Computer Physics Communications 319 (2026) 109901

Contents lists available at ScienceDirect

Computer Physics Communications

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

Computational Physics

A detailed guide to an open-source implementation of the hybrid phase field method for 3D fracture modeling in deal.II

Wasim Niyaz Munshi [a,b], Marc Fehling [c], Wolfgang Bangerth [b,d], Chandrasekhar Annavarapu [a,*]

[a] Department of Civil Engineering, IIT Madras, Chennai, 600036, Tamil Nadu, India
[b] Department of Mathematics, Colorado State University, Fort Collins, 80521, Colorado, USA
[c] Department of Mathematical Analysis, Faculty of Mathematics and Physics, Charles University, Sokolovská 49/83, Prague 8, 186 75, Czech Republic
[d] Department of Geosciences, Colorado State University, Fort Collins, 80521, Colorado, USA

ARTICLE INFO

Prof. Andrew Hazel

Keywords:
Phase field method
Three-dimensional fractures
`deal.II` Implementation
Parallel framework
Adaptive mesh refinement

ABSTRACT

Phase-field models for fracture have demonstrated significant power in simulating realistic fractures, including complex behaviors like crack branching, coalescing, and fragmentation. Despite this, these models have mostly remained in the realm of proof-of-concept studies rather than being applied to practical problems. This paper introduces a computationally efficient implementation of the phase-field method based on the open source finite element library `deal.II`, incorporating parallel computing and adaptive mesh refinement. We provide a detailed outline of the steps required to implement the phase field model in `deal.II`. We then validate our implementation through a benchmark 3D boundary value problem and finally demonstrate the computational capabilities by running field scale problems involving complicated fracture patterns in 3D. This open-source code offers a framework that enables engineers and researchers to simulate diffuse crack growth within a widely-used computational environment.

1. Introduction

Modeling fracture and cracks in materials is difficult because it involves describing lower-dimensional surfaces that cut through a higher-dimensional material. Moreover, the complex geometry of the resulting object changes over time as cracks grow (or perhaps heal), and resolving these phenomena by remeshing both the fracture surfaces and the domain is not practically feasible for all but the simplest situations.

The phase-field method (PFM) for fracture mechanics addresses this issue by modeling fractures not as discrete surfaces, but instead as extended volumetric regions that are weaker than the surrounding material and that can accommodate crack opening via strain accumulation. Phase field methods employ a continuous scalar field (the "damage" field) to represent cracks as diffused surfaces. Crack evolution is governed by an additional equation for the damage field. Based on an energy minimization principle, the phase-field method allows cracks to evolve naturally to minimize the system's total energy, without requiring explicit criteria for crack propagation. This makes PFMs particularly effective for modeling complex fracture scenarios, including crack branching, merging, and fragmentation, especially in 3D [1–3], where meshing such complex and evolving scenarios is not reasonable.

Despite its flexibility and recent popularity in research, there are only few publicly available implementations of the phase-field method for fracture models that researchers can base their own work on when exploring alternative formulations, or to actually simulate concrete applications. The existing implementations of PFM typically fall into two categories: commercial software and ones built on open-source finite element libraries. Let us review what we have found in the available literature:

- *Commercial Software:* PFMs for fracture have been implemented in various commercial platforms like `ABAQUS` [4], `ANSYS` [5], and `COMSOL` [6,7] as part of their extensive set of built-in tools for simulating fracture mechanics [3,7–11]. However, most commercial implementations are limited to 2D due to the high computational demands of PFM, which requires fine meshing for an accurate crack representation. 3D simulations are rare and typically restricted to simple cases. As examples, Molnár and Gravouil model a 3D single-notched plate under uniaxial tension, utilizing a prepared mesh that is fine only in regions where crack growth is anticipated [4]. Similarly, Navitehrani et al. leverage symmetry by modeling just one-eighth of the 3D Brazilian test, applying appropriate boundary conditions

* Corresponding author.
E-mail address: annavarapuc@civil.iitm.ac.in (C. Annavarapu).

https://doi.org/10.1016/j.cpc.2025.109901
Received 29 April 2025; Received in revised form 9 October 2025; Accepted 13 October 2025
Available online 26 October 2025

[3]. While these simulations serve as valuable proof-of-concept examples, they fall short of showcasing the full capabilities of PFMs in capturing complex fracture patterns in 3D. Additionally, while commercial software simplifies PFM use, it also lacks the customization required for advanced research or novel methodologies – for example, it may be restricted to isotropic materials, or specific kinds of anisotropic materials, excluding the simulation of layered fabrics.

- *Codes built on Open-Source Finite Element Libraries:* Open-source libraries like `MOOSE` [12], `FEniCS` [13], `Gridap` [14], `Jive` [15], `Nutils` [16] and `deal.II` [17] provide greater flexibility, particularly for custom implementations. `MOOSE` has been used for phase-field fracture modeling due to its efficient implementation of nonlinear PDE solvers, adaptive mesh refinement (AMR), and parallel computing capabilities [18,19]. Likewise, `FEniCS`, with its Python interface, has seen PFM implementations from various researchers [20–24]. `FEniCS` (and, to some extent, `MOOSE`) focus on simplicity and automation and consequently make it very easy to create a first version of a solver for a new model or formulation; at the same time, the automation also sometimes makes it difficult to control details. `Gridap` was designed to strike a balance between computational efficiency, ease of use, and streamlined workflow productivity and has been successfully used to model phase-field fracture in research such as [25–30]. `Jive` is another open-source FEM toolkit designed for parallel and multi-core computing. It supports domain decomposition, allowing the full simulation process – including both assembly and solution – to be executed in parallel. May et al. [31], Li et al. [32], and Mandal et al. [33] used `Jive` [15] to assess phase-field models for brittle and cohesive fracture; however, none of these works seem to use MPI or adaptive mesh refinement. `Nutils` is an open-source Python-based library for finite element computations, offering efficient vectorization and built-in parallelism. It allows for seamless transition from academic models to real-world applications. Singh et al. [34] used Nutils to solve simplified 2D fracture problems using predefined non-uniform meshes guided by the expected crack path.

  The `deal.II` library, which we use here, is a popular open-source finite element library that emphasizes high-performance computing (HPC) and strong parallel computing capabilities, making it ideal for complex, large-scale models such as those necessary for three-dimensional phase-field fracture simulations. The work of Heister et al. [35] on *pfm-cracks* [36] is one of the earliest and pioneering papers of PFM in `deal.II`. Several other applications of the phase-field fracture model in `deal.II` have followed and demonstrated multi-physics coupling (for example, [37–44]). However, most of these papers focused on efficient numerical strategies to advance the phase-field fracture model itself, and the authors consequently did not provide a detailed guide for implementing the phase-field fracture model in `deal.II`. More recently, Jin et al. [45,46] proposed two novel phase-field formulations to address difficulties in solving non-convex energy minimization problems. While both incorporate adaptive mesh refinement, their parallelization strategies remain limited to using multi-threading via the Threading Building Blocks (TBB) library only during element-level operations, with the rest of the code remaining sequential.

Our conclusion from this overview is that despite the growing popularity of the phase-field fracture method, a detailed implementation guide tailored for early-career researchers using `deal.II` is still lacking. While such comprehensive resources exist for commercial software like `ABAQUS`, and for other open-source finite element libraries, there remains a clear need for a similarly well-documented, open-source reference based on `deal.II`. To address this gap, we herein provide an implementation-focused guide to developing a phase-field fracture model using `deal.II`. Specifically, we here discuss the steps necessary to implement the hybrid PFM model, as described in Ambati et al. [47], in `deal.II`. The hybrid PFM is an easily extendible framework for PFM solvers that integrates aspects of the early phase-field models – which exhibit identical responses in compression and tension – with the more advanced approaches that later introduced tension-compression splitting to reflect asymmetric fracture behavior. When solved using a staggered solution approach, the hybrid PFM keeps the subproblems associated with the various physical effects linear and enables the incorporation of other physical effects (such as thermal or electromagnetic stresses) as well. Having said that, we would like to acknowledge that while PFM offers several advantages, it is not without its limitations. For instance, although PFM excels at simulating complex crack behaviors such as propagation, branching, and coalescence [48–54], predicting crack nucleation remains a significant challenge [55,56]. In this paper, our primary objective is to provide a detailed guide for implementing PFM in `deal.II`. We do not address any inherent limitations of the model. We choose `deal.II` for its state-of-the-art numerical methods including adaptive mesh refinement and parallel computing. Specifically, our goals for this paper are:

- Describe a *concrete implementation* of a phase-field model, along with the underlying algorithms;
- Demonstrate the *versatility* of the implementation using non-trivial benchmarks;
- Illustrate the *performance* of our implementation using large-scale parallel and adaptive simulations.

The code accompanying this paper, see [57], is available as part of the `deal.II` "code gallery" (a collection of user-contributed codes based on the `deal.II` library) and can be used as an open-source basis for further extension to more complex, physics-driven problems. It is licensed under the Lesser GNU Public License (LGPL) version 2.1 or later, allowing for re-use in a wide range of scenarios. The code can be found as a link in the list of code gallery programs at https://dealii.org/developer/doxygen/deal.II/CodeGallery.html, or directly at https://dealii.org/developer/doxygen/deal.II/code_gallery_Phase_field_fracture_model_in_3D.html.

Outline. The remainder of this paper is structured as follows: In Section 2 we describe the specific model our code solves, including the governing equations, the overall algorithm structure, and our goals for adaptive mesh refinement and parallel computing. Section 3 then discusses algorithmic and software-specific details of the implementation. We illustrate our implementation with a sequence of numerical experiments in Section 4 with which we validate the correctness of our implementation, show how adaptive mesh refinement helps us solve nontrivial examples, and demonstrate parallel scalability. Finally, we conclude in Section 5.

2. Methodology

Let us begin outlining the methodology that underlies our code by describing the basic premises of our code. Namely, in the following, we will discuss the mathematical model (Section 2.1), the staggered algorithm to solve these equations (Section 2.2), the spatial discretization and adaptive mesh refinement (Section 2.3), and finally our approach to parallel computing (Section 2.4).

2.1. Governing equations

We use the hybrid phase-field method that combines the quasi-static evolution of a displacement field as a result of load stepping, with the accumulation of damage to the material due to tensile stress. We refer to Ambati et al. [47] for a detailed discussion of this model, and provide its outline below.

The elastic model. Phase-field models describe crack propagation via the interplay of elastic (or perhaps plastic) deformation and the build-up

of a "damage" field that describes the weakening of the elastic body. Each of these two building blocks is represented by a partial differential equation that form a coupled system. To fix notation, let us consider a domain Ω with an external boundary $\partial\Omega$, where $\boldsymbol{n}$ denotes the outward-pointing normal vector to the boundary.

