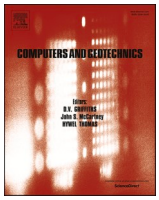
Computers and Geotechnics 187 (2025) 107481

Contents lists available at ScienceDirect

Computers and Geotechnics

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



Research Paper

Full-process GPU-parallelized finite volume framework for phase field modeling of brittle fracture

Tao Yu [a,*], Yuntian Zhao [b], Jidong Zhao [a,*]

[a] Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong
[b] China Institute of Water Resources and Hydropower Research, China

ARTICLE INFO

Keywords:
Phase field
Finite volume method
GPU acceleration
Fracture simulation

ABSTRACT

The phase-field method (PFM) offers a robust framework for fracture simulation by transforming discrete cracks into continuous phase-field representations, thereby eliminating geometric complexities associated with traditional fracture modeling. While the finite volume method (FVM) has emerged as a promising approach for implementing PFM in multi-physics fracturing problems, its scalability remains constrained by inherent computational inefficiencies. This paper presents a full-process GPU-parallelized finite volume framework designed to accelerate PFM-based fracture computations, enabling large-scale, high-resolution simulations and integration of multi-physics couplings. The framework is rigorously verified against three benchmark crack propagation scenarios, confirming its numerical accuracy and computational performance. A comprehensive comparative analysis demonstrates that the full-process GPU-parallelized FVM achieves a significant speedup (up to 12 times) compared to conventional CPU-parallelized implementations. Furthermore, its extensibility is showcased through two two-phase hydraulic fracturing case studies, where it is coupled with a two-phase computational fluid dynamics (CFD). The results highlight the framework's capacity to resolve coupled multi-physics fracturing phenomena without compromising computational efficiency. This advancement opens new avenues for high-fidelity, computationally tractable simulations of complex fracture-dominated systems in engineering and geoscience applications.

1. Introduction

Fracture is a common failure mode in engineering materials, driven by the competition between elastic energy release and the material's resistance to crack growth, as described by energy-based fracture mechanics (Griffith, 1921). Computational modeling has become an indispensable tool for understanding and predicting fracture behavior, particularly when full-scale experiments are prohibitively expensive or impractical (Wu et al., 2020). These models not only aid in forecasting the failure of cracked structures but also provide valuable insights into fracture processes across diverse materials, including concrete, rock, and ceramics. However, modeling crack propagation remains a significant challenge due to the discontinuous nature of cracks and the intrinsic heterogeneity of materials (Wu et al., 2020).

Existing fracture simulation methods can be broadly classified into discontinuous and continuous approaches, each facing inherent challenges. Discrete methods require additional criteria to define crack initiation, propagation, direction, and bifurcation in dynamic fracture problems (Griffith, 1921, Irwin, 1957, Dugdale, 1960, Barenblatt, 1962). Moreover, explicitly tracking complex crack paths remains a significant challenge, particularly in three-dimensional cases (Bordas et al., 2008, Sukumar et al., 2015). Traditional continuous approaches encounter difficulties in introducing displacement discontinuities into an otherwise continuous displacement field (Wu et al., 2020, Rashid, 1968, Wu et al., 2015, Belytschko and Lin, 1987, Xu and Needleman, 1994, Wang et al., 2020a) and some of them still necessitate explicit tracking of crack surfaces (Moës et al., 1999, Wells and Sluys, 2001, Wu and Li, 2015). As a continuous approach, Peridynamics (PD) (Silling, 2000, Fan et al., 2022) offers a natural framework for handling displacement discontinuities. However, its computational cost is significantly higher than FEM, and it lacks direct compatibility with traditional mechanics-based methods such as FEM and CFD.

To address the challenges associated with discrete crack modeling, the phase field method (PFM) (Bourdin et al., 2008, Miehe et al., 2010) has been developed to automatically determine crack paths (Ambati et al., 2015, Tanné et al., 2018). PFM integrates fracture mechanics and

* Corresponding authors.
E-mail addresses: tyuak@connect.ust.hk (T. Yu), jzhao@ust.hk (J. Zhao).

https://doi.org/10.1016/j.compgeo.2025.107481
Received 6 April 2025; Received in revised form 3 July 2025; Accepted 4 July 2025
Available online 10 July 2025

damage mechanics, extending Griffith's theory (Griffith, 1921) by simultaneously solving for both the displacement field and the crack set through the minimization of the material's total potential energy. Formulated using partial differential equations (PDEs) (Ambati et al., 2015), PFM is particularly effective in capturing complex fracture behaviors (Amor et al., 2009, Miehe et al., 2010) and can be seamlessly coupled with traditional numerical methods, such as CFD (Yi et al., 2024). It has been adopted to study fracture modes and dynamic crack propagation in various materials, including the rock materials (Zhang et al., 2017, Fei and Choo, 2020), composite materials (Li et al., 2023), and piezoelectric materials (Kiran et al., 2023).

However, in solving the coupled displacement and phase-field equations in PFM, the primary computational cost arises from the need to resolve the regularization length scale, which necessitates fine spatial discretization throughout the domain (Yue et al., 2023, Tian et al., 2020). This requirement is intrinsic to all numerical frameworks employed in phase-field models, including both FEM (Tian et al., 2020) and FVM (Yang et al., 2024b). In addition, the staggered solution scheme (Seleš et al., 2019), widely adopted for its robustness, involves alternating updates between displacement and phase-field fields, often requiring a large number of iterations to achieve convergence. When combined with a high-resolution mesh, this iterative procedure imposes a significant computational burden, making simulation speed a key bottleneck in large-scale fracture modeling (Yang et al., 2024b). Existing approaches to accelerating the PFM primarily focus on algorithm optimization, such as the high order scheme (Nguyen-Thanh et al., 2020) and the hybrid isotropic-anisotropic scheme (Ambati et al., 2015). Recently, the FVM has been explored for solving PFM, achieving significant efficiency improvements (Yang et al., 2024b, Yang et al., 2024c). Compared to the FEM, which requires assembling a global stiffness matrix with high computational costs, the FVM constructs a sparser matrix with lower computational complexity, making it more suitable for large-scale and parallel computations (Yang et al., 2024b, Yang et al., 2024c). Additionally, FVM is naturally compatible with CFD, facilitating seamless coupling between the two methods. However, further improving computational efficiency from the perspective of algorithm optimization within the FVM and PFM has become increasingly difficult.

In recent years, GPU parallel acceleration has been increasingly applied to scientific computing, a paradigm known as General-Purpose Graphics Processing Unit (GPGPU) computing (Michalakes and Vachharajani, 2008). Leveraging the massive parallelism with thousands of cores, GPU-based acceleration has demonstrated substantial efficiency improvements over CPU parallelism. Currently, GPU acceleration is being extensively adopted in both particle-based methods (Peng et al., 2019, Xia and Liang, 2016, Xiong et al., 2013, Tubbs and Tsai, 2011, Shu et al., 2020, Wang et al., 2023, Zhang et al., 2023b, Wang et al., 2020b, Gao et al., 2018) and structured mesh-based methods (Aissa et al., 2017, Kuo et al., 2020). For particle-based methods, each particle can be assigned an independent GPU thread, a strategy that aligns well with the GPU hardware architecture. Similarly, for structured mesh-based algorithms, retrieving information from neighboring cells is straightforward during the independent computation of each mesh cell, further enhancing computational efficiency. However, this algorithm is challenging to extend to problems with complex geometries.

Recent advances have demonstrated the great potential of GPU-based parallel computing in accelerating fracture simulations across different numerical paradigms. In particular, mesh-free methods such as the discrete element method (DEM) (Liu et al., 2021), peridynamics (Zhong et al., 2024), and the finite point method (FPM) (Zhao et al., 2020, Kang et al., 2024) have all been successfully adapted to GPU architecture, leading to significant performance gains. Extended finite element methods have also benefited from GPU acceleration. For instance, Shin et al. (2023) developed a GPU-parallelized framework based on the GraFEA model to simulate both impact and quasi-static fractures efficiently. While these studies confirm the effectiveness of GPU computing in fracture modeling, most existing work focuses on mesh-free strategies.

Despite the increasing adoption of GPU acceleration, its application in implicit, unstructured mesh-based methods, such as the FVM, remains relatively limited. Recent studies have explored heterogeneous computing that integrates both CPU and GPU parallelism (Lei et al., 2019, Afzal et al., 2021) to leverage existing CPU-parallelized algorithms, where GPU parallelism is primarily employed for solving linear systems (Jespersen, 2010, Zhang et al., 2023a). However, this approach necessitates frequent data transfers between the CPU and GPU during iterative computations, thereby limiting the full potential of GPU acceleration. In this work, we present, for the first time, a fully GPU-parallelized finite volume framework for the phase-field method and apply it to fracture simulations. The proposed framework completely abandons the original domain decomposition-based CPU-parallelized algorithm and adopts an innovative design fully optimized for GPU hardware architecture.

The full-process GPU-parallelized FVM for the PFM overcomes three major challenges in fracture simulations. First, conventional PFM suffers from high computational costs due to the extensive iterations required to solve both the phase-field and solid mechanics equations, rendering large-scale simulations impractical without acceleration. Second, in coupled fracture-fluid flow simulations, the computational bottleneck shifts predominantly to fracture modeling, significantly limiting the efficiency and feasibility of multiphysics simulations. Third, developing a robust and extensible multiphysics framework necessitates a unified numerical approach capable of efficiently handling multiple governing equations. In summary, the proposed full-process GPU-parallelized FVM offers a computationally efficient solution that not only accelerates fracture simulations but also seamlessly integrates with CFD within a unified numerical framework, enabling large-scale multiphysics modeling.

Fracture processes in geomaterials, such as rock cracking, hydraulic fracturing, and desiccation cracking in unsaturated soils, are of fundamental interest in geotechnical engineering (Xu et al., 2018). These problems often involve highly nonlinear fracture propagation under complex loading and multiphase fluid conditions, posing significant computational challenges. The proposed framework is particularly well-suited for such applications, as it enables efficient and robust simulation of large-scale three-dimensional fracture processes in porous and fractured geomaterials. This highlights the relevance of the method to a broad range of geotechnical applications, including CO_2 sequestration (Stevens et al., 1999) and reservoir stimulation (Economides and Nolte, 1989).

The remainder of this paper is organized as follows. Section 2 introduces the PFM for the crack simulation and the full-process GPU-parallelized FVM for the PFM. This section provides a detailed explanation of the full-process GPU-parallelized computational framework, including the grid structure, sparse matrix handling, explicit term computation, governing equation discretization and assembly, linear solver, and iterative algorithms. Section 3 presents three verification cases to demonstrate the accuracy and efficiency of the proposed algorithm, along with two two-phase hydraulic fracturing examples to showcase its potential for coupling with two-phase CFD. A detailed performance comparison between full-process GPU-parallelized FVM for PFM (GPU-PFM) and CPU-accelerated FVM for PFM (CPU-PFM) is provided, demonstrating the significant acceleration achieved by the fully GPU-parallel approach. Finally, Section 4 concludes the study.

2. Methodology: Full-process GPU-parallelized FVM for PFM

The phase-field method (PFM) is a widely recognized approach for simulating brittle fractures, where cracks are represented as diffuse interfaces characterized by a continuous phase-field variable d. This variable ranges from 0 to 1, with $d = 0$ indicating intact material and $d = 1$ representing fully damaged regions. This section provides an overview

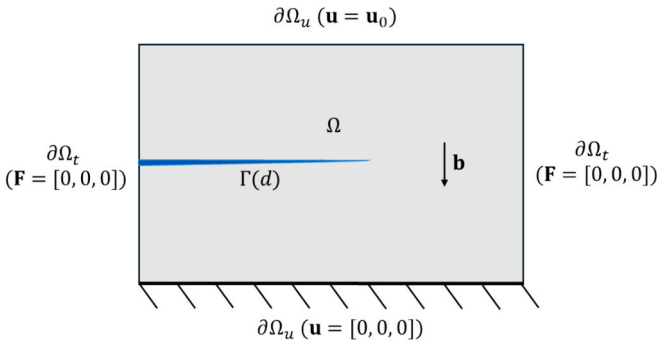


Fig. 1. Schematic illustration of cracks and boundary conditions.

of a typical PFM formulation, primarily based on the variational framework introduced by Miehe et al. (2010), which ensures that crack evolution minimizes the total free energy while adhering to irreversibility constraints. Additionally, the finite volume implementation of the PFM proposed by Yang et al. (2024b), coupled with an innovative GPU-accelerated framework, will be introduced.

2.1. Phase field method for brittle fractures

2.1.1. Problem description

Consider an embedded fracture Γ in a linear elastic solid $\Omega \subset R^n$ (where n denotes the number of dimensions). Fig. 1 shows the fracture region Γ and a solid region Ω in a single-edge notched tension test. The outer boundary of the solid Ω is $\partial\Omega \subset R^{n-1}$ and is subjected to two conditions: prescribed displacement on $\partial\Omega_u$ and prescribed force/stress on $\partial\Omega_t$, which are called as Dirichlet boundary condition and Neumann boundary condition, respectively. Regions $\partial\Omega_u$ and $\partial\Omega_t$ satisfy $\partial\Omega_u \cup \partial\Omega_t = \partial\Omega$ and $\partial\Omega_u \cap \partial\Omega_t = \varnothing$. The body force per unit volume acted on the solid Ω is denoted by $\mathbf{b}$. The embedded fracture Γ is denoted by the phase field variable d.

2.1.2. Governing equations

The applied phase-field model for a purely solid phase is based on the variational principle outlined by Miehe et al. (2010) and rooted in classical fracture theory (Griffith, 1921) and the minimum energy variational method (Francfort and Marigo, 1998). The potential energy functional for the crack fracturing process can be written as:

$$\Pi(\mathbf{u}, \Gamma) = \int_\Omega \Psi(\varepsilon) \mathrm{d}\Omega + \int_\Gamma G_c \mathrm{d}\Gamma - \int_\Omega \mathbf{b}\cdot\mathbf{u} \mathrm{d}\Omega - \int_{\partial\Omega_t} \mathbf{t}\cdot\mathbf{u} dS \tag{1}$$

where Ψ is the elastic strain energy density, ε is the small strain tensor, G_c is the critical energy release rate of the material, $\mathbf{t}$ is the surface traction, $\mathbf{b}$ is the body force, and $\mathbf{u}$ is the displacement.

The strain tensor ε and the strain energy Ψ are functions of the displacement $\mathbf{u}$. For an isotropic linear elastic material, they are defined by the following expressions:

$$\varepsilon(\mathbf{u}) = ((\nabla\mathbf{u})^{\mathrm{T}} + \nabla\mathbf{u})/2 \tag{2}$$

$$\Psi(\varepsilon) = \frac{\lambda}{2}\mathrm{tr}(\varepsilon(\mathbf{u}))^2 + \mu\varepsilon(\mathbf{u}) : \varepsilon(\mathbf{u}) \tag{3}$$

where λ and μ are the Lamé constants.

In the phase-field method, the crack surface $\Gamma(d)$ is denoted by the phase field variable d and is calculated by the following equation: the crack surface energy density per unit volume of the solid is calculated (Miehe et al., 2010)

$$\int_\Gamma \mathrm{d}\Gamma = \int_\Omega \gamma(d) \mathrm{d}\Omega \tag{4}$$

$$\gamma(d) = \frac{d^2}{2l_0} + \frac{l_0}{2}|\nabla d|^2 \tag{5}$$

where γ is the crack surface density per unit volume, ∇d is the spatial gradient of the phase field, and $l_0 \in R^+$ is the characteristic length that controls the width of the diffuse crack transition region.

The total crack surface energy of the elastic solid in Eq. (1) can be expressed as:

$$\int_\Gamma G_c \mathrm{d}\Gamma \approx \int_\Omega G_c \left(\frac{d^2}{2l_0} + \frac{l_0}{2}|\nabla d|^2\right) \mathrm{d}\Omega \tag{6}$$

Further considering the effect of cracks on reducing the stored energy within the solid, the total energy functional (Eq. (1)) can be written as:

$$\Pi(\mathbf{u}, d) = \int_\Omega g(d)\Psi(\varepsilon(\mathbf{u}))\mathrm{d}\Omega + \int_\Omega G_c \left(\frac{d^2}{2l_0} + \frac{l_0}{2}|\nabla d|^2\right) \mathrm{d}\Omega$$
$$- \int_\Omega \mathbf{b}\cdot\mathbf{u} \mathrm{d}\Omega - \int_{\partial\Omega_t} \mathbf{t}\cdot\mathbf{u} dS \tag{7}$$

where $g(d)$ is the stress degradation function and $g(d) = \left[(1-d)^2 + k\right]$ (Wu et al., 2020). $k = 10^{-6}$ is a constant to avoid numerical singularity as d approaches 1.

Taking the first variation with respect to two variations ($\mathbf{u}$ and d) in Eq. (7), two governing equations can be obtained:

$$\nabla\cdot\sigma + \mathbf{b} = 0 \mathrm{in}\Omega \tag{8}$$

$$-G_c l_0 \nabla^2 d + \left[\frac{G_c}{l_0} + 2\Psi(\varepsilon)\right] d = 2\Psi(\varepsilon) \mathrm{in}\Omega \tag{9}$$

with the following boundary conditions:

$$\sigma\cdot\mathbf{n} = \bar{\mathbf{t}} \mathrm{on}\ \partial\Omega_t \tag{10}$$

$$\mathbf{u} = \bar{\mathbf{u}} \mathrm{on}\ \partial\Omega_u \tag{11}$$

$$\nabla d\cdot\mathbf{n} = 0 \mathrm{on}\ \partial\Omega \tag{12}$$

where $\bar{\mathbf{t}}$ and $\bar{\mathbf{u}}$ are the traction and displacement acted on the boundary, respectively. $\mathbf{n}$ is the outward normal vector of the boundary, and σ is the stress.

