or their derivatives at a points $\mathbf{x}_q$, one typically has to do an operation of the form

$$u_h(\mathbf{x}_q) = \sum_j U_j \varphi_j(\mathbf{x}_q), \quad \nabla u_h(\mathbf{x}_q) = \sum_j U_j \nabla \varphi_j(\mathbf{x}_q).$$

The value or derivatives of shape functions, $\varphi_j(\mathbf{x}_q)$ or $\nabla\varphi_j(\mathbf{x}_q)$ are internally evaluated by classes derived from the `FiniteElement` and `Mapping`, and are computed as scalars or tensors of type `double`. On the other hand, the expansion coefficients of the field, U_j are stored in vectors over scalar types chosen by the user; their underlying representation could be `double`, but also `float`, `long double`, `PetscScalar`, or `std::complex<float>`. It could also be an autodifferentiation type.

To facilitate the correct typing of the computed quantity, deal.II now contains mechanisms by which one can evaluate the *type* of the product of two values, and this type is now consistently used throughout the library in expressions such as those above. Thus, as an example, if the user stores the expansion coefficients in a `Vector<long double>`, then the gradients $\nabla u_h(\mathbf{x}_q)$ will be computed as objects of type `Tensor<1,dim,long double>`. Likewise, if a PETSc vector is used, and PETSc was configured with complex scalar types, then the second derivatives of the solution field will be computed as `SymmetricTensor<2,dim,PetscScalar>`, which should equal `SymmetricTensor<2,dim,std::complex<double> >`.

These improvements make type-correct computations possible in many places. In particular, this enables the use of complex-valued solution vectors in many more places than before. On the other hand, many but not all places have learned what actually to do with complex numbers. This is, in particular, true for the `DataOut` class that generates an intermediate data format that can then be written to graphical output files for visualization. Since no format we are aware of supports complex numbers, future versions still need to learn how to separate real and imaginary parts of complex numbers, and output them as two components.

2.3 A new 'shared' triangulation type for parallel computations

deal.II already has two types of triangulations: the `Triangulation<dim,spacedim>` class works entirely locally, whereas `parallel::distributed::Triangulation<dim,spacedim>` builds on the former, but only stores the local partition that corresponds to a globally distributed triangulation managed across an MPI network. One *could*, however, use the former also for parallel computations in situations where one needs access to *all* cells, not just the subset of cells that correspond to the partition owned by the current processor. This required building the same triangulation on all processors, then manually partitioning it (e.g., via `METIS`), calculating and storing index sets of locally owned and locally relevant degrees of freedom (DoFs), querying the `subdomain_id` of a cell during assembly, etc.

In order to simplify this usage of the `Triangulation` class with MPI and to make its behavior in this context consistent with `parallel::distributed::Triangulation`, a new class `parallel::shared::Triangulation` has been introduced. It extends the `Triangulation` class to automatically partition the triangulation when run with MPI. Shared functionality between the `shared` and `distributed` triangulation classes (e.g., locally owned and relevant DoFs, MPI communicators, etc) is now grouped in a common parent class `parallel::Triangulation`. The main difference between the two classes is that in the case of `parallel::shared::Triangulation` each process stores all cells of the triangulation. Consequently, by default there are no artificial cells. That is, cells which are attributed to the current processor are marked as locally owned (`cell->is_locally_owned()` returns `true`) and the rest are ghost cells. This behavior can be altered via an additional boolean flag provided to the constructor of the class. In this case, the set of ghost cells will consist of a halo layer of cells around locally owned cells. Cells which are neither ghost nor locally owned are marked as artificial. This is consistent with the behavior of `parallel::distributed::Triangulation`, although in the latter case the size of the set of artificial cells will be much smaller.

The introduction of the `parallel::shared::Triangulation` class together with the optional artificial cells and parent `parallel::Triangulation` class facilitates writing algorithms that are indifferent to the way a triangulation is stored in the MPI context. For example, the function `DoFTools::locally_active_dofs()` will return the appropriate subset of all DoF indices for both triangulations. Assmebly routines can use predicates such as `cell->is_locally_owned()` for both triangulations.

Based on the new triangulation class, the (non-*hp*) `DoFHandler` manages degrees of freedom in the same way as it has already done for a long time for sequential and parallel distributed triangulations.