On this domain, we consider the deformation of an elastic solid in quasi-static equilibrium, driven by external loading that is applied in small steps. The equations for linear momentum balance in the absence of external body forces are then defined as

$$\nabla \cdot \boldsymbol{\sigma}(\boldsymbol{\varepsilon}(\boldsymbol{u}), d) = \mathbf{0} \qquad \text{in } \Omega, \tag{1a}$$

$$\boldsymbol{u} = \boldsymbol{u}_D \qquad \text{on } \Gamma_D, \tag{1b}$$

$$\boldsymbol{\sigma} \cdot \boldsymbol{n} = \boldsymbol{t}_N \qquad \text{on } \Gamma_N. \tag{1c}$$

Here, $\boldsymbol{\sigma}$ is the Cauchy stress tensor, $\boldsymbol{\varepsilon}(\mathbf{u})$ is the small-strain tensor and $\boldsymbol{u}$ is the displacement vector. d is the spatially variable damage field we will discuss below. Eqs. (1b) and (1c) represent the standard Dirichlet and Neumann boundary conditions, where $\boldsymbol{u}_D$ and $\boldsymbol{t}_N$ refer to prescribed displacements and tractions, respectively.

The Cauchy stress tensor, $\boldsymbol{\sigma}$, is defined as

$$\boldsymbol{\sigma} = g(d)\,\frac{\partial \psi_0(\boldsymbol{\varepsilon}(\mathbf{u}))}{\partial \boldsymbol{\varepsilon}}, \tag{2a}$$

$$g(d) = \begin{cases} (1-d)^2 & \text{if } \ \psi_0^+ > \psi_0^-, \\ 1 & \text{otherwise.} \end{cases} \tag{2b}$$

Here, the "degradation function" $g(d)$ describes the effect of damage on the stiffness of the material. Specifically, the damage field d is a spatially variable function with values between zero (undamaged material) and one (completely destroyed material with no remaining internal cohesion); $g(d)$ as defined in (2b) then describes a situation in which the stress for a given strain is reduced by a factor $(1-d)^2$ if the material is under tension (if $\psi_0^+ > \psi_0^-$), corresponding to the notion that fractures under tension do not provide the material with stiffness. In contrast, a fracture under compression (if $\psi_0^+ < \psi_0^-$) is assumed to be just as stiff as undamaged material. This approach ensures that the hybrid model prevents unphysical cracks in compression while keeping the equation of momentum balance linear. In our code, we only implement the first branch of Eq. (2b) since we only consider tensile cases; this is also the approach taken in [2].

In our experiments, we use isotropic linear materials for which the representative total, tensile, and compressive strain energy densities ψ_0, ψ_0^+ and ψ_0^- are expressed in terms of the strain $\boldsymbol{\varepsilon}$ and the Lamé parameters λ and μ as

$$\psi_0(\boldsymbol{\varepsilon}) = \frac{1}{2}\lambda\,\mathrm{tr}(\boldsymbol{\varepsilon})^2 + \mu\,\mathrm{tr}\left(\boldsymbol{\varepsilon}^2\right), \tag{3a}$$

$$\psi_0^{\pm}(\boldsymbol{\varepsilon}) = \frac{1}{2}\lambda\langle \mathrm{tr}(\boldsymbol{\varepsilon})\rangle_{\pm}^2 + \mu\,\mathrm{tr}\left(\boldsymbol{\varepsilon}_{\pm}^2\right). \tag{3b}$$

Here, we use a tension-compression split based on the spectral decomposition of the strain tensor (see Miehe et al. [58]) to suppress nonphysical crack growth in compression. It is defined via the Macaulay brackets $\langle\bullet\rangle_{\pm} = \frac{1}{2}(\bullet \pm |\bullet|)$ that map a number to its positive/negative part, and using this to define the positive/negative parts of a symmetric tensor via its principal strain decomposition

$$\boldsymbol{\varepsilon}_{\pm} = \sum_{i=1}^{n_{sd}} \langle \varepsilon_i \rangle_{\pm} \boldsymbol{e}_i \otimes \boldsymbol{e}_i. \tag{4}$$

Here, ε_i are the principal strains, $\boldsymbol{e}_i$ are the principal directions, and $n_{sd} = 2, 3$ is the number of space dimensions.

Finally, note that while the equations above are all time-independent, they do depend on the loading of the object as provided by successive load steps applied through the boundary conditions. The loading history is also reflected in the damage variable d we will discuss next.

The damage model. In order to describe the damage the material incurs, we compute a monotonically increasing "history field" $\mathcal{H}^+$ that at each point in the domain represents the irreversible damage (see Miehe et al. [58]) and that is defined as

$$\mathcal{H}^+ = \max_{\tau \in [0,t]} \psi_0^+(\boldsymbol{\varepsilon}(\tau)), \tag{5}$$

where we use the symbol t to denote the loading history variable.

Using $\mathcal{H}^+$, we can define a damage field d that is, in essence, a smoothed out and scaled version of $\mathcal{H}^+$:

$$-G_c l \nabla^2 d + \frac{G_c}{l} d = 2(1-d)\mathcal{H}^+ \qquad \text{in } \Omega, \tag{6a}$$

$$\left(G_c l\right)\nabla d \cdot \boldsymbol{n} = \mathbf{0} \qquad \text{on } \partial\Omega, \tag{6b}$$

Here, l denotes the phase-field length scale parameter, and G_c represents the critical energy release rate that can be a spatially variable function. Eq. (6b) defines flux-free boundary conditions over the entire external boundary.

2.2. The staggered solution approach

The model described in the previous section consists of two partial differential equations that are coupled and nonlinear. The nonlinearity is also non-smooth through the tension-compression split as well as the use of the maximum operation in the definition of $\mathcal{H}^+$. The nonlinearity and non-smoothness pose difficulties for the numerical solution of the coupled system. These difficulties can be overcome through sophisticated mathematical schemes exploiting details of the formulation (see, for example, [36]), but this limits how easy it is to extend our implementation to other formulations. As a consequence, we have opted for a simpler approach that alternates solving the two equations individually, until convergence is reached.

The key to our scheme is the observation that individually, the two equations are linear – i.e., for a fixed d, Equation (1) is linear in the displacement $\mathbf{u}$, and for a fixed displacement $\mathbf{u}$ and history variable $\mathcal{H}^+$, Equation (6) is linear in d. The linearity of Equation (2) is a feature of the "hybrid" phase-field model because, in this model, the degradation function is computed as a postprocessed quantity from the previous iteration. This degradation function is then used to degrade the entire strain energy density thus ensuring the linearity of Equation (1) while simultaneously enforcing tension-compression asymmetry. As both Equations (1) and (6) are linear, they can consequently be solved efficiently in a staggered scheme until a convergence criterion is satisfied. It bears emphasis that damage irreversibility can be enforced through other, more rigorous, means such as the primal-dual active set strategy described in [35]. However, the use of a history field variable is arguably more beneficial in the hybrid phase-field model as it maintains the linearity of the field equations. Ambati et al. [47] present a detailed comparison of the performance of the hybrid model with older phase-field models that result in nonlinear momentum balance equations due to the tension-compression split.

We utilize the linearity of the two governing field equations in the design of the algorithm herein. Within a load step, we therefore start the iteration with $\mathbf{u}^0, d^0$ corresponding to the final solutions of the previous load step. For the ith iteration within this load step, we then first solve

$$\nabla \cdot \boldsymbol{\sigma}(\boldsymbol{\varepsilon}(\boldsymbol{u}^{i+1}), d^i) = \mathbf{0} \qquad \text{in } \Omega \tag{7}$$

for $\boldsymbol{u}^{i+1}$ using the current loading displacements and tractions; update the history field $\mathcal{H}^{+,i+1}$; and finally solve

$$-l^2\nabla^2 d^{i+1} + d^{i+1} = \frac{2l}{G_c}(1-d^{i+1})\mathcal{H}^{+,i+1} \qquad \text{in } \Omega \tag{8}$$

for d^{i+1}. The iterative procedure is stopped when the relative error in the nodal solution vectors from two successive iterations is smaller than a specified tolerance

$$\frac{||\mathbf{u}^{i+1} - \mathbf{u}^i||}{||\mathbf{u}^i||} \leq \text{tol.} \tag{9}$$

and

$$\frac{||\mathbf{d}^{i+1} - \mathbf{d}^i||}{||\mathbf{d}^i||} \leq \text{tol.} \tag{10}$$

This overall approach, including the load stepping loop around the nonlinear iteration is outlined Algorithm 1.

Algorithm 1 Load stepping and staggered nonlinear solver.

```
for load step t = 1 to t_max do
    iteration ← 0
    stoppingCriterion ← false
    while stoppingCriterion = false do
        assemble and solve elastic system
        update H^+
        assemble and solve damage system
        if iteration > 0 then
            stoppingCriterion ← check_convergence(solution,
            solution_old)
        end if
        solution_damage_old ← solution_damage
        solution_elastic_old ← solution_elastic
        if stoppingCriterion = false then
            refine_grid()
        end if
        iteration ← iteration + 1
    end while
    post_process_solution(t)
end for
```

2.3. Spatial discretization and adaptive mesh refinement

As mentioned, (7) and (8) are both linear partial differential equations. We utilize the finite element method for spatial discretization (for details, see Section 3), using the same mesh for both the displacement and the phase field variables. Specifically, we use trilinear, continuous elements for both variables. The discretization utilizes a computational mesh composed of hexahedra that we modify in each nonlinear iteration via adaptive mesh refinement (AMR) to optimize computational efficiency; the resulting mesh is refined only in regions where the crack exists or is expected to propagate. Our AMR strategy consists of the following steps:

1. *Error estimation*: We first estimate some measure of error on each cell. Specifically, we use the `KellyErrorEstimator` class of `deal.II` (see [59,60]), which provides a measure of how well each cell of the mesh is suited to approximating the damage field. Because damage accumulates in places where stresses are large, a mesh adapted to resolving the phase field is also well suited to resolving the displacement field.
2. *Cell marking*: We mark a certain top percentile of cells with the largest error estimates for refinement.
3. *Refinement and coarsening*: Marked cells are refined using isotropic cell division. As a result of this step, the mesh density is adjusted where needed.
4. *Solution transfer*: Solutions from the previous mesh are interpolated onto the refined mesh.

2.4. Parallel computing

Fracture problems are only really relevant in three dimensions as that is the dimensionality in which the objects engineers want to simulate tend to live. On the other hand, in three-dimensions appropriately resolving the phase-field length scale typically requires meshes with hundreds of thousands, millions, or even more cells – resulting in problems that can no longer be efficiently solved on a single computer. As a consequence, our implementation of all of the steps outlined above needs to use parallel algorithms that scale to both large problem sizes and to large numbers of processor cores.

3. Implementation in `deal.II`

Let us now move on to a discussion of the implementation of the phase-field fracture model described above within the finite element framework of `deal.II` [17,61]. `deal.II` is a widely used, open source finite element framework written in C++.

As discussed in Section 2.1, our solution scheme involves iterating between the solution of two equations for the displacement field and the phase-field variable; the mesh is adapted after every iteration until convergence is achieved. We will discuss our implementation in the following sections in three steps: First, we describe the implementation of the discretization and staggered solver. Next, we discuss how we incorporate the Message Passing Interface (MPI) in our implementation for a solver that can run in parallel. Finally, we outline how we incorporate Adaptive Mesh Refinement (AMR) in our code.