The spectral decomposition approach proposed by Miehe et al. (2010) is used to avoid crack development under compression. This approach adopts the spectral decomposition of strain to calculate the effective stress σ.

$$\sigma = \left[(1-d)^2 + k\right]\left\{\lambda\langle\mathrm{tr}(\varepsilon)\rangle_+ \mathbf{I} + 2\mu\varepsilon_+\right\} + \lambda\langle\mathrm{tr}(\varepsilon)\rangle_- \mathbf{I} + 2\mu\varepsilon_- \tag{13}$$

where ε_+ and ε_- represent the positive and negative parts of elastic strain, respectively, satisfying $\varepsilon = \varepsilon_+ + \varepsilon_-$. They are calculated based on the principal strains ε_p and principal directions $\mathbf{n}_p$ as following:

$$\varepsilon_\pm = \sum_{p=1}^{n} \langle\varepsilon_p\rangle_\pm \mathbf{n}_p \otimes \mathbf{n}_p \tag{14}$$

where the Macaulay brackets $\langle\ \rangle_\pm$ are expressed as: $\langle x\rangle_\pm = (x \pm |x|)/2$.

To avoid crack healing in the elastic solid under compression or unloading, a history state variable H is adopted, which is expressed as:

$$H^{n+1}(\mathbf{x}) = \max\left(H^n(\mathbf{x}), \Psi^+\left(\varepsilon^{n+1}(\mathbf{x})\right)\right) \tag{15}$$

where Ψ^+ is the positive part of elastic strain energy derived from the spectral decomposition approach:

$$\Psi^+ = \frac{\lambda}{2}\langle\mathrm{tr}(\varepsilon)\rangle_+^2 + \mu\mathrm{tr}\left[\varepsilon_+^2\right] \tag{16}$$

The history state variable H is assumed to be constant within each finite volume cell and is updated once at every time step. This treatment ensures that H accurately captures the maximum driving energy up to the current time. This assumption aligns with the conventional finite volume framework, wherein field variables are represented as cell-averaged quantities and assumed constant within each control volume. While this may introduce mesh-dependent approximation errors, such a treatment has been widely adopted in the literature (Yang et al., 2024b; Cardiff et al., 2017; Jasak and Weller, 2000) and has been shown, through extensive benchmark studies and verification cases, to yield sufficiently accurate results for a broad range of brittle and hydraulic fracture problems. Therefore, the assumption is considered both reasonable and computationally efficient within the context of the current implementation.

Eq. (7) can be further rewritten as Eq. (17) by replacing the elastic energy $\Psi(\varepsilon)$ with the history state variable H.

$$-G_c l_0 \nabla^2 d + \left[\frac{G_c}{l_0} + 2H\right] d = 2H \text{in} \Omega \tag{17}$$

Jasak and Weller (2000) proposes a separation solution algorithm to stabilize the numerical oscillations of Eq. (8), which is expressed as:

$$\underbrace{K_f \nabla^2 \mathbf{u}}_{\text{implicit}} \underbrace{- K_f \nabla^2 \mathbf{u} + \nabla \cdot \sigma + \mathbf{b}}_{\text{explicit}} = 0 \tag{18}$$

where the coefficient K_f is selected as $K_f = \lambda + 2\mu$ in this work (Cardiff et al., 2017), controlling the smoothing effect. In this context, "implicit" means that the unknown variable will be adopted to discretize this term and assemble the algebraic linear system, while "explicit" means that only the variable at the last time step or iteration will be used for the discretization process.

The separation solution algorithm adopted in this work follows established practices (Cardiff et al., 2017, Jasak and Weller, 2000, Yang et al., 2024b) for solving PFM using the FVM. In particular, the implicit formulation is known to potentially suffer from checker-boarding artifacts, manifesting as spurious oscillations in the displacement field (Cardiff and Demirdžić, 2021). To suppress these non-physical modes, a stabilizing diffusion term, analogous to the Rhie-Chow correction originally proposed for pressure–velocity coupling in fluid dynamics (Rhie and Chow, 1983), is introduced. This term appears as the first two terms of Eq. (40) and effectively mitigates displacement oscillations by enhancing numerical coupling between adjacent control volumes.

It is worth noting that this stabilization technique typically requires a large number of iterations to achieve convergence, especially in regions with strong phase-field gradients or evolving crack fronts. This iterative burden significantly contributes to the overall computational cost and partially explains the increased runtime observed in FVM-based PFM solvers. Nevertheless, upon sufficient convergence, the method satisfies the mechanical equilibrium condition $\nabla \cdot \sigma + \mathbf{b} = 0$, ensuring a physically consistent solution.

Eqs. (17) and (18), in conjunction with boundary conditions (Eqs. (10)–(12)), comprise the governing equations of the phase-field model, including the displacement field $\mathbf{u}$ and the phase field d.

To further account for the transient nature of hydraulic fracturing and the fluid-induced forces acting on the solid, Eq. (18) can be extended to the following form.

$$\underbrace{\frac{\partial^2}{\partial t^2}(\rho_s \mathbf{u}) + K_f \nabla^2 \mathbf{u}}_{\text{implicit}} \underbrace{- K_f \nabla^2 \mathbf{u} + \nabla \cdot \sigma - \alpha_{\text{biot}} \nabla p - k(\mathbf{u}_f - \mathbf{v}_s)}_{\text{explicit}} = 0 \tag{19}$$

where ρ_s is the density of the solid, and $\mathbf{b} = -\alpha_{\text{biot}} \nabla p - k(\mathbf{u}_f - \mathbf{v}_s)$ is the source term representing the combined effects of fluid pressure and the drag force induced by the relative motion between the fluid and the solid. α_{biot} is the Biot's coefficient, a dimensionless parameter (typically between 0 and 1) that quantifies the proportion of fluid pressure transmitted to the solid matrix in porous media (Miehe et al., 2010). p denotes the fluid pressure, and k is the drag coefficient, determined from a relative permeability model (Carrillo and Bourg, 2021). u_f and $\mathbf{v}_s$ are the fluid and solid velocities, respectively. The solid velocity $\mathbf{v}_s$ is approximated as $\mathbf{v}_s^n = \left(\mathbf{u}^n - \mathbf{u}^{n-1}\right)/\Delta t$, representing the incremental displacement of the solid over one time step.

2.2. Two-phase CFD with porous media

2.2.1. Two-phase model with porous media

Two-phase flow in porous media is typically modeled using the Volume of Fluid (VOF) method in conjunction with the Multidimensional Universal Limiter for Explicit Solution (MULES) algorithm (Roenby et al., 2017, Rusche, 2002). This method tracks the distribution of individual fluid phases through phase volume fractions α_i, and introduces a compressive flux term within the advection equation to enhance the sharpness of the fluid interface. The evolution of the volume fraction field is governed by the following continuity equation, which includes the compressive flux contribution (Eq. (20)), as proposed in Carrillo et al. (2020). The fluids are considered incompressible, and their continuity behavior is described by:

$$\frac{\partial \phi \alpha_i}{dt} + \nabla \bullet \left(\alpha_i \mathbf{u}_f\right) + \nabla \bullet \left(\phi \alpha_i (1 - \alpha_i) \mathbf{u}_c\right) = 0, \tag{20}$$

where ϕ is porosity of porous media, $\mathbf{u}_f$ is the fluid velocity, and $\mathbf{u}_c$ is the compressed velocity.

The evaluation of the compressive flux term is context-dependent, varying between pure fluid regions and porous media. In regions occupied by pure fluids, the term is computed using the standard MULES algorithm. In contrast, within porous domains, a modified formulation is employed following the method proposed by Carrillo et al. (2020), which accounts for the effects of porosity and permeability on interface capturing.

$$\mathbf{u}_c = \begin{cases} c|\mathbf{u}|\nabla\alpha_i/|\nabla\alpha_i|, in pure fluid regions, \\ \phi^{-1}\left[-\left(\frac{M_i}{\alpha_i} - \frac{M_j}{\alpha_j}\right)\nabla p + \left(\frac{\rho_i M_i}{\alpha_i} - \frac{\rho_j M_j}{\alpha_j}\right)\boldsymbol{g}\right], in porous media, \end{cases} \tag{21}$$

where $c \in [0, 1]$ is a coefficient that controls the magnitude of the compressive velocity. The symbols $\boldsymbol{g}$, M and ρ represent the gravitational acceleration, fluid mobility, and fluid density, respectively. Subscripts i and j correspond to fluid phases i and j, respectively.

The mobility M is computed using relative permeability models. In this study, we employ the formulations proposed by Van Genuchten (1980), as defined in Eqs. (22)–(24). These relative permeability expressions are derived by combining Mualem's model (Mualem, 1976) with the van Genuchten water retention function (Van Genuchten, 1980). The van Genuchten model, although originally developed within the framework of unsaturated soil mechanics, is highly relevant to the geotechnical hydraulic fracturing problems addressed in this work. Specifically, the scenarios considered involve water–gas two-phase flow in unsaturated geomaterials, where the permeability-saturation relationship plays a central role in governing the flow behavior. This model offers a well-established and widely adopted constitutive relation for capturing these effects and has been previously employed in hydraulic fracturing studies involving fractured unsaturated media (Carrillo and Bourg, 2021).

$$\begin{cases} M_i = \dfrac{k_0}{\mu_i} k_{ri} \\ M_j = \dfrac{k_0}{\mu_j} k_{rj} \end{cases}, \tag{22}$$

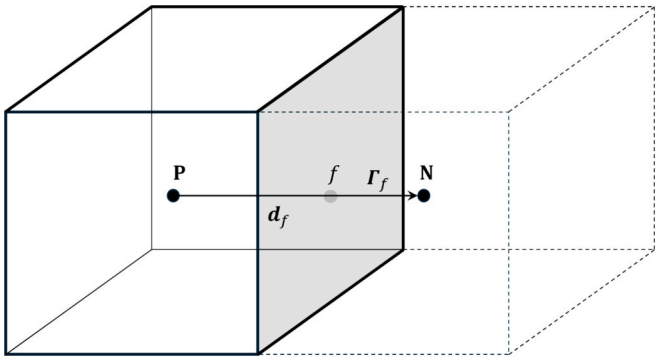


Fig. 2. Schematic illustration of hexahedron control volumes in FVM.

$$\begin{cases} k_{ri} = \alpha_{i,eff}^{\frac{1}{2}}\left(1-\left(1-\alpha_{i,eff}^{\frac{1}{m}}\right)^{m}\right)^{2} \\ k_{rj} = \left(1-\alpha_{i,eff}\right)^{\frac{1}{2}}\left(1-\alpha_{i,eff}^{\frac{1}{m}}\right)^{2m} \end{cases}, \tag{23}$$

$$\alpha_{i,eff} = \frac{\alpha_i - \alpha_{i,irr}}{1-\alpha_{j,irr}-\alpha_{i,irr}} \tag{24}$$

where k_0 is the absolute permeability of the porous media. m is a model parameter that reflects the heterogeneity of the pore structure. A smaller m value indicates greater heterogeneity, meaning there is a wider variation in pore sizes. μ is the fluid viscosity. α_{irr} denotes the irreducible saturation of the fluid phase.

In this study, the van Genuchten model is utilized primarily as a hydraulic model, providing a functional dependence of the absolute permeability coefficient on the local saturation. This formulation allows for realistic representation of multiphase transport processes in unsaturated porous media. It is worth emphasizing that the primary contribution of this work lies in the development of a fully GPU-accelerated coupled computational framework. Within this framework, the van Genuchten model serves as one possible closure relation for unsaturated permeability. It can be readily replaced by alternative models in future studies without affecting the structure, modularity, or general applicability of the overall computational methodology. In particular, replacing this model would only influence the local evaluation of the permeability-induced body force in the momentum equations, without requiring changes to the coupling algorithm or numerical implementation.

The effective density ρ and viscosity μ throughout the computational domain are updated based on the Volume of Fluid (VOF) method, in conjunction with the relative permeability model.

$$\rho = \begin{cases} \alpha_1\rho_1 + \alpha_2\rho_2, \text{in pure fluid regions}, \\ (M_1+M_2)^{-1}(M_1\rho_1+M_2\rho_2), \text{in porous media}, \end{cases} \tag{25}$$

$$\mu = \alpha_1\mu_1 + \alpha_2\mu_2 \tag{26}$$

2.2.2. Governing equations

Based on the Navier-Stokes equations, the following momentum equation is developed to incorporate key physical effects, including the presence of porous media characterized by porosity ϕ, surface tension, and interaction forces between the fluid and the porous matrix.

$$\phi^{-1}\left(\frac{\partial}{\partial t}(\rho\mathbf{u}_f) + \nabla\cdot(\phi^{-1}\rho\mathbf{u}_f\otimes\mathbf{u}_f)\right) =$$

$$-\nabla p + \rho\mathbf{g} + \nabla\cdot(\phi^{-1}\mu\cdot(\nabla\mathbf{u}_f)) + \phi^{-1}c\sigma|\nabla\alpha_1|\mathbf{n} - k(\mathbf{u}_f-\mathbf{v}_s) \tag{27}$$

Here, ρ, $\mathbf{u}_f$ and p denote the fluid density, velocity, and pressure, respectively. $\mathbf{g}$ is gravitational acceleration. The surface tension coefficient is denoted by σ, and the curvature of the fluid-gas interface is represented by c, defined as $c = -\nabla\bullet\mathbf{n}$, where $\mathbf{n}$ is the unit normal vector at the interface. For a free surface, $\mathbf{n}$ is computed as $\mathbf{n} = \nabla\alpha_1/|\nabla\alpha_1|$, with α_1 being the volume fraction of the fluid phase.

The coefficient k is determined in accordance with the Van Genuchten model, expressed as:

$$k = k_0^{-1}\left(\frac{k_{ri}}{\mu_i}+\frac{k_{rj}}{\mu_j}\right)^{-1}, \tag{28}$$

where coefficients k_{ri} and k_{rj} can be calculated using Eq. (23).

The Pressure-Implicit with Splitting of Operators (PISO) algorithm (Issa, 1986) is adopted to solve the momentum equation. This method is a widely validated approach for pressure–velocity coupling in transient flow simulations. As the primary focus of this study is not on solving the CFD equations themselves, the discretization and solution procedures are not discussed in detail.

It is worth noting that both the CFD and PFM modules are implemented within a unified finite volume method (FVM) framework on the GPU. They operate on the same computational mesh and utilize a common memory layout. All physical quantities, including those required for coupling (e.g., pressure, velocity, and displacement fields), reside in global memory and are directly accessible to all computational kernels. Consequently, the exchange of interaction forces incurs no additional memory transfer or inter-process communication overhead. This architecture avoids the performance penalties typically associated with heterogeneous or loosely coupled implementations and represents a major advantage of the proposed fully GPU-accelerated FVM framework, particularly in the context of tightly coupled multiphysics problems such as hydraulic fracturing.

2.3. Full-process GPU-parallelized finite volume method

The finite volume method (FVM) is a numerical technique widely used for solving partial differential equations (PDEs) that arise in engineering and physical sciences, particularly in fluid dynamics and heat transfer. The FVM discretizes control equations by integrating their conservation form over a finite control volume and approximating the resulting integrals. It is a robust and versatile method that emphasizes the conservation of physical quantities, such as mass, momentum, or energy, across discrete control volumes. Fig. 2 illustrates two hexahedron control volumes used in FVM, in conjunction with a neighboring face f.

Eqs. (29) and (30) can be obtained by integrating Eqs. (17) and (18) over the cell Ω_P and applying the divergence theorem.

$$\oint_{\partial\Omega_P} K_f(\nabla\mathbf{u}^{n+1})_f \mathrm{d}\Gamma_f - \oint_{\partial\Omega_P} K_f(\nabla\mathbf{u}^{n})_f \mathrm{d}\Gamma_f + \oint_{\partial\Omega_P}\sigma_f\mathrm{d}\Gamma_f + \int_{\Omega_P}\mathbf{b}\mathrm{d}\Omega = 0 \tag{29}$$

$$-\oint_{\partial\Omega_P} G_c l_0 \nabla d^{n+1}\mathrm{d}\Gamma_f + \int_{\Omega_P}\left[\frac{G_c}{l_0}+2H^n\right]d^{n+1}\mathrm{d}\Omega = \int_{\Omega_P}2H^n\mathrm{d}\Omega \tag{30}$$

By adopting the integration in each FVM control volume, Eqs. (29) and (30) can be further rewritten as:

$$\sum K_f(\nabla\mathbf{u}^{n+1})_f \bullet\Gamma_f - \sum K_f(\nabla\mathbf{u}^{n})_f\bullet\Gamma_f + \sum\sigma_f\bullet\Gamma_f + \mathbf{b}V = 0 \tag{31}$$

$$-\sum G_c l_0(\nabla d^{n+1})_f\bullet\Gamma_f + \left[\frac{G_c}{l_0}+2H^n\right]Vd^{n+1} = 2H^nV \tag{32}$$

where V is the volume of the control cell Ω_P, Γ_f is the area normal vector of face f, and the subscript denotes the value at the face f, as shown in Fig. 2.

