Computer Physics Communications 249 (2020) 106997

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

High-performance dune modules for solving large-scale, strongly anisotropic elliptic problems with applications to aerospace composites ☆

R. Butler [a], T. Dodwell [b,c], A. Reinarz [d,*], A. Sandhu [b], R. Scheichl [e,f], L. Seelinger [e]

[a] Department of Mechanical Engineering, University of Bath, UK
[b] College of Engineering, Mathematics and Physical Sciences, University of Exeter, UK
[c] The Alan Turing Institute, London, NW1 2DB, UK
[d] Institute of Informatics, Technical University of Munich, Germany
[e] Institute for Scientific Computing, University of Heidelberg, Germany
[f] Department of Mathematical Sciences, University of Bath, UK

ARTICLE INFO

Article history:
Received 16 January 2019
Received in revised form 12 September 2019
Accepted 16 October 2019
Available online 25 October 2019

Keywords:
Composites
Parallel iterative solvers
Domain decomposition
High performance computing

ABSTRACT

The key innovation in this paper is an open-source, high-performance iterative solver for high contrast, strongly anisotropic elliptic partial differential equations implemented within `dune-pdelab`. The iterative solver exploits a robust, scalable two-level additive Schwarz preconditioner, GenEO (Spillane et al., 2014). The development of this solver has been motivated by the need to overcome the limitations of commercially available modelling tools for solving structural analysis simulations in aerospace composite applications. Our software toolbox `dune-composites` encapsulates the mathematical complexities of the underlying packages within an efficient C++ framework, providing an application interface to our new high-performance solver. We illustrate its use on a range of industrially motivated examples, which should enable other scientists to build on and extend `dune-composites` and the GenEO preconditioner for use in their own applications. We demonstrate the scalability of the solver on more than 15,000 cores of the UK national supercomputer ARCHER, solving an aerospace composite problem with over 200 million degrees of freedom in a few minutes. This scale of computation brings composites problems that would otherwise be unthinkable into the feasible range. To demonstrate the wider applicability of the new solver, we also confirm the robustness and scalability of the solver on SPE10, a challenging benchmark in subsurface flow/reservoir simulation.

Program summary

Program Title: `dune-composites`
Program Files doi: http://dx.doi.org/10.17632/96mtdcmjsb.1
Licensing provisions: BSD 3-clause
Programming language: C++
Nature of problem: `dune-composites` is designed to solve anisotropic linear elasticity equations for anisotropic, heterogeneous materials, e.g. composite materials. To achieve this, our contribution also implements a new preconditioner in `dune-pdelab`.
Solution method: The anisotropic elliptic partial differential equations are solved via the finite element method. The resulting linear system is solved via an iterative solver with a robust, scalable two-level overlapping Schwarz preconditioner: GenEO.

1. Introduction

Across the physical sciences, elliptic partial differential equations (PDEs) naturally arise as mathematical models of the equilibrium state of a system. Classical examples include the distribution of temperature in a body, the flow of fluid in a porous medium and, the particular application of interest in this paper, the equilibrium of forces acting on a material. The most widely used approach to solve such PDEs is the finite element

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655)..

* Corresponding author.
E-mail address: reinarz@in.tum.de (A. Reinarz).

https://doi.org/10.1016/j.cpc.2019.106997

method [1], which results in a sparse system of equations. For systems which exhibit multiple scales and large spatial variations in model parameters the system of equations can be very ill-conditioned and extremely large, e.g. contain $> 10^7$ degrees of freedom. The design of robust and scalable solvers that do not require laborious tuning and are capable of exploiting the power of modern distributed computers, is essential. In this paper, we describe the design and implementation of a robust two-level additive Schwarz preconditioner for parallel Krylov based iterative solvers in `dune-pdelab` [2]. By also creating a bespoke module for analysis of composite structures `dune-composites`, we demonstrate the capabilities of this new solver on industrially motivated aerospace composite problems with over 200 million degrees of freedom. To also demonstrate the wider applicability of the new solver, we demonstrate the robustness and scalability of the solver on the challenging, classical SPE10 benchmark [3,4] in subsurface flow/reservoir simulation.

1.1. Motivating computational challenge in aerospace composites

Scientific advances in aerospace composite design and materials offer exciting engineering opportunities, making them the material of choice for many modern aircraft (e.g Airbus A350, Boeing 787). However, composite manufacturers face huge challenges in designing and making complex components quickly enough to remain commercially competitive. There is a growing realisation in both academia and industry that to meet ambitious global growth targets, composites manufacturers should *'reduce time, cost and risk to market through the use of validated simulation packages'* [5]. Currently simulation capabilities allowing high-fidelity full-scale analysis of a composite structure are not openly available. But, why is this? What makes this analysis of large scale composite structure so challenging?

When we apply classical finite element (FE) analysis to a composite structure the problem reduces to finding a vector of displacements $\mathbf{u}^{(i)} \in \mathbb{R}^3$ at each of the N nodes within a FE mesh. This leads to the sparse system of FE equations [1]:

$$\mathbf{A}\tilde{\mathbf{u}} = \mathbf{b}, \quad \text{where} \quad \tilde{\mathbf{u}} = [\mathbf{u}_h^{(1)}, \dots, \mathbf{u}_h^{(N)}]^T, \tag{1}$$

$\mathbf{A}$ is the global stiffness matrix and $\mathbf{b}$ is the load vector arising from the applied boundary conditions or loading. In solving the linear system (1), we face two significant mathematical challenges:

- **Scale of calculations.** Composite materials are manufactured from thin fibrous layers, less than 1 mm thick, separated by even thinner resin interfaces of thickness less than 0.05 mm, yet entire component parts are generally several metres long, Fig. 1. To resolve stresses and accurately predict failure, several elements need to be placed through each layer [6]. Naturally this means that the number of nodes N is very large. As an example in this paper, we model a 1 m section of a wing box given in Section 5.3, while resolving the resin interfaces, giving in total 200 million degrees of freedom. Solving linear systems of this size requires specialised, parallel solvers. Current industry standard tools, such as Abaqus [7], are not able to deal with these problem sizes, largely due to limitations of the parallel solvers employed.
- **Material anisotropy.** Central to the benefits of composite structures is the inclusion of directional fibres, which gives them an excellent weight to stiffness ratio under a particular loading. This means, there is a large contrast ($\sim 1 : 40$) in mechanical properties within a single layer of composite, related to the fibre direction(s) and those directions dominated by the stiffness of the matrix material (typically a toughened epoxy resin). In the FE discretisation, this leads to a stronger coupling between degrees of freedom in the fibre direction, as opposed to those in the orthogonal directions. The fibrous layers are stacked with different fibre orientations to form a laminate, adding an additional level of complexity. The fibre directions act as stiff constraints on the deformation, whilst the weak connections give rise to low-energy mechanics within the structure. Mathematically, this causes significant numerical challenges in solving Eq. (1) via iterative solvers, since the system is very ill-conditioned. For such cases, classical iterative solvers (required to address Challenge 1) converge very slowly.

The usual approach to tackle both these challenges is to apply the parallel iterative solvers to a *preconditioned* version of Eq. (1). The task then is to develop an operator $\mathbf{M}^{-1}$, which is computationally cheap to construct, such that $\mathbf{M}^{-1}\mathbf{A}\tilde{\mathbf{u}} = \mathbf{M}^{-1}\mathbf{b}$ is better conditioned. The most widely used preconditioners for iterative solvers for (1) in both commercial and scientific FE codes are Algebraic Multigrid (AMG) methods [8,9]. They have demonstrated excellent scalability for a broad class of problems over thousands of processors, and have the advantage of working only on the matrix equations (1), so that they can be applied 'black-box'. As a preconditioner, AMG constructs the matrix $\mathbf{M}$ by repeatedly coarsening the full matrix $\mathbf{A}$ through recursive aggregation over the degrees of freedom. The aggregation process is algebraic and based on the fact that the solution at two neighbouring nodes will be similar if they are 'strongly connected'. The success of an AMG preconditioner depends on this aggregation process. As discussed above, the connectivity of degrees of freedom within a laminate is very complex even for a simple laminate and in our numerical experiments the performance of all AMG preconditioners that we tested was prohibitively poor. In particular, this includes off-the-shelf AMG used in the commercial software Abaqus [7], as well as tailored aggregation strategies in the AMG preconditioners provided through `dune-istl` [8] and `Hypre` [9].

This initial testing of AMG preconditioners highlighted the need for the development and implementation of a robust preconditioner with respect to both problem size, material heterogeneity and anisotropy for large-scale composite based applications. Over the last decade there has been significant effort from the domain decomposition community to develop scalable and robust preconditioners suitable for parallel computation. One such preconditioning approach is provided by the additive Schwarz framework [10]. The domain is decomposed into overlapping subdomains, which in our case each correspond to one processor, and the subdomain's local stiffness matrix is inverted on each processor using a direct solver. This "one level" approach is not sufficient for very large problems and global information in the form of a coarse space must be added. In `dune-composites`, we use GenEO [11] to construct a coarse space by combining low energy eigenvectors of the local subdomain stiffness matrices using a partition of unity. The resulting preconditioner leads to an almost optimal scaling with respect to problem size and number of processors, allowing us to successfully tackle large industrially important problems with over 200 million degrees of freedom.

1.2. The contributions of this paper

`dune-composites` is a high-performance composite FE package built on top of Dune (Distributed and Unified Numerics Environment), an open source modular toolbox for solving partial differential equations (PDEs) with grid-based methods [12–14]. Based on the core Dune philosophy, `dune-composites` is written using C++ and exploits modern inheritance and templating programming paradigms. It is open-source and publicly available at https://dune-project.org/modules/dune-composites/. The package provides a codebase with the following key features:

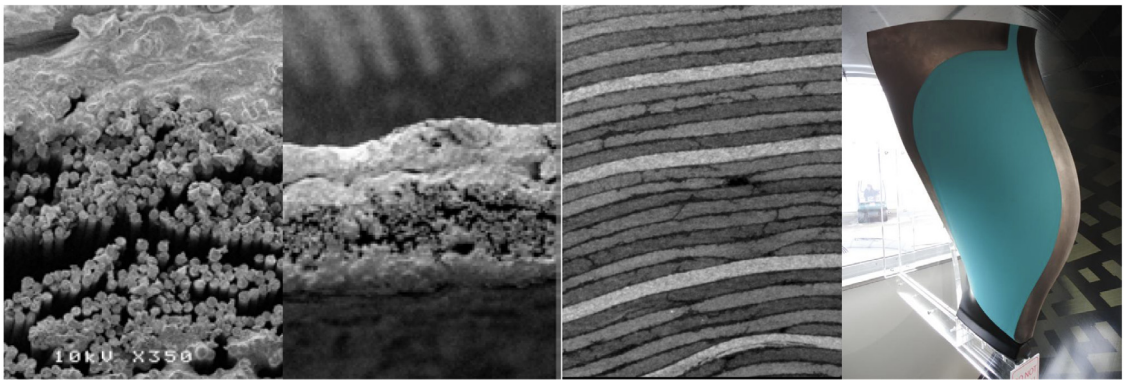

Fig. 1. The range of scales introduces significant complexity in the analysis of aerospace composites. From Left to Right: fibre/resin scale (5 μm), ply scale (0.25 mm), laminate scale (5–30 mm) to the structure (> 1 m)(Composite Fan Blade, Right).

- implementation and interface to a novel, robust preconditioner called GenEO [6,11] for parallel Krylov solvers, which exhibits excellent scalability over thousands of processors on Archer, the UK national HPC system. Since release 2.6, the preconditioner is provided as part of `dune-pdelab`, available at https://dune-project.org/modules/dune-pdelab/ (initially it had been developed within the `dune-composites` module);
- interfaces to handle composite applications, including stacking sequences, complex part geometries, defects and non-standard boundary conditions, such as multi-point constraints or periodicity;
- to overcome shear locking of standard FEs, mesh stabilisation strategies to support reduced integration [15], as well as a new 20-node 3D serendipity element (with full integration) have been implemented;
- interfaces to other state-of-the-art parallel solvers (& preconditioners) in `dune-istl` [8] and `Hypre` [9];
- a code structure which supports both engineering end-users, and those requiring flexibility to extend any aspect of the code in a modular way to introduce new applications, solvers or material models.

