The deal.II Library, Version 8.5

Daniel Arndt[1], Wolfgang Bangerth[2], Denis Davydov[3], Timo Heister[4], Luca Heltai[5], Martin Kronbichler[6], Matthias Maier[7], Jean-Paul Pelteret[8], Bruno Turcksin[*9], and David Wells[10]

[1]Interdisciplinary Center for Scientific Computing, Heidelberg University, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany. daniel.arndt@iwr.uni-heidelberg.de
[2]Department of Mathematics, Colorado State University, Fort Collins, CO 80523-1874, USA. bangerth@colostate.edu
[3]Chair of Applied Mechanics, University of Erlangen-Nuremberg, Egerlandstr. 5, 91058 Erlangen, Germany. denis.davydov@fau.de
[4]Mathematical Sciences, O-110 Martin Hall, Clemson University, Clemson, SC 29634, USA. heister@clemson.edu
[5]SISSA, International School for Advanced Studies, Via Bonomea 265, 34136, Trieste, Italy. luca.heltai@sissa.it
[6]Institute for Computational Mechanics, Technical University of Munich, Boltzmannstr. 15, 85748 Garching, Germany. kronbichler@lnm.mw.tum.de
[7]School of Mathematics, University of Minnesota, 127 Vincent Hall, 206 Church Street SE, Minneapolis, MN 55455, USA. msmaier@umn.edu
[8]Chair of Applied Mechanics, University of Erlangen-Nuremberg, Egerlandstr. 5, 91058 Erlangen, Germany. jean-paul.pelteret@fau.de
[9]Computational Engineering and Energy Sciences Group, Computional Sciences and Engineering Division, Oak Ridge National Laboratory, 1 Bethel Valley Rd., TN 37831, USA turcksinbr@ornl.gov
[10]Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, USA wellsd2@rpi.edu

Abstract: This paper provides an overview of the new features of the finite element library deal.II version 8.5.

1 Overview

deal.II version 8.5.0 was released April 6, 2017. 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 8.5. deal.II is an object-oriented finite element library used around the world in the development of finite element solvers. It is available for free under the GNU Lesser General Public License (LGPL) from the deal.II homepage at http://www.dealii.org/.

The major changes of this release are:

– The CellDataStorage class provides a mechanism to store and communicate user-defined data on each cell.

*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).

- The `MappingManifold` class provides mappings between the reference cell and a mesh cell that is "exact", rather than the usual polynomial approximations of a manifold.
- Various improvements for high-order elements, including a switch of support points in `FE_Q` and `FE_DGQ` to Gauss-Lobatto support points, stable evaluation of high-order Legendre polynomials, and several bugfixes for high-order polynomial mappings defined through the `MappingQ` class.
- The `LinearOperator` class has been extended by a generic "payload" mechanism that allows the attachment of arbitrary additional information to a `LinearOperator`.
- A dedicated physics module has been created to provide some standard definitions and operations used in continuum mechanics.
- The `FE_Enriched` class implements the operation of enriching the finite element space of an underlying element.
- The `FESeries` namespace provides expansions of a finite element solution in terms of different, hierarchical bases.
- New tutorial programs step-55, step-56, and step-57; as well as updates to step-27, step-37, and step-44. In addition, the separate code gallery of `deal.II` has gained a number of new entries.
- Static code analyzers are valuable tools to improve and maintain the quality of the code in our library in addition to build and regression tests for a variety of setups using CDash. This release was tested with Cppcheck, PVS-Studio and Coverity-Scan.
- More than 240 other features and bugfixes.

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

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 [1]. (The file is also linked to from the web site of each release as well as the release announcement.)

2.1 The `CellDataStorage` class and friends

The `CellDataStorage` templated class is an integrated mechanism to store a vector of user-defined data within each cell, such as a coefficient at each quadrature point. Although the same was previously achieved through the use of a cell's `user_pointer`, it required users to manage this data themselves, including transfer of this data if ownership of a cell is transferred from one processor to another in parallel computations. Instead, this data is now treated as a first-class citizen in `deal.II`.

`CellDataStorage` can work with arbitrary kinds of data defined to live on a cell as long as the corresponding data type implements a default constructor. Additionally, when a user's custom data type is derived from `TransferableQuadraturePointData` and thereby implements certain interfaces, then the `parallel::distributed::ContinuousQuadratureDataTransfer` class can transfer this data during *h*-adaptive refinement from parent cell to children (interpreting the data on quadrature points as discrete representations of an underlying continuous field), as well as from one processor to another during repartitioning.