Fig. 3 presents the overall flowchart of the full-process GPU-parallelized FVM for PFM in this study. Details involved will be elaborated in the subsequent subsections. Iterations are required to solve two governing equations as they are dependent on each other and more

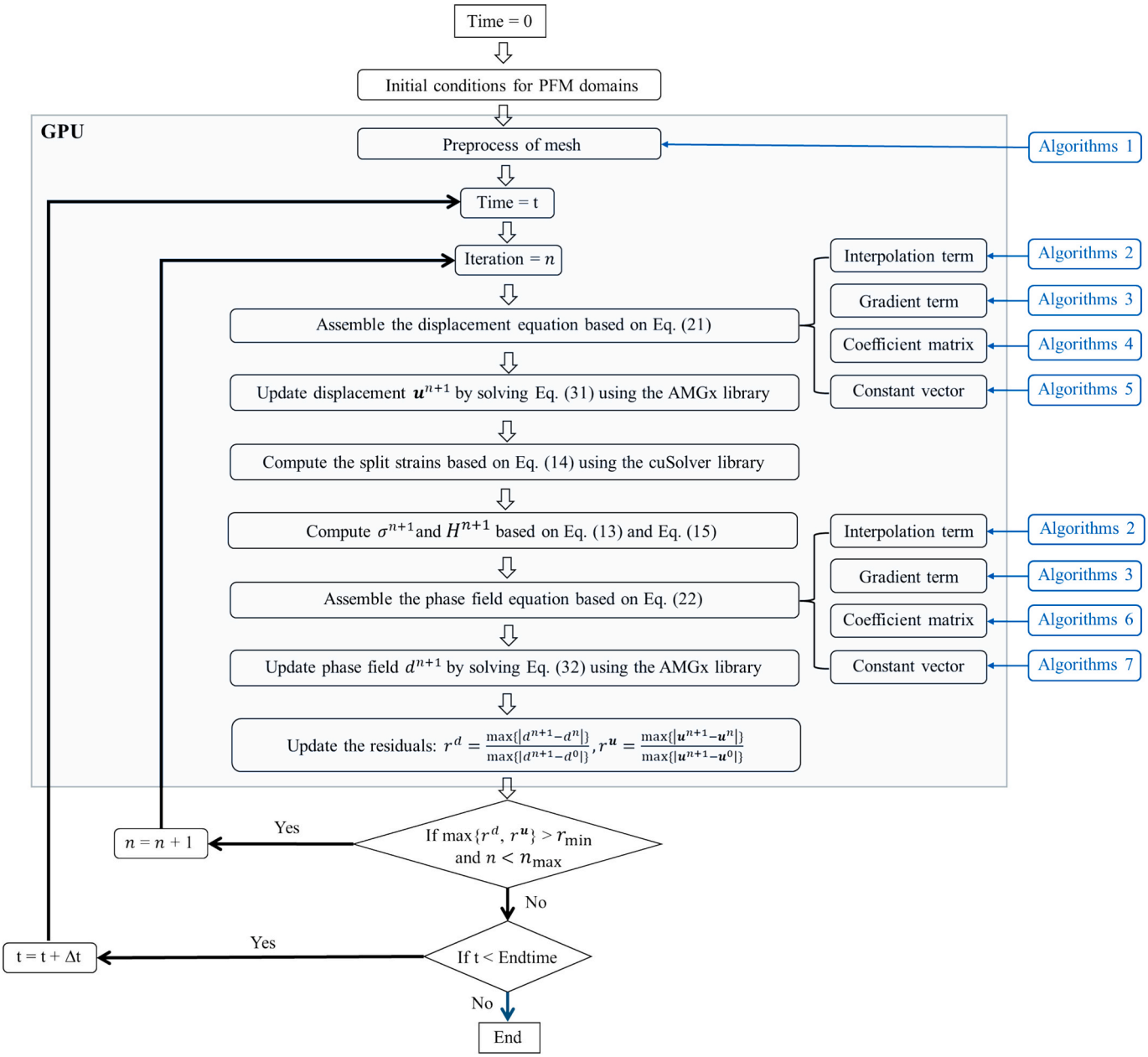


Fig. 3. Overall flowchart of the full-process GPU-parallelized FVM-based phase-field model.

iterations can lead to higher accuracy. The relative residual tolerance for iterations is set to $r_{\min} = 10^{-4}$, and the maximum number of iterations is set to $n_{\max} = 1,000$.

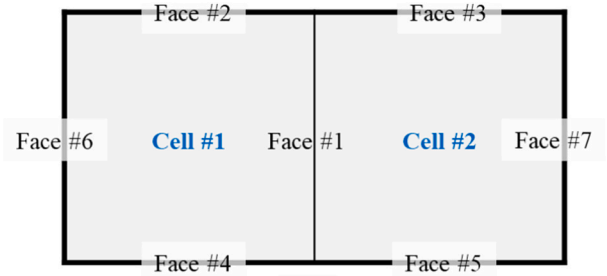
2.3.1. *Data structure of mesh*

In the GPU-parallelized finite volume method (FVM), the discretization and assembly of equations, as well as the GPU parallel algorithm, strongly depend on the mesh data structure. Therefore, it is essential to first introduce the storage scheme for the mesh information in GPU memory.

There are two typical data structures to describe the mesh information, i.e., the cell-based structure and the face-based structure. Specifically, the cell-based structure is widely used in the structured mesh, where the surrounding cell indices can be easily obtained according to the current cell index due to the regular distribution of the structured mesh (Kuo et al., 2020). However, the structured mesh is hard to be used in cases with complex geometry, such as complex topography and mechanical structures. Face-based structure is commonly used in the unstructured mesh, which is suitable for arbitrary geometry.

In this study, the face-based structure commonly used in OpenFOAM is employed to support complex computational domains and facilitate direct coupling with CFD. Three lists of basic information are selected to describe the mesh topology, including the point coordinates, the point indices of faces, and the boundary information, and two lists of face relationships, including the owner and neighbor cells of faces, as shown in Fig. 4. Note that, there is no neighbor cell for the boundary face as there is only one cell adjacent to the boundary face and this cell is defined as the owner cell. By iterating all the faces, the connection relationship between the cells can be obtained.

Different from the CPU parallel algorithm, where the computational domain should be divided into multiple subdomains for each CPU thread, the proposed GPU parallel algorithm sets a new corresponding relation, i.e., one GPU thread is responsible for one cell/face. Therefore, in the subsequent GPU parallel algorithm, the required computations can be efficiently performed by directly iterating over each mesh cell or face. It means that extra treatment for domain decomposition is not



(a)

Inner face index	Owner cell index	Neighbor cell index
1	1	2

(b)

Boundary face index	Owner cell index
2	1
3	2
4	1
5	2
6	1
7	2

(c)

Fig. 4. Schematic illustration of the face-based structure of mesh: (a) the face indices and cell indices; (b) the owner cell indices and neighbor cell indices for the inner face; (c) the owner cell indices for the neighbor faces.

required in the proposed GPU-accelerated finite volume framework and slow communications between CPU threads can be further avoided and replaced by the fast global memory in GPU.

2.3.2. Interpolation term

The interpolation scheme is required to calculate the terms $(\nabla\mathbf{u})_f$, $(\nabla d)_f$, and σ_f) in Eqs. (31) and (32) for the further equation discretization and assembly. In this work, a linear interpolation scheme is adopted to calculate a variable at the face center, which means that the variable is assumed to vary linearly within a cell. Take the stress σ for example:

$$(\sigma)_f \approx \gamma_f(\sigma)_P + \left(1-\gamma_f\right)(\sigma)_N \tag{33}$$

where the weight γ_f is calculated based on the following formula:

$$\gamma_f = \frac{(\mathbf{x}_N - \mathbf{x}_f)\bullet\mathbf{d}_f}{(\mathbf{x}_N - \mathbf{x}_P)\bullet\mathbf{d}_f} \tag{34}$$

where $\mathbf{x}_P$ and $\mathbf{x}_N$ are the center coordinates of the cells P and N, respectively. $\mathbf{x}_f$ is the center coordinate of the shared face f between cells P and N, as shown in Fig. 2. $\Delta = \cos\theta|\mathbf{d}_f|$.

Algorithm 1 shows the steps to calculate the interpolation term (Eq. (33)) using the GPU parallel algorithm. $(\nabla\mathbf{u}^n)_f$, σ_f, and $(\nabla d^n)_f$ in Eqs. (31) and (32) are calculated using this algorithm. Since a boundary face has only one owner cell, whereas an inner face has both an owner cell and a neighbor cell, two separate GPU kernels are utilized to process inner faces and boundary faces, respectively.

Algorithm 1. *Computation of the interpolation term*

1: **1*st* GPU kernel:** *Iterate over inner faces*
2: *Get the owner cell index* $\mathrm{ID_o}$ *and neighbor cell index* $\mathrm{ID_n}$ *for inner face* i;
3: *Calculate the weight* γ_f *using Eq.* (34);
4: *Calculate the interpolation* $\mathrm{f}(x)$: $\mathrm{f}(x) = \gamma_f x[\mathrm{ID_o}] + \left(1-\gamma_f\right)x[\mathrm{ID_n}]$;
5: **2*nd* GPU kernel:** *Iterate over boundary faces*
6: *Get the owner cell index* $\mathrm{ID_o}$ *for inner face* i;
7: *Calculate the interpolation* $\mathrm{f}(x)$: $\mathrm{f}(x) = \mathrm{f_b}(x)$;

Note: $\mathrm{f_b}(\mathrm{x})$ means the boundary value of the variable, which is dominated by the boundary condition shown in Eqs. (10)–(12).

2.3.3. Explicit gradient term

The gradient scheme is required to calculate the terms ($\nabla\mathbf{u}$ and ∇d) in Eqs. (31) and (32) for the further equation discretization and assembly. There are two typical algorithms in FVM to calculate the gradient term, including the Green-Gauss gradient scheme and the least-square gradient scheme. The least-square gradient scheme gets more accurate results than the Green-Gauss gradient scheme, especially for the unstructured mesh. Take displacement for example, its gradient calculated by the least-square gradient scheme is shown in Eq. (35)

$$(\nabla\mathbf{u})_P \approx \left[\sum\nolimits_f w_f^2\mathbf{d}_f\mathbf{d}_f\right]^{-1}\sum\nolimits_f\left[w_f^2\mathbf{d}_f(\mathbf{u}_N-\mathbf{u}_P)\right] \tag{35}$$

where $\mathbf{d}_f$ is the vector from the cell center P to the cell center N, as shown in Fig. 2. w_f is the weight and is taken as $|\mathbf{d}_f|^{-1}$ (Jasak and Weller, 2000) n this work.

Algorithm 2 shows the steps to calculate the explicit gradient term (Eq. (35)) using the GPU parallel algorithm. $\nabla\mathbf{u}^n$ and ∇d^n in Eqs. (31) and (32) are calculated using this algorithm. Two GPU kernels are adopted to calculate two terms in Eq. (35). Two GPU kernels are employed to compute Eq. (35) in a stepwise manner. First, $\left[\sum_f w_f^2\mathbf{d}_f\mathbf{d}_f\right]^{-1}$ is computed using the 1st GPU kernel, and then it is used as a known value to solve the entire equation for $(\nabla\mathbf{u})_P$.

Algorithm 2. Computation of the gradient term

1: **1*st* GPU kernel:** *Iterate over faces*
2: *Get the owner cell index* $\mathrm{ID_o}$ *and neighbor cell index* $\mathrm{ID_n}$ *for inner face* i;
3: *Calculate the coefficient* C_Δ : $\mathrm{C}_\Delta[i] += w_f^2\mathbf{d}_f\mathbf{d}_f$ *using the atomic operation*;
4: **2*nd* GPU kernel:** *Iterate over inner faces*
5: *Get the owner cell index* $\mathrm{ID_o}$ *and neighbor cell index* $\mathrm{ID_n}$ *for inner face* i;
6: *Calculate* ∇x : $\nabla x += \left[w_f^2\mathbf{d}_f(x[\mathrm{ID_n}]-x[\mathrm{ID_o}])\right]/\mathrm{C}_\Delta$ *using the atomic operation*;
7: **3*rd* GPU kernel:** *Iterate over boundary faces*
8: *Get the owner cell index* $\mathrm{ID_o}$ *for boundary face* i;
9: *Calculate* ∇x : $\nabla x += \left[w_f^2\mathbf{d}_f(x[i]-x[\mathrm{ID_o}])\right]/\mathrm{C}_\Delta[i]$ *using the atomic operation*;

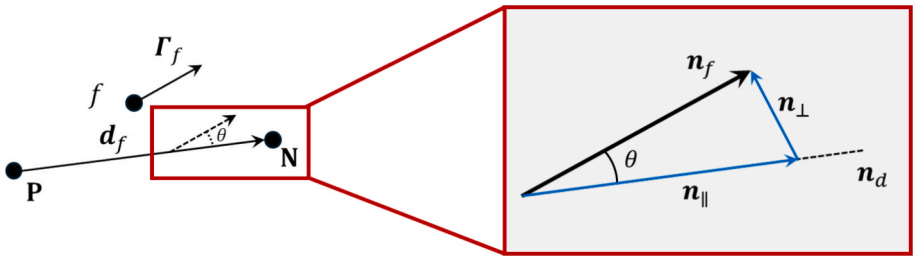


Fig. 5. The overrelaxation scheme of non-orthogonal treatment. $\boldsymbol{n}_d$ is the normalized direction of $\mathbf{d}_f$ and $\boldsymbol{n}_f$ is the normalized direction of Γ_f.

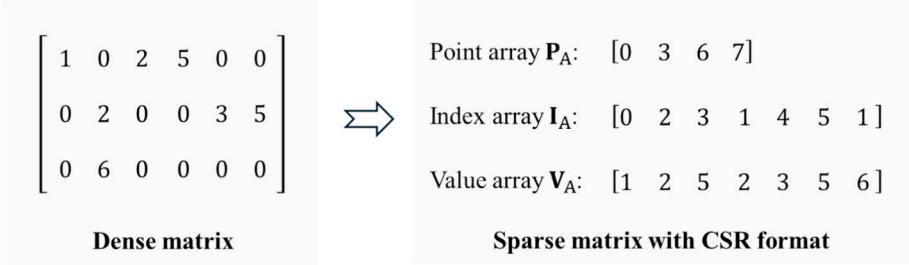


Fig. 6. A transition example from a dense matrix to a sparse matrix with CSR format.

2.3.4. Implicit gradient term

FVM adopts a face-based gradient scheme to implicitly discrete the gradient term at the face *f*. Take a structured mesh for example, i.e., the directions of the face vector Γ_f and the cell vector $\mathbf{d}_f$ are the same, the displacement gradient $(\nabla \mathbf{u}^{n+1})_f$ of the displacement at the face *f* can be discretized as:

$$(\nabla \mathbf{u}^{n+1})_f \bullet \Gamma_f \approx \frac{\mathbf{u}_N^{n+1} - \mathbf{u}_P^{n+1}}{|\mathbf{d}_f|^2}\mathbf{d}_f \bullet \Gamma_f = \frac{\mathbf{u}_N^{n+1} - \mathbf{u}_P^{n+1}}{|\mathbf{d}_f|} \tag{36}$$

However, for the unstructured mesh, the discretization of the Implicit gradient term cannot be simplified as Eq. (36). An extra non-orthogonal treatment called overrelaxation scheme is commonly used to improve discretization accuracy. The basic idea is to discretize the face vector Γ_f using Eq. (37), as shown in Fig. 5.

$$\Gamma_f = A\boldsymbol{n}_f = A(\boldsymbol{n}\| + \boldsymbol{n}_\perp) \tag{37}$$

where A is the face area. $\boldsymbol{n}_f$ is the normalized face vector. $\boldsymbol{n}\|$ is the vector parallel to the $\boldsymbol{n}_d$, which is the normalized cell vector d_f. $\boldsymbol{n}_\perp$ is the vector perpendicular to the $\boldsymbol{n}_d$. θ is the angle between $\boldsymbol{n}_f$ and d_f. $\boldsymbol{n}\|$ satisfies $\boldsymbol{n}\| = \boldsymbol{n}_d/\cos\theta$ and $\boldsymbol{n}_\perp$ satisfies $\boldsymbol{n}_\perp = \boldsymbol{n}_f - \boldsymbol{n}_d/\cos\theta$.

Therefore, the implicit gradient term in Eq. (36) can be rewritten as Eq. (38). The term $\left(\nabla d^{n+1}\right)_f \bullet \Gamma_f$ in Eq. (32) can be calculated in the same way.

$$\begin{aligned}(\nabla \mathbf{u}^{n+1})_f \bullet \Gamma_f &= (\nabla \mathbf{u}^{n+1})_f A(\boldsymbol{n}\| + \boldsymbol{n}_\perp) \\ &\approx A\frac{\mathbf{u}_N^{n+1} - \mathbf{u}_P^{n+1}}{|\mathbf{d}_f|^2}\mathbf{d}_f \bullet \boldsymbol{n}\| + A(\nabla \mathbf{u}^n)_f \boldsymbol{n}_\perp \\ &= A\frac{\mathbf{u}_N^{n+1} - \mathbf{u}_P^{n+1}}{\cos\theta|\mathbf{d}_f|} + A(\nabla \mathbf{u}^n)_f\left(\boldsymbol{n}_f - \frac{\mathbf{d}_f}{\cos\theta|\mathbf{d}_f|}\right) \\ &= A\frac{\mathbf{u}_N^{n+1} - \mathbf{u}_P^{n+1}}{\Delta} + A(\nabla \mathbf{u}^n)_f\left(\boldsymbol{n}_f - \frac{\mathbf{d}_f}{\Delta}\right)\end{aligned} \tag{38}$$

where $\Delta = \cos\theta|\mathbf{d}_f|$ and $(\nabla \mathbf{u}^n)_f$ means the explicit gradient term of the displacement calculated from the current displacement field.