The purpose of this paper is to highlight the novel mathematical aspects of the code and document its structure. We illustrate its use through a range of industry motivated examples, which should enable other scientists to build on and extend *dune-composites* for use in their own applications. We begin by outlining the mathematical formulation of the new robust preconditioner and its implementation on a distributed memory computer in Section 3. We then provide details of the structure and salient features of the code in Section 4. Finally in Section 5, through the use of a series of example problems, we provide details of how to implement, build and run your own applications. We also use these examples as an opportunity to demonstrate the computational efficiency of `dune-composites`.

2. Preliminaries: Anisotropic elasticity equations and their finite element discretisation

A composite structure occupies the domain $\Omega \subset \mathbb{R}^3$ with the boundary Γ and a unit, outward normal vector $\mathbf{n} \in \mathbb{R}^3$. At each point $\mathbf{x} \in \Omega$ we define a vector-valued displacement $\mathbf{u}(\mathbf{x}) : \Omega \to \mathbb{R}^3$. In each of these three global directions the boundary may contain a Dirichlet component $\Gamma_D^{(i)}$ and a Neumann component $\Gamma_N^{(i)}$, such that

$$\Gamma = \Gamma_D^{(i)} \cup \Gamma_N^{(i)} \quad \text{and} \quad \Gamma_D^{(i)} \cap \Gamma_N^{(i)} = \emptyset, \quad i = x, y, z. \tag{2}$$

Let σ_{ij} denote the Cauchy stress tensor and $\mathbf{f}(\mathbf{x}) : \Omega \to \mathbb{R}^3$ the body force per unit volume. The infinitesimal strain tensor, is defined as the symmetric part of the displacement gradients

$$\epsilon_{ij}(\mathbf{u}) = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right), \tag{3}$$

where $u_{i,j} = \frac{\partial u_i}{\partial x_j}$. The strain tensor is connected to Cauchy stress tensor via the generalised Hooke's law

$$\sigma_{ij}(\mathbf{u}) = C_{ijkl}(\mathbf{x})\epsilon_{kl}(\mathbf{u}). \tag{4}$$

$C_{ijkl}(\mathbf{x})$ is a symmetric, positive definite fourth order tensor. A composite laminate is made up of a stack of composite layers (or plies $\sim$ 0.2 mm), separated by a very thin layer of resin (15 μm). A single composite layer is modelled as a homogeneous orthotropic elastic material, characterised in general by 9 parameters and a vector of orientations θ. Resin interfaces are assumed isotropic, defined by just 2 scalar (Lamé) parameters. These fibres are aligned in local coordinates and can be rotated in any direction using standard tensor rotations, for more details see e.g. [16].

Given functions $h_i : \Gamma_D^{(i)} \to \mathbb{R}$ and $g_i : \Gamma_N^{(i)} \to \mathbb{R}$, prescribing the Dirichlet and Neumann boundary data (for each component), we seek the unknown displacement field $\mathbf{u}(\mathbf{x})$, which satisfies the force equilibrium equations and the boundary conditions,

$$\begin{aligned} &\nabla \cdot \underline{\sigma}(\mathbf{u}) + \mathbf{f} = 0, \quad \mathbf{x} \in \Omega, \quad u_i = h_i \text{ for } \mathbf{x} \in \Gamma_D^{(i)} \quad \text{and} \\ &\times \ \sigma_{ij} n_j = g_i \quad \text{for } \mathbf{x} \in \Gamma_N^{(i)}, \end{aligned} \tag{5}$$

as well as Eqs. (3) and (4). Then, we define the function space for each component of displacement u_i to be

$$V^{(i)} := \{v \in H^1(\Omega) : v_i(\mathbf{x}) = h_i \ , \ \mathbf{x} \in \Gamma_D^{(i)}\}, \tag{6}$$

leading to the weak formulation of (5), of finding $\mathbf{u} \in V := V^{(1)} \otimes V^{(2)} \otimes V^{(3)}$ such that

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) &:= \int_\Omega \sigma_{ij}(\mathbf{u})\epsilon_{ij}(\mathbf{v})\, dx = \int_{\partial\Omega_n} \sigma_{ij} n_j v_i \, ds - \int_\Omega f_i v_i \, dx \\ &:= b(\mathbf{v}), \quad \forall \mathbf{v} \in V. \end{aligned} \tag{7}$$

We consider the discretisation of the variational equations (7) with conforming FEs on a mesh $\mathcal{T}_h$ on Ω. Let $V_h \subset V$ denote the restriction of V onto a FE space on $\mathcal{T}_h$ and seek an approximation $\mathbf{u}_h \in V_h$ such that

$$a(\mathbf{u}_h, \mathbf{v}_h) - b(\mathbf{v}_h) = 0, \quad \text{for all} \quad \mathbf{v}_h \in V_h. \tag{8}$$

We block together displacements from all three space dimensions, so that $\mathbf{u}_h^{(i)} \in \mathbb{B} = \mathbb{R}^3$ denotes the vector of displacement coefficients containing all space components associated with the ith basis function. We introduce the (vector-valued) FE basis for

V_h defined by the spanning set of (vector-valued) shape functions $\{\boldsymbol{\phi}^{(i)}(\mathbf{x})\}_{i=1}^{N}$. These are the normal scalar shape functions, repeated for each displacement component. Therefore the vector displacement at a point is given by $\mathbf{u}_h(\mathbf{x}) = \sum_{i=1}^{N}(\mathbf{u}_h^{(i)})^T\ \boldsymbol{\phi}^{(i)}(\mathbf{x})$. The choice of basis converts (8) into a symmetric positive-definite (spd) system of algebraic equations

$$\mathbf{A}\tilde{\mathbf{u}} = \mathbf{b} \quad \text{where} \quad \mathbf{A} \in \mathbb{B}^N \times \mathbb{B}^N \quad \text{and} \quad \mathbf{b} \in \mathbb{B}^N \tag{9}$$

where the blocks in the global stiffness matrix and in the load vector, for any $i, j = 1, \ldots, N$, are given by $\mathbf{A}_{ij} = a(\boldsymbol{\phi}^{(i)}, \boldsymbol{\phi}^{(j)})$ and $\mathbf{b}_i = b(\boldsymbol{\phi}_i)$. The vector $\tilde{\mathbf{u}} = [\mathbf{u}_h^{(1)}, \ldots, \mathbf{u}_h^{(N)}]^T \in \mathbb{B}^N$ is the block vector of unknown FE coefficients. System (9) is assembled element-wise from (7), using Gaussian integration.

3. A robust, scalable, parallel iterative solver for composite structures

The key innovation of *dune-composites*, as a software package, is the design and implementation of a highly robust, scalable parallel iterative solver for composite applications. This solver is applicable to a more general class of problems and has been made available in the *dune-pdelab* module. In this section, we provide the mathematical and implementation details of the new solver. Apart from new types of FEs that had to be implemented in DUNE, the remainder of the package largely provides interfaces to handle the set-up for complex composite applications.

3.1. Krylov subspace methods preconditioned with two-level additive Schwarz methods

In `dune-composites` we use *preconditioned Krylov subspace methods* both in sequential and parallel, as provided by DUNE's "Iterative Solver Template Library" `dune-istl` [8]. Krylov subspace methods are iterative solvers which construct a sequence of approximations $\mathbf{u}^{(k)}$ in the k-dimensional subspace:

$$\mathcal{K}_k = \text{span}\{\mathbf{r}, \mathbf{A}\mathbf{r}, \mathbf{A}^2\mathbf{r}, \ldots, \mathbf{A}^{k-1}\mathbf{r}\} \subset \mathbb{R}^n \tag{10}$$

where $\mathbf{r} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{u}}^{(0)}$ is the initial residual. The simplest Krylov subspace method for a symmetric positive-definite matrix $\mathbf{A}$ is the Conjugate Gradient method (CG), first introduced by Hestenes and Stiefel (1952). In each step, the approximate solution $\tilde{\mathbf{u}}^{(k)}$ is updated by adding the search direction $\mathbf{d}^{(k)}$ scaled by a factor chosen to minimise the energy norm over the space $\mathcal{K}_k$. The search directions are chosen to be $\mathbf{A}$-orthogonal to all previous direction i.e. $\langle \mathbf{d}^{(k)}, \mathbf{A}\mathbf{d}^{(k')} \rangle = 0$ for $k' < k$. The method iterates until the residual norm (or "energy") $\|\mathbf{r}^{(k)}\|$ reduces below a user defined tolerance. Importantly, the convergence rate of CG depends on the spectral properties of the matrix $\mathbf{A}$, see e.g. [17]. In particular, it can be bounded proportionally to the square root of the condition number κ, defined as the ratio between its largest and smallest eigenvalue. A large value, as usually seen in composites, indicates that the system $\mathbf{A}\tilde{\mathbf{u}} = \mathbf{b}$ is ill-conditioned. This means that $\mathbf{u}$ is very sensitive to small changes in $\mathbf{b}$. For such cases, iterative solvers converge very slowly or even not at all, particularly when the problem size increases. A remedy is to *precondition* the system, that is to develop an operation $\mathbf{M}^{-1}$ which is computationally cheap to construct and apply (in parallel) such that $\mathbf{M}^{-1}\mathbf{A}\tilde{\mathbf{u}} = \mathbf{M}^{-1}\mathbf{b}$ is better conditioned and CG solvers converge quickly.

In `dune-composites` our main preconditioner is a two level additive Schwarz method. To construct this method we partition our domain Ω into a set of non-overlapping subdomains Ω_j' for $j = 1$ to N resolved by $\mathcal{T}_h$, as shown in Fig. 2 (left). Each subdomain Ω_j' is extended by O-layers of elements to give the overlapping subdomains Ω_j, Fig. 2 (middle). For each subdomain $1 \leqslant j \leqslant N$, we denote the restriction of V_h to Ω_j by $V_h(\Omega_j)$, whilst the space of FE functions with support contained in Ω_j is called $V_{h,0}(\Omega_j)$.

Remark. In `dune-composites` the user can define the initial non-overlapping decomposition (or a default is used), the overlapping process is handled by DUNE's parallel structured grid class `Dune::YaspGrid` [12].

Any function $v \in V_{h,0}(\Omega_j)$ is mapped onto V_h by the prolongation operator $R_j^T : V_{h,0}(\Omega_j) \to V_h$, which extends v by zeros, so that

$$R_j^T v(\mathbf{x}) = \begin{cases} v(\mathbf{x}), & \mathbf{x} \in \Omega_j \\ 0, & \mathbf{x} \in \Omega \backslash \Omega_j. \end{cases}$$

We therefore note that the restriction operator $R_j : V_h \to V_{h,0}(\Omega_j)$. In matrix form the restriction and prolongation operators R_j and R_j^T, are denoted $\mathbf{R}_j$ and $\mathbf{R}_j^T$ respectively. This allows us to define the subdomain stiffness matrices restricted to $V_{h,0}(\Omega_j)$ as $\mathbf{A}_j := \mathbf{R}_j\mathbf{A}\mathbf{R}_j^T$ for $j = 1, \ldots, N$. In practice, we do not compute $\mathbf{A}_j$ from $\mathbf{A}$ via this double matrix product. Instead, we can equivalently assemble $\mathbf{A}_j$ directly from the bilinear form (8) on Ω_j with homogeneous Dirichlet boundary conditions on all artificial interior subdomain boundaries, i.e all points $\mathbf{x} \in \partial\Omega_j$ that satisfy $\mathbf{x} \in \Omega_{j'}$ for some other $j' \neq j$.