2.2 The `MappingManifold` class

The `MappingManifold` class implements the functionality of the `Mapping` interface for *manifold conforming* mappings. In other words, instead of using polynomial approximations of the objects that describe the boundaries (or interiors) of cells, this class computes the transformation between the reference and real cell by exploiting the geometrical information coming from the underlying `Manifold` object.

When using this class, quadrature points lie on the *exact* geometrical objects, and tangent and normal vectors computed are tangent and normal to the underlying geometry. This is in contrast with the `MappingQ` and `MappingQGeneric` classes, which approximate the geometry using a polynomial of some order, and then compute the normals and tangents using the approximated surface.

The class currently only implements the information that relates to the mapping itself, as well to its derivatives (such as the Jacobian of the mapping, or the determinant thereof). Information related to higher order derivatives – such as the Hessian of the mapping – will result in an exception.

2.3 Extension of the `LinearOperator` class

We have extended the `LinearOperator` class by a generic "payload" mechanism that allows for the attachment of arbitrary additional information to a `LinearOperator`. This was achieved by introducing a generic `Payload` base class. The main use case of the new mechanism is to extend the `LinearOperator` class to seamlessly exploit the native features and operations offered by external linear algebra libraries. We have thus developed a `TrilinosPayload` class that provides full support for the suite of `Trilinos` parallel iterative solvers and preconditioners.

A particularly interesting case is the construction of an `inverse_operator`. Whereas previously only `deal.II`'s built-in solvers were compatible with this operator, one can now also select those offered by `Trilinos`. This has been achieved by using the `Epetra_Operator` as the basis for the `TrilinosPayload`, for which the result of standard and composite operations involving forward (`Apply()`) and inverse (`ApplyInverse()`) matrix-vector multiplication are collated using lambda functions.

We envisage that, in the future, similar extensions can be implemented for the `PETSc` iterative solvers.

Another additional in this release related to the `LinearOperator` suite is the definition of a `schur_complement` operator and its associated condensation and post-processing `PackagedOperations`. An operator representing the Schur complement of a block system can be declared and, through the delayed evaluation offered by `PackagedOperations`, reused on any number of vector systems.

2.4 The physics module

We have created a dedicated physics module to facilitate the implementation of functions and classes that relate to continuum mechanics, physical fields and material constitutive laws. To date, it includes transformations of scalar or tensorial quantities between any two configurations (by user-specification of a linear map $\mathbf{F}$), and some definitions typically utilized in both linear and finite-strain nonlinear elasticity.

The `Physics::Transformations` namespace offers push-forward and pull-back operations in the context of contravariant, covariant and Piola transformations, as well as rotation operations for the Euclidean space. Although these transformations are defined in a general manner, one typical use of them in finite-strain elasticity would be the determination of the Cauchy stress tensor $\boldsymbol{\sigma} = \boldsymbol{\sigma}(\mathbf{x})$ defined at a spatial position $\mathbf{x} \in \mathcal{B}$ from its fully referential counterpart, namely the Piola-Kirchhoff stress tensor $\mathbf{S} = \mathbf{S}(\mathbf{X})$ computed at the material coordinate $\mathbf{X} \in \mathcal{B}_0$. By choosing $\mathbf{F}(\mathbf{X}) = \dfrac{\partial \mathbf{x}(\mathbf{X})}{\partial \mathbf{X}}$, this is achieved through the action of the Piola push-forward $\boldsymbol{\sigma} = \dfrac{1}{\det \mathbf{F}} \mathbf{F} \cdot \mathbf{S} \cdot \mathbf{F}^T$.