For the two classical boundary conditions, Dirichlet and Neumann boundary conditions, Eq. (38) can be simplified as follows:

$$(\nabla \mathbf{u}^{n+1})_f \bullet \Gamma_f = \begin{cases} \nabla \mathbf{u}_b for Neumann boundary \\ A\dfrac{\mathbf{u}_b - \mathbf{u}_P^{n+1}}{\Delta} + A(\nabla \mathbf{u}^n)_P\left(\boldsymbol{n}_f - \dfrac{\mathbf{d}_f}{\Delta}\right) for Dirichlet boundary \end{cases} \tag{39}$$

Where $\nabla \mathbf{u}_b$ represents the displacement gradient at the Neumann boundary and $\mathbf{u}_b$ denotes the displacement at the Dirichlet boundary. The term $\Delta = \cos\theta|\mathbf{d}_f|$, where $|\mathbf{d}_f|$ is the distance from the boundary face center to the owner cell center.

2.3.5. Algebraic linear equation

By incorporating the explicit interpolation and gradient terms, along with the implicit gradient term, as presented in Eqs. (35), (38), and (33), the two governing equations (Eqs. (31) and (32)) can be reformulated as follows:

$$\sum K_f A\frac{\mathbf{u}_N^{n+1} - \mathbf{u}_P^{n+1}}{\Delta} = \sum K_f A(\nabla \mathbf{u}^n)_f \frac{\mathbf{d}_f}{\Delta} - \sum \sigma_f \bullet \Gamma_f - \mathbf{b}V \tag{40}$$

$$\left[1 + \frac{2Hl_0}{G_c}\right]Vd_P^{n+1} - \sum \frac{l_0^2 A}{\Delta}\left(d_N^{n+1} - d_P^{n+1}\right) = \sum l_0^2 A(\nabla d^n)_f\left(\boldsymbol{n}_f - \frac{\mathbf{d}_f}{\Delta}\right) + \frac{2HV}{G_c} \tag{41}$$

where the left-hand side denotes the implicit part, and the right-hand side denotes the explicit part.

Eqs. (40) and (41) can be finally rewritten as two algebraic linear equations (Eqs. (42) and (43)).

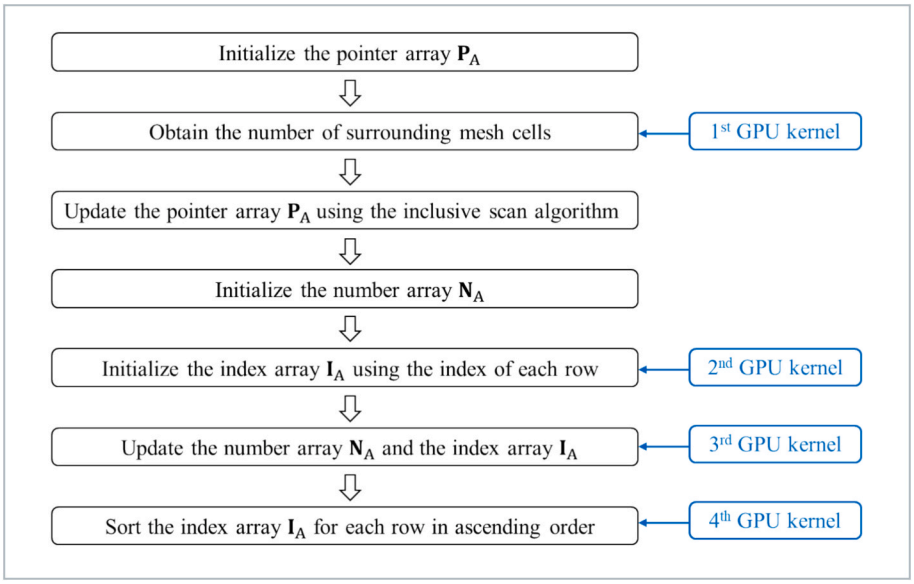


Fig. 7. Flowchart for the construction of the index array and pointer array.

Table 1
AmgX setup for the displacement equation and the phase-field equation.

Variable	Solver	Max PCG iterations	Convergence mode	PCG tolerance	Preconditioner	Max AMG levels
u	PCG	100	Absolute	10^{-5}	AMG	100
d	PCG	100	Absolute	10^{-15}	AMG	100

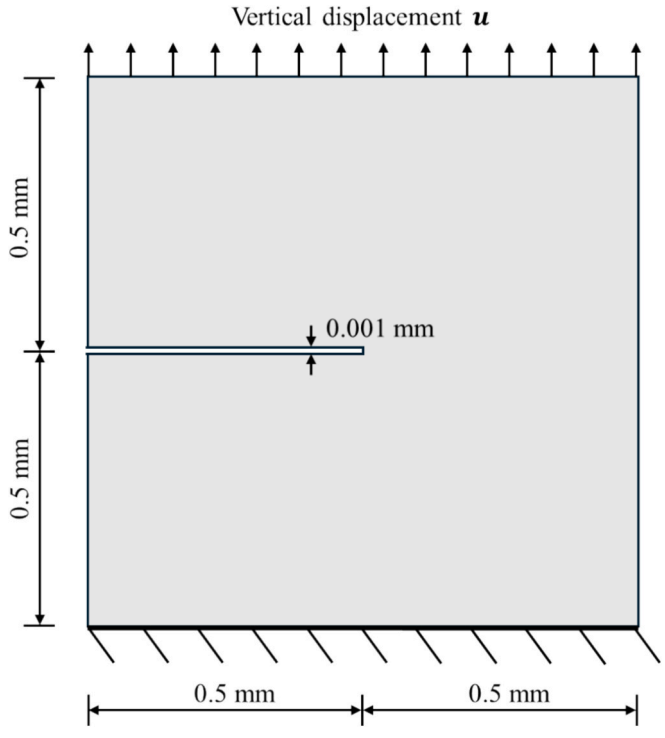


Fig. 8. Schematic illustration of the single-edge notched plate.

$$[\mathrm{A}^{\mathbf{u}}][\mathbf{u}] = [\mathrm{B}^{\mathbf{u}}] \tag{42}$$

$$[\mathrm{A}^{d}][d] = [\mathrm{B}^{d}] \tag{43}$$

where $[\mathrm{A}^{\mathbf{u}}]$ and $[\mathrm{A}^{d}]$ are the assembled coefficient matrices. $[\mathrm{B}^{\mathbf{u}}]$ and $[\mathrm{B}^{d}]$ are the constant vectors.

In this study, a sparse matrix format called Compressed Sparse Row (CSR) is adopted to store the coefficient matrix [A], e.g., $[\mathrm{A}^{\mathbf{u}}]$ and $[\mathrm{A}^{d}]$. CSR is an efficient storage format for sparse matrices that is optimized for memory saving, fast row slicing and matrix–vector products. In the CSR format, the coefficient matrix [A] is represented by three arrays: value $\mathbf{V}_A$, index $\mathbf{I}_A$, and pointer $\mathbf{P}_A$ (Dutto et al., 2000).

Value array $\mathbf{V}_A$ stores all the non-zero elements of the coefficient matrix [A]. Index array $\mathbf{I}_A$ stores the column indices of the non-zero elements in the value array $\mathbf{V}_A$. Pointer array $\mathbf{P}_A$ stores the starting position of each row in the value array. Specifically, $\mathbf{P}_A[0]$ is zero, and $\mathbf{P}_A[n+1] - \mathbf{P}_A[n]$ is the number of non-zero elements in the n^{th} row of the coefficient matrix [A]. Fig. 6 shows a simple example of the transition from a dense matrix to a sparse matrix with CSR format. The size of the pointer array $\mathbf{P}_A$ is $(n_{row} + 1)$, where n_{row} is the row number of the dense matrix. The size of the index array $\mathbf{I}_A$ and the value array $\mathbf{V}_A$ is the number of non-zero elements of the dense matrix.

It should be noted that the point array $\mathbf{P}_A$ and index array $\mathbf{I}_A$ are only related to the mesh topology, which means that these two arrays can be calculated in advance. This is because the non-zero elements only occur when two cells are connected. Algorithm 3 lists the process to calculate the point array $\mathbf{P}_A$ and index array $\mathbf{I}_A$ based on the mesh data using the GPU parallel algorithm. Fig. 7 further presents the flowchart for the construction of the index array and pointer array, highlighting the GPU

kernels from Algorithm 3 used in each step. Here, the number array $\mathbf{N}_A$ serves as a temporary variable used to determine the position of values within the index array $\mathbf{I}_A$, thereby preventing read-write conflicts during the GPU parallel execution.

Algorithm 3. Construction of the index array and pointer array

Initialize the pointer array $\mathbf{P}_A$;

1st GPU kernel: Iterate over inner faces
  Get the owner cell index ID_o and neighbor cell index ID_n for inner face i;
  Compute $\mathbf{P}_A$: $\mathbf{P}_A[ID_o]$ += 1 and $\mathbf{P}_A[ID_n]$ += 1 using the atomic operation;

Use the inclusive scan algorithm from the thrust library to update $\mathbf{P}_A$;
Initialize the index array $\mathbf{I}_A$ and the number array $\mathbf{N}_A$;

2nd GPU kernel: iterate over cells
  Initialize index array $\mathbf{I}_A$ with $\mathbf{I}_A[\mathbf{P}_A[i]] = i$ for cell i;

3rd GPU kernel: Iterate over inner faces
  Get the owner cell index ID_o and neighbor cell index ID_n for inner face i;
  Compute $\mathbf{N}_A$: $\mathbf{N}_A[ID_o]$+= 1 and $\mathbf{N}_A[ID_n]$+= 1 using the atomic operation;
  Update $\mathbf{I}_A$: $\mathbf{I}_A[\mathbf{P}_A[ID_o]+\mathbf{N}_A[ID_o]]$=$ID_n$ and $\mathbf{I}_A[\mathbf{P}_A[ID_n]+\mathbf{N}_A[ID_n]]$= ID_o;

4th GPU kernel: iterate over cells
  Sort the array ($\mathbf{I}_A[\mathbf{P}_A[i]]$ to $\mathbf{I}_A[\mathbf{P}_A[i+1]-1]$) in ascending order;

Eqs. (40) and (41) are utilized to assemble two algebraic linear equations (Eqs. (42) and (43)) within the GPU-parallelized framework. The following algorithms outline the detailed steps for computing the value arrays ($\mathbf{V}_A^{\mathbf{u}}$ and $\mathbf{V}_A^{d}$), as well as the constant vectors ($\mathrm{B}^{\mathbf{u}}$ and B^{d}), based on Eqs. (40) and (41) using the GPU parallel algorithm. It is important to note that the constant vector $\mathrm{B}^{\mathbf{u}}$ consists of three components, corresponding to the three displacement components.

The governing equations (Eqs. (40) and (41)) for both displacement and the phase field contain an implicit gradient term, differing only in their coefficients. Therefore, the same discretization and assembly strategy can be applied to both. By iterating over inner faces and boundary faces separately and utilizing the implicit gradient discretization formula from Eq. (38), the coefficient contributions corresponding to the owner or neighbor cells of each face can be directly assembled into the coefficient matrix (1st and 2nd GPU kernels in Algorithm 4 and Algorithm 5). Specifically, the contribution of the implicit gradient term to the owner cell is $A\left(\mathbf{u}_N^{n+1}-\mathbf{u}_P^{n+1}\right)/\Delta$, while its contribution to the neighbor cell is $-A\left(\mathbf{u}_N^{n+1}-\mathbf{u}_P^{n+1}\right)/\Delta$, where the negative sign for the neighbor cell arises from the convention that the positive direction of the face normal is defined as pointing from the owner cell to the neighbor cell.

Compared to the displacement equation, the phase-field equation includes an additional implicit linear term. Consequently, an extra GPU kernel is introduced to handle the assembly of the implicit linear term (3rd GPU kernel in Algorithm 5). Additionally, in the face-based parallel algorithm, atomic operations must be utilized when writing data associated with either the owner or neighbor cells. This prevents memory read-write conflicts, which could otherwise result in erroneous computations.

Algorithm 4. Computation of the coefficient matrix $[\mathrm{A}^{\mathbf{u}}]$ for the displacement equation

1st GPU kernel: Iterate over inner faces
  Get the owner cell index ID_o and neighbor cell index ID_n for inner face i;
  Find $\mathbf{I}_A[j] = ID_o$ for $j \in [\mathbf{P}_A[ID_o], \mathbf{P}_A[ID_o])$;
  Calculate $\mathbf{V}_A^{\mathbf{u}}$: $\mathbf{V}_A[j] -= K_f A/\Delta$ using the atomic operation;
  Find $\mathbf{I}_A[j] = ID_n$for $j \in [\mathbf{P}_A[ID_o], \mathbf{P}_A[ID_o])$;
  Calculate $\mathbf{V}_A^{\mathbf{u}}$: $\mathbf{V}_A^{\mathbf{u}}[j] += K_f A/\Delta$ using the atomic operation;
  Find $\mathbf{I}_A[j] = ID_o$ for $j \in [\mathbf{P}_A[ID_n], \mathbf{P}_A[ID_n])$;
  Calculate $\mathbf{V}_A^{\mathbf{u}}$: $\mathbf{V}_A^{\mathbf{u}}[j] += K_f A/\Delta$ using the atomic operation;
  Find $\mathbf{I}_A[j] = ID_n$for $j \in [\mathbf{P}_A[ID_n], \mathbf{P}_A[ID_n])$;
  Calculate $\mathbf{V}_A^{\mathbf{u}}$: $\mathbf{V}_A^{\mathbf{u}}[j] -= K_f A/\Delta$ using the atomic operation;

(*continued on next column*)

(*continued*)

Algorithm 4. Computation of the coefficient matrix $[\mathrm{A}^{\mathbf{u}}]$ for the displacement equation

2nd GPU kernel: Iterate boundary faces
  Get the owner cell index ID_o for boundary face i;
  Find $\mathbf{I}_A[j] = ID_o$ for $j \in [\mathbf{P}_A[ID_o], \mathbf{P}_A[ID_o])$;
  If the boundary condition = Dirichlet:
    Calculate $\mathbf{V}_A^{\mathbf{u}}$: $\mathbf{V}_A^{\mathbf{u}}[j] -= K_f A/\Delta$ using the atomic operation;
    Calculate $\mathrm{B}^{\mathbf{u}}$: $\mathrm{B}^{\mathbf{u}}[ID_o] -= K_f A\mathbf{u}_b/\Delta$ using the atomic operation;
  If the boundary condition = Neumann:
    Calculate $\mathrm{B}^{\mathbf{u}}$: $\mathrm{B}^{\mathbf{u}}[ID_o] -= K_f A\nabla\mathbf{u}_b$ using the atomic operation;

Algorithm 5. Computation of the coefficient matrix $[\mathrm{A}^{d}]$ for the displacement equation

1st GPU kernel: Iterate over inner faces
  Get the owner cell index ID_o and neighbor cell index ID_n for inner face i;
  Find $\mathbf{I}_A[j] = ID_o$ for $j \in [\mathbf{P}_A[ID_o], \mathbf{P}_A[ID_o])$;
  Calculate $\mathbf{V}_A^{d}$: $\mathbf{V}_A^{d}[j] += l_0^2 A/\Delta$ using the atomic operation;
  Find $\mathbf{I}_A[j] = ID_n$for $j \in [\mathbf{P}_A[ID_o], \mathbf{P}_A[ID_o])$;
  Calculate $\mathbf{V}_A^{d}$: $\mathbf{V}_A^{d}[j] -= l_0^2 A/\Delta$ using the atomic operation;
  Find $\mathbf{I}_A[j] = ID_o$ for $j \in [\mathbf{P}_A[ID_n], \mathbf{P}_A[ID_n])$;
  Calculate $\mathbf{V}_A^{d}$: $\mathbf{V}_A^{d}[j] -= l_0^2 A/\Delta$ using the atomic operation;
  Find $\mathbf{I}_A[j] = ID_n$for $j \in [\mathbf{P}_A[ID_n], \mathbf{P}_A[ID_n])$;
  Calculate $\mathbf{V}_A^{d}$: $\mathbf{V}_A^{d}[j] += l_0^2 A/\Delta$ using the atomic operation;

2nd GPU kernel: Iterate boundary faces
  Get the owner cell index ID_o for boundary face i;
  Find $\mathbf{I}_A[j] = ID_o$ for $j \in [\mathbf{P}_A[ID_o], \mathbf{P}_A[ID_o])$;
  If the boundary condition = Dirichlet:
    Calculate $\mathbf{V}_A^{d}$: $\mathbf{V}_A^{d}[j] += l_0^2 A/\Delta$ using the atomic operation;
    Calculate B^{d} : $\mathrm{B}^{d}[ID_o] += l_0^2 A d_b/\Delta$ using the atomic operation;
  If the boundary condition = Neumann:
    Calculate B^{d} : $\mathrm{B}^{d}[ID_o] += l_0^2 A\nabla d_b$ using the atomic operation;

3rd GPU kernel: Iterate cells
  Calculate $\mathbf{V}_A^{d}$: $\mathbf{V}_A^{d}[i] += [1+2Hl_0/G_c]V$ for cell i;

The sources of the constant vectors in both the displacement and phase-field equations can be categorized into two types based on their storage locations. The first type is stored on the mesh faces, including the non-orthogonal correction of the implicit gradient term $K_f A(\nabla\mathbf{u}^n)_f\mathbf{d}_f/\Delta$ and the face stress σ_f. The second type is stored at the cell centers, including the body force $\mathbf{b}$ and the history state variable H. In the GPU-parallel algorithm, the former requires face-based iteration, where the contributions from each face are accumulated into the corresponding owner or neighbor cell, as implemented in the 1st GPU kernel in Algorithm 6 and Algorithm 7. In contrast, the latter is more straightforward, as it directly accumulates the contribution of each cell through cell-based iteration, as executed in the 2nd GPU kernel in Algorithm 6 and Algorithm 7.