The 1-level Additive Schwarz method can then be defined as a preconditioner of (9) via the operator

$$\mathbf{M}_{AS,1}^{-1} = \sum_{j=1}^{N} \mathbf{R}_j^T \mathbf{A}_j^{-1} \mathbf{R}_j \tag{11}$$

Here, in the subscript, the *AS* denotes additive Schwarz and the 1 denotes a one-level method. In this case the preconditioner $\mathbf{M}_{AS,1}^{-1}$ approximates the inverse operator $\mathbf{A}^{-1}$ by a sum of local solves on overlapping subdomains, with homogeneous Dirichlet boundary conditions on interior boundaries. We will see in the numerical examples to follow that, for large problems, a single-level method is not sufficient, causing stagnation (high iteration counts) of the iterative solver. This stagnation of the iterative solver is caused by a few very small eigenvalues in the spectrum of the preconditioned problem. They are due to the lack of a global exchange of information in the preconditioner in the single-level method. A classical remedy is the introduction of a coarse grid problem that couples all subdomains at the second level [10]. To define our coarse problem, we introduce a coarse space $V_H \subset V_h$ (which we define below). We denote the restriction from the fine to the coarse space by the operator $R_H : V_h \to V_H$, with matrix representation $\mathbf{R}_H$. The two-level additive Schwarz preconditioner (in matrix form) is given by

$$\mathbf{M}_{AS,2}^{-1} = \mathbf{R}_H^T \mathbf{A}_H^{-1} \mathbf{R}_\mathbf{H} + \mathbf{M}_{AS,1}^{-1} \quad \text{where} \quad \mathbf{A}_H = \mathbf{R}_\mathbf{H}\mathbf{A}\mathbf{R}_\mathbf{H}^T. \tag{12}$$

Two natural questions arise:

- What is a good choice of coarse space V_H for composite applications?
- How do we construct $\mathbf{A}_H$ efficiently on a distributed memory computer without assembling $\mathbf{A}$ directly?

3.2. A robust coarse space via generalised eigenproblems in the overlaps (GENEO)

The ideal coarse space would capture the global low energy modes of $\mathbf{A}$ that jeopardise the performance of Krylov solvers. Specifically, in the two-level additive Schwarz setting, the modes not captured by the local solves are of interest. Yet, to compute those low-energy modes explicitly would be more expensive than inverting $\mathbf{A}$ itself. Instead, the global low-energy modes can be approximated by stitching together local (optimal) approximations. These local approximations are solutions of specific GENERALISED EIGENPROBLEMS IN THE OVERLAPS, hence named GENEO,

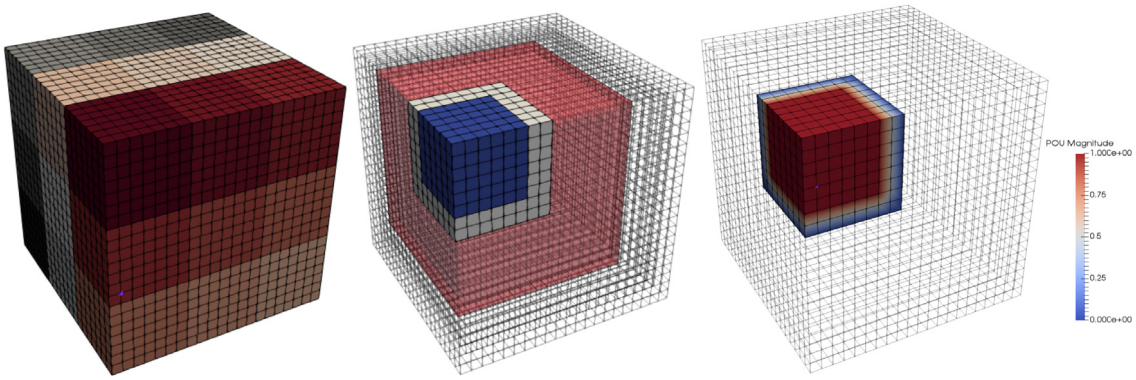

Fig. 2. (Left) Domain Ω partitioned into non-overlapping subdomains Ω_j' where colouring differentiates independent subdomains. (Middle) Shows overlapping subdomain Ω_j with a single layer of overlap ($O = 1$). Overlap region Ω_j° is shown in white. Transparent red regions show cells of the grid which belong to 'nearest neighbour' processors. (Right) Shows partition of unity (PoU) operator Ξ_j on a single processor, defined as in 3.2.

defined below. Importantly the local eigenproblems are independent and can trivially be computed in parallel. The robustness of GenEO has been proven for isotropic elasticity problems, Spillane et al. [11], and numerically verified by the authors for anisotropic variants [6].

The construction of the GenEO coarse space has two key steps: the definition of the generalised eigenproblems on the subdomains and the stitching together of the resulting local eigenmodes from each subdomain to form a global basis. This stitching process by means of *partition of unity* (PoU) operators is also incorporated in the local eigenproblems, therefore we construct the PoU operators first.

Definition 3.1 (*Subdomain Overlap*)**.** For each subdomain Ω_j, the overlap region is defined by the set

$$\Omega_j^\circ := \{\mathbf{x} \in \Omega_j : \exists j' \neq j \text{ s.t. } \mathbf{x} \in \Omega_{j'}\},$$

i.e. the subset of Ω_j which belongs to at least one other subdomain.

Definition 3.2 (*Partition of Unity*)**.** The family of operators $\Xi_j : V_h(\Omega_j) \to V_{h,0}(\Omega_j)$, $j = 1, \ldots, N$, defines a Partition of Unity if

$$\sum_{j=1}^{N} R_j^T \Xi_j(v|_{\Omega_j}) = v, \quad \forall v \in V_h.$$

Since $R_{j'}^T \Xi_{j'}(v|_{\Omega_{j'}}) = 0$ on $\Omega_j \backslash \Omega_j^o$ for all $j' \neq j$, it follows from this definition that restricted to $\Omega_j \backslash \Omega_j^o$ each Ξ_j has to be the identity operator. In the overlaps, the choice of Ξ_j is not unique. The simplest approach is to define $\Xi_j(v)$ such that each coefficient of the FE function v is scaled by the number of subdomains the corresponding degree of freedom belongs to (see [11] for details). However, we also provide a smoother PoU as defined by Sarkis [18] in our implementation, but observe little difference in the performance of the overall solver (at most one iteration); we therefore keep the presentation here as simple as possible.

Given this set of local PoU operators $\Xi_j(\cdot)$, we can construct any global FE function $v_h \in V_h$ from local functions $v_h^{(j)} \in V_h(\Omega)$ as follows:

$$v_h = \sum_{j=1}^{N} R_j^T \Xi_j(v_h^{(j)}). \tag{13}$$

In particular, we can define the local generalised eigenproblems that (once collected from each subdomain) provide the basis of the GenEO coarse space. The following definition can be rigorously motivated from theoretical considerations and we refer again to [11]. For each subdomain Ω_j, $j = 1, \ldots, N$, we define the generalised eigenproblem: Find $(\lambda, p) \in \mathbb{R}^+ \times V_h(\Omega_j)$ such that

$$a_{\Omega_j}(p, v) = \lambda a_{\Omega_j^o}(\Xi_j(p), \Xi_j(v)), \qquad \forall v \in V_h(\Omega_j), \tag{14}$$

where, for any $D \subset \Omega$, the bilinear form a_D is defined like a in (7) with the integral restricted to D.

Definition 3.3 (*GenEO Coarse Space*)**.** For each subdomain Ω_k let p_k^j be the eigenfunctions from (14) with associated eigenvalues λ_k^j in ascending order. Then, for some choice of $m_j \in \mathbb{N}$, the GenEO coarse space is defined as

$$V_H := \text{span}\{R_j^T \Xi_j(p_k^j) \ : \ k = 1, \ldots, m_j,\ j = 1, \ldots, N\}.$$

The only parameter that remains to be chosen is the number of eigenmodes m_j to be included in each subdomain Ω_j. In order to ensure robustness and scalability of the solver, the condition number of the preconditioned system needs to be bounded from above, independent of N, h and of the material properties. It has been shown in [11] that

$$\kappa(\mathbf{M}_{\mathbf{AS},2}^{-1}\mathbf{A}) \leqslant C \max_{1 \leqslant j \leqslant N} \left(1 + \frac{1}{\lambda_{m_j+1}^j}\right). \tag{15}$$

where $C > 0$ is a constant depending only on the geometry of the subdomains and where $\lambda_{m_j+1}^j$ is the lowest eigenvalue whose eigenfunction is not added to the coarse space on Ω_j. Thus, the desired robustness can be achieved by including all eigenfunctions in the coarse space whose eigenvalues are below an a priori chosen threshold. A particular threshold that turns out to provide an effective black-box choice for m_j and also depends only on the geometry of the subdomain partition is to include all eigenfunctions with $\lambda_k^j \leq \text{diam}(\Omega_j)/\text{width}(\Omega_j^o)$.[1] This simple threshold can be scaled by a constant factor, thus also scaling the condition bound of the preconditioned system by the same factor. As the number of iterations of the Krylov solver depends directly on the condition number, this allows us to balance the time spent in the iterative solver with the time spent on setting up the preconditioner.

The number of eigenfunctions that are used in the coarse space is problem-specific, but it turns out that for strongly structured coefficient distributions only a small number is typically sufficient. We will see in Section 5 that the calculation of these local eigenmodes is not prohibitively expensive, while yielding excellent condition numbers and, due to the independence of the individual eigenproblems, parallel scalability.

[1] For any $D \subset \Omega$, diam(D) and width(D) refer to the radius of the largest circumscribed and inscribed circle, respectively.

3.3. Implementation of GenEO on a high performance computer

The two-level additive Schwarz preconditioner with GenEO coarse space is implemented within a collection of header files, which are located in `dune-pdelab` from `releases/2.6` and in the folder `solvers/geneo/` in prior releases. Here we describe our implementation. We are aware of only one other high performance implementation of GenEO, which can be found in the package `HPDDM` for which details are provided in *Jolivet et al.* [19,20].

It is a main goal of such an implementation to fully exploit the excellent parallel scalability promised by the method's construction and theoretical properties. Therefore, each process j will be assigned to subdomain Ω_j and only store relevant fine-level operators and functions in the form of local restrictions to Ω_j. Further, per-subdomain stiffness matrix and eigenproblem solves will be run in parallel on the respective processes. Only scalable nearest-neighbour communication is needed, with the exception of setting up the coarse matrix, which consequently requires particular care.

3.3.1. Partition of Unity (PoU) operator

The partition of unity operator as defined by Definition 3.2 is stored locally on each processor (see Fig. 2). In practice, the partition of unity operator Ξ_j is represented as a diagonal matrix $\mathbf{X}^{(j)}$. In the simplest case, each diagonal entry of $\mathbf{X}^{(j)}$ is set to one divided by the number of subdomains containing the associated degree of freedom, except for the subdomain boundary where entries are set to zero. Therefore, if $\mathbf{v}^{(j)}$ is a vector containing all nodal degrees of freedom of the FE function $v_h \in V_h(\Omega_j)$ in subdomain Ω_j, the operation $\mathbf{X}^{(j)}\mathbf{v}^{(j)}$ automatically maps v_h into $V_{h,0}(\Omega_j)$. Such a PoU can be generated using existing parallel data structures in Dune by adding a vector of ones and by enforcing both global and subdomain boundary conditions before and after communication. The implementation of the PoU operators is within the header file `geneo/partitionofunity.hh` under the function `standardPartitionofUnity(...)`. As the choice of partition of unity operator is not unique, we also provide the PoU in [18], which is implemented in the same header file under the function `sarkisPartitionofUnity(...)`. This gives a 'smoother' PoU operator, which is however restricted to equally distributed subdomain sizes. Under testing, we noted no significant difference in the performance of the preconditioner when changing between the two different PoU operators.

3.3.2. Subdomain eigenproblems

The local generalised eigenvalue problems (14) can be rewritten in matrix form as follows: Find eigenpairs $(\lambda_i^{(j)}, \mathbf{p}_i^j)$ with eigenvalues in ascending order and $\|\mathbf{p}_i^j\| = 1$ such that

$$\mathbf{A}_{\Omega_j}\mathbf{p}_i^j = \lambda_i^j\left(\mathbf{X}^{(j)}\mathbf{A}_{\Omega_j^\circ}\mathbf{X}^{(j)}\right)\mathbf{p}_i^j, \quad \text{for} \quad j = 1,\dots,N \quad \text{and} \quad i = 1,\dots,m_j \tag{16}$$

where $\mathbf{A}_{\Omega_j}$ and $\mathbf{A}_{\Omega_j^\circ}$ denote the stiffness matrices corresponding to the bilinear forms $a_{\Omega_j}(\cdot,\cdot)$ and $a_{\Omega_j^\circ}(\cdot,\cdot)$ on $V_h(\Omega_j)$ and $V_h(\Omega_j^o)$, respectively. They are solved using `ARPACK++` [21].

