Summary of *The DEAL.II Finite Element Library: Design, Features, and Insights*

Abdullah Naci Bodur

Abstract

`DEAL.II` is a state-of-the-art finite element library emphasizing generality, dimension-independent programming, parallelism, and extensibility. The library provides sophisticated features such as distributed meshes, *hp*-adaptivity, support for complex geometries, and matrix-free algorithms. More than just a software tool, `DEAL.II` is supported by a diverse, worldwide community of developers and users, and serves as an educational platform. The paper outlines the library's primary design considerations and technical features, and shares insights and lessons learned from running a large, community-driven software project over two decades, addressing both technical and social challenges.

1 Design and Core Concepts

The design of the `DEAL.II` library is guided by principles that position it as a complete toolbox for finite element codes, allowing users to combine generic tools freely in their application codes. It is explicitly designed as a **library**, not a framework, meaning it provides the building blocks but does not dictate the overall structure or logic of the user's program.

A central mathematical concept is the bilinear form for the weak formulation of the Laplace equation, often written independently of dimension d as:

$$a(u, v) := (\nabla u, \nabla v)_\Omega = \int_\Omega \nabla u \cdot \nabla v \, dx, \tag{1}$$

where the gradient ∇u is a d-dimensional vector, and $\Omega \subset \mathbb{R}^d$.

The library's core tenets are summarized in Table 1.

Table 1: Key Design Principles of the `DEAL.II` Finite Element Library

Principle	Description
Complete Toolbox	Provides all generic functionality (meshes, linear algebra, etc.) for finite element discretization.
Library, not Framework	Provides building blocks; user dictates the overall program logic to ensure maximum flexibility.
Dimension-Independent Programming (DIP)	Uses templates to make the spatial dimension (d) a compile-time constant, improving compiler optimization and allowing mixed-dimension programming.
Iterator-based Programming	Accesses mesh objects (cells, faces) via iterators pointing to "accessor" objects, hiding internal data structures and enhancing cache locality.
Large-scale Parallelism	Supports parallel computation using task-based programming (shared memory) and distributed memory parallelization via MPI.

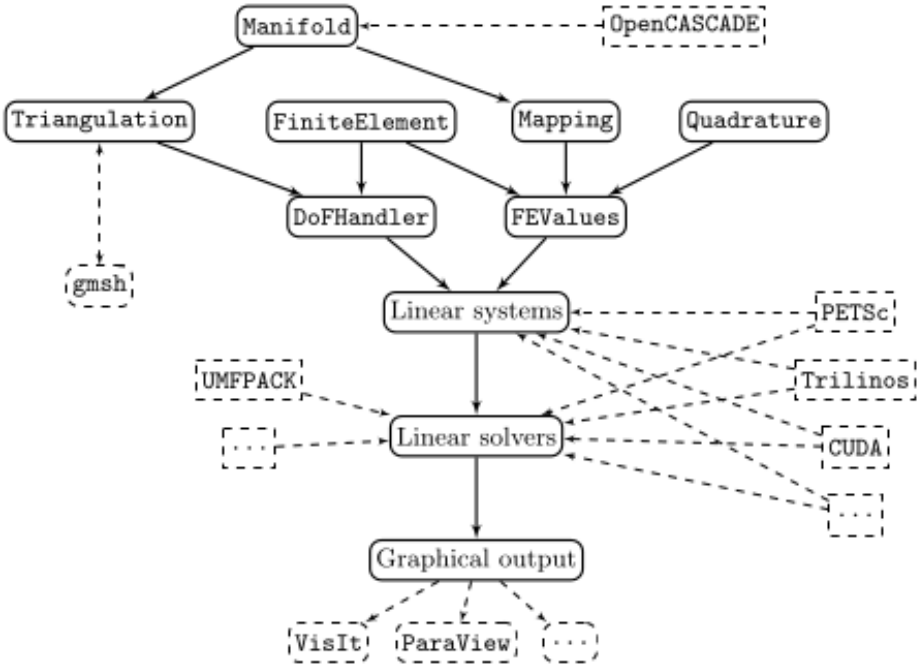


Figure 1: Core components of `deal.II` and its connection to external libraries and functionalities [1].

2 Dimension-Independent Programming

A key technique is DIP, where the dimension is a template argument, such as in the core `Point` class definition (simplified):

```
template <int dim> class Point {
  private:
    double coordinates[dim];
  public:
    double operator[](const unsigned int i);
};
```

This design choice allows the compiler to unroll loops and allocate memory on the stack efficiently, even when mixing different dimensional objects in the same program (e.g., `Point<2>` and `Point<3>`). This enables code to be a near-literal translation of mathematical notation, such as the process for assembling a matrix corresponding to $a(\cdot, \cdot)$ from Eq. 1 on one cell.