Algorithm 6. Computation of the constant vector $[\mathrm{B}^{\mathbf{u}}]$ for the displacement equation

1st GPU kernel: Iterate over faces
  Get the owner cell index ID_o and neighbor cell index ID_n for inner face i;
  Calculate $\mathrm{B}^{\mathbf{u}}$: $\mathrm{B}^{\mathbf{u}}[ID_o] += \left(K_f A(\nabla\mathbf{u}^n)_f\frac{\mathbf{d}_f}{\Delta}-\sigma_f\bullet\Gamma_f\right)$ using the atomic operation;
  Calculate $\mathrm{B}^{\mathbf{u}}$: $\mathrm{B}^{\mathbf{u}}[ID_n] -= \left(K_f A(\nabla\mathbf{u}^n)_f\frac{\mathbf{d}_f}{\Delta}-\sigma_f\bullet\Gamma_f\right)$ using the atomic operation;

2nd GPU kernel: Iterate over cells
  Calculate $\mathrm{B}^{\mathbf{u}}$: $\mathrm{B}^{\mathbf{u}}[i] -= \mathbf{b}[i]V[i]$ for cell i;

Algorithm 7. Computation of the constant vector $[\mathrm{B}^{d}]$ for the phase field equation

1st GPU kernel: Iterate over faces
  Get the owner cell index ID_o and neighbor cell index ID_n for inner face i;
  Calculate B^{d} : $\mathrm{B}^{d}[ID_o] += l_0^2 A(\nabla d^n)_f(\boldsymbol{n}_f-\frac{\mathbf{d}_f}{\Delta})$ using the atomic operation;

(*continued on next page*)

(*continued*)

Algorithm 7. Computation of the constant vector $[\mathrm{B}^d]$ for the phase field equation

4: Calculate $\mathrm{B}^d : \mathrm{B}^d[\mathrm{ID_n}] -= l_0^2 A(\nabla d^n)_f(\boldsymbol{n}_f - \frac{\mathbf{d}_f}{\Delta})$ using the atomic operation;

5: **2nd GPU kernel:** Iterate over cells

6: Calculate $\mathrm{B}^d : \mathrm{B}^d[i] += 2H[i]V[i]/G_c$ for cell i;

2.3.6. Linear solver

The solver of two algebraic linear equations (Eqs. (42) and (43)) dominates the total efficiency of the GPU-accelerated FVM framework. In this work, the algebraic multigrid solver (AmgX) library is implemented to solve the linear equation, and the CUDA version is 12.1. AmgX is a high-performance library developed by NVIDIA that provides accelerated core solver technology on NVIDIA GPUs.

For the following cases in this paper, the algebraic multigrid method (AMG) with a V-cycle is selected as a preconditioner of the conjugate gradient method when solving the linear equation. Table 1 shows the AmgX setup used for solving the displacement equation and the phase-field equation. Since the absolute convergence mode is employed, the selection of tolerance is based on the magnitude of the coefficient matrix in the equation. According to the test results of the TFluid software (https://www.t-fluid.com), the time cost of solving two linear equations always accounts for around 80 % of the total time cost.

2.3.7. GPU implementation considerations

The proposed full-process GPU-accelerated finite volume framework is designed to minimize CPU-GPU communication by transferring all mesh data and initialization information to the GPU's global memory at the beginning of the simulation. During the computation phase, no further data transfer between CPU and GPU is required, except for exporting selected variables such as displacement, velocity, and pressure fields. This approach significantly reduces the communication overhead and improves computational throughput. Besides the input information, memory management across mesh scales is handled through a pre-allocated strategy. All memory allocations are performed once at the start of the simulation and remain in global memory throughout the run. This avoids the costly allocation and deallocation operations during time stepping. Data structures primarily consist of quantities stored at cell centers and face centers, whose total sizes are determined by the number of cells and faces in the unstructured mesh. These memory demands can thus be predicted and allocated in advance.

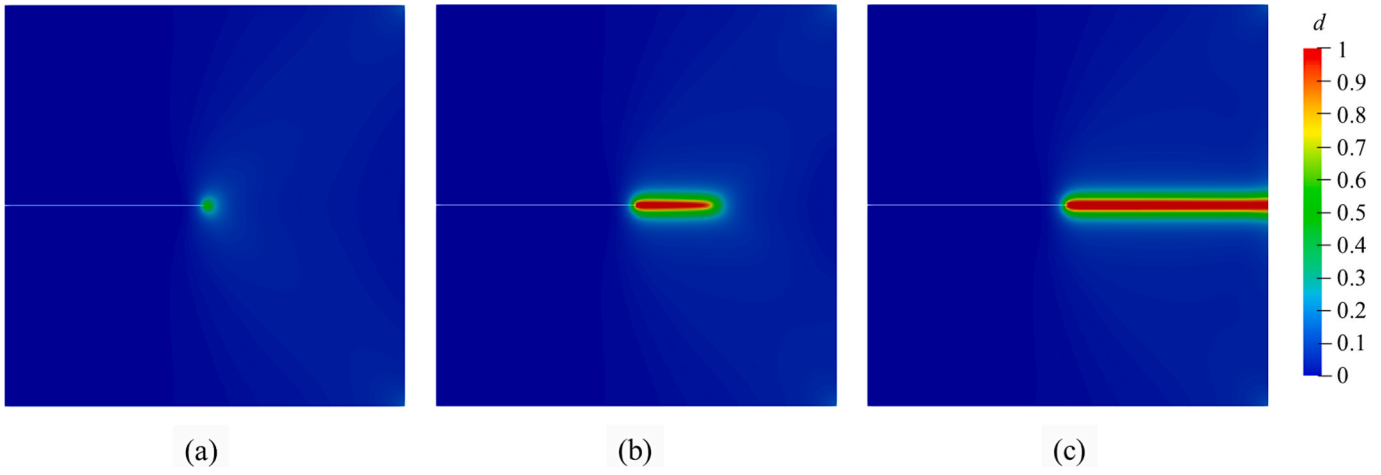


Fig. 9. Simulation results of crack patterns in single-edge notched tension test at displacements ($l_0/h = 8.2$): (a) $\boldsymbol{u} = 5.40 \times 10^{-3}$mm, (b) $\boldsymbol{u} = 5.70 \times 10^{-3}$mm, (c) $\boldsymbol{u} = 6.01 \times 10^{-3}$mm.

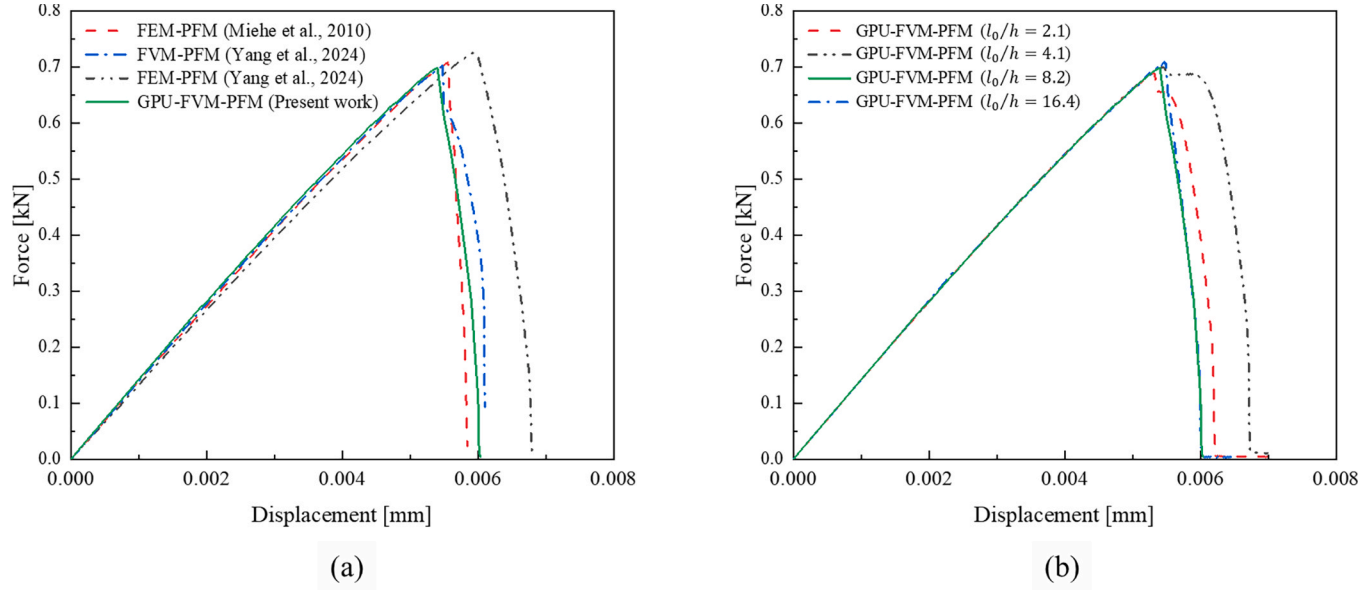


Fig. 10. Force-displacement curves of the single-edge notched tension test: (a) Comparison with existing simulation results; (b) Results for four different cell sizes.

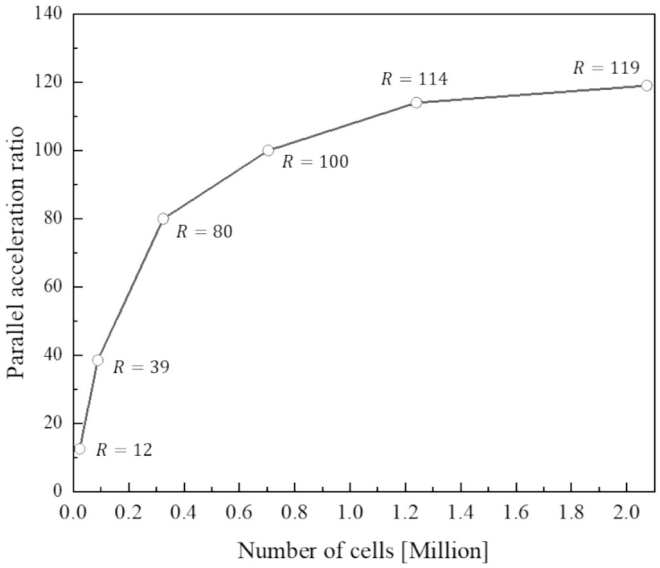


Fig. 11. GPU parallel acceleration ratio for different numbers of cells.

Table 2
Comparison of computational time between CPU and GPU parallel FVM-PFM.

Algorithm	Device	Mesh number	Run time [s]	Acceleration ratio R
CPU-FVM-PFM (Yang et al., 2024b)	AMD® Ryzen® CPU R7-5800X (1 CPU core)	22,308	5790	1
GPU-FVM-PFM (this work)	RTX 3060 Ti	22,540	117	12
		87,584	148	39
		324,324	260	80
		703,888	456	100
		1,240,420	704	114
		2,073,116	1130	119

Throughout the assembly of governing equations and variable updates, all data are accessed and manipulated in global memory. Notably, due to the irregular access patterns associated with unstructured meshes and the absence of highly localized, reuse-heavy computation blocks, shared memory or coalesced memory strategies were not found to offer measurable advantages in our current implementation. Instead, we adopt direct global memory access. To avoid race conditions when updating shared quantities, such as face contributions to cell-centered values, we employ atomic operations, as illustrated in Algorithm 2.

In the context of unstructured mesh data structures, synchronization presents additional challenges. While the computation within a single cell can proceed independently, operations involving face-cell relationships require traversing mesh faces to obtain complete topological information. This contrasts with structured grids, where neighboring cell access is trivial. In the proposed framework, many computational procedures are split into multiple GPU kernels to enforce synchronization points, ensuring consistency of intermediate quantities. This strategy is exemplified in Algorithm 1 to Algorithm 3.

For solving the resulting sparse linear systems, the NVIDIA AmgX library is integrated. The assembled matrices are stored in CSR format to match AmgX's data interface requirements. The solution process is fully GPU-resident, with the CSR row pointer, column index, and value arrays, along with the right-hand-side vector, directly passed to AmgX as described in Section 2.3.5. This avoids the need for host-device data exchange and allows efficient parallel linear solves.

3. Numerical simulations

This chapter presents three benchmark cases to verify the accuracy of the proposed GPU-accelerated finite volume framework for phase field modeling. Specifically, by comparing the results with those reported in previously published studies, we demonstrate the significant improvements in computational efficiency achieved by the GPU-accelerated algorithm.

3.1. Single-edge notched tension test

3.1.1. Comparison of simulation results

The single-edge notched plate is a classic benchmark problem commonly used for verification in various studies (Wu et al., 2020, Yang et al., 2024b), as shown in Fig. 8. The problem in this work consists of a plate with an initial crack located at the left-center, with a length of 0.5 mm and a width of 0.001 mm. The bottom of the plate is fixed, while the top is subjected to a vertical displacement-constrained load. The vertical displacement increment is $\Delta \boldsymbol{u} = 10^{-5}$ mm during the loading steps. The initial crack is modeled as a gap in the mesh domain. The model is truncated with a uniformly wide (0.001 mm) region designated as the initial crack, meaning that no mesh is applied in this region, as shown in Fig. 8. The material properties are characterized by a Young's modulus of $E = 210$GPa, a Poisson's ratio of $\nu = 0.3$, a critical energy release rate of $G_c = 2.7$N/mm, and a characteristic length of $l_0 = 0.015$mm. To facilitate the comparison of GPU performance under different numbers of grid points, this study employs fixed uniform mesh.

Fig. 9 illustrates the simulated crack development process as displacement increases, with a characteristic length-to-minimum cell size ratio (l_0/h) of 8.2. The crack initiates at a displacement of approximately 5.40×10^{-3} mm. As the displacement increases, the crack rapidly propagates and eventually penetrates the entire specimen at a displacement of 5.70×10^{-3} mm. This process is further depicted in Fig. 10(a), where a significant decrease in force is observed when the displacement reaches 5.40×10^{-3} mm.

The simulation results obtained using GPU-FVM-PFM align closely with those reported in the literature (Yang et al., 2024b, Miehe et al., 2010). Specifically, Specifically, we uniformly divided the force values into 130 intervals and extracted the corresponding 130 displacement values for comparison. The root mean square error (RMSE) of the displacement values is computed using Eq. (44). The results indicate that, compared to the solutions of Yang et al. (2024b) and Miehe et al. (2010), the RMSE of the present results is 2.47 % and 2.32 %, respectively. Additionally, the peak force calculated by GPU-FVM-PFM is 698.6 N, while Yang et al. (2024b) reported a peak force of 702.4 N using CPU-FVM-PFM, representing a difference of only 0.5 %. Moreover, the displacement corresponding to the peak force is 5.40×10^{-3} mm for GPU-FVM-PFM and 5.47×10^{-3} mm for CPU-FVM-PFM, with a discrepancy of approximately 1 %. Considering these results and accounting for differences in mesh resolution, it can be concluded that GPU-FVM-PFM achieves accuracy comparable to that of CPU-FVM-PFM.

$$\mathrm{RMSE} = \sqrt{\frac{1}{130}\sum_{i=1}^{130}\left(\frac{\mathbf{u}_i^{\mathrm{ref}} - \mathbf{u}_i}{\mathbf{u}_i^{\mathrm{ref}}}\right)^2} \tag{44}$$

where $\mathbf{u}_i$ represents the displacement obtained in the present study, while $\mathbf{u}_i^{\mathrm{ref}}$ denotes the displacement reported by Yang et al. (2024b) or Miehe et al. (2010).

Fig. 10(b) further investigates the influence of mesh size on the simulation results. Four different mesh sizes were selected, with characteristic length ratio to cell size l_0/h of approximately 2.1, 4.1, 8.2, and 16.4, representing a near-doubling progression for better representativeness. The results show that as the mesh is refined, the force–displacement curves gradually converge. In particular, the curves corresponding to $l_0/h = 8.2$ and 16.4 are nearly identical, indicating

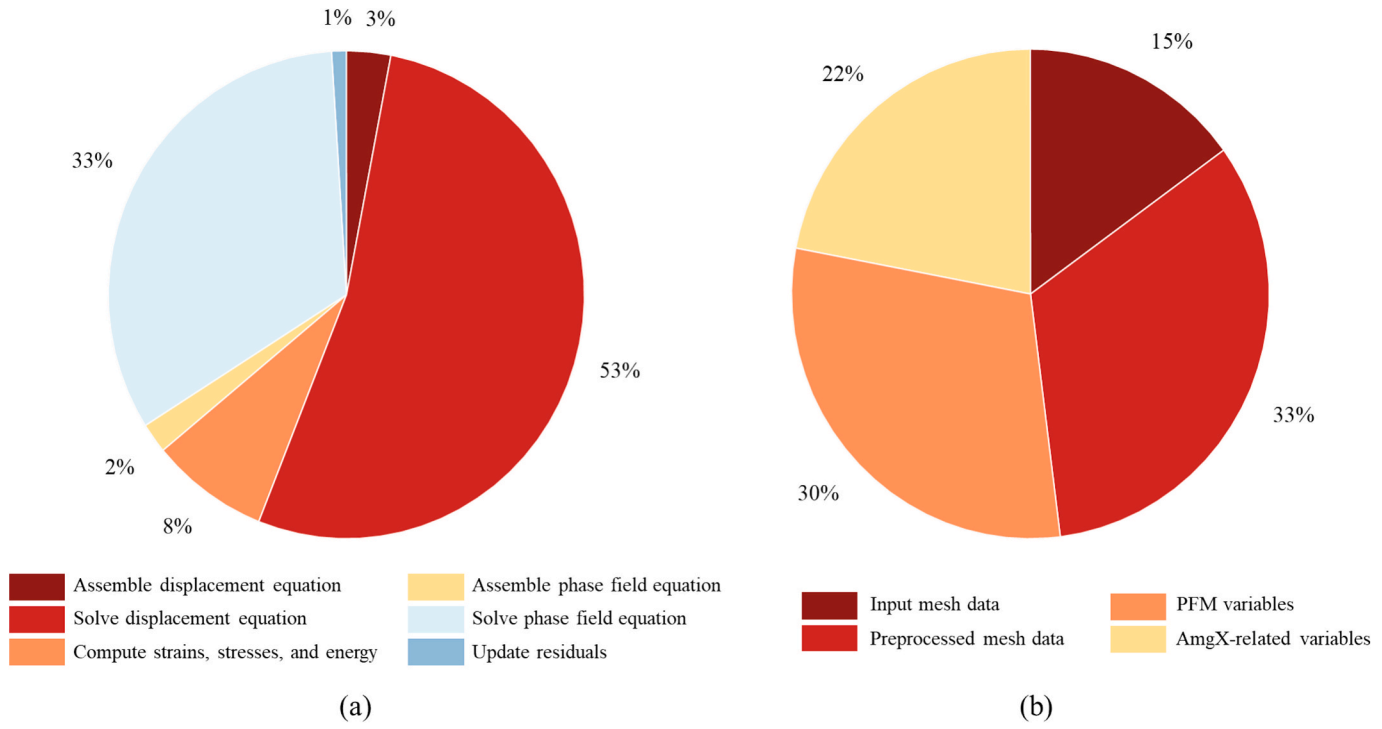


Fig. 12. (a) Average computation time distribution across different computational modules, based on the benchmark case with 2.07 million cells. (b) GPU memory usage per million cells for each module. The total memory consumption is approximately 1.2 GB, obtained from the same 2.07 million-cell case.