A customised wrapper has been developed to convert Dune data structures into a suitable format for `ARPACK++` [21]. In order to regularise the problem, we employ `ARPACK++`'s *shift and invert spectral transformation mode* and, since we are interested in the smallest m_j eigenvalues, we choose a small shift factor. The global coarse basis vectors $\Phi_1,\dots,\Phi_{N_H}$ are obtained from the local eigenvectors by applying the PoU operator, i.e., $\Phi_{i(j,k)} := \mathbf{X}^{(j)}\mathbf{p}_k^j$, and padding the rest of the global vector (outside Ω_j) with zeros. Here, $(j,k) \mapsto i(j,k)$ is a one-to-one mapping from the local numbering of the eigenvectors on Ω_j to a global numbering, with $1 \le i(j,k) \le N_H = \sum_{j=1}^{N} m_j$.

3.3.3. Coarse space assembly

The parallel assembly of the coarse system $\mathbf{A_H} = \mathbf{R}_H\mathbf{A}\mathbf{R_H}^T$ is not trivial in practice since process j only has local access to rows and columns of $\mathbf{A}$ associated to degrees of freedom on subdomain Ω_j. We denote this submatrix $\tilde{\mathbf{A}}_j$. Note that $\tilde{\mathbf{A}}_j$ differs from the matrix $\mathbf{A}_j$ in (11) in that it does not incorporate Dirichlet conditions on interior subdomain boundaries.

Furthermore the coarse space prolongation matrix $\mathbf{R}_H^T$ is only available in a distributed manner. Each basis vector Φ_i, $i \in \{1,\dots,N_H\}$, is available only on process $j(i)$, where the unique $j(i) \in \{1,\dots,N\}$ denotes the index of the subdomain $\Omega_{j(i)}$ associated with the eigenproblem (16) corresponding to Φ_i. However, due to the local support of the basis functions, one can break down the global matrix product into local products

$$(\mathbf{A}_H)_{i,\ell} = (\mathbf{R}_H\mathbf{A}\mathbf{R}_H^T)_{i,\ell} = \left(\Phi_i^T\tilde{\mathbf{A}}_{j(i)}\right)\Phi_\ell, \quad \text{for} \quad i,\ell = 1,\dots,N_H, \tag{17}$$

with a slight abuse of notation, denoting the local parts of the global vectors Φ_i and Φ_ℓ restricted to $\Omega_{j(i)}$ again by Φ_i and Φ_ℓ. In the implementation, the matrix vector product in the bracket is local whereas the scalar product requires communication of (parts of) vector Φ_ℓ from processor $j(\ell)$ to processor $j(i)$. This avoids having to communicate the local matrices $\tilde{\mathbf{A}}_j$.

We note that the locality of the basis functions implies $\Phi_i\tilde{\mathbf{A}}_{j(\ell)}\Phi_\ell = 0$ whenever $\Omega_{j(i)} \cap \Omega_{j(\ell)} = \emptyset$. Therefore, the parallel assembly of $\mathbf{A_H}$ requires only communication between processes assigned to overlapping subdomains. This locality of communication, which is demonstrated in Fig. 2 (middle), can be exploited to set up $\mathbf{A_H}$ as a banded sparse matrix. Its parallel communication is implemented within `geneo/multicommdatahandle.hh`, allowing to pass basis functions between all processes at the same time and therefore make best use of available bandwidth. Combining sparsity and efficient communication, linear complexity in basis size can be achieved for this step.

Since the number of coarse degrees of freedom per process is too small, i.e., m_j on process j, after assembly we distribute the resulting global coarse matrix $\mathbf{A}_H$ to all processors and duplicate the coarse solve on all processors to avoid fine-grained communication. The communication of $\mathbf{A}_H$ is achieved directly with `MPI` calls, as the `dune-pdelab` communication infrastructure is not designed for such global operations.

In case of the restriction of a distributed vector $\mathbf{v_h}$ representing a function $v_h \in V_h$, it follows that

$$(\mathbf{R}_H\mathbf{v_h})_i = \Phi_i^T\mathbf{v_h}. \tag{18}$$

So, each row i can be computed by the process associated with Φ_i, and the rows can be exchanged among all processes via `MPI_Allgatherv`. Again, the communication effort increases with the dimension of V_H. On the other hand, the prolongation $\mathbf{R}_H^T\mathbf{v_H}$ of a vector $\mathbf{v_H}$ that is globally available on all processors, representing a $v_H \in V_H$, consists only of local contributions and hence can be computed in parallel without communication. Since the result lies in $V_h(\Omega)$, the regular PDELab communication patterns can be used. This only involves communication between adjacent subdomains, making this a highly scalable process.

Each process executes its associated subdomain solve as well as the coarse space solve (redundantly). Where possible we use a sparse direct solver (`UMFPack`) [22]. For very large problems and a large number of parallel processors the coarse space becomes too large, and must itself be solved with a preconditioned iterative solver; in that case we use by default preconditioned CG with the BoomerAMG [9] preconditioner. It is important to note that in such cases since the coarse solve is inexact, the preconditioner for the (overall) Krylov method is now instationary. It is therefore necessary to switch from a standard preconditioned CG to a flexible Krylov solver. In our case we use Flexible GMRES as provided by `dune-istl` [8].

4. Using and extending dune-composites

In this section we provide an overview of the dune-composites code, sufficient to enable other scientists to leverage the framework. The code is structured so that little additional knowledge of Dune and/or C++ is required to apply the code within the existing functionality. Fig. 3 shows the code structure. A user can extend any of the functionality e.g. implement a new solver, define a more complex nonlinear problem (e.g. cohesive zone) or introduce new types of elements.

4.1. Defining a Model

At the highest level an analysis is defined by a user-defined baseStructuredGridModel class shown in green in Fig. 3. This class defines all the key variables, functions and classes which describe the analysis, as well as storing any variables that are required for postprocessing or any later calculations. A base model class is provided, which can be inherited by each example. This provides default variables and functions, so that the user need only overwrite those functions which deviate from this base class. The Model class also defines the general loading on the structure and the boundary conditions, these include Dirichlet and Neumann conditions, but also thermal loading and multi-point constraints. Periodic boundary conditions are defined within the grid data structure, using Model::LayerCake(). Examples of user defined Model classes for a series of applications are provided in Section 5 which follows.

4.2. Internals of dune-composites

The functions provided in the subfolder /Setup provide support for the geometric setup of the grid geometry, material properties and boundary conditions. This includes the composite layering (or stacking sequence), the structural geometric shape of the component and in some cases adding a perturbation to the geometry to form a defect (see for example Section 5.1). Because of uniform layering and planar anisotropy in composite laminates, our first version has focused on structured, overlapping grid implementations using Dune::YaspGrid and Dune::GeometryGrid. The Dune::GeometryGrid functionality allows us to apply any continuous transformation to the basic Cartesian mesh provided by Dune::YaspGrid. Development of unstructured grid implementation using Dune::UGGrid [23] are described in the future work Section 7.

The folder /Driver contains the key functions and classes which relate to the FE calculations beyond what is available directly from Dune::PDELab. In particular these include all element calculations, the definitions of new FEs, solvers and preconditioners. The functions and classes are split between three folders:

- /localOperators define the weak form of the equations to be solved on an element, along with any support functions. In our case for anisotropic linear elasticity equations we define the local operator linearelasticity.hh which returns the element stiffness matrix, load vector and residual as defined by Eq. (7).
- /FEM defines specialist finite elements beyond those defined by Dune::PDELab. In our case, these are the family of serendipity elements [24]. The use of these elements are then defined explicitly in the Driver class, where the FE space is set up on the grid.
- /Solvers defines specialist solvers and preconditioners beyond those defined in Dune::PDELab::istl [8]. The Driver uses the solver as defined by the Model class, defined by the templated class function Model::solve(). By default, as defined by baseStructuredGridModel, for parallel calculations we use a CG Krylov solver, preconditioned with either a one or two level additive Schwarz method. Two-level methods use GenEO as the coarse space as long as ARPACK++ is available. If not, the coarse space consists of only the zero energy modes [10]. In sequential mode, in particular for the coarse and the local solves, a sparse direct solver UMFPack [22] and the iterative CG preconditioned with AMG as provided by Dune::PDELab::ISTLBackend_SEQ_CG_AMG_SSOR, are also available. In /Solvers/hypre we provide a wrapper to the external parallel solvers provided by hypre [25], including boomerAMG [9].

The analysis is defined via a Driver. This is a class which provides the complete solution procedure, from the setup of the grid, the definition of a finite element space, assembly of the global stiffness matrix and load vector, and finally, calling the solver, followed by the PostProcessing routines, as defined by the Model class. In this version, we provide two driver classes for the examples considered in Section 5: /FEMDriver/linearStatic.hh and /FEMDriver/ThermalStatic.hh.

The folder /PostProcessing contains various classes and functions for the postprocessing of results. By default, after the solution has been calculated the driver initiates certain postProcessing steps, in particular the calculation of stresses, the creation of the necessary data for any plots and finally the calculation of any further quantities of interest. All of these routines can be modified by the user.

5. dune-composites - examples

In this section, we introduce and demonstrate the functionalities of dune-composites using a series of examples of increasing complexity. The examples are intended as a starting point for researchers implementing their own studies, whilst also demonstrating the significant computational gains dune-composites is able to achieve in comparison to the commercial package Abaqus [7].

To simplify the definition of the examples in the following we assume that all cases use the same material properties. The orthotropic fibrous layers are assumed to be of thickness $t_p = 0.23$ mm, with elastic moduli

$$E_{11} = 162 \text{ GPa}, \quad E_{22} = E_{33} = 10 \text{ GPa}, \quad G_{12} = G_{13} = 5.2 \text{ GPa}, \quad G_{23} = 3.5 \text{ GPa}, \quad \nu_{12} = \nu_{13} = 0.35 \quad \text{and} \quad \nu_{23} = 0.5, \tag{19}$$

whereas the isotropic resin rich layers are assumed to be $t_i = 0.02$ mm thick, with isotropic properties $E = 10$ GPa and $\nu = 0.35$. These particular values are taken from a previous study by the authors [26].