3 Features and Scaling

The library provides extensive features, including sequential, shared, and distributed triangulations for parallelism, and uses `Manifold` classes to handle complex geometries and adaptive mesh refinement. It supports both h- and hp-adaptivity for individual cell refinement and polynomial degree selection.

4 Large-Scale Performance

For optimal performance on modern hardware, `DEAL.II` supports geometric multigrid (GMG) methods and matrix-free operators. The matrix-free approach avoids building and storing a global sparse matrix, instead solving linear systems by the action of the linear operator on a vector using the weak form integrals. This is crucial for high-performance computing, as it reduces the memory transfer bottleneck.

A study on the parallel scaling of one GMG V-cycle for the 3D Laplace equation demonstrates the library's capability. The experiment scaled up to 304,128 cores and 2.15×10^{12} unknowns, achieving arithmetic throughputs of 5.9 PFlop/s. The plot in Fig. 2 illustrates the scaling behavior across various problem sizes.

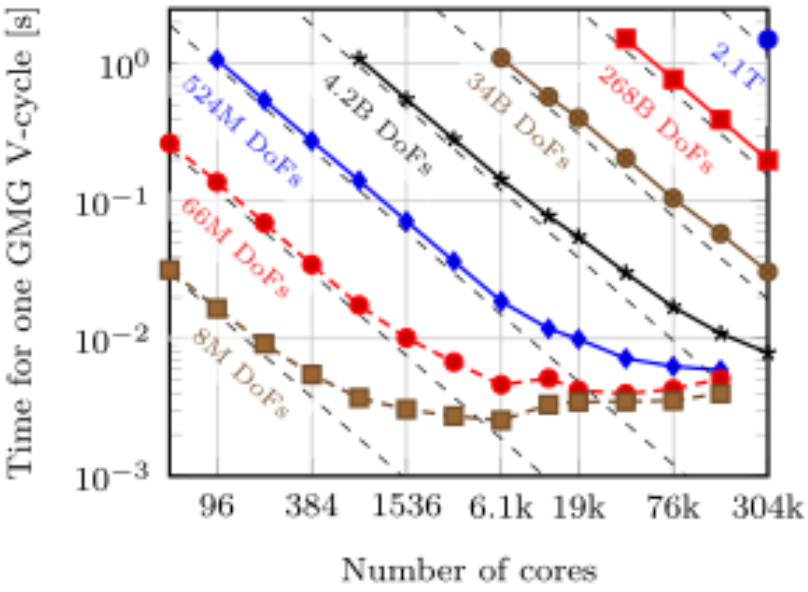
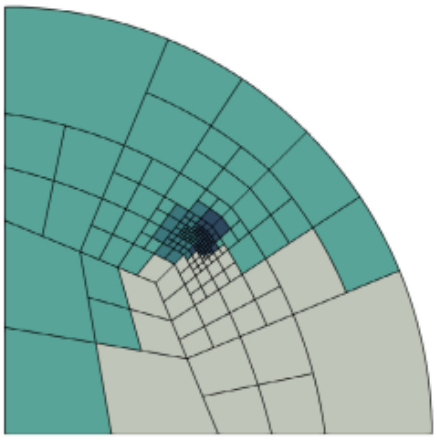


Figure 2: Parallel scaling of a GMG V-cycle on up to 304,128 cores and up to 2×10^{12} unknowns. The dashed lines show ideal scaling.

5 Geometry and Adaptivity

The geometric abstractions and adaptive refinement capabilities are illustrated in Fig. 3a, which shows an adaptively refined mesh with hanging nodes and curved faces, where the boundary geometry is an exact circle extended into the interior using transfinite interpolation.



(a) Adaptively refined mesh with hanging nodes.

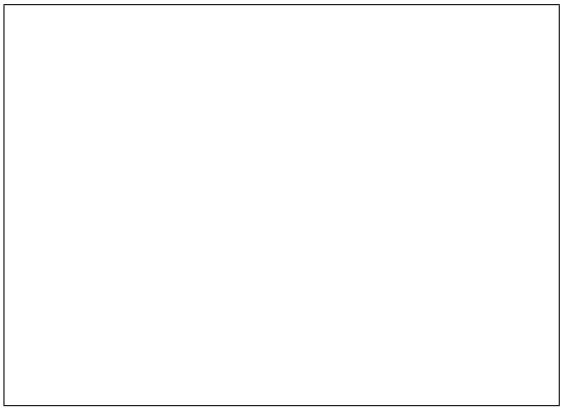


Figure 3: Adaptive mesh refinement and domain decomposition in `deal.II` [1].

References

[1] Daniel Arndt, Wolfgang Bangerth, Denis Davydov, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Jean-Paul Pelteret, Bruno Turcksin, and David Wells, *The DEAL.II finite element library: Design, features, and insights*, *Computers and Mathematics with Applications*, 81 (2021) 407–422.