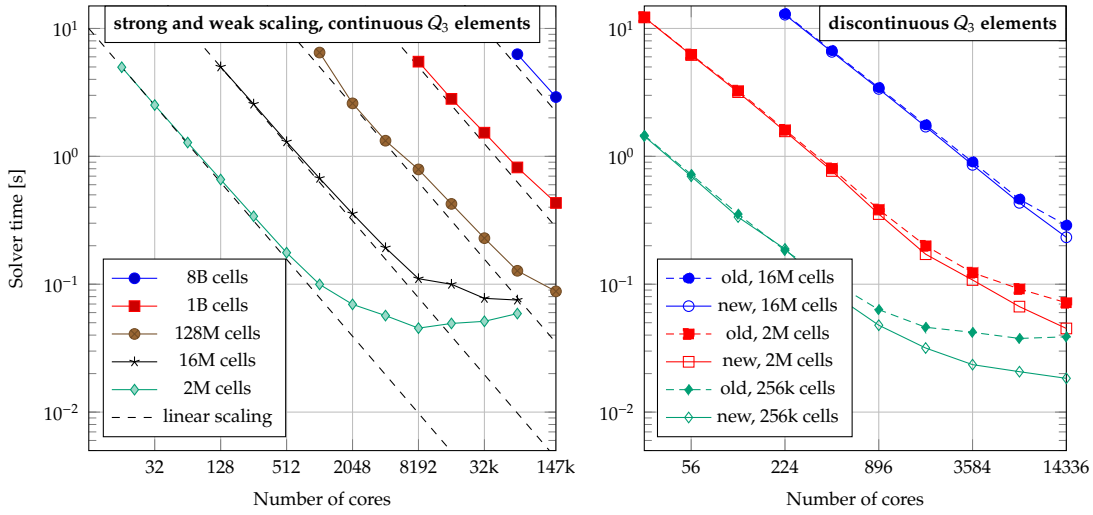

Figure 1: Scaling of `deal.II`'s geometric multigrid algorithms on SuperMUC. Each line corresponds to a strong scaling experiment, increasing the number of processor cores for a fixed-size problem. Comparing corresponding data points on different lines yields weak scaling information.

In the `Physics::Elasticity::Kinematics` namespace, a selection of deformation, strain and strain rate tensors are defined. The `Physics::Elasticity::StandardTensors` class provides some frequently used second and fourth order metric tensors, and defines a number of referential and spatial projection operators and tensor derivatives that are commonly required in the definition of material laws.

The updated step-44 tutorial program demonstrates the use of the physics module in the context of a solid mechanics problem. Additional updates to this tutorial include application of the new `CellDataStorage` class to store and retrieve local quadrature point data. An alternative approach to solving the linear system, previously performed by global static condensation of two of the three field variables, has been implemented using the `LinearOperator` class.

2.5 Scalability of geometric multigrid framework

For the new release, the geometric multigrid facilities in `deal.II` have been thoroughly overhauled regarding their scalability on large-scale parallel systems. During this process, a geometric multigrid implementation based on the fast matrix-free kernels from [28] has been benchmarked on up to 147,456 cores. The fast matrix-vector products revealed several scalability bottlenecks in the other multigrid components, including unnecessary inner products inside the Chebyshev smoother and $O(n_{\text{levels}})$ global communication steps during the restriction process rather than the single unavoidable global communication step inherent to going to the coarsest grid and the coarse solve. We implemented new matrix-free transfer implementations called `MGTransferMatrixFree` that can replace the matrix-based `MGTransferPrebuilt` class for tensor product elements. Besides better scalability than the Trilinos Epetra matrices underlying the latter, the matrix-free transfer is also a much faster for high-order elements with a complexity per degree of freedom of $O(dp)$ in the polynomial degree p in d dimensions rather than $O(p^d)$ for the matrices.

The scalability of the improved geometric multigrid framework is shown in Fig. 1, including a combined strong and weak scaling plot in the left panel using continuous Q_3 elements with 57 million to 232 billion degrees of freedom for discretizing the Laplacian. Along each line, the same problem size is solved with an increasing number of cores, whereas different lines are a factor of eight apart and always start at 3.5 million degrees of freedom per core with an absolute

performance of around 650,000 degrees of freedom per core and second. Almost ideal scalability down to approximately 0.1 seconds can be observed also on 147k cores. The right panel of Fig. 1 shows the effect of the aforementioned algorithmic improvements on a setup with discontinuous DG elements, clearly improving the latency of the multigrid V-cycle.

The updated step-37 tutorial programs presents the updated algorithms and the MPI-parallel multigrid setting with matrix-free operator evaluation.

2.6 Matrix-free operators

In order to facilitate the usage of matrix-free methods, a `MatrixFreeOperator::Base` class has been introduced, implementing functionality for matrix-vector products and the necessary operations for the interface residuals for multigrid on adaptively refined meshes (see [25]). Furthermore, the class is compatible with the linear operator framework and provides an interface to a Jacobi preconditioner. Derived classes only need to implement the `apply_add()` method that is used in the `vmult()` functions, and a method to compute the diagonal entries of the underlying matrix. The `MatrixFreeOperator` namespace contains implementations of `MatrixFreeOperators::LaplaceOperator` and `MatrixFreeOperators::MassOperator`.