5.1. Example 1: A flat composite plate

For the first two examples we consider a flat composite plate $[0, 100 \text{ mm}] \times [0, 20 \text{ mm}]$ under various loading conditions. The laminate is made up of 12 identical composite layers arranged in the following composite stacking sequence

$$[\mp 45^\circ/0^\circ/90^\circ/\pm 45^\circ/\mp 45^\circ/90^\circ/0^\circ/\pm 45^\circ]. \tag{20}$$

The composite layers are separated by 11 isotropic resin interface layers, giving a total thickness of $T = 2.98$ mm. In

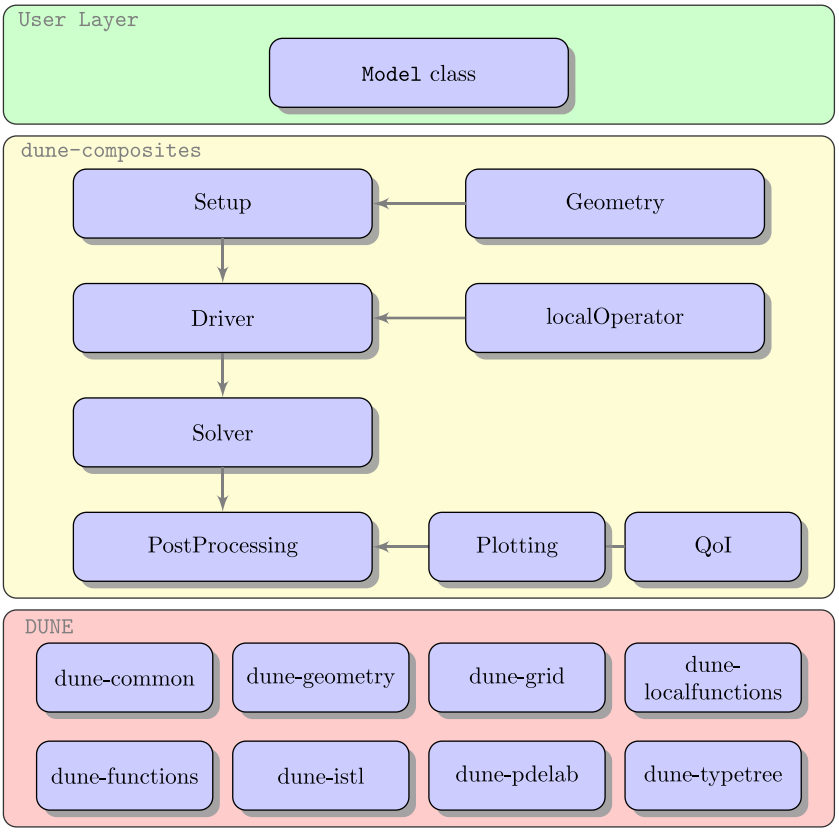

Fig. 3. Code structure.

each of the examples, we discretise the geometry with quadratic, 20-node serendipity elements (with full Gaussian integration). For the base mesh, which will be refined, we take 20 elements in the x-direction, 5 in the y-direction and through thickness 2 per composite layer and 1 per interface layer. This gives a total number of 3,500 elements, with 13,608 degrees of freedom.

The geometry, the stacking sequence and the initial finite element mesh are introduced into a model by overwriting the base class function `Model::LayerCake()`, with the following user defined function.

Here the geometry and grid are defined by a file ‘‘`stackingSequences/example1.csv`’’.

In these first two examples, we demonstrate a very simple setup and run on a single processor as well as on a few processors. We consider a cantilever beam with a uniform pressure of 0.01 MPa applied to the top face and the following boundary conditions:

$$u_1 = u_2 = u_3 = 0 \quad \text{at} \quad x = 0 \quad \text{and} \quad \sigma_{33} \cdot \mathbf{n}_3 = -q \quad \text{at} \quad z = T. \tag{21}$$

All other boundary conditions are assumed to be homogeneous Neumann conditions, i.e

$$\sigma_{ij} \cdot n_j = 0. \tag{22}$$

Boundary conditions are implemented by overwriting the two class functions `Model::isDirichlet()` and `Model::evaluateNeumann()` as follows

```
bool inline isDirichlet(FieldVec& x, const int i){
   return (x[0] < 1e-6);
}
```

Here i refers to the direction being restricted, see (2).

```
inline void evaluateNeumann(const FieldVec& x, FieldVec& h,
                            const FieldVec& normal) const{
    h = 0.0; // initialise to zero
    double T = R[0].L[2]; // Thickness
    if (x[2] > T - 1e-6){
        h[2] += q;
    }
}
```

We note that `Model::evaluateDirichlet()` need not be overwritten since by default it returns homogeneous boundary conditions (i.e. $\mathbf{u}(\mathbf{x}) = \mathbf{0}$) for all those points x marked as Dirichlet boundary conditions by `isDirichlet()`. Furthermore, by default loading under the weight of the structure is included by providing density as an input parameter. We do not wish to include it in this example and therefore we must also overwrite the function `Model::evaluateWeight()`

```
inline void evaluateWeight(FieldVec& f, int id) const{
    f = 0;
}
```

Here f is the output density in a given element *id*.

5.1.1. Example 1a: A flat composite plate — getting started

Our first study computes the maximum vertical deflection of the cantilever beam as our quantity of interest

$$Q(\mathbf{u}) = \max_{\mathbf{x} \in \Omega} u_3(\mathbf{x}).$$

This is done by providing the following user defined function

This function loops over the solution at each vertex `native(u)[i]`, and records the maximum vertical displacement `native(u)[i][2]`. Since for a parallel run, this maximum is only the maximum on the local subdomain associated with a

```
void inline LayerCake(){
   std::string example01a_Geometry = "stackingSequences/example1a.csv";
   LayerCakeFromFile(example01a_Geometry);
   GeometryBuilder();
}
```

```
template<class GO, class V, class GFS, class C>
void inline postprocess(const GO& go, V& u, const GFS& gfs, const C& cg){
    using Dune::PDELab::Backend::native;
    double local_u3_max = 0.0;
    for (int i = 0; i < native(u).size(); i++){ // Loop over each vertex
        double u3 = std::abs(native(u)[i][2]);
        if (local_u3_max < u3) { local_u3_max = u3; }
    }
    MPI_Allreduce(&local_u3_max, &QoI, 1,
                  MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
}
```

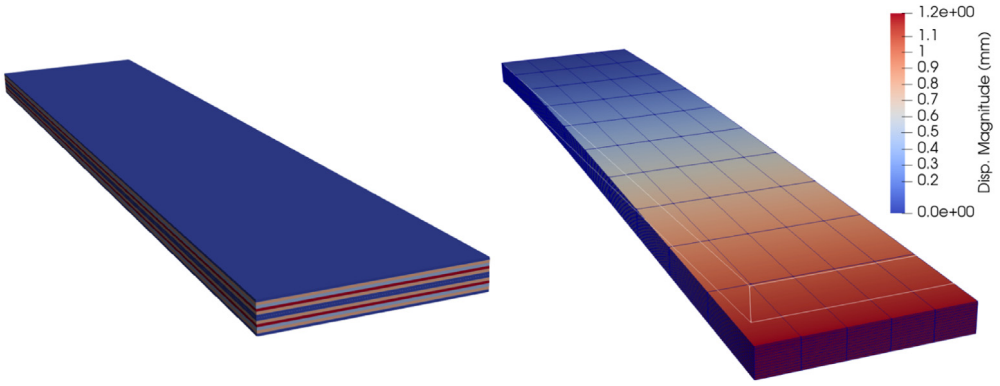

Fig. 4. Visualisation of results for `Example01a` using ParaView (left) Visual output of laminate and stacking sequence using `plotProperties()` function (right) Visualisation of solution, in deformed coordinates (scalar factor of displacement is 4).

given processor, the final command `MPI_Allreduce()` finds the maximum vertical displacement over all subdomains (processors). The final result is stored in `QoI`, a member of the `baseStructuredGridModel` class.

We use the default sequential and parallel solvers. On a single processor (e.g. with the call `./Example1a`) the sparse direct solver `UMFPack` [22] is used. Otherwise, if more than one processor is used (e.g. with the call `mpirun -np 8 ./Example1a`) the equations will be solved with CG, preconditioned with a one-level additive Schwarz preconditioner, as defined by (11) using `UMFPack` as the local solver on each subdomain.

As output, the quantity of interest is printed to the screen (`Maximum Vertical Displacement inExample01a = 1.23992mm`). Furthermore, the data for plots of the laminate stacking sequence, the solution (deformation) and the stress field are generated and provided in three files named `Example01a_xxx.vtu`. The stacking sequence and solution are shown in Fig. 4.

5.1.2. Example 1b: A flat composite plate — testing preconditioners (up to 32 cores)

In `Example01b`, we test our new preconditioner GenEO on up to 32 processors. In this example we also demonstrate the inclusion of a failure criterion. To do this we change the quantity of interest to be the pressure $q = q^\star$ in the boundary condition (21) at which the laminate fails according to the Camanho criterion [27], defined by the functional

$$\mathcal{F}(\sigma(\mathbf{x})) = \sqrt{\left(\frac{\sigma_{33}^{+}}{s_{33}}\right)^2 + \left(\frac{\sigma_{13}}{s_{13}}\right)^2 + \left(\frac{\sigma_{23}}{s_{23}}\right)^2}. \tag{23}$$

We apply the Camanho criterion only in the resin-rich interface layers and we say that failure occurs at a load $q^\star$ if $\max_{\mathbf{x}\in\Omega^{\text{Inter}}} \mathcal{F}(\sigma(\mathbf{x})) = 1$. However, since the problem is linear it suffices to solve only one problem with an arbitrary load q. The failure load is then given by $q^\star := q/\max_{\mathbf{x}\in\Omega^{\text{Inter}}} \mathcal{F}(\sigma(\mathbf{x}))$. Expression (23) is implemented in the file `PostProcessing/FailureCriterion/Camanho.hh` within the code. Within `linearStaticDriver`, by default, the stress field (per element) is stored within the container `stress_mech` (a 6×1 vector). To compute $q^\star$, the Camanho functional is first calculated in each element. The maximum is then found by once again overwriting the class function `Model::postprocess`. The material allowables, $s_{33} = 61$ MPa, $S_{13} = 97$ MPa and $s_{23} = 94$ MPa, in Eq. (23) are stored in a `std::vector<double>` p.

Different failure criteria can be implemented by defining other user-defined functionals of the stress tensor, similar to `Dune::Composites::Camanho()`. In this simple test, we note that failure initiates due to high through thickness stresses in the interface between layers (σ_{13} and σ_{23}) as the laminate bends. For further engineering discussion of the failure of composites under the Camanho criterion we point the reader to the original paper [27] and to [6,26].

```
template<....>
void inline postprocess(...){
    //material allowables in MPa
    const std::vector<double> p = {61., 97., 94.};
    double Fm = 0.0;
    for (int i = 0; i < stress_mech.size(); i++){
        double F = Camanho(stress_mech[i], elemIndx2PG[i], p);
        if (Fm < F) { Fm = F; }
    }
    double Fm_all;
    MPI_Allreduce(&Fm, &Fm_all, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
    Q = pressure / Fm_all; // Failure load
}
```

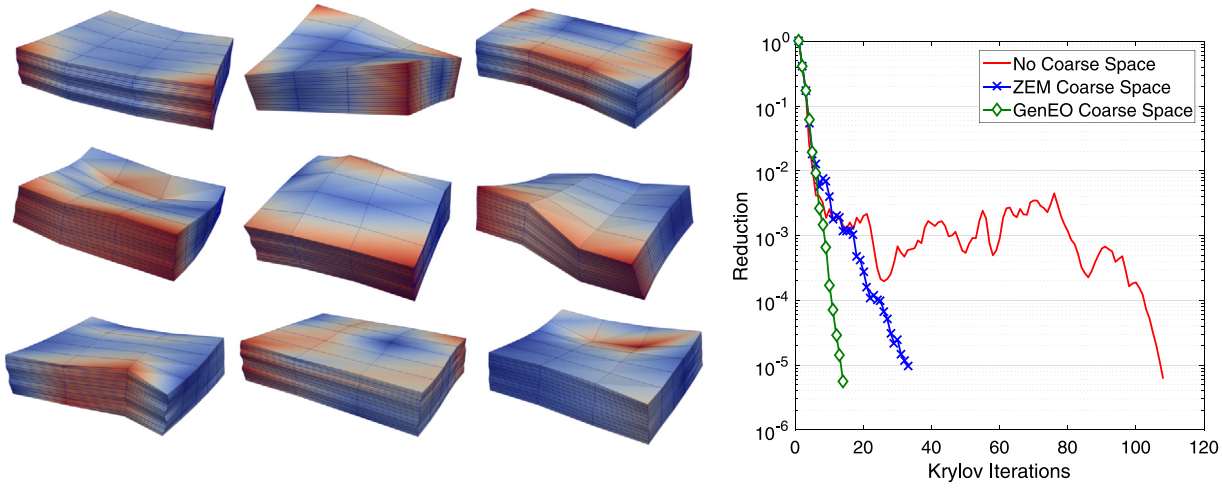

Fig. 5. (Left) The eigenvectors corresponding to the first nine non-zero eigenvalues on a subdomain with no global Dirichlet boundary. (Right) The reduction of the residual against CG iterations for `Example01b` using no coarse space, only zero energy modes (ZEM) and the full GenEO coarse space.