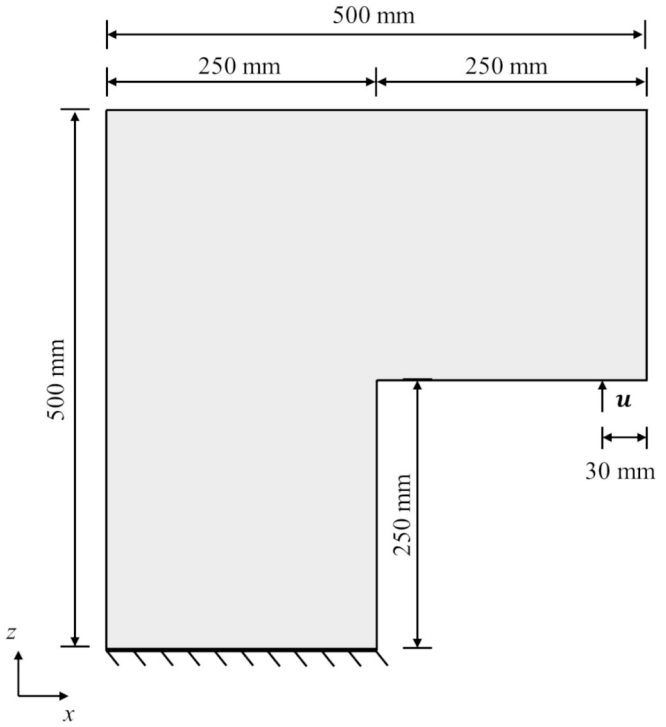


Fig. 13. Schematic illustration of the L-shaped panel test.

that a mesh resolution of one-eighth the characteristic length is sufficient to ensure accuracy and convergence of the results.

3.1.2. *Performance of the GPU-accelerated framework*

To further compare the efficiency of GPU-FVM-PFM and CPU-FVM-PFM, we use the results reported in Yang et al. (2024b) as a reference. In their study, Yang et al. (2024b) performed simulations using a single-core CPU (AMD® Ryzen® R7-5800X) for a test case with 22,308 mesh elements, completing the single-edge notched tension test in 5790 s. For a fair comparison, we selected the NVIDIA RTX 3060 Ti for benchmarking. This GPU was released in late 2020, around the same release time of AMD® Ryzen® R7-5800X CPU, and is priced slightly lower than the CPU.

In this study, we define a parameter, the parallel acceleration ratio R, to evaluate the efficiency of GPU parallelization. The parallel acceleration ratio is given by $R = \bar{t}_{\mathrm{CPU}}/\bar{t}_{\mathrm{GPU}}$, where $\bar{t}_{\mathrm{CPU}}$ and $\bar{t}_{\mathrm{GPU}}$ represent the computational time per unit grid on the CPU and GPU, respectively. According to Yang et al. (2024b), $\bar{t}_{\mathrm{CPU}}$ is computed as $\bar{t}_{\mathrm{GPU}} = 5790/22308 = 0.26$. For the GPU, $\bar{t}_{\mathrm{GPU}}$ is calculated as $\bar{t}_{\mathrm{GPU}} = t/N_c$, where N_c is the number of mesh cells and t is the total runtime. This ratio quantifies the equivalent computational capacity of the GPU in terms of the number of CPU cores.

To ensure a fair comparison, three key points are emphasized: (1) Comparable baseline performance: In CPU parallelization, the performance of each core is influenced by the number of mesh elements it processes. With each core handling tens of thousands of elements, the single-core CPU performance for 22,308 elements represents an efficient and fair baseline for comparison, avoiding comparisons with less optimal scenarios. (2) Consistent algorithms and models: The GPU-FVM-PFM and CPU-FVM-PFM simulations share the same underlying algorithms and model settings, with results that align closely. The only difference lies in the parallelization approach, ensuring the comparison is fair. (3) Suitability for large-scale computation: Considering that GPU parallelization is particularly well-suited for large-scale computations, we compare GPU parallel efficiency across varying mesh sizes while keeping other settings constant to provide a comprehensive evaluation.

Fig. 11 illustrates the variation in the parallel acceleration ratio across different grid sizes, with detailed results summarized in Table 2. The data indicates that the parallel acceleration ratio increases steadily with the grid size, eventually reaching a peak. This growth is more pronounced for grid sizes below 1 million. For example, at a grid size of approximately 20,000, the parallel acceleration ratio is around 25,

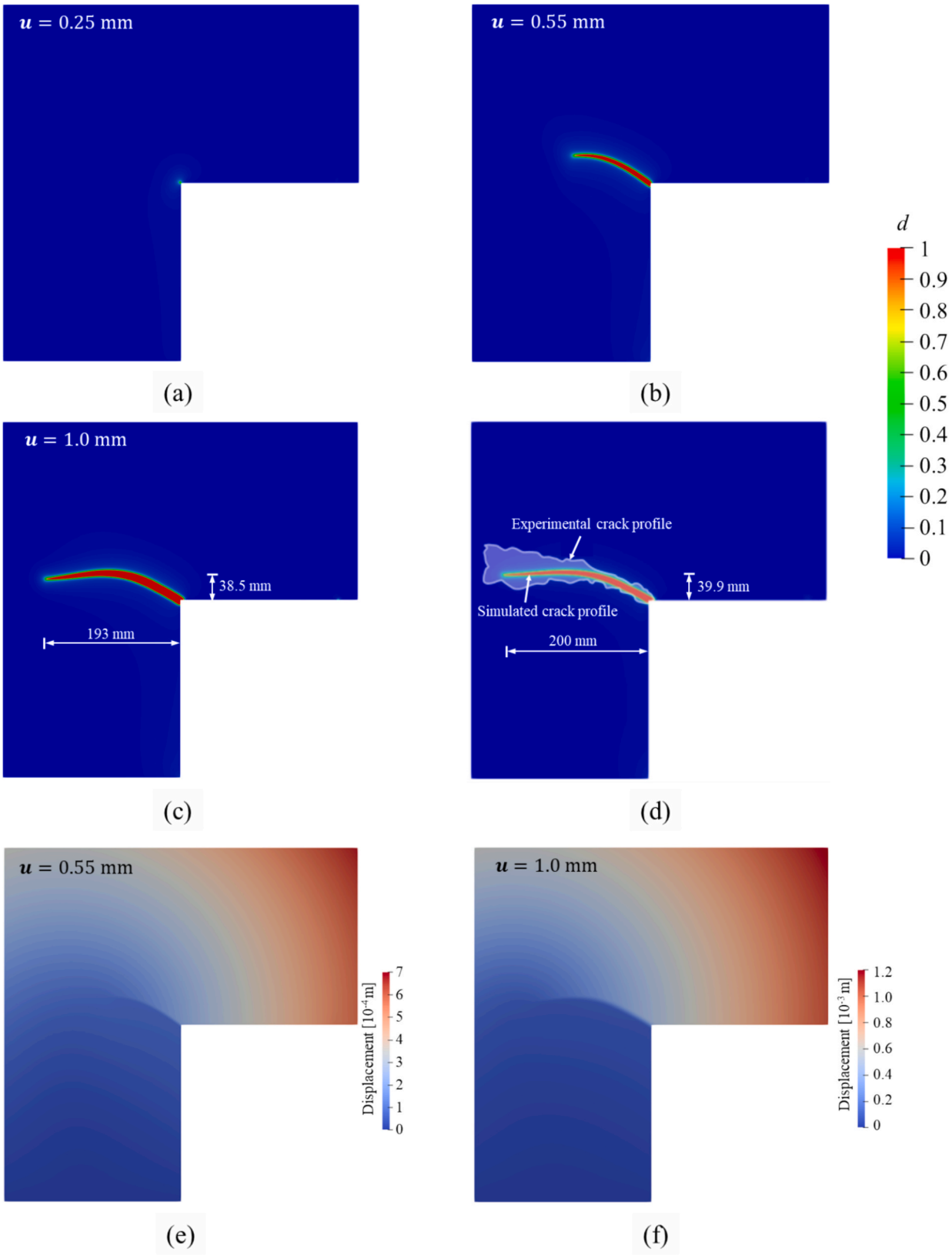


Fig. 14. Simulation results of crack patterns (a-c) and displacement fields (e-f) in L-shaped panel test at different displacements. (d) Experimental crack profile (Ambati et al., 2015) and simulated crack profile at displacement $\boldsymbol{u} = 1$mm from the literature (Yang et al., 2024b).

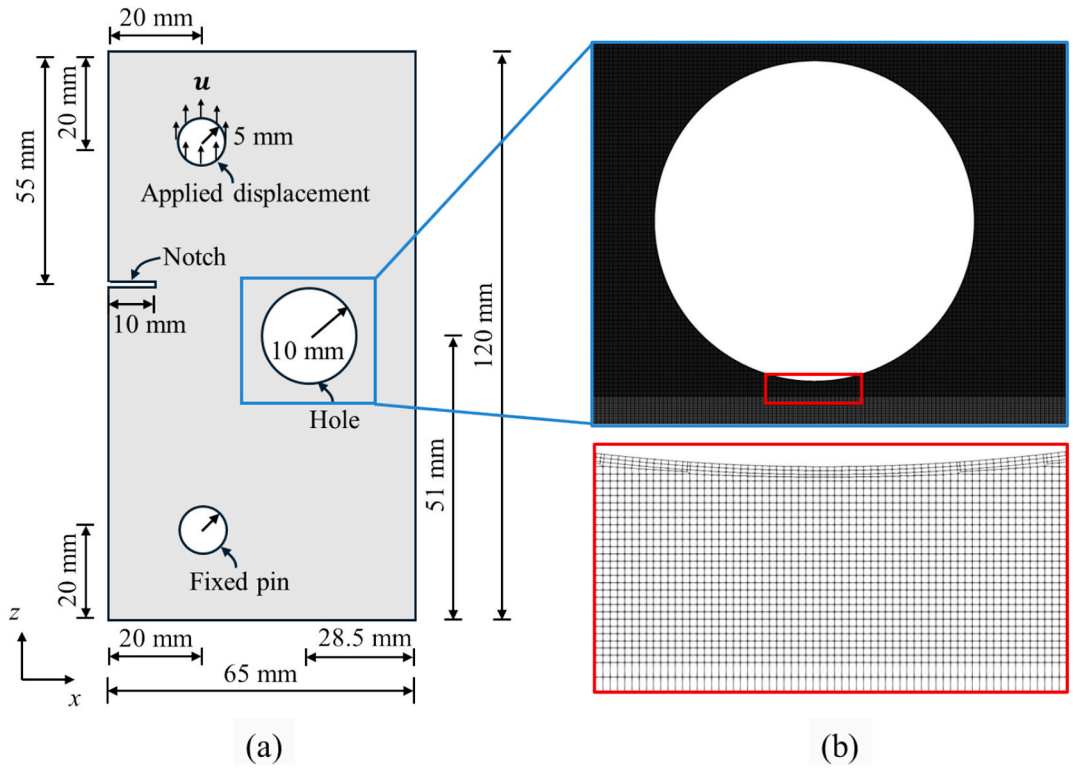


Fig. 15. (a) Schematic illustration and (b) mesh discretization of the notched plate with a hole (Case I). To better capture the fractures around the hole, FVM mesh is refined in its vicinity and added three boundary layers.

whereas for a grid size of 2 million, the ratio rises to 250, reaching a tenfold difference.

These findings further validate that GPU acceleration is particularly advantageous for large-scale computations, such as those involving over 2 million grids. For instance, in the 2-million-grid case, a single NVIDIA RTX 3060Ti GPU demonstrates computational capability equivalent to 119 cores of the AMD® Ryzen® CPU R7-5800X. Given that this CPU has 8 cores, a single RTX 3060Ti is effectively comparable to 15 AMD® Ryzen® R7-5800X processors. In terms of cost efficiency, the launch price of the RTX 3060Ti is $399, while the AMD® Ryzen® R7-5800X is priced at $449. The hardware cost for GPU parallelization is only 6 % of that for CPU parallelization with equivalent computational power.

It is also worth emphasizing that the reported parallel acceleration ratio represents a conservative estimate. As the number of CPU cores increases, communication overhead between cores often limits performance scalability. For example, a 32 CPU cores does not typically achieve a 32-fold speedup over one CPU core, making GPU-based parallelization even more advantageous in practice.

To provide a clearer understanding of the performance characteristics of the proposed framework, we further decompose the computational procedure for detailed analysis. The case involving 2.07 million cells, as listed in Table 2, is selected as a representative benchmark, as it fully utilizes the GPU's parallel computing capabilities and is thus well-suited for in-depth performance profiling. Throughout the entire computation, the GPU achieves an average occupancy of approximately 91 %, indicating a high level of parallel resource utilization.

Due to the unstructured nature of the mesh and the fine-grained design of the computational routines, a large number of kernel functions are invoked, each typically exhibiting very short execution times. Consequently, kernel-level profiling would offer limited insight while introducing excessive verbosity. For this reason, we do not report kernel-level timing or memory bandwidth statistics for individual kernels. Instead, we focus on profiling the major computational stages to more effectively capture the overall performance characteristics.

Fig. 12a presents the average computation time distribution and GPU memory usage across the main stages of the solution process, as outlined in the flowchart in Fig. 3. It is evident that solving the displacement and phase-field equations dominates the total runtime, accounting for approximately 86 %. This confirms that, in implicit methods, the solution of large-scale linear systems constitutes the primary computational bottleneck. The second most time-consuming stage involves the evaluation of stress, strain, and energy terms, which contributes around 8 % to the total runtime. This relatively high cost arises from the strain decomposition procedure (see Eq. (13)), which involves more complex operations than standard assembly routines. In contrast, the assembly of the governing equations requires only about 6 % of the total runtime, reflecting the high parallel efficiency enabled by our atomic-operation-based implementation. Finally, residual norm evaluation, which involves global extrema computations, accounts for approximately 1 % of the runtime.

Fig. 12b shows the breakdown of GPU memory usage into four primary categories: Input mesh data (15 %), Preprocessed mesh data (33 %), PFM variables (30 %), and AmgX-related variables (22 %). The "Input mesh data" category includes the basic unstructured mesh information described in Section 2.3.1. "Preprocessed mesh data" comprises auxiliary geometric and topological data derived from the input mesh, which are precomputed and stored prior to simulation to avoid redundant computations. "PFM variables" encompass all simulation-dependent quantities, such as displacement, strain, and stress. Lastly, "AmgX-related variables" refer to the temporary data structures generated during the linear system solution via the AmgX library, including coarse-level matrix representations and interpolation operators required by the AMG algorithm.

3.2. *L-shaped panel test*

The L-shaped panel test is a common benchmark in fracture mechanics used to evaluate crack propagation and stress intensity factors.