In the explanations below, we will frequently refer to one or the other of `deal.II`'s extensively documented tutorial programs in which the relevant functionality is discussed and demonstrated. The `deal.II` tutorial enumerates these programs as step-1, step-2, etc. Our code follows the general structure of these programs, in particular using a single principal class; a function called `run()` that contains the top-level loops; and using the common naming scheme for functions and variables. Our code will consequently look familiar to everyone who has read through the first few tutorial programs. Table 1 at the end of this section also summarizes which tutorial programs we drew from in writing the code, along with the modifications we needed to make.

Our code is licensed under the Lesser GNU Public License (LGPL) version 2.1 or later, allowing for re-use in a wide range of scenarios.

3.1. Discretization framework

We have to discretize and solve two principal equations iteratively for our scheme. The first describes the elastostatic equilibrium (7), and can be rewritten into

$$\nabla \cdot \left(g(d^i)\mathbb{C}\boldsymbol{\varepsilon}(\boldsymbol{u}^{i+1}) \right) = \mathbf{0}, \tag{11}$$

to make clear that we consider it as an equation for the displacement $\mathbf{u}^{i+1}$ in a one-field displacement formulation. Here, $\mathbb{C}$ is the tensor that relates strain to stress. The equation is a standard elliptic equation not dissimilar to the Poisson equation, though it is vector-valued. The second key component is Eq. (8) that describes the damage field d^{i+1} and that can be rewritten as

$$-\nabla^2 d^{i+1} + \frac{(1 + 2\mathcal{H}^{+,i+1} l/G_c)}{l^2} d^{i+1} = \frac{2}{G_c l} \mathcal{H}^{+,i+1} \tag{12}$$

to make clear that it is a Helmholtz equation. Since the coefficient in front of d^{i+1} is strictly positive, the equation is also elliptic.

Both of these equations are easy to discretize using the finite element method. In our implementation, we lean heavily on the fact that these two equations are solved in the step-8 and step-7 tutorial programs of the `deal.II` library, respectively. These are two exceedingly well documented programs with dozens of pages of explanation for the mathematical background and the actual implementation. As a consequence, we do not feel that it is necessary to discuss their general structure or approach and consequently only comment on the changes necessary to accommodate the specifics of our problem.

Compared to the implementation in step-8, the key change is the incorporation of the damage factor $g(d^i)$ in (11) when assembling the stiffness matrix. This factor needs to be computed at each quadrature point; this is straightforward as the factor only depends on the previous iteration's damage variable d, which is a known quantity when solving the momentum balance equation. Evaluating a finite element field at quadrature points of a cell is efficiently done with the `FEValues::get_function_values()` function.

Finally, let us also discuss how we deal with the history variable $\mathcal{H}^+$. Strictly speaking, $\mathcal{H}^+$ is a spatially variable function like $\mathbf{u}$ and d, and

Table 1
Details of the `deal.II` specific changes to the load stepping and staggered nonlinear solver of Algorithm 1, along with the `deal.II` tutorials from which parts of the algorithm are derived.

Step	Description
Solve Equation (1)	Based on `step-8` with following changes: – Incorporate $g(d^i)$ to degrade stiffness matrix – Replace `Vector` and `SparseMatrix` objects with `PETScWrappers::MPI::Vector` and `PETScWrappers::MPI::SparseMatrix` objects or Trilinos equivalents – Use CG solver with AMG preconditioner
Update $\mathcal{H}^+$	Postprocess from u^{i+1} Store using `CellDataStorage`
Solve Equation (6)	Based on `step-7` with following changes: – Compute α and f using Eq. (12) – Replace `Vector` and `SparseMatrix` objects with `PETScWrappers::MPI::Vector` and `PETScWrappers::MPI::SparseMatrix` objects or Trilinos equivalents – Use CG solver with AMG preconditioner
Refine Grid	Use `KellyErrorEstimator` on d^{i+1} Interpolate solutions from old mesh to new mesh (adapted from `step-15`, `step-26`) Interpolate $\mathcal{H}^+$ using `CellDataStorage`, `TransferableQuadraturePointData`, and `ContinuousQuadratureDataTransfer` classes

it could be discretized in the same way. At the same time, $\mathcal{H}^+$ is not described by a partial differential equation, but the point-wise equality (5) and so there is no expectation that it will, for example, have to be a continuous function of space. Moreover, in the phase-field model, $\mathcal{H}^+$ appears as a coefficient and in the right hand side term of the damage equation; in practice, this means that it is only needed at *quadrature points* during assembly of the damage linear system, rather than as a function defined everywhere. As is common in computational mechanics, we therefore do not store $\mathcal{H}^+$ as a finite element field defined at node points, but instead as values defined at quadrature points. The necessary functionality is provided in `deal.II` via the `CellDataStorage` class, which associates scalar or vector-valued data with each cell in the mesh, and specifically with the quadrature points on each cell.

3.2. Considerations for parallel computing based on MPI

The key consideration in writing programs that can run in parallel on clusters is that every data structure that is used in a finite element program must be split up so that every MPI (Message Passing Interface) process only stores a part it "owns", along with perhaps a small amount of data on adjacent cells (the "ghost cells"). We then say that this data structure is "distributed" across the collection of MPI processes. As a consequence of this approach, all operations where things are added up – say, the assembly of the system matrix and right hand side – require each process to only compute on that part of the input it actually owns (say, the "locally owned" cells), followed by a reduction step where if a process wants to add its result to an element of the output data structure that it does not own itself, that result is instead sent to the owning process for addition. `deal.II` has had the ability to support this programming paradigm for nearly twenty years. The key tutorial program that explains the concepts we use in our implementation of the phase-field method is step-40, which solves the Poisson equation in a parallel distributed computing environment. We follow essentially the same approach in our implementation. Specifically, we utilize the building blocks discussed in the following.

Triangulations. We use the `parallel::distributed::Triangulation` mesh class that distributes its cells across multiple processes. This class partitions the mesh such that each process owns a roughly equally-sized portion of the domain. A layer of ghost cells is used to handle the overlap needed for communication between processes. This class utilizes the `p4est` [62] library to handle partitioning and parallel adaptive mesh refinement.

Refining and coarsening a mesh also triggers a re-balancing operation that ensures that each process in a parallel computation always has a roughly equal share of both the overall workload and the overall memory usage, even on locally refined meshes. The scalability of this approach to very large computations is described in detail in [63].

Linear systems. We replace the `Vector` and `SparseMatrix` classes of step-7 and step-8 by the `PETScWrappers::MPI::Vector` and `PETScWrappers::MPI::SparseMatrix` classes, or their `Trilinos` equivalents; both `PETSc` [64,65] and `Trilinos` [66,67] support parallel operations and our code uses whatever the underlying `deal.II` library is configured with. These classes distribute data across MPI processes, meaning each process stores only a portion of a global vector or matrix.

Where necessary, vectors can be instructed to also store vector entries that correspond to (remotely owned) degrees of freedom located on ghost cells or at the interface between locally owned and ghost cells. Such vectors are called "ghosted" vectors and are needed whenever it is necessary to evaluate the solution on locally owned cells (which may not own the degrees of freedom at an interface to a ghost cell).

In practice, managing where to use ghosted and where to use non-ghosted vectors is a frequent source of complexity in distributed-memory parallel computing. This is because some operations can only be performed on one kind of vector whereas other operations require the opposite kind. In practice, *computing* a vector – say, as the solution of a linear system or during the assembly of the right hand side – generally requires vectors that have no ghost elements because the underlying operations only ever *write* into a vector (or *add* to its elements). On the other hand, *using* a vector, for example to postprocess the solution to obtain load-displacement curves or for graphical output – requires *reading* elements that correspond to degrees of freedom on locally owned or ghost cells and therefore requires vectors with ghost elements.

In our program, we generally store information only in one kind of vector. For example, the right hand side vector is never read from, and so it is stored as a non-ghosted vector. The solution vector is computed as a non-ghosted vector during the solution of the linear system, but it is immediately converted into a ghosted vector which then serves as the input for both the assembly of the next linear system and for post-processing. Ensuring that our code only ever stores information in one kind of vector also avoids the common issue of duplicated information going out of sync, and the difficult-to-find bugs that result.

Assembly of system matrices and right hand sides. Following the paradigm outlined above, parallel assembly of the system matrix and right-hand side vector requires that each process assembles the contributions only on those elements it owns. This includes contributions to matrix and vector entries owned by different processes, and these contributions are communicated between processes in a process called "compression" at the end of assembly.

Parallel solvers. In parallel programs, every MPI process only stores a subset of the rows of a matrix, and a subset of the elements of vectors.

This puts limitations on what solvers one can use – for example, direct solvers computing lower-upper (LU) decompositions of a matrix can not be implemented efficiently, whereas iterative solvers that only ever use matrix-vector products do work efficiently. As a consequence, we use such iterative methods – specifically, we use the Conjugate Gradient (CG) method for both the elasticity and the damage linear systems given that the matrices involved in both linear systems are symmetric and positive definite.

Parallel preconditioners. Most iterative linear solvers are only efficient if paired with good preconditioners. As with solvers, parallel programs challenge the design of preconditioners because preconditioning operations must run in parallel and cannot require access to data stored elsewhere. As a consequence, many common preconditioners such as symmetric successive over-relaxation (SSOR) are not an option. Instead, we rely on the parallel implementations of algebraic multigrid (AMG) methods that are available via `PETSc` [64,65] and `Trilinos` [66,67]. AMG is a methodology that builds a hierarchy of matrices, each of which is a lower-dimensional approximation of the next higher-dimensional one. Specifically, we use the `hypre` [68] implementation of AMG if `deal.II` was configured to use `PETSc`, and `ML` [69] or `MueLu` [70] when using `Trilinos`-based linear algebra. All of these are highly optimized, parallel implementations of the AMG idea that are widely used as black-box preconditioners for symmetric and positive definite matrices such as those we are solving with for both of the linear systems we consider.

Communication. An important part of parallel programs is that they have to exchange (communicate) data computed locally on one process but required elsewhere on a different process. A specific kind of communication is *synchronization*, where processes exchange the information that they have all reached a specific point in the program, only after which point they can continue with their work.

In practice, `deal.II` handles almost all necessary communication and synchronization. However, there are places where we post-process the solution – notably for computing the load-displacement curve where we need to compute integrals to compute a traction force; these integrals are over all boundary faces, which may be owned by different processes. In such situations, every MPI process only computes the contributions to the integral from the boundary faces of cells it owns, after which the complete integral is computed by explicitly summing up the contributions from all processes via an MPI reduction operation.

3.3. Considerations for incorporating adaptive mesh refinement

As outlined in Algorithm 1 and in Section 2.3, we adapt the mesh after each cycle of solving for the displacement $\mathbf{u}^i$, the accumulated history field $\mathcal{H}^{+,i}$, and the damage field d^i. We do this so that the mesh continues to accurately resolve these variables without the need for a globally refined mesh that is almost certainly going to require far more cells to reach the same resolution.

As for parallelization, `deal.II` provides many of the tools to make this possible, but it is worth discussing the specific implementation details one has to consider. We will do so in the following sub-sections.