We use this test example to demonstrate the influence of the GenEO coarse space on the parallel iterative solver. For the first experiment we use 16 processors. Fig. 5 (left) shows the first nine non-zero energy modes of a subdomain with no global Dirichlet boundary. Linear combinations of these functions together with the zero energy modes (six rigid body translations and rotations) provide a good low dimensional representation of the system on that subdomain consisting of those modes most easily energetically excited. Fig. 5 (right) shows the influence of the coarse space on the number of iterations for the preconditioned Krylov Solver (pCG), comparing no coarse space (one-level additive Schwarz), only the zero energy modes (ZEM) and the GenEO coarse space. The need for a coarse space is clear; with no coarse space, even in this simple test case we observe the well-documented stagnation phenomenon for iterative solvers [10] between Iteration $10-100$. With a coarse space (ZEM or GenEO), the convergence shows no stagnation and it is much faster — close to optimal with GenEO.

Next we want to study the robustness of GenEO as a function of the number of subdomains in comparison to one-level additive Schwarz (AS) and ZEM. We consider a fixed size problem and increase the number of subdomains. We note that the tests can be run with

`mpirun -np 16 ./Example01b` or `mpirun -np 16 ./Example01bBoomerAMG`

In each case we record the condition number, the dimension of the coarse space $\dim(V_H)$ (if applicable) and the number of CG iterations to achieve a residual reduction of 10^{-5}. The results are summarised in Table 1. We see that the iteration counts (and the condition number estimates) increase steadily with the number of subdomains when no coarse space is used. The condition number estimate is still fairly big if only the zero energy modes are used and the iteration counts also increase steadily with the number of subdomains. In contrast, the iterations and the condition number estimates remain constant for the GenEO preconditioner. We also add a comparison with `boomerAMG` [9] for this test problem. `BoomerAMG` provides a large number of parameters to fine-tune. We retained the defaults for most parameters (HMIS coarsening without aggressive refinement levels and a hybrid Gauss–Seidel smoother). We used blocked aggregation with block size 3 as recommended for elasticity problems. A strong threshold of 0.75 was chosen after testing values in the range from 0.4 to 0.9. Due to a lower setup cost with this parameter setting, the `boomerAMG` solver is faster in actual CPU time, but the numbers of iterations – albeit also constant – are more than $10\times$ bigger. For more complex geometries, `boomerAMG` does not perform very well and in our tests it does not scale beyond about 100 cores in composite applications.

5.2. Example 2: Corner unfolding — validation & performance comparison with Abaqus (up to 32 cores)

This example is motivated by the industrial challenge of certifying the corner-bend strength of a wingspar as its corner unfolds due to the internal fuel pressure in an aeroplane wing. We use this example to demonstrate the validity of the results of `dune-composites` by comparing the stresses computed with those given by Abaqus. We also make a cost comparison between the two software packages up to 32 cores.

Table 1
Demonstration of performance of different preconditioners for `Example01b` for fixed problem size (30,000 DOFs) but increasing the number of subdomains: Number of pCG iterations (it), coarse space dimension (dim(V_h)), an estimate of the condition number κ of the preconditioned system matrix $\mathbf{M}_{AS,2}^{-1}\mathbf{A}$.

N	AS		ZEM			GenEO			BoomerAMG	
	it	cond κ	it	cond κ	dim(V_H)	it	cond κ	dim(V_H)	it	Num. levels
4	89	79,735	26	394	12	16	10	78	258	10
8	97	84,023	30	245	42	15	9	126	258	11
16*	107	98,579	36	177	84	16	10	182	257	12
32	158	226,871	42	230	168	16	9	526	263	12

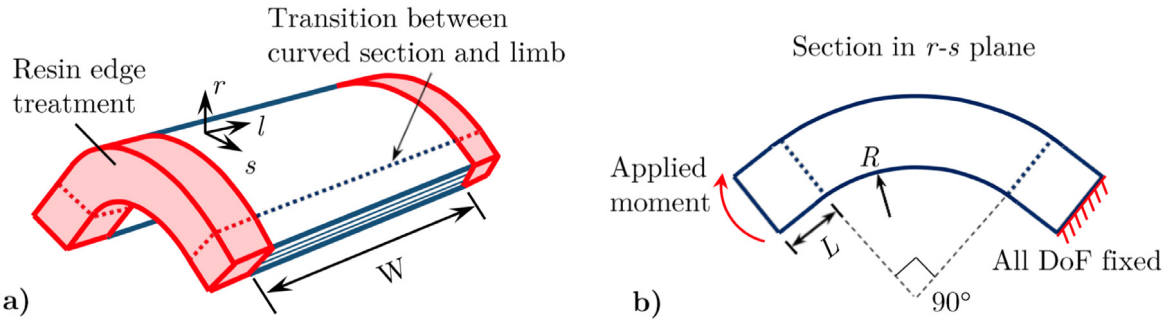

Fig. 6. (Left) Diagram of the corner bend specimen with resin edge treatment. (Right) Cross section of the corner showing the loading conditions.

The model setup is shown in Fig. 6. We consider the curvilinear coordinate system (s, r, ℓ), where s is around the radius, r is outwards (or normal) to the laminate and ℓ runs along the length of the sample. For our particular test, the two limbs of the coupon are of length $L = 3$ mm and border a corner with a radius of $R = 6.6$ mm. The width is taken to be $W = 15$ mm. The 12 plies and the 11 interfaces have the same properties as in `Example01`, but the stacking sequence is slightly different, given by

$$[\mp 45^\circ/90^\circ/0^\circ/\mp 45^\circ/\mp 45^\circ/0^\circ/90^\circ/\pm 45^\circ]\,. \tag{24}$$

Furthermore, we apply a resin treatment of 2 mm to the free edges of the laminate as shown in Fig. 6 (left). The advantages of edge treatment have been shown in [26]. It reduces conservatism in the design of aircraft structures, as well as making the analyses more reliable by eliminating stress singularities at the free edges.

Away from the points of contact, a standard four-point bend test as detailed in ASTM D6415 [28] generates a pure moment on the corner. To simulate such a moment, all degrees of freedom at the boundary of one limb are clamped. At the other end, all nodes are tied with a multipoint constraint where a running moment of 96.8 Nmm/mm is applied. This is achieved by applying an offset load from the mid-plane of the laminate as a Neumann boundary condition, implemented with the user-defined function `evaluateNeumann()`.

The finite element mesh consists of 56 × 56 columns of hexahedral 20-node serendipity elements (element `C3D20R` in ABAQUS, [7]) in its local l and s coordinates. In the r direction, each (fibrous and resin) layer is discretised into 6 elements, leading to an overall number of 432,768 elements. All of the geometry and mesh parameters are defined in `stackingSequenes/example2.csv`. To ensure a sufficient resolution of the strong gradients of the solution at the free edges and at the material discontinuities, the mesh is graded along the width towards both free edges and in the radial direction towards each of the fibre–resin interfaces. Grading is defined by the ratio between largest and smallest elements in the mesh, called the bias ratio and chosen to be 400 between the centre and the edge in the l direction and 10 between the layer centres and interfaces in the r direction. The specification of the geometry and of the mesh grading can be defined in the function `gridTransformation()`.

In Fig. 7 (left & middle), we compare the radius stress (denoted by σ_r) and the through-thickness shear stress (denoted by $\tau_{s\ell}$) recovered from `dune-composites` and Abaqus. We see good agreement between the two codes. There are two small differences: Abaqus uses reduced integration while our example uses full integration and the stresses are not recovered in an identical way from the displacements in the two codes. In Fig. 7 (right) we see the absolute cost and the parallel scalability of the sparse direct solver in Abaqus and the iterative CG solver in `dune-composites` for a fixed total problem size, i.e. a strong scaling test. The red and blue curves show one-level and two level overlapping Schwarz methods respectively, both of which perform better than the sparse direct solver (green) in Abaqus. However, both codes show optimal parallel scalability up to 32 cores. In `dune-composites` the problem is decomposed into 8 subdomains for $1 - 8$ cores, distributed evenly to the available cores. On 16 and 32 cores, each core is passed exactly one subdomain, i.e., the number of subdomains is 16 and 32, respectively. The local problems on each subdomain are solved using the sparse direct solver `UMFPack` [22]. The simulations in Abaqus are with a parallel sparse direct solver, based on a parallel multi-frontal method similar to [29]. Abaqus's iterative solver, which is based on CG preconditioned with `ML` [30] (another black-box AMG preconditioner), does not converge in a reasonable time for this problem. Therefore the computational gains observed here are really the difference between using a direct and a robust iterative solver. Importantly, we note that the parallel sparse direct solver, available in Abaqus does not scale beyond 64 cores [7], making it unsuitable for problems much bigger than that considered here, and reinforcing the need for robust iterative solvers and therefore `dune-composites` as a package.

5.3. Example 3: Large composite structure — parallel efficiency of `dune-composites` (up to 15,360 cores)

The industrially motivated problem described in this section is to assess the strength of a wingbox with a small localised wrinkle defect. Wrinkle defects, which can form during the manufacturing process [31,32], occur at the layer scale. They lead to strong local stress concentrations [6,33], causing premature failure. Naturally, good mesh resolution around the defect is required, leading to

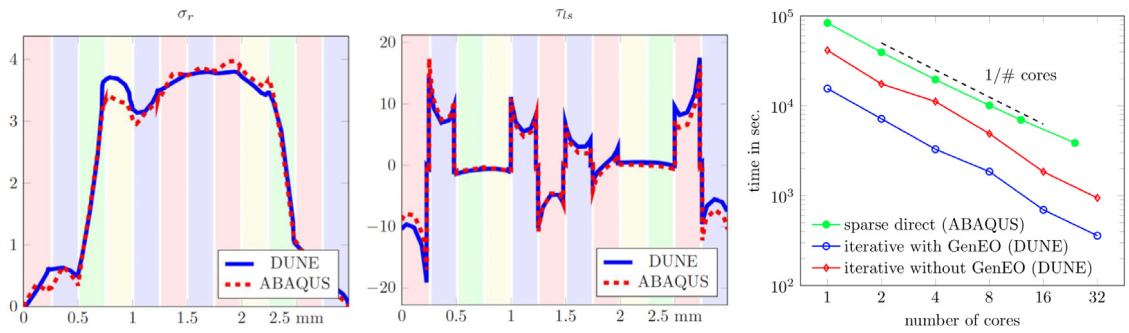

Fig. 7. (Left & Middle) Stresses (in MPa) as functions of the distance r from the outer radius at the apex of the curve, at 2.156 mm from the edge of the resin-edge-treated laminate (`dune-composites`, solid blue; ABAQUS, dotted red). The background colours indicate the stacking sequence: $+45°$ = red, $-45°$ = blue, $90°$ = green, $0°$ = yellow. (Right) Cost comparison between the sparse direct solver implemented in ABAQUS and the iterative preconditioned CG solver in `dune-composites`. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

finite element calculations with very large number of degrees of freedom. We leave the engineering discussion of the results to a future engineering publication, using it instead to demonstrate both weak and strong scalability of `dune-composites` up to 15,360 subdomains. The experiments in this section were performed using the UK national HPC cluster ARCHER, which has 4920 Cray XC30 nodes with two 2.7 GHz, 12-core E5-2697 v2 CPUs each.