The updated step-37 tutorial program makes use of these facilities and explains their usage in detail. Using the matrix-free mass operator, `VectorTools::project` has become much faster than the previous matrix-based approach for elements supported by `MatrixFree` and also works for parallel computations based on MPI.

2.7 The `FE_Enriched` class

The `FE_Enriched` finite element implements a partition of unity finite element method (PUM) by Babuska and Melenk which enriches a standard finite element with an enrichment function multiplied with another (usually linear) finite element. This allows including a priori knowledge about the partial differential equation being solved in the finite element space, which in turn improves the local approximation properties of the spaces. Programs can also use enriched and non-enriched finite elements in different parts of the domain.

The `DoFTools::make_hanging_node_constraints()` function can automatically make the resulting space C^0 continuous. The existing `SolutionTransfer` class can be used to transfer the solution during h-adaptive refinement from a coarse to a fine mesh under the condition that all child elements are also enriched.

2.8 The `FESeries` namespace

The `FESeries` namespace offers functions to calculate expansion series of the solution on the reference element. Coefficients of expansion are often used to estimate local smoothness of the underlying finite element field to decide on a h- or p-adaptive refinement strategy. Specifically, `FESeries::Legendre` calculates expansion of a scalar finite element field into series of Legendre functions on the reference element, whereas `FESeries::Fourier` calculates Fourier coefficients. Programs using this functionality have to specify the required number of coefficients in each direction as well as provide a collection of finite elements and quadrature rules; these will be used for the calculation of the transformation matrices.

The updated step-27 tutorial program demonstrates the use of `FESeries::Fourier`.

2.9 New and updated tutorial programs

In addition to the updated tutorial programs mentioned in the previous section, this release of `deal.II` includes three new tutorials:

- **step-55** explains how to solve the Stokes equations efficiently in parallel. It is a good introduction to solving systems of PDEs in parallel, discusses optimal block preconditioners, and demonstrates other aspects like error computation. Inverses of individual blocks of the linear system are approximated with an algebraic multigrid preconditioner.
- **step-56** shows how to apply geometric multigrid preconditioners on a subset of a system of PDEs. The problem solved here is the Stokes equations, like in step-55.
- **step-57** solves the stationary Navier-Stokes equations. The nonlinear system is solved using Newton's method on a sequence of adaptively refined grids. The preconditioner is again built on a block factorization of the saddle point system like in step-55 and step-56, but the non-symmetric terms stemming from the nonlinear convective part requires more sophisticated solvers. The benchmark problem, flow in the 2d lid-driven cavity, requires a continuation method for high Reynolds numbers.

In addition to tutorials, `deal.II` has a separate "code gallery" that consists of programs shared by users as examples of what can be done with `deal.II`. While not part of the release process, it is nonetheless worth mentioning that the set of new programs since the last release covers the following topics:

- Quasi-static quasi-incompressible visco-elastic material behavior;
- Multiphase Navier-Stokes flow;
- The evolution of global-scale topography on planetary bodies;
- Goal-oriented elastoplasticity.

2.10 Incompatible changes

The 8.5 release includes around 20 incompatible changes; see [1]. 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, three incompatible changes are worth mentioning:

- High-order Lagrange elements, both continuous `FE_Q` and discontinuous `FE_DGQ` types, now use the nodal points of the Gauss-Lobatto quadrature formula as support points by default, rather than the previous equidistant ones. For cubic polynomials and higher, the point distribution has thus changed and, consequently, the entries in solution vectors will be different compared to previous versions of `deal.II`. Note, however, that using the Gauss-Lobatto points as nodal points results in a much more stable interpolation, including better iteration counts in most iterative solvers.
- The library no longer instantiates template classes with `long double`. These were rarely used, but took up a significant fraction of compile and link time, as well as library size. Application programs can, however, still instantiate all template classes with `long double` as long as they include the corresponding `.templates.h` header files.
- The `ParameterGUI` has been moved to a separate repository.