2.4 Second and third derivatives of finite element fields are now computed exactly

Second derivatives of solution fields, i.e.,

$$\nabla^2 u_h(\mathbf{x}_q) = \sum_j U_j \nabla^2 \varphi_j(\mathbf{x}_q)$$

were previously computed by finite differencing of first derivatives. The reason for this approach is that in order to compute the second derivatives of shape functions, $\nabla^2 \varphi_j(\mathbf{x}_q)$ one needs (at least) derivatives of the inverse of the Jacobian of the mapping. This is easy to see because, for the usual Q_p Lagrange elements, one has that $\nabla \varphi_j(\mathbf{x}_q) = J^{-1} \hat{\nabla} \hat{\varphi}_j(\hat{\mathbf{x}}_q)$, where quantities with a hat refer to coordinates and functions on the reference cell, and J is the Jacobian of the mapping from reference to real cell. Thus, the second derivatives satisfy

$$\nabla^2 \varphi_j(\mathbf{x}_q) = J^{-1} \hat{\nabla} \left[J^{-1} \hat{\nabla} \hat{\varphi}_j(\hat{\mathbf{x}}_q) \right] = J^{-1} J^{-1} \hat{\nabla}^2 \hat{\varphi}_j(\hat{\mathbf{x}}_q) + J^{-1} \left(\hat{\nabla} [J^{-1}] \right) \hat{\nabla} \hat{\varphi}_j(\hat{\mathbf{x}}_q)$$

where in the last expression, matrices and tensors of rank 1 and 3 have to be appropriately contracted. Here, the difficulty lies in computing the derivative $\hat{\nabla}[J^{-1}]$: for the usual mappings on quadrilaterals and hexahedra, J is in general a polynomial in the reference coordinates $\hat{\mathbf{x}}$, so J^{-1} is a rational function. It is possible to compute the derivatives of this object, but they are difficult and cumbersome to evaluate, especially for higher order mappings.

The key to computing second (and higher) derivatives of shape functions is to recognize that $\hat{\nabla}[J^{-1}]$ can be expressed more conveniently by observing that

$$0 = \hat{\nabla} I = \hat{\nabla}(JJ^{-1}) = J(\hat{\nabla}[J^{-1}]) + (\hat{\nabla} J)J^{-1}$$

and consequently, $\hat{\nabla}[J^{-1}] = -J^{-1}(\hat{\nabla} J)J^{-1}$. Here, J^{-1} is a matrix that is already available from computing first derivatives, and $\hat{\nabla} J$ is an easily computed rank-3 tensor with polynomial entries. Using this approach, we have

$$\nabla^2 \varphi_j(\mathbf{x}_q) = J^{-1}J^{-1}\hat{\nabla}^2 \hat{\varphi}_j(\hat{\mathbf{x}}_q) - J^{-1}J^{-1}(\hat{\nabla} J)J^{-1}\hat{\nabla}\hat{\varphi}_j(\hat{\mathbf{x}}_q)$$

again with an appropriate set of contractions over the indices of the objects on the right.

This, and corresponding extensions to compute third derivatives, have now been implemented in several of the finite element and mapping classes by Maien Hamed, and are available through the `FEValues` interface to shape functions and their derivatives.

2.5 Visual C++ support

The library can now be compiled under Windows with Visual C++ 2013 and 2015. The support is still experimental for the following reasons: First, we currently only support static linking. This will slow down linking of application code immensly. Second, only a minimal testsuite is working, which is mainly because static linking of thousands of test executables is not viable. Therefore, we can not exclude the possibility of subtle bugs in the library. Finally, there is of course limited support for external packages.

2.6 Incompatible changes

2.6.1 Revision of the interface between finite elements, quadratures, and mappings

Finite element classes describe shape functions as continuous (as opposed to discrete) objects on the reference cell. On the other hand, in actual practice, one only needs information about shape functions at finitely many quadrature points, and these are typically the same on every cell in a loop over all cells. Furthermore, the evaluation of shape functions at quadrature points then also needs to be mapped to the real cell, typically using a polynomial mapping.

To facilitate this complex interplay, deal.II has three class hierarchies rooted in the base classes `FiniteElement` (for the description of shape functions on the reference cell), `Quadrature` (for the locations and weights of quadrature points on the reference cell), and `Mapping` (for the description of mappings from the reference to the real cell). In almost all cases, users create an object of a derived class for each of these categories, but they never access any of the members of these classes and instead leave this task to the `FEValues` class (and `FEFaceValues`, `FESubfaceValues`) that presents all one typically needs: the mapped values, gradients, and higher order derivatives of shape functions at the quadrature points of the real cell.