Choices for mesh refinement. One can refine finite element meshes in a number of different ways [71]. For triangular and tetrahedral meshes, one often uses red-green refinement or longest-edge bisection to obtain meshes that remain conforming, i.e., for which every face of a cell not at the boundary of the domain is a *complete* face of a neighboring cell. This is not easily feasible with quadrilateral and hexahedral meshes like the ones we use here. Instead, we use `deal.II`'s approach of splitting each cell into four children (for quadrilaterals in 2d) or eight children (for hexahedra in 3d) via bisection in each of the cells' principal directions. This results in meshes with "hanging nodes" in which neighboring cells may differ in size by one refinement level. (We will show examples of how such meshes look like in Section 4.) We will briefly discuss how to treat hanging nodes in the next section.

Our approach to choosing which cells to refine is based on an "error indicator" that provides an estimate of how large the error may be on a given cell. Specifically, we use the error criterion by Kelly et al. [59] applied to the damage field to determine which cells should be refined, as implemented in the `KellyErrorEstimator` class of `deal.II`. This criterion computes an estimate of the second derivative of a finite element field, times a power of the mesh size, as the resulting quantity is the dominant factor in estimating the interpolation error, and is widely used in the literature for the quality of the locally refined meshes it produces.

Ensuring continuity of the solutions. As mentioned above, the scheme for mesh refinement we utilize results in meshes with "hanging nodes" at which, unless special care is taken, finite element functions would be discontinuous because the shape functions associated with vertices, mid-edge, or mid-face nodes do not match from the two adjacent cells. On the other hand, we are using continuous finite elements in the discretization of both $\mathbf{u}$ and d, and we need to ensure continuity of the discretized functions even at hanging nodes. We enforce the continuity requirement via *constraints* that augment the linear systems we obtain via assembly. These constraints are stored in an `AffineConstraints` object (along with other constraints resulting from boundary conditions, for example), and obtained via the `DoFTools::make_hanging_node_constraints()` function after mesh refinement. Our approach fundamentally follows the example set in `deal.II`'s step-6 tutorial program and its extension to parallel computations in step-40.

Interpolating solutions from the old to the new mesh. In the finite element method, solutions are piecewise polynomial functions defined by their nodal values – typically the values at vertices and interpolation points along edges, faces, and in cell interiors. When the mesh is changed through mesh adaptation, it is necessary to transfer solutions from the old to the new mesh. This can be done in different ways, one of which is to *interpolate* solutions from the old to the new mesh; this is made particularly convenient in `deal.II` because the old and new meshes are related by the fact that mesh refinement replaces a cell by its isotropically refined children, and mesh coarsening replaces a number of small cells by their erstwhile parent cell – all of which guarantees that the new nodal points at which we need to evaluate the old solution have known locations in the old mesh.

In practice, the interpolation is achieved by using the `SolutionTransfer` class in ways that resemble its use in step-15 and step-26.

Interpolating the history variable $\mathcal{H}^+$ from the old to the new mesh. As discussed in Section 3.1, we do not store the history variable $\mathcal{H}^+$ as a finite element field, but rather as values associated with quadrature points. This poses questions when adapting the mesh because we need $\mathcal{H}^+$ at the quadrature points of the new mesh, and these generally do not line

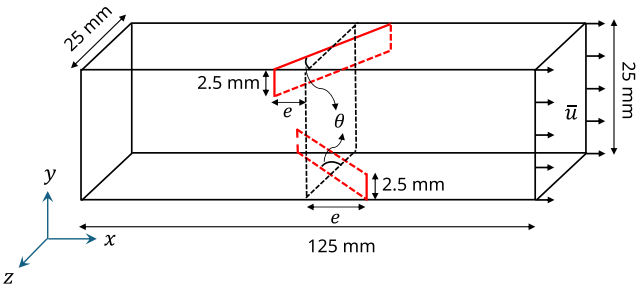


Fig. 1. Validation test case: Geometry, boundary conditions and initial cracks for the 3D double-notched test of Muixí et al. [2]. The angle θ between the notches and the plane $x = 62.5$ mm is $\pi/6$, leading to an eccentricity of $e = 12.5 \times \tan(\pi/6)$ mm.

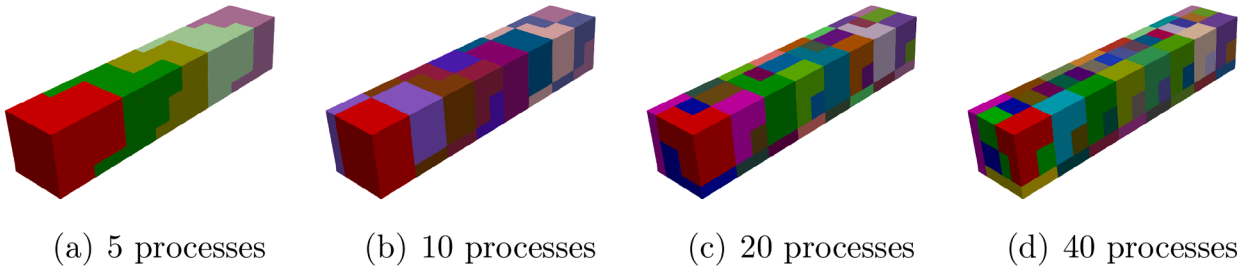


Fig. 2. Validation test case: Domain decomposition using different numbers of processors. Each color represents that part of the overall domain owned by one process.

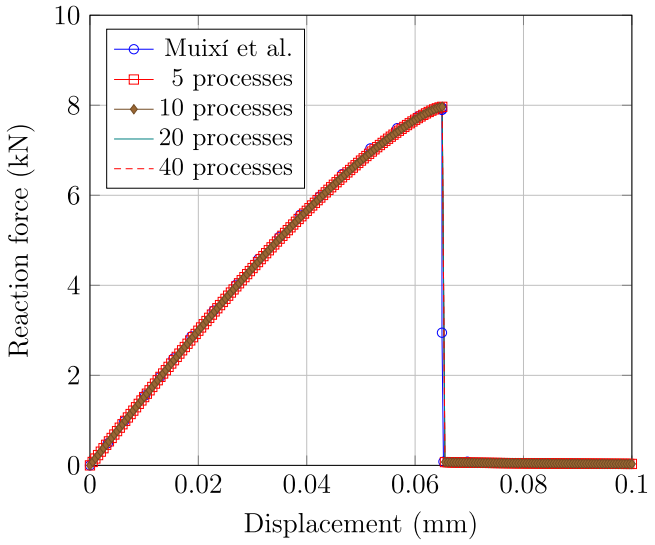


Fig. 3. Validation test case: Comparison of load-displacement curves. We compare our results using 5, 10, 20, and 40 processors with the results from Muixí et al. [2]. The results are clearly in excellent agreement.

up with the quadrature points of the old mesh. Conceptually, the most reasonable approach to dealing with the issue is to first find a piecewise polynomial function that matches or approximates the values stored at the quadrature points of the old mesh, interpolate this function to the new mesh, and then evaluate the function at the quadrature points of the new mesh.

It is clear that it is not trivial to implement such an approach, in particular if both the new and old mesh are partitioned across MPI processes. In practice, the `CellDataStorage` class we use to store $\mathcal{H}^+$ can be combined with the `TransferableQuadraturePointData` and `ContinuousQuadratureDataTransfer` classes to facilitate this process, though there are at the moment no tutorial programs that illustrate how this can be done. Consequently, we think of our program also as a contribution to the documentation of `deal.II` showing how to achieve transfer of quadrature point data between meshes.

4. Results

Having discussed details of the implementation of our program in the previous section, let us now demonstrate how it works in practice. To this end, we will consider two test cases: (i) In Section 4.1 a three-dimensional crack propagation problem that has been considered in the literature before and for which we consequently have a known solution against which we can compare to validate our implementation's correctness; and (ii) in Section 4.2 a test case that results in complex fracture patterns and that we can utilize to demonstrate our code's ability to solve complicated, real-world examples typical of what applications look like.

4.1. Validation test case: Fracture propagation in a 3D beam

As a validation example, we consider a 3D problem discussed by Muixí et al. in [2] and shown in Fig. 1. The setup is a beam with a square section $\Omega = [0, 125] \times [0, 25] \times [0, 25]\,\text{mm}^3$ with two notches on the surfaces $y = 0\,\text{mm}$ and $y = 25\,\text{mm}$ as shown in Fig. 1. The notch on the surface $y = 25\,\text{mm}$ is oriented at an inclination of $\theta = \pi/6 = 30°$ with respect to the $y - z$ plane in the counter-clockwise direction, while the notch on the surface $y = 0\,\text{mm}$ is oriented at the same inclination, but in the clockwise direction. The beam is clamped along the left end (at $x = 0\,\text{mm}$), and prescribed displacements are applied incrementally in the x-direction at the right end (at $x = 125\,\text{mm}$). We use load steps of $\Delta u = 5 \times 10^{-4}\,\text{mm}$ and $\text{tol} = 10^{-2}$ as the numerical tolerance for convergence. The Young's modulus, Poisson's ratio, and critical energy release rate are $E = 32\,\text{GPa}, \nu = 0.25, G_c = 1.6 \times 10^{-4}\,\text{kN/mm}$. We choose the characteristic length scale l to be 2 mm. We validate the correctness of our implementation by comparing the damage profiles and the displacement-load curves against the results shown in [2].

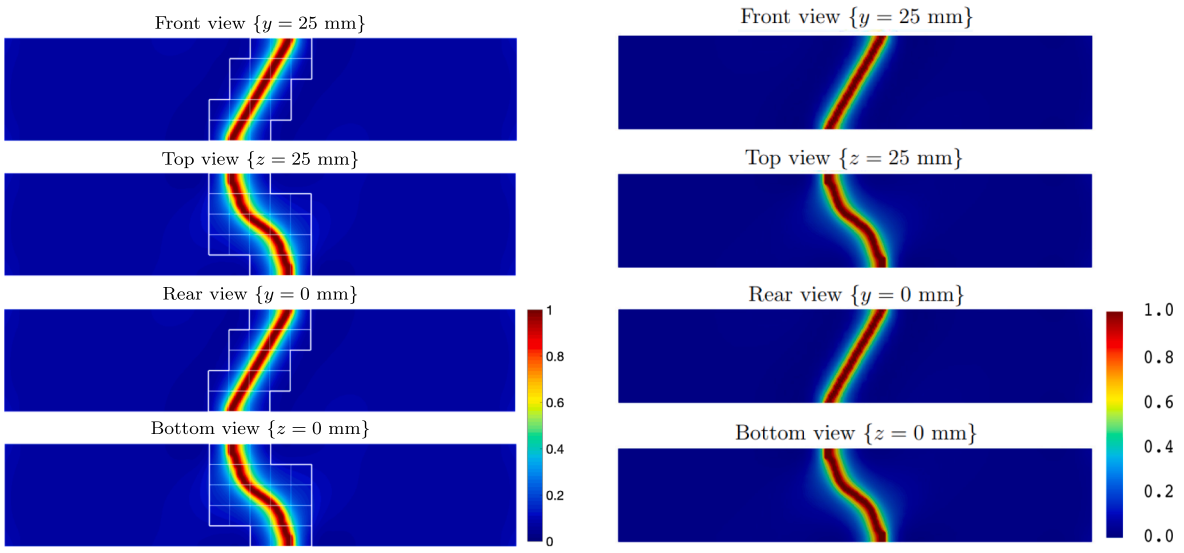
4.1.1. Validating the correctness of our MPI implementation

To validate our parallel framework, we first consider a uniform mesh of $256 \times 40 \times 40 = 409\,600$ trilinear hexahedral elements and compare the results of our code using increasingly larger numbers of processors. This mesh ensures that the phase-field length scale l is appropriately resolved as the ratio l/h (h being the mesh size) is around 3. The general recommendation is to use an l/h ratio between 2–5 (see Wu et al. [72]). The domain decomposition for 5, 10, 20, and 40 processors is shown in Fig. 2. Fig. 3 shows a comparison of load-displacement curves against the results of Muixí et al. [2]. It is clear that there is excellent agreement between our implementation and that of Muixí et al. In particular, it is reassuring (and expected) that the number of MPI processes used has no influence on the results – it should, after all, only affect how *long* the program runs, not *what* it computes.