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 bibtex snippets for this document see:

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

The original deal.II paper containing an overview of its architecture is [9]. If you rely on specific features of the library, please consider citing any of the following:

- For geometric multigrid: [26, 25];
- For distributed parallel computing: [7];
- For *hp* adaptivity: [15];
- For *PUM* and enrichment of the FE space: [18];
- For matrix-free and fast assembly techniques: [28];
- For computations on lower-dimensional manifolds: [19];
- For integration with CAD files and tools: [21];
- For `LinearOperator` and `PackagedOperation` facilities: [30, 31].
- For uses of the `WorkStream` interface: [38].

deal.II can interface with many other libraries:

- ARPACK [29]
- BLAS, LAPACK
- GSL [20]
- HDF5 [37]
- METIS [27]
- MUMPS [2, 3, 4, 32]
- muparser [33]
- NetCDF [36]
- OpenCASCADE [34]
- p4est [16]
- PETSc [5, 6]
- SLEPc [22]
- Threading Building Blocks [35]
- Trilinos [23, 24]
- UMFPACK [17]

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

Older releases of deal.II can be cited as [11, 12, 13, 10, 8].

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:

Rajat Arora, Mauro Bardelloni, Conrad Clevenger, Sam Cox, Toby D. Young, Juliane Dannberg, Nicola Demo, Patrick Esser, Niklas Fehn, Rene Gassmoeller, Joscha Gedicke, Nicola Giuliani, Sebastian Gonzalez-Pintor, Ryan Grove, Michael Harmon, Daniel Jodlbauer, Guido Kanschat, Justin Kauffman, Eldar Khattatov , Uwe Koecher, Alex Kokomov, Paul Kuberry, Dustin Kumor, Konstantin Ladutenko, Karl Ljungkvist, Andrew McBride, Mathias Mentler, Andrea Mola, Dragan Nikolic, Vaibhav Palkar, Spencer Patty, Jonathan Perry-Houts, Giuseppe Pitton, Ce Qin, Jonathan Robey, Mayank Sabharwal, Ali Samii, Alberto Sartori, Daniel Shapero, Martin Steigemann, Jihuan Tian, Jaeryun Yim, Liang Zhao

Their contributions are much appreciated!

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

D. Arndt was 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 was 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 Awards No. EAR-0949446 and EAR-1550901 and The University of California – Davis.

D. Davydov was supported by the European Research Council (ERC) through the Advanced Grant 289049 MOCOPOLY and the Competence Network for Technical and Scientific High Performance Computing in Bavaria (KONWIHR).

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.

M. Kronbichler was 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), grant agreement no. KR4661/2-1, the Bayerisches Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen (KONWIHR), and the Gauss Centre for Supercomputing e.V. by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ) through project id pr83te.

J-P. Pelteret was supported by the European Research Council (ERC) through the Advanced Grant 289049 MOCOPOLY.

B. Turcksin: This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC05-00OR22725.

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 and the SVN archive.

References

[1] List of changes. https://www.dealii.org/developer/doxygen/deal.II/changes_between_8_4_and_8_5.html.

[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] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.

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

[7] 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.

[8] W. Bangerth, D. Davydov, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and D. Wells. The `deal.II` library, version 8.4. *Journal of Numerical Mathematics*, 24(3):135–141, 2016.

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

[10] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, and B. Turcksin. The `deal.II` library, version 8.3. *Archive of Numerical Software*, 4(100):1–11, 2016.

[11] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young. The `deal.II` library, version 8.0. *arXiv preprint* http://arxiv.org/abs/1312.2266v3, 2013.

[12] 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.

[13] 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 Numerical Software*, 3, 2015.

[14] W. Bangerth and G. Kanschat. Concepts for object-oriented finite element software – the `deal.II` library. Preprint 1999-43, SFB 359, Heidelberg, 1999.

[15] 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.

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

[18] D. Davydov, T. Gerasimov, J.-P. Pelteret, and P. Steinmann. On the h-adaptive PUM and hp-adaptive FEM approaches applied to PDEs in quantum mechanics. arXiv:1612.02305 [physics.comp-ph], 2016.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] M. A. Heroux et al. Trilinos web page, 2014. http://trilinos.sandia.gov.

[25] 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.

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

[27] 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.

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

[29] 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.

[30] M. Maier, M. Bardelloni, and L. Heltai. LinearOperator – a generic, high-level expression syntax for linear algebra. *Computers and Mathematics with Applications*, 2016. To appear.

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

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

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

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

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

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

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

[38] 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.