The interplay between these classes in the `FEValues` interface is one of the oldest parts of the library. It was, at the time, not designed based on specifications we explicitly or implicitly knew this class had to satisfy, but instead organically grew to its current state. These interfaces have now been fundamentally rewritten: some functions have been replaced; several others had their argument lists shuffled, sorted, and made more uniform; and everything has generally been far better documented.

None of the changes in this arena is visible to the average user. The only user codes that are affected in an incompatible way are those that implement finite element or mapping classes.

2.6.2 Other incompatible changes

The file that lists all changes for this release (see [26]) lists another 18 incompatible changes, but none of these should in fact be visible in typical user codes. Some remove previously deprecated classes and functions, and the majority change internal interfaces that are not typically used in user codes.

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 a particular version of the library, reference the present document. For up to date information and bibtex snippets for this document see:

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

The original deal.II paper containing an overview of its architecture is [7]. If you rely on specific features of the library, please consider citing any of the following:

- For geometric multigrid: [21, 22];
- For distributed parallel computing: [6];
- For *hp* adaptivity: [13];
- For matrix-free and fast assembly techniques: [24];
- For computations on lower-dimensional manifolds: [16];
- For integration with CAD files and tools: [17];
- For `LinearOperator` and `PackagedOperation` facilities: [27, 28].
- For uses of the `WorkStream` interface: [35].

deal.II can interface with many other libraries:

- ARPACK [25]
- BLAS, LAPACK
- HDF5 [34]
- METIS [23]
- MUMPS [1–3, 29]
- muparser [30]
- NetCDF [33]
- OpenCASCADE [31]
- p4est [14]
- PETSc [4, 5]
- SLEPc [18]
- Threading Building Blocks [32]
- Trilinos [19, 20]
- UMFPACK [15]

Please consider citing the appropriate references if you use interfaces to these libraries. Older releases of deal.II can be cited as [8–11].

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: Daniel Arndt, Mauro Bardelloni, Alistair Bentley, Andrea Bonito, Claire Bruna-Rosso, Krzysztof Bzowski, Praveen Chandrashekar, Conrad Clevenger, Patrick Esser, Rene Gassmoeller, Arezou Ghesmati, Maien Hamed, Alexander Grayver, Lukas Korous, Aslan Kosakian, Adam Kosik, Konstantin Ladutenko, Jean-Paul Pelteret, Lei Qiao, Gennadiy Rishin, Angel Rodriguez, Alberto Sartori, Daniel Shapero, Jason Sheldon, Jan Stebel, Florian Sonner, Zhen Tao, Heikki Virtanen, Daniel Weygand.

Their contributions are much appreciated!

Funding: deal.II and its developers are financially supported through a variety of funding sources. W. Bangerth and B. Turcksin were partially supported by the National Science Foundation under award OCI-1148116 as part of the Software Infrastructure for Sustained Innovation (SI2) program; and by the Computational Infrastructure in Geodynamics initiative (CIG), through the National Science Foundation under Award No. EAR-0949446 and The University of California–Davis.

D. Davydov was supported by the ERC Advanced Grant MOCOPOLY and the Competence Network for Technical and Scientific High Performance Computing in Bavaria (KONWIHR).

L. Heltai was partially supported by the project OpenViewSHIP, 'Sviluppo di un ecosistema computazionale per la progettazione idrodinamica del sistema elica-carena', financed by Regione FVG–PAR FSC 2007–2013, Fondo per lo Sviluppo e la Coesione, and by the project TRIM 'Tecnologia e Ricerca Industriale per la Mobilità Marina', CTN01-00176-163601, funded by MIUR — Ministero dell'Istruzione, dell'Università e della Ricerca.

T. Heister was partially supported by the Computational Infrastructure in Geodynamics initiative (CIG), through the National Science Foundation under Award No. EAR-0949446 and The University of California—Davis, and National Science Foundation grant DMS1522191.

G. Kanschat and M. Kronbichler were partially supported by the German Research Foundation (DFG) through the project ExaDG. M. Kronbichler also acknowledges the Gauss Centre for Supercomputing e.V. for funding algorithm enhancements by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ) through project id pr83te.

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

References

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling, *SIAM J. Matrix Anal. Appl.* **23** (2001), 15–41.

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

[3] P.R. Amestoy, I.S. Duff and J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, *Comput. Methods Appl. Mech. Engrg.* **184** (2000), 501–520.