Fig. 4 shows the damage profiles for the front, back, top, and bottom surfaces of the domain obtained in a computation with 40 processes, compared against the results of Muixí et al. [2]. Again, there is excellent agreement between our implementation and theirs.

4.1.2. Validating the convergence of results with mesh refinement

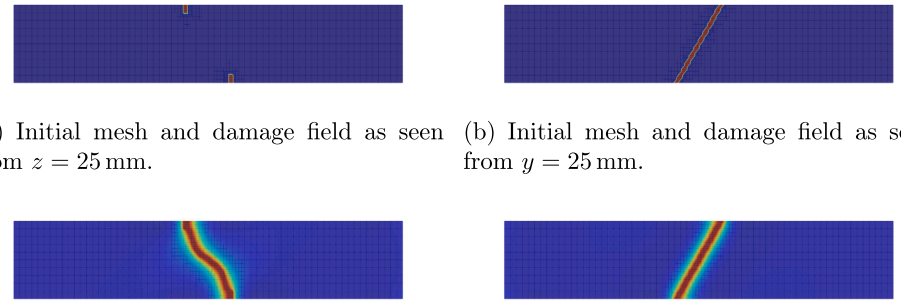
Having validated the parallel framework on the fixed mesh of the previous section, we now perform a mesh convergence study by comparing results from four simulations using uniform meshes of increasing resolution: $64 \times 10 \times 10$, $128 \times 20 \times 20$, $256 \times 40 \times 40$, and $512 \times 80 \times 80$. Fig. 7 presents the corresponding load-displacement curves. We observe that the curves for the coarser meshes ($64 \times 10 \times 10$ and $128 \times 20 \times 20$) deviate noticeably from the benchmark solution obtained earlier using the $256 \times 40 \times 40$ mesh. This discrepancy is expected, as these coarser meshes have a length scale ratio $l/h < 2$ and therefore lack the resolution needed to adequately capture the phase-field length scale. In contrast, the results for the finest mesh ($512 \times 80 \times 80$) closely overlap with those of the $256 \times 40 \times 40$ mesh. This indicates that the solution has converged and that further mesh refinement beyond $l/h \geq 3$ does not yield



(a) Damage profiles from the PhD thesis of Muixí

(b) Damage profiles from the current implementation using 40 processors.

Fig. 4. Validation test case: Comparison of damage profiles on different faces of the domain for an applied displacement of 0.066 mm.



(a) Initial mesh and damage field as seen from $z = 25\,\text{mm}$.

(b) Initial mesh and damage field as seen from $y = 25\,\text{mm}$.

(c) Final mesh and damage field as seen from $z = 25\,\text{mm}$.

(d) Final mesh and damage field as seen from $y = 25\,\text{mm}$.

Fig. 5. Validation test case: Initial (top row) and final meshes (bottom row), along with the damage profile of a computation using adaptive mesh refinement, as seen from $z = 25\,\text{mm}$ (left) and $y = 25\,\text{mm}$ faces (right).

any significant improvement in accuracy. Overall, this study confirms that our implementation converges under mesh refinement, and demonstrates that a resolution of $l/h \geq 3$ is sufficient to accurately resolve the length scale in phase-field fracture simulations. Further refinement beyond this threshold offers minimal additional benefit.

Staggered schemes to iterate out the nonlinearity of the coupled model, such as the scheme we use herein, are often considered inefficient compared to monolithic methods such as those in [35]. Yet, for the benchmark of this section, the computations on the $256 \times 40 \times 40$ mesh always converge in two iterations (one to solve the system, one to realize that the error criterion is now below the threshold) with the exception of a single load step in which our method requires 25 iterations to achieve a convergence tolerance of 10^{-2}; that one load step is the one in which the specimen fails. In other words, the cost for iterating out the nonlinearity instead of tackling it directly in a monolithic approach is clearly acceptable. If crack growth occurred in a more controlled manner, rather than in a single load-step, a staggered approach could indeed be much more expensive as discussed in [46].

4.1.3. Validating the adaptive mesh refinement implementation

Next, we validate the AMR implementation by solving the above problem using an adaptively refined mesh. While adaptive mesh refinement is often seen as selectively *increasing* mesh resolution in certain parts of the domain, it can also be seen as selectively *decreasing* resolution where high resolution is not necessary, and consequently dramatically reducing the computational effort without compromising accuracy. In order to test this, let us examine the accuracy we obtain from the meshes shown in Fig. 5 that have the same cell size for the *smallest* cells as the uniformly refined mesh of the previous section, though the vast majority of cells is far coarser. For example, the initial mesh has only 11048 cells compared to 409 600 cells of the uniformly refined mesh. As the crack propagates, the mesh adaptively refines in regions with high damage gradients; the final mesh has 24278 cells.

The final damage profile along with the mesh is shown in Fig. 5. These damage profiles are in good agreement with the results of Muixí et al. [2] and those in Fig. 4. We compare the load-displacement in Fig. 6, again showing a largely comparable behavior, with the peak loads and maximal displacements before failure having differences of 1% and 3%, respectively. These small differences should perhaps not be too surprising given that the computational problem is 20 to 40 times smaller than the one of the previous section, and correspondingly faster. It is also worth emphasizing that the accuracy might be further improved by utilizing more sophisticated adaptive mesh-refinement strategies such as

the predictor-corrector strategy advocated in [35]. However, the Kelly error estimator-based AMR utilized here provides a good trade-off between accuracy, cost, and implementational ease.

4.2. The oreo test case: A three-layered medium under biaxial loading

Having convinced ourselves that our implementation of the model matches results previously presented in the literature, let us turn to our second test case: An example where we showcase the complexity of situations we can simulate with our code. Specifically, we investigate fracture propagation in a three-layered medium, subjected to biaxial loading. The problem setup, shown in Fig. 8, consists of a rectangular prism $\Omega = [0, 30] \times [0, 30] \times [0, 13]\,\text{mm}^3$ with two planar material interfaces at $z = 5\,\text{mm}$ and $z = 8\,\text{mm}$ that separate the specimen into three layers. All three layers are homogeneous with Young's modulus $E = 37.5\,\text{GPa}$ and Poisson's ratio $\nu = 0.25$. The top and bottom layers have a critical energy release rate $G_c = 1 \times 10^{-3}\,\text{kN/mm}$, while the middle layer is considered more brittle by a factor of 25 such that its critical energy release rate is

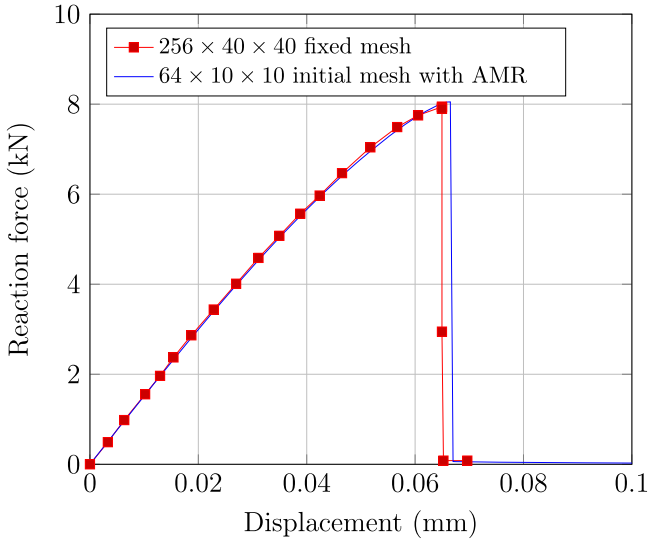


Fig. 6. Validation test case: Comparison of load-displacement curves for two computations, one using a fixed mesh of $256 \times 40 \times 40$ hexahedra and the other using an initial mesh of $64 \times 10 \times 10$ hexahedra which is then adaptively refined.

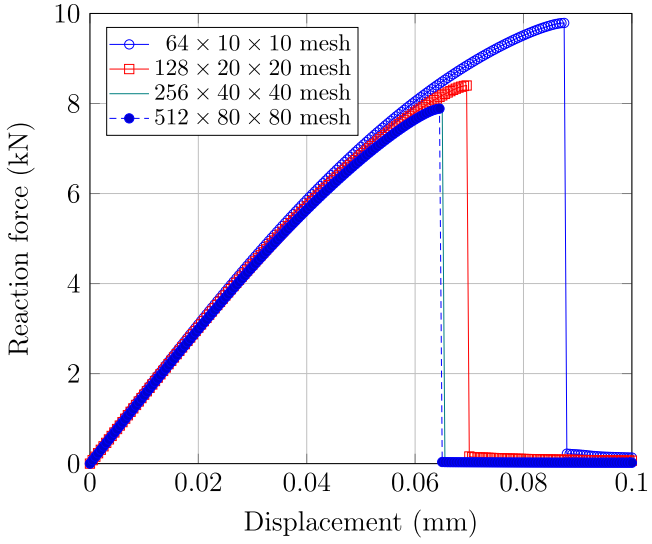


Fig. 7. Mesh convergence study: Comparison of load-displacement curves for four computations, each using a fixed mesh of increasingly higher resolution.

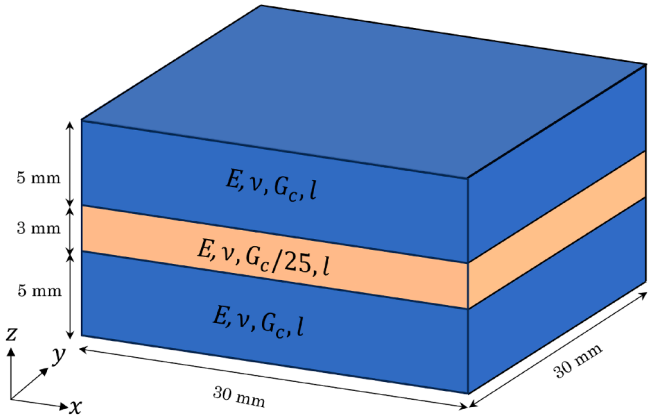


Fig. 8. Oreo test case: Problem setup, showing the geometry and the material properties for the Oreo test. The specimen is pulled in positive x and y-direction. Displacement constraints are applied on the $x = 0\,\text{mm}$ and $y = 0\,\text{mm}$ faces to prevent rigid body motion.