For these tests we model a single bay of a wingbox of width $W = 1$ m, height $H = 300$ cm and length $L = 1$ m, as shown by the schematics in Fig. 8 (left). The laminates were assumed to be of constant thickness, made up of 39 composite layers (as well as 38 interfaces) giving a total thickness of $T = 9.93$ cm with an internal radius of 15 mm in the corners. As in a typical aerospace application, the stacking sequence differs in the covers (top and bottom), corners and in the spar (sides) with the following approximate percentage breakdowns of 0°, ±45° and 90°:

$$[50\%, 40\%, 10\%] \quad \text{(covers)}; \quad [20\%, 60\%, 20\%] \quad \text{(corners)}; \quad \text{and} \quad [15\%, 70\%, 15\%] \quad \text{(spars)}. \tag{25}$$

We reiterate that this example serves to represent structural scale modelling. Therefore, sub-structural phenomena, such as stiffening of the upper and lower covers, are not modelled here. The specific layer-sequencing has been chosen, using a discrete optimiser, to ensure that each laminate is balanced, symmetric with no bend–twist coupling, whilst maximising the number of continuous orientations around the wing box. Transitions between each of the stacking sequences are achieved over a relatively short segment of 5 cm, and the chosen stacking sequence is in no way optimised for strength in these regions, as considered for example by Dillinger et al. [34]. In practice, this change of stacking sequence is easy for the user to specify using a `.csv` file specifying different `Regions` for each segment of the wingbox and providing the required different stacking sequence. The wingbox geometry is again achieved by specifying a `gridTransformation()`, which now becomes slightly more complex, in order to handle each of the different regions. To create the closed curve of this wingbox, periodic boundary conditions are imposed. In this application, we consider two forms of loading. Firstly, an internal pressure of 0.109 MPa, arising from the fuel, is applied to the internal surface. Secondly, a thermal pre-stress induced by the manufacturing process is imposed, using the user-defined function `evaluateHeat()` (see Fig. 9).

We approximate the influence of the ribs that constrain the wingbox in the y direction, by clamping all degrees of freedom at one end, whilst tying all other degrees of freedom at the other end using a multipoint constraint. Elements to be included in the multipoint constraint are marked with the user-defined function `isMPC(FieldVec& x)`. A localised wrinkle defect is introduced into one of the corner radii, as shown in Fig. 8. The defect is introduced by adapting the function `gridTransformation()`. The wrinkle geometry is defined by a random field, parameterised by a Karhunen–Loéve expansion. The actual parametrisation of the wrinkle is chosen to match an observed defect in a CT-Scan of a real corner section. Further details of this methodology are provided in Sandhu et al. [33, Sec. 3].

We firstly carry out a weak scaling experiment, increasing the problem size proportionally to the number of cores used. For iterative solvers that scale optimally with respect to problem size and with respect to the number of cores, the computational time should remain constant. To scale the problem size as the number of cores N_{cores} grows, we refine the mesh, doubling the number of elements as we double the number of cores. The number or elements for each setup are detailed in Table 2, separately listing the number of elements across the spar and the cover, around the corners and along the length of the wingbox. The defective corner, denoted R_d, contains twice as many elements as the other three corners, denoted by R_{nd}. Table 2 also details the resulting number of degrees of freedom, iteration numbers for the preconditioned CG, an estimate of the condition number of the preconditioned system matrix $\mathbf{M}_{\mathbf{AS},2}^{-1}\mathbf{A}$, the dimension of the coarse space dim V_H, as well as the total run time. Fig. 10 (left) shows that the weak scaling of the iterative CG solver in `dune-composites` with GENEO preconditioner is indeed almost optimal up to at least 15,360 cores (the limiting capacity available on ARCHER for our experiments). We also include a more detailed subdivision of the computational time into Setup Time (for the assembly of the FE stiffness matrix and for the construction of the GENEO coarse space) and Iteration Iime (for the preconditioned CG iteration). Both scale almost optimally. This test demonstrates the capability of increasing the size of the tests at a nearly constant run time and thus, to solve a problem with 200 million degrees of freedom in just over 13 min.

Next we carry out a small strong scaling experiment. The mesh is that of Setup 5 in Table 2 and the results of the strong scaling test are given in Fig. 10 (right) and in Table 3. We see that the iterative CG solver in `dune-composites` with GenEO preconditioner scales almost optimally to at least 11 320 cores, with the time taken approximately halving as the number of cores is doubled.

Again, both the Setup and the Iteration scale optimally.

6. Subsurface flow application: Strong scaling for the SPE10 benchmark

In this section, we apply the GenEO solver to an elliptic partial differential problem outside of composites modelling, demonstrating its scalability and robustness on a subsurface flow problem in a highly heterogeneous medium. A challenging test case in

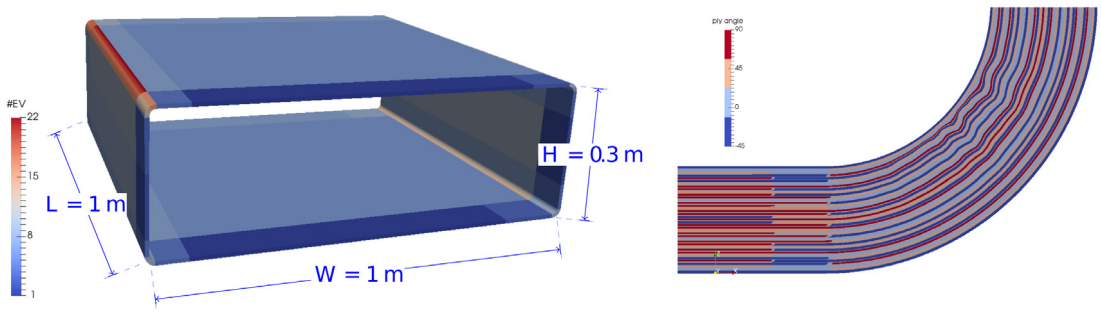

Fig. 8. (Left) Geometry of the wingbox with dimensions; the colouring shows the number of eigenmodes used in GenEO in each of the subdomains of Setup 6 in Table 2. (Right) Close-up plot of the corner of the wingbox using `plotProperties()`, which shows the wrinkle and the inter-lacing of the different stacking sequences in the corner, cover and spar regions.

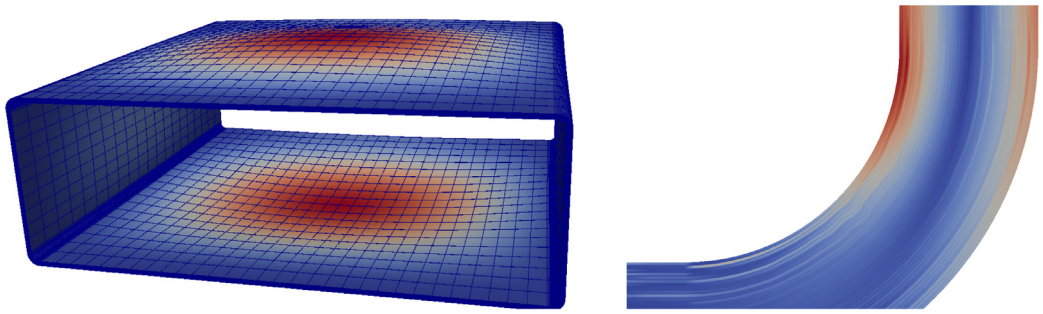

Fig. 9. FE solution for Example 3: (Left) Overall deformation of the wingbox with colours showing the magnitude of the displacements in cm. (Right) Camanho failure criterion (23) in a close-up of the corner containing the wrinkle defect.

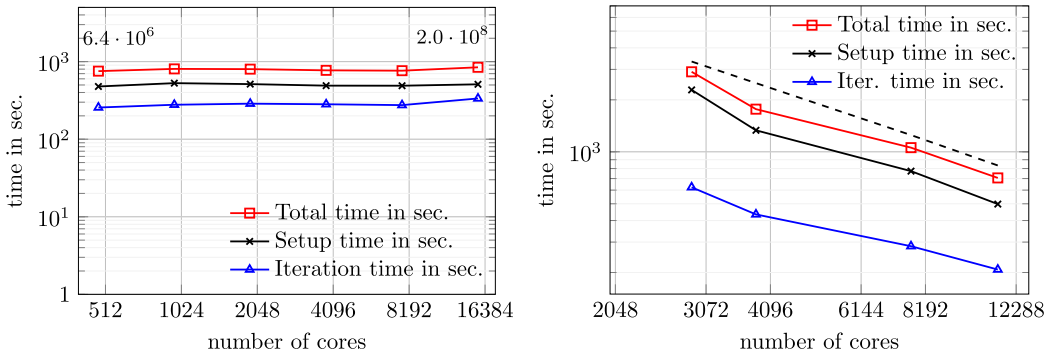

Fig. 10. Parallel performance of `dune-composites` on Archer: (Left) A weak scaling test, as summarised in Table 2. (Right) A strong scaling test using Setup 5 in Table 2, with the dashed line showing perfect scaling, as summarised in Table 3.

Table 2
Details of the six setups and results used in the weak scaling test. In all of the tests, we used two layers of 20-node serendipity elements per fibrous layer and only one layer of elements in each of the interface layers. The number of elements per core was fixed at 2808.

Setup	N_{cores}	Spar	Cover	R_d	R_{nd}	Length	DOF	iter.	κ	dim V_H	Time (s)
1	480	34	14	40	20	20	$6.4 \cdot 10^6$	156	445	5 025	734
2	960	34	14	40	20	40	$1.3 \cdot 10^7$	154	421	7 840	806
3	1 920	68	28	80	40	40	$2.6 \cdot 10^7$	152	322	18 752	800
4	3 840	68	28	80	40	80	$5.1 \cdot 10^7$	144	287	29 444	772
5	7 680	216	64	80	40	80	$1.0 \cdot 10^8$	132	303	50 930	764
6	15 360	216	64	80	40	160	$2.0 \cdot 10^8$	102	245	94 527	845

the computational geosciences is the SPE10 benchmark [3]. This problem features high contrast, heterogeneous coefficients which challenge most iterative solvers [4].

We consider the SPE10 domain $\Omega := [0, 1200] \times [0, 2200] \times [0, 170]$ (feet), divided into a tensor product grid $\mathcal{T}_h$ with $60 \times 220 \times 85 = 1.122 \times 10^6$ cells. The domain Ω has the boundary $\partial\Omega = \Gamma_D \cup \Gamma_N$, where we define $\Gamma_D := \{\mathbf{x} \in \partial\Omega : z = 0\}$ as the Dirichlet part of the boundary and $\mathbf{n} \in \mathbb{R}^3$ as the outward normal to $\partial\Omega$. We calculate the steady-state fluid pressure $u(\mathbf{x}) \in \Omega$ which obeys Darcy's law. This is given by the linear, scalar elliptic partial differential equation

$$-\nabla \cdot (\mathbf{K}(\mathbf{x})\nabla u) = f, \quad \forall \mathbf{x} \in \Omega \tag{26}$$

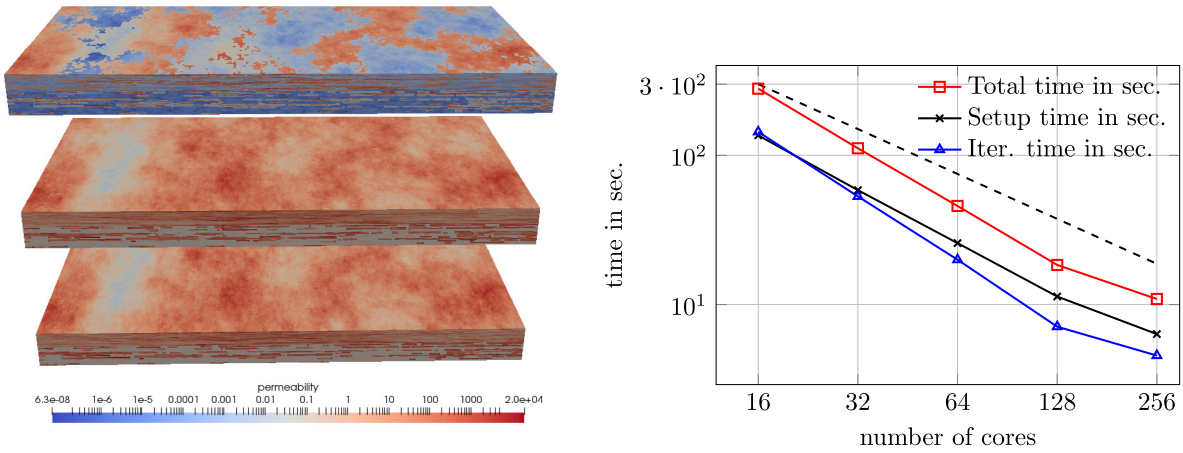

Fig. 11. Left: Logarithm of the permeability field K for the SPE10 benchmark, from bottom to top: K_x, K_y and K_z. Right: A strong scaling test using the SPE10 dataset, with the dashed line showing perfect scaling.