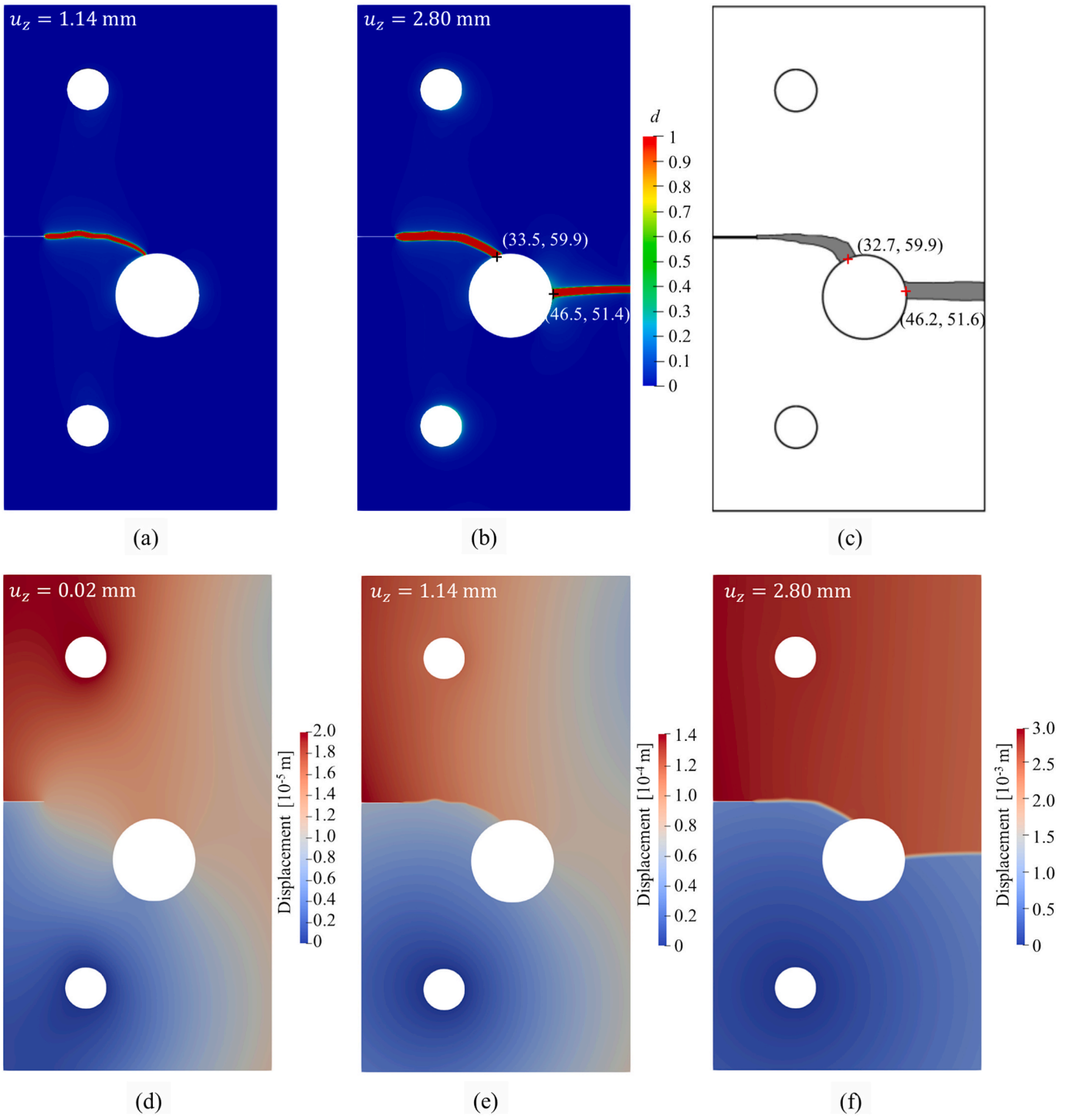


Fig. 16. Simulation results of crack patterns (a-b) and displacement fields (d-f) in notched plate with a hole at different displacements. (c) Experimental crack profile (Ambati et al., 2015).

Fig. 13 shows a concrete L-shaped panel subjected to a progressively increasing point displacement constraint. In the simulation, the point constraint is simplified by applying the displacement constraint over a region with a small width, equivalent to two mesh sizes. The vertical displacement increment is $\Delta \boldsymbol{u} = 3.33 \times 10^{-4}$ mm, applied until the displacement reaches 10 mm. The material properties are characterized by a Young's modulus of $E = 25.85$GPa, a Poisson's ratio of $\nu = 0.18$, a critical energy release rate of $G_c = 0.089$N/mm, and a characteristic length of $l_0 = 1.1875$mm. Uniform mesh is adopted in this case and the mesh size is 0.25 mm.

Fig. 14 presents the simulation results of the L-shaped panel test, along with comparisons to experimental results (Ambati et al., 2015) and simulated results (Yang et al., 2024b) from previous studies. The PFM enables high-fidelity simulations of the discontinuous displacement field across the fracture surfaces (Fig. 14(e) and (f)). The crack trajectory obtained using the GPU-PFM is highly consistent with the experimental results and the simulated results from the earlier studies. Naturally, minor and acceptable discrepancies, such as those in crack length and width, are observed between the two sets of simulation results. Specifically, when the displacement reaches 1 mm, the crack lengths in the x-

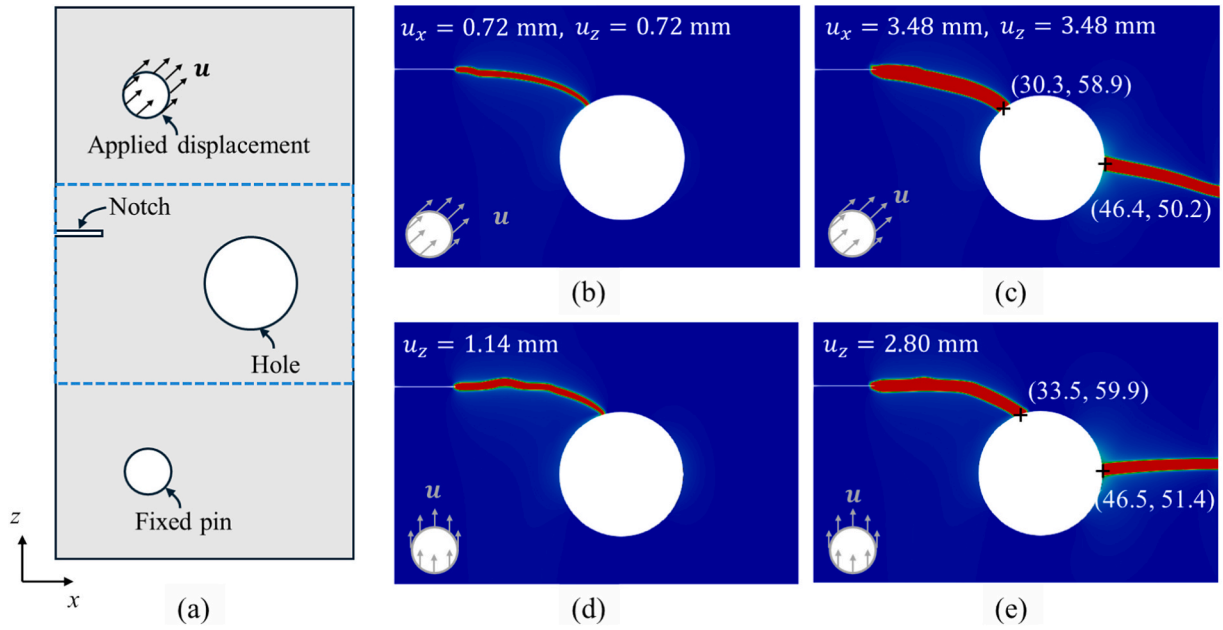


Fig. 17. (a) Schematic illustration of the notched plate with a central hole (Case II), showing a different loading direction compared to Fig. 15(a). (b–c) Simulated crack patterns for Case II under varying displacement levels. (d–e) Simulated crack patterns for Case I under varying displacement levels.

direction are 193 mm and 200 mm, respectively, yielding an error of only 3.5 %. Similarly, the crack lengths in the z-direction are 38.5 mm and 39.9 mm, respectively, also corresponding to an error of 3.5 %. These discrepancies are likely attributable to differences in mesh types. Previous study employed an adaptive refinement approach with a mesh size of 0.3125 mm in the crack propagation region, whereas this study used a uniform mesh with a minimum size of 0.25 mm.

3.3. Notched plate with a hole

The plate shown in Fig. 15(a) contains an initial crack and three holes. The hole at the bottom left is fixed (with displacement constrained to zero), a displacement load is applied to the hole at the top left, and the hole on the right serves as a free boundary. This experiment test was presented by Ambati et al. (2015). Two different displacement loading conditions were investigated. In Case I, a vertical displacement increment of $\Delta u = 0.001$ mm was applied until complete failure, as illustrated in Fig. 15(a). Case II builds upon Case I by additionally introducing a horizontal displacement increment, resulting in a total displacement vector oriented at a 45° angle with respect to the x-axis, as shown in Fig. 17(a). The material parameters used in both cases are identical, characterized by a Young's modulus of $E = 4.9$GPa, a Poisson's ratio of $\nu = 0.22$, a critical energy release rate of $G_c = 2.28$N/mm, and a characteristic length of $l_0 = 0.1$mm. Fixed non-uniform mesh is adopted in this case to save the computational cost.

The entire computational domain is divided into three sections based on the z-coordinate: 0−0.04 m, 0.04−0.07 m, and 0.07−0.12 m. The region 0.04−0.07 m, where cracks may develop, is a refined mesh region with a uniform mesh size of 0.037 mm. In the other two regions, the mesh size is 0.037 mm in the x-direction and 0.074 mm in the z-direction. Additionally, three layers of boundary layer mesh are applied around the circular domain to ensure smoothness and continuity at the boundaries. The boundary layer mesh thickness is 0.02 mm for the circular domain in the second region, and 0.033 mm for the circular domains in the other two regions. The mesh distribution is shown in Fig. 15 (b).

Fig. 16 presents the crack propagation process from the simulation alongside the experimental results. The outcomes demonstrate that GPU-PFM accurately replicates the crack growth trajectory, confirming the robustness and reliability of the GPU-PFM model. It is worth noting that, unlike discrete methods, PFM, as a continuum-based approach, may predict a gradually expanding crack region over time, as exemplified by the left-side crack in Fig. 16. This behavior aligns with findings reported in previously published studies (Ambati et al., 2015, Yang et al., 2024b) on PFM methodologies. Specifically, the observed crack initiation points on the center hole in the simulation are located at (33.5 mm, 59.9 mm) and (46.5 mm, 51.5 mm), while the experimentally measured crack initiation points are (32.7 mm, 59.9 mm) and (46.2 mm, 51.6 mm). The corresponding errors are 2.4 %, 0 %, 0.6 %, and 0.2 %, respectively, all within an acceptable range. Upon completion of the through-crack propagation, a notable difference in the displacement field between the upper and lower regions becomes evident (Fig. 16f). The lower region shows minimal displacement due to constraints, while the upper region, being unconstrained, experiences significantly larger displacements.

The accuracy of the adopted numerical algorithm is validated by comparing the simulation results of Case I with the corresponding experimental observations. Additionally, a comparative analysis between Case I and Case II, as illustrated in Fig. 17, allows for a deeper investigation into the fracture patterns of the specimen under different loading directions. In Case II, the introduction of a horizontal displacement component means that the resulting cracks are no longer solely attributed to tensile failure. The shear effect induced by the horizontal displacement causes a pronounced inclination in the fracture paths, particularly on the right side of the specimen. The displacement fields and vectors shown in Fig. 18 and Fig. 19 further clarify the mechanisms by which different loading directions result in distinct fracture patterns.

Even before crack formation, the displacement distributions in Case I and Case II show significant differences (Fig. 18). Particular attention is given to Region I and Region II, where cracks are subsequently observed. In Case I, which is subjected solely to vertical tensile loading, the displacement field is relatively uniform, with displacements predominantly oriented in the vertical direction throughout the specimen, including both Region I and Region II. In contrast, Case II introduces a horizontal displacement component, which notably alters the displacement directions. In Region I, the displacement direction shifts from the upper-left in Case I to the upper-right in Case II. The difference is even

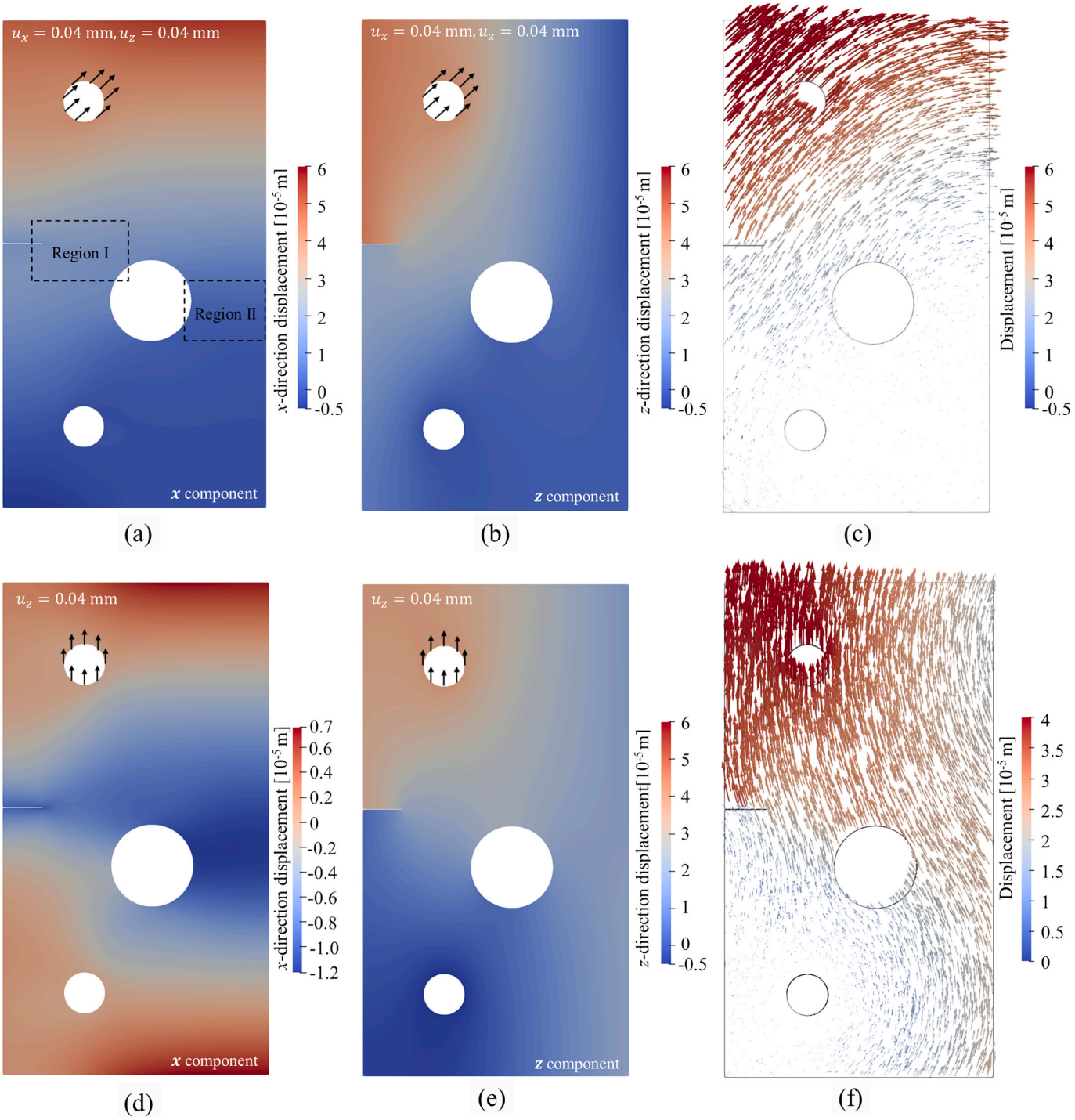


Fig. 18. Simulated results of the x- and z-components of the displacement field, as well as the displacement vectors, for Case II at applied displacements of u_x = 0.04 mm and u_z = 0.04 mm (a–c), and for Case I at u_z = 0.04 mm (d–f).

more pronounced in Region II, where the upward displacement component observed in Case I nearly vanishes in Case II. This change is attributed to the horizontal displacement introducing a compressive effect in Region II, counteracting the vertical tensile effect and thereby suppressing the upward displacement in that area. This compensating effect persists even after cracks have formed in Region I (Fig. 19c), resulting in a delayed onset of cracking in Region II for Case II compared to Case I, as shown in Fig. 17(c and e).