$G_c^{\text{m}} = 4 \times 10^{-5}\text{kN/mm}$. The phase-field characteristic length l is chosen to be the same in all three layers, with $l = 0.6\,\text{mm}$. Because the test case resembles the make-up of a common sweet treat available globally – in which a weak layer is sandwiched between two tougher layers –, we call this example the "Oreo test case".

Incremental extensional displacements $\Delta u = 1 \times 10^{-3}\,\text{mm}$ are applied along the positive x- and positive y-direction on the surface $x = 30\,\text{mm}$ and $y = 30\,\text{mm}$, respectively. To prevent rigid body motion, normal displacements are constrained for the faces $x = 0\,\text{mm}$ and $y = 0\,\text{mm}$, while the degrees of freedom along the line defined by the $z = 6.5\,\text{mm}$ and $x = 0\,\text{mm}$ are constrained in the z-direction.

Fig. 9 shows a visualization of the damage that results from the loading of the specimen. It clearly shows the complexity of the fracture network induced.

In the following, let us present results obtained on both a fine uniform, and an adaptively refined mesh, that will illustrate both the complexity of simulations possible with our code and its parallel scalability to these kinds of large computations.

4.2.1. Parallel scalability on a uniform mesh

We first perform a strong scaling study on a fixed mesh with $160 \times 160 \times 80 = 2048\,000$ cells, without any adaptive mesh refinement. This problem has 6298 803 degrees of freedom for the linear momentum balance equation and about 2099 601 degrees of freedom for the damage evolution equation. We solve this fixed-size problem with different numbers of MPI processes for a single load step, while timing those parts of the program that occupy the most run time. We repeated each run five times to mitigate fluctuations and took the minimum in each category as our result. For our study, we used an in-house computing cluster. Each standard computing node is equipped with two Intel® Xeon® Gold 6140 processors (with 18 cores each, running at 2.30 GHz) and 132 GB of memory. Communication between nodes happens via an InfiniBand network operating at up 100GB/s. More information on the configuration of the machine can be found at https://cluster.karlin.mff.cuni.cz/.

Experience with other `deal.II`-based codes indicates that each MPI process should ideally have at least 100 000 unknowns in order to ensure that communication costs do not outweigh computation costs [63,73,74]. This suggests that for the problem sizes mentioned above, we should expect that run times are roughly proportional to one over the number P of MPI processes as long as $P \lesssim 20$ for the damage equation and $P \lesssim 60$ for the elasticity equation. We confirm this behavior with the timing results of our experiments displayed in Fig. 10; in fact, scaling seems to extend substantially further than these limits. In the example shown in the figure, the use of 128 MPI processes reduces the average time to solve one load step of the "for" loop in Algorithm 1 from

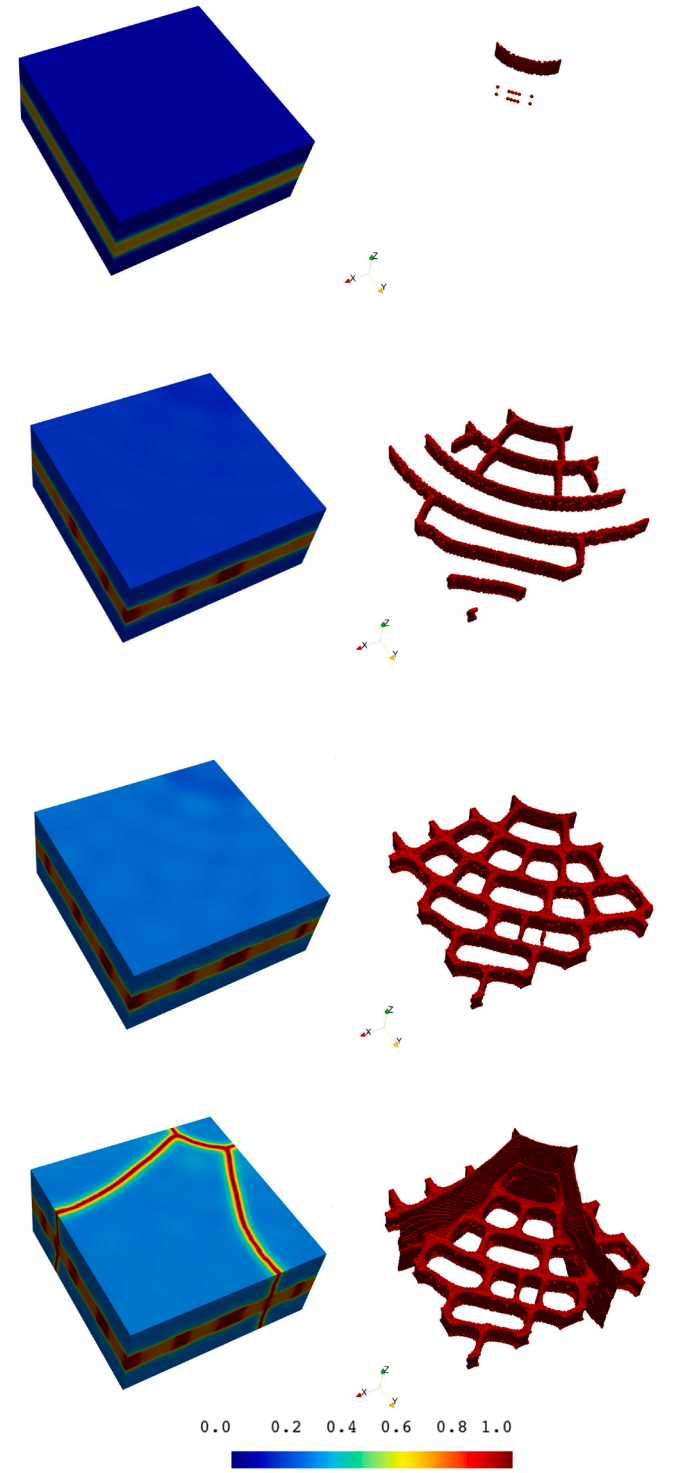


Fig. 9. Oreo test case: The left column shows the scalar damage field over the entire three-dimensional domain for applied displacements of 4×10^{-2} mm, 6×10^{-2} mm, 8×10^{-2} mm and 9×10^{-2} mm (top to bottom). The right column shows the corresponding crack surface plotted by visualizing the damage field only in regions where it exceeds a threshold value of $d_* = 0.9$. We use a $20 \times 20 \times 10$ uniform initial mesh, followed by AMR.

slightly more than one hour to just under a minute, a speed-up of about 70.

4.2.2. Results for a parallel simulation on an adaptively refined mesh

We conclude by showcasing the computational efficiency of our framework, utilizing both AMR and MPI. Starting with a coarse initial mesh of $20 \times 20 \times 10$ cells, adaptive refinement is applied as the damage evolves. This process grows the number of cells from 4000 cells to 432 006 cells in the final refined mesh, shown in Fig. 11. This final mesh has the same minimal cell size as the uniformly refined one, but has only about one fifth the number of cells. Furthermore, due to the brittle fracture nature of the problem, most mesh refinement occurs late in the simulation as the fracture network grows, allowing a coarse mesh to be used for the majority of the computation. Combining the savings of adaptive meshes and parallel processing, the simulation using adaptive mesh refinement takes only 983 seconds (approximately 16 minutes) running on 40 MPI processes; the same simulation using a uniformly refined mesh and a single MPI process takes 433 856 seconds (approximately 5 days), highlighting the potential for combining AMR and parallel processing to solve very large problems.

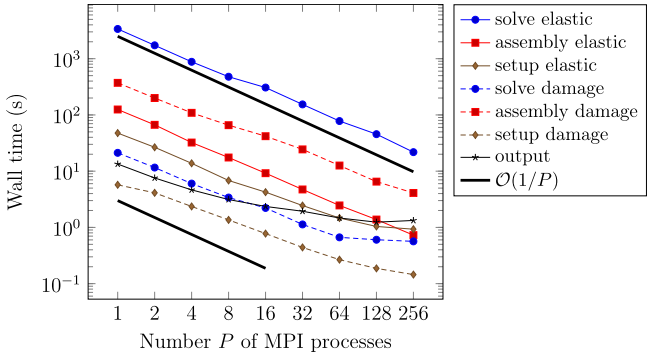


Fig. 10. Oreo test case: Strong scaling results showing the run time for solving a fixed-size problem as a function of the number P of MPI processes used. The run times for each operation are corresponding to the two iterations of the first load step of the simulation. Until $P = 16$, all MPI processes fit onto a single node; beyond that, we utilize several nodes running 16 MPI processes each.

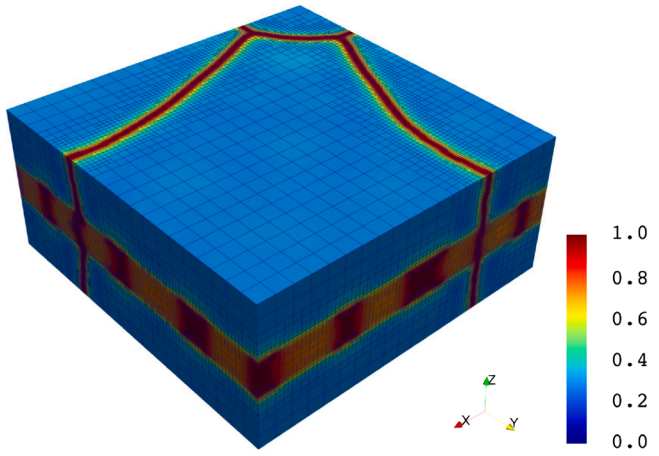


Fig. 11. Oreo test case: Final mesh and damage field on the adaptively refined mesh of Section 4.2.2, showing the highly resolved mesh around fractures.

5. Conclusions

Herein, we have presented the design and implementation of an open-source code that solves a widely-used formulation of fracture propagation using the phase-field method. It couples an equation of quasi-static elastic equilibrium driven by stepped loads with a model that describes damage that the material has incurred as a result of the deformation, and alternates solving these two equations until convergence is achieved within each load step. We have also provided numerical evidence that our computations yield simulation results which match those that can be found in the literature. Our numerical experiments also show

that, using adaptive mesh refinement and parallel computing based on MPI, we can efficiently solve large three-dimensional problems of substantial complexity, using many millions of degrees of freedom.

Our goals for this manuscript were (i) to demonstrate how one can implement solvers for phase-field-based fracture models with `deal.II`, and (ii) to provide a basis for future experiments by other members in our community. Indeed, while the iterative algorithm we use – alternating between solving the elasticity and damage equations – could likely be improved upon to achieve faster convergence using algorithms such as those described in [36], our implementation has the advantage of flexibility for future extensions of this program. In particular, it is easy to exchange the equations that describe damage d as a function of the history variable $\mathcal{H}^+$, to include anisotropy or nonlinearity into the elasticity equation, or indeed to replace elasticity by plasticity or other forms of anelastic behavior. By compartmentalizing the solvers for $\mathbf{u}$, $\mathcal{H}^+$, and d, the code can also easily use much more complicated descriptions of how damage is actually created by deformation.

We look forward to seeing the ways in which our code will serve as a resource for the community.