Table 3
Strong scaling test with Setup 5 in Table 2, demonstrating near optimal strong scaling up to at least 11,320 cores.

N_{cores}	Elements per core	dim(V_H)	it.	T_{it}	T_{setup}	T_{total}	Total core time (days)
2 880	3132	18 843	167	623	2283	2906	96.9
3 840	2340	26 333	153	434	1332	1766	78.5
7 680	2008	52 622	132	284	773	1057	94.0
11 320	1392	78 233	162	208	498	706	92.5

Table 4
A strong scaling test using the SPE10 dataset.

N_{cores}	dim(V_H)	it.	T_{it}	T_{setup}	T_{total}
16	149	167	136.11	143.621	279.721
32	225	203	58.42	53.065	111.485
64	379	206	25.81	19.982	45.787
128	527	224	11.32	7.107	18.427
256	930	232	6.34	4.552	10.892
512	1737	234	5.18	3.795	8.975

subject the boundary conditions

$$u(\mathbf{x}) = 0 \quad \text{on} \quad \Gamma_D \quad \text{and} \quad -\mathbf{K}(\mathbf{x})\nabla u \cdot \mathbf{n} = 0 \quad \text{on} \quad \Gamma_N = \partial\Omega \backslash \Gamma_D \,. \tag{27}$$

The SPE10 dataset gives a spatially varying permeability tensor

$$\mathbf{K}(\mathbf{x}) = \begin{bmatrix} K_x(\mathbf{x}) & 0 & 0 \\ 0 & K_y(\mathbf{x}) & 0 \\ 0 & 0 & K_z(\mathbf{x}) \end{bmatrix} \quad \forall \mathbf{x} \in \Omega \,.$$

Fig. 11(left) shows the permeability field, it is constant in each cell, but varies strongly over the domain. The parameters K_x and K_y vary from 6.65×10^{-4} to 2.0×10^4 and the parameter K_z varies from 6.65×10^{-8} to 6.0×10^3.

We define the function space for the pressure u as $V := \{v \in H^1(\Omega) : v(\mathbf{x}) = 0 \,,\ \mathbf{x} \in \Gamma_D\}$, and choose the finite element space $V_h \subset V$ to be the set of continuous, piecewise linear functions on $\mathcal{T}_h$. The finite element discretisation of (26) then reads: Find $u_h \in V_h$ such that

$$\int_\Omega \mathbf{K}(\mathbf{x})\nabla u_h \cdot \nabla v_h \, dx = \int_\Omega f v_h \, dx \quad \forall \mathbf{v}_h \in V_h. \tag{28}$$

By defining $u_h = \sum_{i=1}^N u^{(i)}\phi_i(\mathbf{x})$, again we obtain the sparse system of equations

$$\mathbf{Au} = \mathbf{b}, \quad \text{where} \quad \mathbf{u} = [u^{(1)}, u^{(2)}, \ldots, u^{(N)}]^T$$

is a vector pressures at each cell vertex, which is assembled element-wise from (28) using standard Gaussian integration. The source term in our experiments is assumed to be uniform $f \equiv 1$.

In Fig. 11(right) and Table 4 we show a small strong scaling experiment performed with this challenging setup. The parameter contrast for this benchmark is on the order of 10^{11}. Nevertheless, we see that the iterative CG solver in `dune-composites` with GenEO preconditioner scales almost optimally to at least 256 cores. At 512 cores with only around 2000 elements per core the strong scaling begins to break down due to the communication overhead. Due to the layered structure of the material parameters our domain decomposition is two dimensional. Each subdomain includes the full length in z-direction. We used a minimal overlap of only one element. Table 4 also details the number of cores, size of the coarse space dim V_H, iteration numbers for the preconditioned CG, as well as the time spent in CG iterations, setup time and total run time.

7. Discussion & future developments

In this paper, we describe the new high performance package `dune-composites`, designed to solve massive finite element problems for the anisotropic linear elasticity equations. The paper provides both the mathematical foundations of the methods, their implementation within a state-of-the-art software platform on modern distributed memory computer architectures, as well as details of how to set up a problem and carry out an analysis, illustrated via a series of increasingly complex examples. In addition, we demonstrate the scalability of the new solver on over 15,000 cores on the UK national supercomputer ARCHER, solving industrially motivated problems with over 200 million degrees of freedom within minutes. This scale of computations brings composites problems that would otherwise be unthinkable into the feasible range.

The disadvantage of `dune-composites` as a package over commercial counterparts is currently the limited functionality in considering more general problems; this includes complex geometries, unstructured grids and nonlinear problems. This is the first release of `dune-composites`, and therefore the functionality is naturally still limited, but it will increase over time, driven by the industrial questions we seek to solve as a community of developers. There are currently four key areas of active development:

- **Multiscale methods**: There is a strong connection between coarse spaces for domain decomposition methods, as developed and implemented within the GenEO preconditioner, and multiscale discretisation methods, such as generalised multiscale FEs (GMsFE) [35]. In fact, the GenEO coarse space provides and natural multiscale method. Current research [36] is aiming to provide this functionality as an embedded solution scheme within `dune-composites`.
- **Nonlinear mechanics**: Currently the analysis implemented in the package is linear (static and thermal). Naturally, modelling failure propagation in composites is a nonlinear problem. Current research in implementing nonlinear material models includes cohesive zone models and ply damage models, for integration within the existing framework.
- **Uncertainty quantification**: A significant motivation for developing efficient robust solvers is to enable stochastic studies in which many simulations of a model with different parameters are required. The package has already been used to study the effects of wrinkle defects in composite strength [33], and ongoing research is exploring Bayesian parameter estimation using multilevel Markov chain Monte Carlo methods [37,38].
- **GUI development**: A current limitation of the package is that its application to new models or geometries requires a basic knowledge of `C++` and command line programming. Active development with software engineers is seeking to provide a simple Graphical User Interface (GUI) to enable application-focused research with `dune-composites` without extensive programming experience.

Acknowledgments

This work was supported by an EPSRC Maths for Manufacturing grant (EP/K031368/1). Richard Butler holds a Royal Academy of Engineering-GKN Aerospace Research Chair in Composites. This work used the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

References

[1] T.J. Hughes, The Finite Element Method: Linear Static and Dynamic Finite Element Analysis, Courier Corporation, 2012.
[2] M. Blatt, A. Burchardt, A. Dedner, C. Engwer, J. Fahlke, B. Flemisch, C. Gersbacher, C. Gräser, F. Gruber, C. Grüninger, D. Kempf, R. Klöfkorn, T. Malkmus, S. Müthing, M. Nolte, M. Piatkowski, O. Sander, Arch. Numer. Softw. 4 (2016) 13–29.
[3] M. Christie, M. Blunt, Proceedings of SPE Reservoir Simulation Symposium, 11-14 February, Houston, SPE-66599-MS, Society of Petroleum Engineers, 2001.
[4] P. Bastian, E.H. Müller, S. Müthing, M. Piatkowski, Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations, 2018, arXiv e-prints, arXiv:1805.11930.
[5] The 2016 UK composites strategy: Delivering UK growth through the multi-sector application of composites, 2016, URL https://compositesuk.co.uk/system/files/documents/Strategy%20final%20version_1.pdf.
[6] A. Reinarz, T. Dodwell, T. Fletcher, L. Seelinger, R. Butler, R. Scheichl, Compos. Struct. 184 (2018) 269–278.
[7] D. Systèmes, Abaqus analysis user's manual, 2007.
[8] M. Blatt, P. Bastian, International Workshop on Applied Parallel Computing, Springer, 2006, pp. 666–675.
[9] U.M. Yang, V.E. Henson, Appl. Numer. Math. 41 (2002) 155–177.
[10] A. Toselli, O. Widlund, Domain Decomposition Methods-Algorithms and Theory, Vol. 34, Springer Science & Business Media, 2006.
[11] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, R. Scheichl, Numer. Math. 126 (2014) 741–770.
[12] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, O. Sander, Computing 82 (2008) 103–119.
[13] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, O. Sander, Computing 82 (2008) 121–138.
[14] M. Blatt, P. Bastian, Int. J. Comput. Sci. Eng. 4 (2008) 56–69.
[15] T. Belytschko, J.S.-J. Ong, W.K. Liu, J.M. Kennedy, Comput. Methods Appl. Mech. Engrg. 43 (1984) 251–276.
[16] T. Ting, Anisotropic Elasticity: Theory and Applications, in: Applied Mathematics and Engineering Science Texts Series, John Wiley & Sons, Limited, 1992, URL https://books.google.de/books?id=XxemPwAACAAJ.
[17] Y. Saad, Iterative Methods for Sparse Linear Systems, Vol. 82, siam, 2003.
[18] M. Sarkis, Numer. Math. 77 (1997) 383–406.
[19] P. Jolivet, V. Dolean, F. Hecht, F. Nataf, C. Prud'Homme, N. Spillane, J. Numer. Math. 20 (2012) 287–302.
[20] P. Jolivet, F. Hecht, F. Nataf, C. Prud'Homme, Sci. Program. 22 (2014) 157–171.
[21] F.M. Gomes, D.C. Sorensen, Arpack++: A C++ Implementation of ARPACK Eigenvalue Package, Tech. Rep. TR97729, CRPC, Rice University, Houston, TX, 1997.
[22] T.A. Davis, ACM Trans. Math. Softw. (TOMS) 30 (2004) 196–199.
[23] P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuß, H. Rentz-Reichert, C. Wieners, Comput. Vis. Sci. 1 (1997) 27–40.
[24] D.N. Arnold, G. Awanou, Found. Comput. Math. 11 (2011) 337–344.
[25] R.D. Falgout, J.E. Jones, U.M. Yang, Numerical Solution of Partial Differential Equations on Parallel Computers, Springer, 2006, pp. 267–294.
[26] T.A. Fletcher, T. Kim, T.J. Dodwell, R. Butler, R. Scheichl, R. Newley, Compos. Struct. 146 (2016) 26–33.
[27] P.P. Camanho, C.G. Davila, M.F. De Moura, J. Compos. Mater. 37 (2003) 1415–1438.
[28] ASTM D6415 / D6415M-06a(2013), Standard test method for measuring the curved beam strength of a fiber-reinforced polymer-matrix composite, 2013, http://dx.doi.org/10.1520/D6415_D6415M-06AR13.
[29] P.R. Amestoy, A. Guermouche, J.-Y. L'Excellent, S. Pralet, Parallel Comput. 32 (2006) 136–156.
[30] M.W. Gee, C.M. Siefert, J.J. Hu, R.S. Tuminaro, M.G. Sala, ML 5.0 Smoothed Aggregation User's Guide, Technical Report, Technical Report SAND2006-2649, Sandia National Laboratories, 2006.
[31] T.J. Dodwell, R. Butler, G.W. Hunt, Compos. Sci. Technol. 105 (2014) 151–159.
[32] J.P.-H. Belnoue, T. Mesogitis, O.J. Nixon-Pearson, J. Kratz, D.S. Ivanov, I.K. Partridge, K.D. Potter, S.R. Hallett, Composites A 102 (2017) 196–206.
[33] A. Sandhu, A. Reinarz, T. Dodwell, Compos. Struct. 205 (2018).
[34] J. Dillinger, M. Abdalla, T. Klimmek, Z. Gürdal, J. Aircr. 50 (2013) 1159–1168.
[35] Y. Efendiev, J. Galvis, T.Y. Hou, J. Comput. Phys. 251 (2013) 116–135.
[36] T.J. Dodwell, A. Sandhu, R. Scheichl, International Workshop on Bifurcation and Degradation in Geomaterials, Springer, 2017, pp. 577–584.
[37] T.J. Dodwell, C. Ketelsen, R. Scheichl, A.L. Teckentrup, SIAM/ASA J. Uncertain. Quantif. 3 (2015) 1075–1108.
[38] T.J. Dodwell, C. Ketelsen, R. Scheichl, A.L. Teckentrup, SIAM Rev. 61 (2019) 509–545.