[4] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, K. Rupp, B. F. Smith, and H. Zhang, *PETSc Users Manual*, Argonne National Laboratory, Report No. ANL-95/11 - Revision 3.5, 2014.

[5] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, K. Rupp, B. F. Smith, and H. Zhang, *PETSc Web page*, http://www.mcs.anl.gov/petsc, 2014.

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

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

[8] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, and B. Turcksin, The deal.II Library, Version 8.3, *Archive of Numer. Software* **4** (2016), 1–11.
[9] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young, The deal.II Library, Version 8.0, *arXiv preprint http://arxiv.org/abs/1312.2266v3* (2013).
[10] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young, The deal.II Library, Version 8.1, *arXiv preprint http://arxiv.org/abs/1312.2266v4* (2013).
[11] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young, The deal.II Library, Version 8.2, *Archive of Numer. Software* **3** (2015).
[12] W. Bangerth and G. Kanschat, *Concepts for Object-Oriented Finite Element Software – the deal.II library*, SFB 359, Preprint No. 1999-43, Heidelberg, 1999.
[13] W. Bangerth and O. Kayser-Herold, Data structures and requirements for *hp* finite element software, *ACM Trans. Math. Software* **36** (2009), 4/1–4/31.
[14] C. Burstedde, L. C. Wilcox, and O. Ghattas, p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, *SIAM J. Sci. Comput.* **33** (2011), 1103–1133.
[15] T. A. Davis, Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method, *ACM Trans. Math. Software* **30** (2004), 196–199.
[16] A. DeSimone, L. Heltai, and C. Manigrasso, *Tools for the Solution of PDEs Defined on Curved Manifolds with deal.II*, SISSA, Report No. 42/2009/M, 2009.
[17] L. Heltai and A. Mola, *Towards the Integration of CAD and FEM Using Open Source Libraries: a Collection of deal.II Manifold Wrappers for the OpenCASCADE Library*, SISSA, Report, 2015, Submitted.
[18] V. Hernandez, J. E. Roman, and V. Vidal, SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems, *ACM Trans. Math. Software* **31** (2005), 351–362.
[19] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, An overview of the Trilinos project, *ACM Trans. Math. Software* **31** (2005), 397–423.
[20] M. A. Heroux et al., *Trilinos web page*, 2014, http://trilinos.sandia.gov.
[21] B. Janssen and G. Kanschat, Adaptive multilevel methods with local smoothing for H^1- and H^{curl}-conforming high order finite element methods, *SIAM J. Sci. Comput.* **33** (2011), 2095–2114.
[22] G. Kanschat, Multi-level methods for discontinuous Galerkin FEM on locally refined meshes, *Comput. & Struct.* **82** (2004), 2437–2445.
[23] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.* **20** (1998), 359–392.
[24] M. Kronbichler and K. Kormann, A generic interface for parallel cell-based finite element operator application, *Comput. Fluids* **63** (2012), 135–147.
[25] 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.
[26] *List of Changes*, https://www.dealii.org/8.4.0/doxygen/deal.II/changes_between_8_3_and_8_4.html.
[27] Matthias Maier, Mauro Bardelloni, and Luca Heltai, LinearOperator – a generic, high-level expression syntax for linear algebra, *Computers Math. Appl.* (2016), To appear.
[28] Matthias Maier, Mauro Bardelloni, and Luca Heltai, *LinearOperator Benchmarks, Version 1.0.0*, March 2016.
[29] *MUMPS: a MUltifrontal Massively Parallel sparse direct Solver*, http://graal.ens-lyon.fr/MUMPS/.
[30] *muparser: Fast Math Parser Library*, http://muparser.beltoforion.de/.
[31] *OpenCASCADE: Open CASCADE Technology, 3D modeling & numerical simulation*, http://www.opencascade.org/.
[32] J. Reinders, *Intel Threading Building Blocks*, O'Reilly, 2007.
[33] R. Rew and G. Davis, NetCDF: an interface for scientific data access, *Computer Graph. Appl., IEEE* **10** (1990), 76–82.
[34] The HDF Group, *Hierarchical Data Format, version 5*, 1997-NNNN, http://www.hdfgroup.org/HDF5/.
[35] B. Turcksin, M. Kronbichler, and W. Bangerth, *WorkStream* – a design pattern for multicore-enabled finite element computations, *ACM Trans. Math. Software* (2016), (accepted).