CRediT authorship contribution statement

Wasim Niyaz Munshi: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Formal analysis; **Marc Fehling:** Writing – review & editing, Visualization, Software; **Wolfgang Bangerth:** Writing – review & editing, Writing – original draft, Software, Formal analysis; **Chandrasekhar Annavarapu:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Data availability

The code is shared at a public url and is licensed under LGPL license

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

W.N.M. gratefully acknowledges the Prime Minister's Research Fellowship (Project No. SB23242064CEPMRF008957) by the Ministry of Education, Government of India, and the Fulbright Nehru Doctoral Research Fellowship (Award No. 3047/FNDR/2024-2025), funded by United States India Education Foundation (USIEF).

M.F. gratefully acknowledges support by the ERC-CZ grant LL2105 CONTACT, funded by the Czech Ministry of Education, Youth and Sports. M.F. was also supported by the Charles University Research Centre Program No. UNCE/24/SCI/005.

W.B. and M.F. gratefully acknowledge support by the National Science Foundation through awards OAC-1835673 as part of the Cyberinfrastructure for Sustained Scientific Innovation (CSSI) program. W.B. was also supported by the National Science Foundation under awards EAR-1925595 and OAC-2410847.

C.A. gratefully acknowledges the support from the Ministry of Education, Government of India and IIT Madras to the Subsurface Mechanics and Geo-Energy Laboratory under the grant SB20210856CEMHRD008957.

Charles University is acknowledged for providing computing time on the Sněhurka cluster.

References

[1] T. Noll, C. Kuhn, D. Olesch, R. Müller, 3D Phase field simulations of ductile fracture, GAMM-Mitteilungen 43 (2) (2020) e202000008. https://doi.org/10.1002/gamm.202000008

[2] A. Muixí, S. Fernández-Méndez, A. Rodríguez-Ferran, Adaptive refinement for phase-field models of brittle fracture based on nitsche's method, Comput Mech 66 (1) (2020) 69–85. https://doi.org/10.1007/s00466-020-01841-1

[3] Y. Navidtehrani, C. Betegón, E. Martínez-Pañeda, A simple and robust abaqus implementation of the phase field fracture method, Applications in Engineering Science 6 (2021) 100050. https://doi.org/10.1016/j.apples.2021.100050

[4] G. Molnár, A. Gravouil, 2D And 3D abaqus implementation of a robust staggered phase-field solution for modeling brittle fracture, Finite Elem. Anal. Des. 130 (2017) 27–38. https://doi.org/10.1016/j.finel.2017.03.002

[5] D. Kosov, A. Tumanov, V. Shlyannikov, ANSYS Implementation of the phase field fracture approach, Frattura ed Integrità Strutturale 18 (70) (2024) 133–156. https://doi.org/10.3221/IGF-ESIS.70.08

[6] S. Zhou, T. Rabczuk, X. Zhuang, Phase field modeling of quasi-static and dynamic crack propagation: COMSOL implementation and case studies, Adv. Eng. Software 122 (2018) 31–49. https://doi.org/10.1016/j.advengsoft.2018.03.012

[7] W. Zhang, Z. Shen, J. Ren, L. Gan, L. Xu, Y. Sun, Phase-field simulation of crack propagation in quasi-brittle materials: COMSOL implementation and parameter sensitivity analysis, Modell. Simul. Mater. Sci. Eng. 29 (5) (2021) 055020. https://doi.org/10.1088/1361-651X/ac03a4

[8] Y. Navidtehrani, C. Betegon, E. Martinez-Paneda, A unified abaqus implementation of the phase field fracture method using only a user material subroutine, Materials (Basel) 14 (8) (2021) 1913. https://doi.org/10.3390/ma14081913

[9] M.A. Msekh, J.M. Sargado, M. Jamshidian, P.M. Areias, T. Rabczuk, Abaqus implementation of phase-field model for brittle fracture, Comput. Mater. Sci 96 (2015) 472–484. https://doi.org/10.1016/j.commatsci.2014.05.071

[10] P.C. Sidharth, B.N. Rao, Phase-field modeling of brittle fracture in functionally graded materials using exponential finite elements, Eng. Fract. Mech. 291 (2023) 109576. https://doi.org/10.1016/j.engfracmech.2023.109576

[11] P.C. Sidharth, B.N. Rao, Phase-field modeling of brittle fracture using automatically oriented exponential finite elements, Int. J. Fract. 242 (2) (2023) 169–189. https://doi.org/10.1007/s10704-023-00708-9

[12] G. Giudicelli, A. Lindsay, L. Harbour, C. Icenhour, M. Li, J.E. Hansel, P. German, P. Behne, O. Marin, R.H. Stogner, J.M. Miller, D. Schwen, Y. Wang, L. Munday, S. Schunert, B.W. Spencer, D. Yushu, A. Recuero, Z.M. Prince, M. Nezdyur, T. Hu, Y. Miao, Y.S. Jung, C. Matthews, A. Novak, B. Langley, T. Truster, N. Nobre, B. Alger, D. Andrs, F. Kong, R. Carlsen, A.E. Slaughter, J.W. Peterson, D. Gaston, C. Permann, 3.0 - MOOSE: Enabling massively parallel multiphysics simulations, SoftwareX 26 (2024) 101690. https://doi.org/10.1016/j.softx.2024.101690

[13] A. Logg, K.-A. Mardal, G.N. Wells, et al., Automated Solution of Differential Equations by the Finite Element Method, Springer, 2012. https://doi.org/10.1007/978-3-642-23099-8

[14] S. Badia, F. Verdugo, Gridap: an extensible finite element toolbox in julia, Journal of Open Source Software 5 (52) (2020) 2520. https://doi.org/10.21105/joss.02520

[15] C. Nguyen-Thanh, V.P. Nguyen, A. de Vaucorbeil, T.K. Mandal, J.-Y. Wu, Jive: an open source, research-oriented c++ library for solving partial differential equations, Adv. Eng. Software 150 (2020) 102925. https://doi.org/10.1016/j.advengsoft.2020.102925

[16] J.S.B. van Zwieten, G.J. van Zwieten, W. Hoitinga, Nutils v7.0, 2022. https://doi.org/10.5281/zenodo.6006701

[17] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells, The deal.II finite element library: design, features, and insights, Comput. Math. Appl. 81 (2021) 407–422. https://doi.org/10.1016/j.camwa.2020.02.022

[18] W. Jiang, T. Hu, L.K. Aagesen, Y. Zhang, Three-dimensional phase-field modeling of porosity dependent intergranular fracture in UO2, Comput. Mater. Sci 171 (2020) 109269. https://doi.org/10.1016/j.commatsci.2019.109269

[19] P.C. Sidharth, B.N. Rao, An open-source moose implementation of phase-field modeling of fracture in functionally graded materials, Procedia Structural Integrity 58 (2024) 115–121. https://doi.org/10.1016/j.prostr.2024.05.019

[20] Hirshikesh, S. Natarajan, R.K. Annabattula, A FEnics implementation of the phase field method for quasi-static brittle fracture, Frontiers of Structural and Civil Engineering 13 (2019) 380–396. https://doi.org/10.1007/s11709-018-0471-9

[21] D. Schneider, B. Nestler, et al., Realization of adaptive mesh refinement for phase-field model of thermal fracture within the FEnics framework, Eng. Fract. Mech. 293 (2023) 109676. https://doi.org/10.1016/j.engfracmech.2023.109676

[22] F. Freddi, L. Mingazzi, Adaptive mesh refinement for the phase field method: a FEnics implementation, Appl. Eng. Sci. 14 (2023) 100127. https://doi.org/10.1016/j.apples.2023.100127

[23] D.T. Nguyen, A. Gupta, R. Duddu, C. Annavarapu, An adaptive mesh refinement algorithm for stress-based phase field fracture models for heterogeneous media: application using FEniCS to ice-rock cliff failures, Finite Elem. Anal. Des. 244 (2025) 104311. https://doi.org/10.1016/j.finel.2024.104311

[24] S. Natarajan, R.K. Annabattula, E. Martínez-Pañeda, et al., Phase field modelling of crack propagation in functionally graded materials, Composites Part B: Eng. 169 (2019) 239–248. https://doi.org/10.1016/j.compositesb.2019.04.003

[25] M.M. Rahaman, An open-source implementation of a phase-field model for brittle fracture using gridap in julia, Math. Mech. Solids 27 (11) (2022) 2404–2427. https://doi.org/10.1177/10812865211071088

[26] V. Prakash, A.K. Behera, M.M. Rahaman, A phase-field model for thermo-mechanical fracture, Mathematics and Mechanics of Solids 28 (2) (2023) 533–561. https://doi.org/10.1177/10812865221085198

[27] A. Unnikrishna Pillai, M.M. Rahaman, A phase-field length scale insensitive model for fatigue failure in brittle materials, Int. J. Fatigue 196 (2025) 108875. https://doi.org/10.1016/j.ijfatigue.2025.108875

[28] A.K. Behera, A. Unnikrishna Pillai, M.M. Rahaman, A phase-field model for electro-mechanical fracture with an open-source implementation of it using gridap in julia,

Mathematics and Mechanics of Solids 28 (8) (2023) 1877–1908. https://doi.org/10.1177/10812865221133860
[29] A.U. Pillai, A.K. Behera, M.M. Rahaman, Combined diffused material interface and hybrid phase-field model for brittle fracture in heterogeneous composites, Eng. Fract. Mech. 277 (2023) 108957. https://doi.org/10.1016/j.engfracmech.2022.108957
[30] A.K. Behera, K.H. Sudeep, M.M. Rahaman, Thermodynamically consistent volumetric–deviatoric decomposition-based phase-field model for thermo-electro-mechanical fracture, Eng. Fract. Mech. 290 (2023) 109468. https://doi.org/10.1016/j.engfracmech.2023.109468
[31] S. May, J. Vignollet, R. De Borst, A numerical assessment of phase-field models for brittle and cohesive fracture: Γ-convergence and stress oscillations, Eur. J. Mech. A/Solids 52 (2015) 72–84. https://doi.org/10.1016/j.euromechsol.2015.02.002
[32] H. Li, H. Lei, Z. Yang, J. Wu, X. Zhang, S. Li, A hydro-mechanical-damage fully coupled cohesive phase field model for complicated fracking simulations in poroelastic media, Comput. Methods Appl. Mech. Eng. 399 (2022) 115451. https://doi.org/10.1016/j.cma.2022.115451
[33] T.K. Mandal, V.P. Nguyen, J.-Y. Wu, C. Nguyen-Thanh, A. de Vaucorbeil, Fracture of thermo-elastic solids: phase-field modeling and new results with an efficient monolithic solver, Comput. Methods Appl. Mech. Eng. 376 (2021) 113648. https://doi.org/10.1016/j.cma.2020.113648
[34] N. Singh, C.V. Verhoosel, R. De Borst, E.H. Van Brummelen, A fracture-controlled path-following technique for phase-field modeling of brittle fracture, Finite Elem. Anal. Des. 113 (2016) 14–29. https://doi.org/10.1016/j.finel.2015.12.005
[35] T. Heister, M.F. Wheeler, T. Wick, A primal-dual active set method and predictor-corrector mesh adaptivity for computing fracture propagation using a phase-field approach, Comput. Methods Appl. Mech. Eng. 290 (2015) 466–495. https://doi.org/10.1016/j.cma.2015.03.009
[36] T. Heister, T. Wick, Pfm-cracks: a parallel-adaptive framework for phase-field fracture propagation, Software Impacts 6 (2020) 100045. https://doi.org/10.1016/j.simpa.2020.100045
[37] T. Heister, T. Wick, Parallel solution, adaptivity, computational convergence, and open-source code of 2d and 3d pressurized phase-field fracture problems, PAMM 18 (1) (2018) e201800353. https://doi.org/10.1002/pamm.201800353
[38] S. Lee, M.F. Wheeler, T. Wick, A phase-field diffraction model for thermo-hydro-mechanical propagating fractures, Int. J. Heat Mass Transf. 239 (2025) 126487. https://doi.org/10.1016/j.ijheatmasstransfer.2024.126487
[39] J. Yang, H.A. Tchelepi, A.R. Kovscek, Phase-field modeling of rate-dependent fluid-driven fracture initiation and propagation, Int. J. Numer. Anal. Methods Geomech. 45 (8) (2021) 1029–1048. https://doi.org/10.1002/nag.3190
[40] M. Dinachandra, A. Alankar, Adaptive finite element modeling of phase-field fracture driven by hydrogen embrittlement, Comput. Methods Appl. Mech. Eng. 391 (2022) 114509. https://doi.org/10.1016/j.cma.2021.114509
[41] C. Luo, Fast staggered schemes for the phase-field model of brittle fracture based on the fixed-stress concept, Comput. Methods Appl. Mech. Eng. 404 (2023) 115787. https://doi.org/10.1016/j.cma.2022.115787
[42] N. Noii, M. Fan, T. Wick, Y. Jin, A quasi-monolithic phase-field description for orthotropic anisotropic fracture with adaptive mesh refinement and primal-dual active set method, Eng. Fract. Mech. 258 (2021) 108060. https://doi.org/10.1016/j.engfracmech.2021.108060
[43] F. Fei, J. Choo, C. Liu, J.A. White, Phase-field modeling of rock fractures with roughness, Int. J. Numer. Anal Methods Geomech. 46 (5) (2022) 841–868. https://doi.org/10.1002/nag.3317
[44] R. Ma, W. Sun, T. Guo, Phase-field model for ductile fracture in the stress resultant geometrically exact shell, Int J Numer Methods Eng 125 (13) (2024). https://doi.org/10.1002/nme.7462
[45] T. Jin, Z. Li, K. Chen, A novel phase-field monolithic scheme for brittle crack propagation based on the limited-memory BFGS method with adaptive mesh refinement, Int J Numer Methods Eng 125 (22) (2024) e7572. https://doi.org/10.1002/nme.7572
[46] T. Jin, Gradient projection method for enforcing crack irreversibility as box constraints in a robust monolithic phase-field scheme, Comput. Methods Appl. Mech. Eng. 435 (2025) 117622. https://doi.org/10.1016/j.cma.2024.117622
[47] M. Ambati, T. Gerasimov, L. De Lorenzis, A review on phase-field models of brittle fracture and a new fast hybrid formulation, Comput. Mech. 55 (2015) 383–405. https://doi.org/10.1007/s00466-014-1109-y
[48] S. Khan, A. Muixí, C. Annavarapu, A. Rodríguez-Ferran, Adaptive phase-field modeling of fracture propagation in bi-layered materials, Eng. Fract. Mech. 292 (2023) 109650. https://doi.org/10.1016/j.engfracmech.2023.109650
[49] S. Khan, A. Muixí, C. Annavarapu, A. Rodríguez-Ferran, Investigation on the effect of material mismatch between two dissimilar materials using an adaptive phase-field method, Int. J. Adv. Eng. Sci. Appl. Math. (2023).
[50] S. Khan, I. Singh, C. Annavarapu, A. Rodríguez-Ferran, Adaptive phase-field modeling of fracture propagation in layered media: effects of mechanical property mismatches, layer thickness, and interface strength, Eng. Fract. Mech. 314 (2025) 110672. https://doi.org/10.1016/j.engfracmech.2024.110672
[51] I. Jain, A. Muixí, C. Annavarapu,S.S. Mulay, A. Rodríguez-Ferran, Adaptive phase-field modeling of fracture in orthotropic composites, Eng. Fract. Mech. 292 (2023) 109673. https://doi.org/10.1016/j.engfracmech.2023.109673
[52] I. Jain, C. Annavarapu, S.S. Mulay, A. Rodríguez-Ferran, Numerical modeling of fracture propagation in orthotropic composite materials using an adaptive phase-field method, Int. J. Adv. Eng. Sci. Appl. Math. 15 (2023) 144–154.
[53] W.N. Munshi, C. Annavarapu, S.S. Mulay, A. Rodríguez-Ferran, Modeling three-Dimensional crack interaction in layered materials with imperfectly bonded interfaces using a phase-Field model, Eng. Fract. Mech. 329 (2025) 111624. https://doi.org/https://doi.org/10.1016/j.engfracmech.2025.111624
[54] W.N. Munshi, C. Annavarapu, S.S. Mulay, A. Rodriguez-Ferran, Phase-field modeling of crack propagation in layered materials with weak interfaces, Int. J. Comput. Methods Eng. Sci. Mech. (2025). https://doi.org/10.1080/15502287.2025.2568453
[55] A. Kumar, B. Bourdin, G.A. Francfort, O. Lopez-Pamies, Revisiting nucleation in the phase-field approach to brittle fracture, J Mech Phys Solids 142 (2020) 104027. https://doi.org/10.1016/j.jmps.2020.104027
[56] O. Lopez-Pamies, J.E. Dolbow, G.A. Francfort, C.J. Larsen, Classical variational phase-field models cannot predict fracture nucleation, Comput Methods Appl Mech Eng 433 (2025) 117520. https://doi.org/10.1016/j.cma.2024.117520
[57] W.N. Munshi, The deal.II code gallery: Phase-field model for fracture using MPI and AMR, 2025, https://doi.org/10.5281/zenodo.14682543
[58] C. Miehe, F. Welschinger, M. Hofacker, Thermodynamically consistent phase-field models of fracture: variational principles and multi-field FE implementations, Int. J. Numer. Methods Eng. 83 (10) (2010) 1273–1311. https://doi.org/10.1002/nme.2861
[59] D.W. Kelly, J. P. d. S. R. Gago, O.C. Zienkiewicz, I. Babuška, A posteriori error analysis and adaptive processes in the finite element method: part i–Error analysis, Int. J. Num. Meth. Eng. 19 (11) (1983) 1593–1619. https://doi.org/10.1002/nme.1620191103
[60] J. P. d. S. R. Gago, D.W. Kelly, O.C. Zienkiewicz, I. Babuška, A posteriori error analysis and adaptive processes in the finite element method: part II — adaptive mesh refinement, Int. J. Num. Meth. Eng. 19 (11) (1983) 1621–1656. https://doi.org/10.1002/nme.1620191104
[61] D. Arndt, W. Bangerth, M. Bergbauer, M. Feder, M. Fehling, J. Heinz, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, B. Turcksin, D. Wells, S. Zampini, The deal.II library, version 9.5, Journal of Numerical Mathematics 31 (3) (2023) 231–246. https://doi.org/10.1515/jnma-2023-0089
[62] C. Burstedde, L.C. Wilcox, O. Ghattas, P4EST: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, SIAM J. Sci. Comput. 33 (3) (2011) 1103–1133. https://doi.org/10.1137/100791634
[63] W. Bangerth, C. Burstedde, T. Heister, M. Kronbichler, Algorithms and data structures for massively parallel generic adaptive finite element codes, ACM Trans. Math. Software 38 (2011) 14/1–28. https://doi.org/10.1145/2049673.2049678
[64] S. Balay, S. Abhyankar, M.F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W.D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M.G. Knepley, F. Kong, S. Kruger, D.A. May, L.C. McInnes, R.T. Mills, L. Mitchell, T. Munson, J.E. Roman, K. Rupp, P. Sanan, J. Sarich, B.F. Smith, H. Suh, S. Zampini, H. Zhang, H. Zhang, J. Zhang, PETSc/TAO Users Manual, Technical Report ANL-21/39 - Revision 3.22, Argonne National Laboratory, 2023. https://doi.org/10.2172/2205494
[65] S. Balay, S. Abhyankar, M.F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E.M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W.D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M.G. Knepley, F. Kong, S. Kruger, D.A. May, L.C. McInnes, R.T. Mills, L. Mitchell, T. Munson, J.E. Roman, K. Rupp, P. Sanan, J. Sarich, B.F. Smith, S. Zampini, H. Zhang, H. Zhang, J. Zhang, PETSc Web page, 2024, https://petsc.org/.
[66] M.A. Heroux, R.A. Bartlett, V.E. Howle, R.J. Hoekstra, J.J. Hu, T.G. Kolda, R.B. Lehoucq, K.R. Long, R.P. Pawlowski, E.T. Phipps, A.G. Salinger, H.K. Thornquist, R.S. Tuminaro, J.M. Willenbring, A. Williams, K.S. Stanley, An overview of the trilinos project, ACM Trans. Math. Software 31 (3) (2005) 397–423. https://doi.org/10.1145/1089014.1089021
[67] T.T.P. Team, The Trilinos Project Website, 2024. https://trilinos.github.io.
[68] R.D. Falgout, U.M. Yang, Hypre: a library of high performance preconditioners, in: P.M.A. Sloot, A.G. Hoekstra, C.J.K. Tan, J.J. Dongarra (Eds.), Computational Science – ICCS 2002, 2331 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 632–641. https://doi.org/10.1007/3-540-47789-6_66
[69] M. Gee, C. Siefert, J. Hu, R. Tuminaro, M. Sala, ML 5.0 Smoothed Aggregation User's Guide, Technical Report SAND2006-2649, Sandia National Laboratories, 2007. https://trilinos.github.io/pdfs/mlguide5.pdf.
[70] L. Berger-Vergiat, C.A. Glusa, J.J. Hu, C.M. Siefert, R.S. Tuminaro, M. Matthias, A. Prokopenko, T.A. Wiesner, MueLu User's Guide, Technical Report SAND2023-12265, Sandia National Laboratories, 2023. https://trilinos.github.io/pdfs/mueluguide.pdf.
[71] G.F. Carey, Computational Grids: Generation, Adaptation and Solution Strategies, Taylor & Francis, 1997.
[72] J.-Y. Wu, V.P. Nguyen, C.T. Nguyen, D. Sutula, S. Sinaie, S.P.A. Bordas, Phase-field modeling of fracture, Adv. Appl. Mech. 53 (2020) 1–183. https://doi.org/10.1016/bs.aams.2019.08.001
[73] M. Kronbichler, T. Heister, W. Bangerth, High accuracy mantle convection simulation through modern numerical methods, Geophys J. Int. 191 (2012) 12–29. https://doi.org/10.1111/j.1365-246X.2012.05609.x
[74] J. Frohne, T. Heister, W. Bangerth, Efficient numerical methods for the large-scale, parallel solution of elastoplastic contact problems, Int. J. Numer. Methods Eng. 105 (2016) 416–439. https://doi.org/10.1002/nme.4977