3.4. KGD hydraulic fracturing test

The KGD model, proposed by Geertsma and De Klerk (1969), is a classical analytical solution for hydraulic fracture propagation in an impermeable, linear elastic medium under plane strain conditions. It combines linear elasticity, fracture mechanics, and lubrication theory to describe fluid-driven crack growth. Fig. 20 shows the schematic illustration of the KGD hydraulic fracturing test adopted in this study. Fracture propagation can be classified as toughness-dominated or viscosity-dominated, depending on whether energy is primarily consumed by creating new fracture surfaces or by fluid viscous

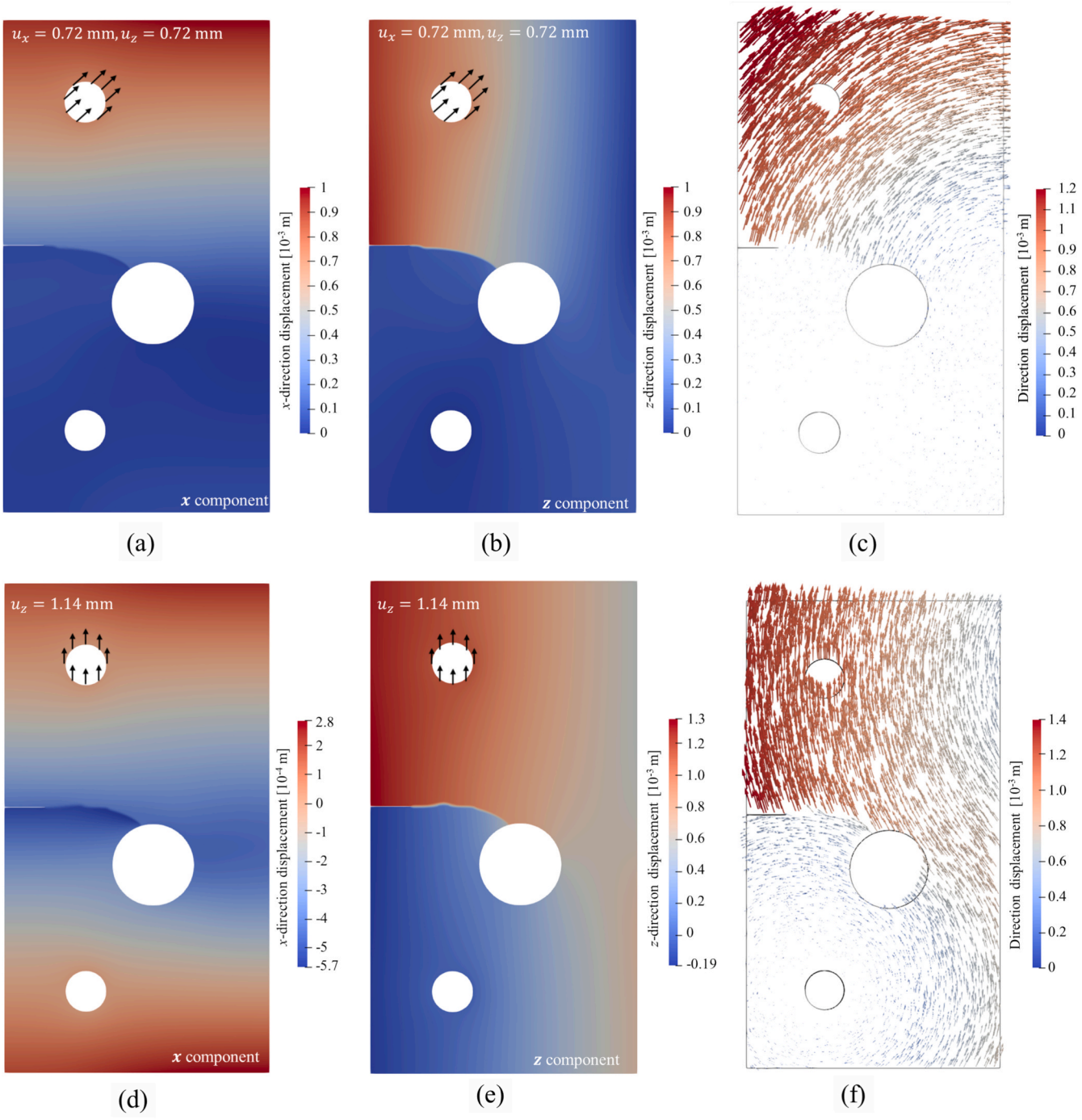


Fig. 19. Simulated results of the x- and z-components of the displacement field, as well as the displacement vectors, for Case II at applied displacements of $u_x = 0.72$ mm and $u_z = 0.72$mm (a–c), and for Case I at $u_z = 0.72$ mm (d–f).

dissipation. When injecting low-viscosity Newtonian fluids such as water, the fracture typically remains in the toughness-dominated regime without transitioning from viscosity-dominated behavior (Bunger et al., 2005). This study focuses on such toughness-dominated fractures, using the analytical expressions for fracture length and injection pressure provided in Bunger et al. (2005).

$$L = \frac{2}{\pi^{\frac{2}{3}}}\left(\frac{E'qt}{K'}\right)^{\frac{2}{3}} \tag{45}$$

$$P = \frac{\pi^{\frac{1}{3}}}{8}\left(\frac{k'^4}{K'qt}\right)^{\frac{1}{3}} \tag{46}$$

where q represents the injection flow rate, t is the time elapsed since injection began, and K' denotes the plane strain modulus. The modulus K' is given by:

$$K' = 4\sqrt{\frac{2}{\pi}}K_{IC} \tag{47}$$

where K_{IC} is the mode-I fracture toughness and $K_{IC} = 1.88\text{MPa} \bullet \text{m}^{0.5}$ in

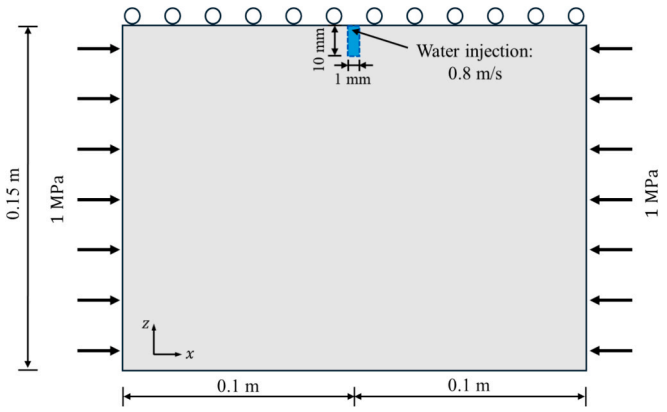


Fig. 20. Schematic illustration of the KGD hydraulic fracturing test.

this case.

The simulation setup is illustrated in Fig. 20. The model consists of a 200 mm × 150 mm solid domain representing a typical brittle material (Aliha et al., 2016). The material properties are characterized by a Young's modulus of $E = 2.8$GPa, a density of 1180 kg/m^3, a Poisson's ratio of $\nu = 0.37$, a critical energy release rate of $G_c = 1079$N/m, and a characteristic length of $l_0 = 1$mm. The porosity and absolute permeability of the solid are 0.01 and 3×10^{-16} m^2, respectively. The density and viscosity of water are 1000 kg/m^3 and 10^{-6} m^2/s, respectively, while those of air are 1 kg/m^3 and 1.48×10^{-5} m^2/s, respectively. The Biot's coefficient is 1. A compressive stress of 1 MPa is applied along the x-direction, while the upper boundary is constrained in the z-direction to suppress rigid body motion caused by fluid injection. Water is introduced through a tube positioned at the top surface, and an initial crack, 10 mm in length and 1 mm in width, is pre-inserted at the center of the upper boundary. The mesh size is uniformly set to 0.125 mm across all materials. The injection velocity is specified as 0.8 m/s, corresponding to a volumetric flow rate of 0.0008 m^3/s (Yang et al., 2024a). The simulation is conducted over a total duration of 0.1 s with a time step of 1×10^{-6} s.

Fig. 21 depicts a comparison of injection pressure at the injection point and fracture length across three approaches: the analytical solution (Bunger et al., 2005), prior PD simulations (Yang et al., 2024a), and the present model. At injection onset, fluid pressure rapidly builds up, reaching the fracture initiation threshold (8.6 MPa) at 4 ms. This peak marks fracture nucleation, triggering a sharp pressure drop due to sudden fracture opening. Subsequently, as injection continues, pressure briefly rises when fluid fills the new void space, then declines gradually alongside stable fracture extension. Critically, the fracture propagation rate decelerates over time, evidenced by the flattening slope in Fig. 21 (b), while the pressure decay rate moderates concomitantly (Fig. 21a).

Overall, fracture length and pressure evolution align well with both the analytical solution and prior PD results. In this study, the RMSE between the simulated and analytical injection pressure is 0.43 MPa, while that for the fracture length is 0.13 mm. Fig. 22 further illustrates this agreement by comparing fracture contours, fluid phase distribution, and pressure fields at four representative time instants between the present model and PD results, also demonstrating high consistency. Notably, the present model generates a smoother pressure profile than the PD approach, exhibiting reduced oscillations and superior numerical stability. The analytical solution, assuming a fully fluid-filled fracture, yields a monotonic pressure decrease, overlooking transient effects captured in our simulation. This contrast underscores our model's capability to resolve dynamic fluid–structure interactions essential to hydraulic fracturing.

3.5. Three-dimensional two-phase hydraulic fracturing example

Fig. 23 further illustrates a three-dimensional two-phase hydraulic fracturing example, demonstrating the potential of the proposed algorithm in coupling with the two-phase CFD. Note that this coupled algorithm is implemented within the GPU-parallelized CFD-DEM software TFluid (https://www.t-fluid.com). It leverages the GPU-accelerated FVM framework in TFluid for solving two-phase flow in porous media, thereby achieving a unified GPU-accelerated FVM framework for the coupled two-phase CFD and PFM.

In this case, roller boundary conditions are applied in all three directions, restricting displacement in the normal direction while allowing free movement in the tangential directions. The computational domain measures 0.8 m × 0.8 m × 0.2 m, with a through-hole along the z-axis located at the center of the domain, as shown in Fig. 23(b). The rest of the domain is solid. Initially, the hole is filled with air, and water is injected from the top of the hole at a velocity of 0.5 m/s.

The material properties are characterized by a Young's modulus of

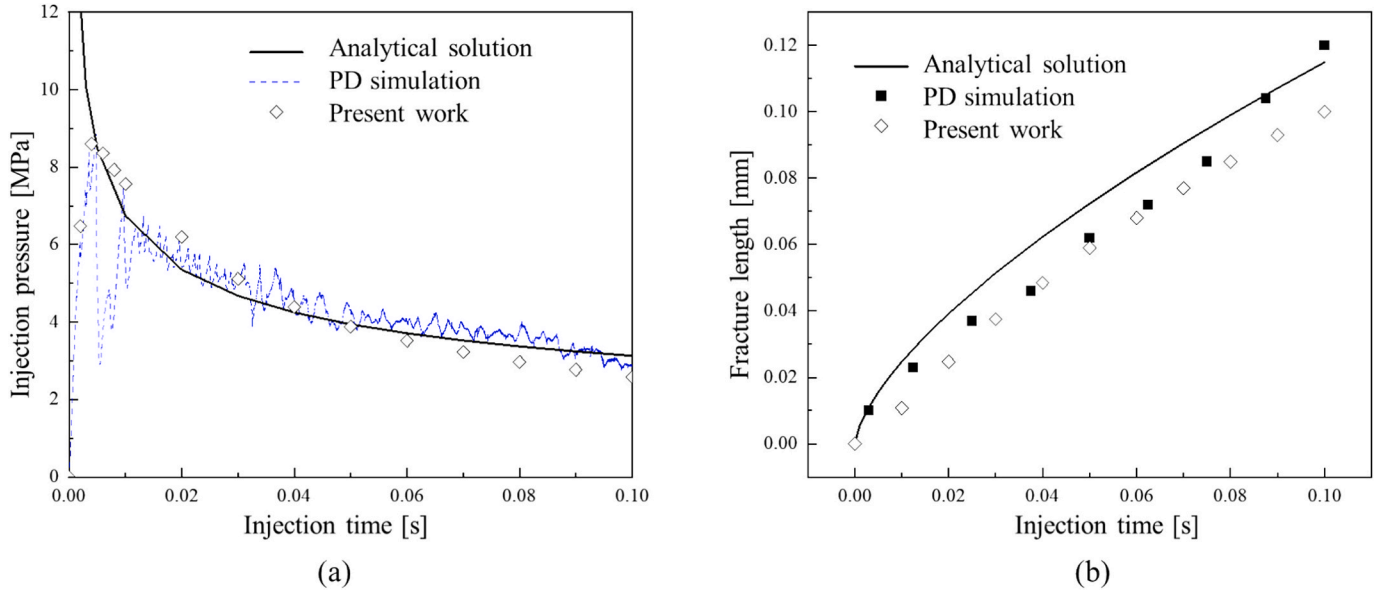


Fig. 21. Comparison of (a) injection pressure at the injection point and (b) fracture length among the analytical solution (Bunger et al., 2005), existing PD simulation results (Yang et al., 2024a), and the present study.

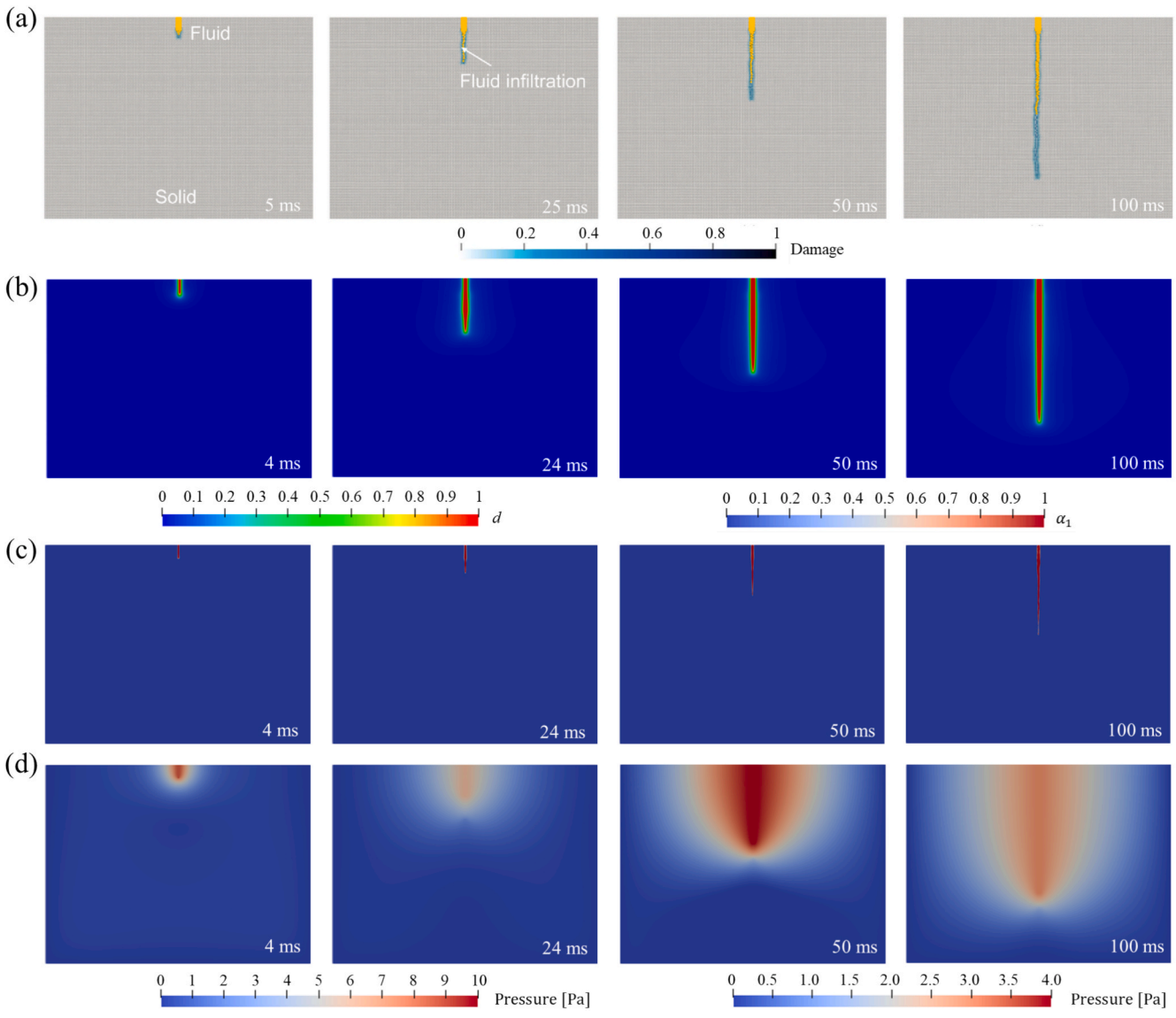


Fig. 22. Fracture development and fluid infiltration at various time steps. (a) Fracture and fluid phase contours from PD simulations (Yang et al., 2024a); (b–d) Results from the present simulation showing fracture contours (b), fluid phase distribution (c), and pressure field (d), respectively.

$E = 25$GPa, a density of 2000 kg/m^3, a Poisson's ratio of $\nu = 0.3$, a critical energy release rate of $G_c = 50$N/m, and a characteristic length of $l_0 = 2$mm. The porosity of the solid is 0.5. The Biot's coefficient is 1. The fluid properties, including the density and viscosity of both water and air, are consistent with those employed in the preceding KGD test. The van Genuchten (Van Genuchten, 1980) model shown in Eqs. (22)–(24) is chosen as the permeability model, with a permeability coefficient k_0 of 1×10^{-14} m^2 and a non-dimensional coefficient m of 0.99.

As fractures are expected to form in the central region along the x-direction, the mesh in this area is refined in a stepwise manner, with the finest grid size reaching 2.5 mm. In the y and z directions, a uniform mesh is employed, also with a grid size of 2.5 mm. The model consists of 1.55 million mesh elements, and the time step is set to 0.005 s. Given the transient nature of the simulation, the maximum number of iterations for the PFM is limited to 1. The average computational time per time step is 1.95 s using the RTX 3060Ti, with approximately 0.61 s allocated to the two-phase CFD component, which includes the porous media module, and 1.34 s dedicated to the PFM component. The PFM computation time is 2.2 times that of the CFD component, making it the primary performance bottleneck of the coupled two-phase CFD-PFM algorithm.

Performance analysis indicates that the high computational cost of PFM mainly arises from solving four equations: three displacement equations and the phase-field equation. Additionally, the strain decomposition process further contributes to the overall computational burden. Notably, increasing the number of maximum PFM iterations would lead to a significant rise in PFM computation time, making it the dominant factor in the overall performance of the coupled two-phase CFD-PFM algorithm. Consequently, GPU acceleration for PFM is crucial for efficiently solving two-phase hydraulic fracturing problems, substantially improving the computational efficiency of the coupled two-phase CFD-PFM framework.

Hydraulic fracturing involves complex physical phenomena, including two-phase flow, porous flow, solid mechanics, and fracture propagation. However, in existing simulations, two-phase flow and porous flow are often approximated as an equivalent pressure to simplify computations and avoid solving the Navier-Stokes (N-S) equations (Zhou et al., 2018, Lu et al., 2025). While this simplification reduces computational costs, it fails to accurately capture the distribution of fluid phase fractions and flow velocities. In contrast, the application of

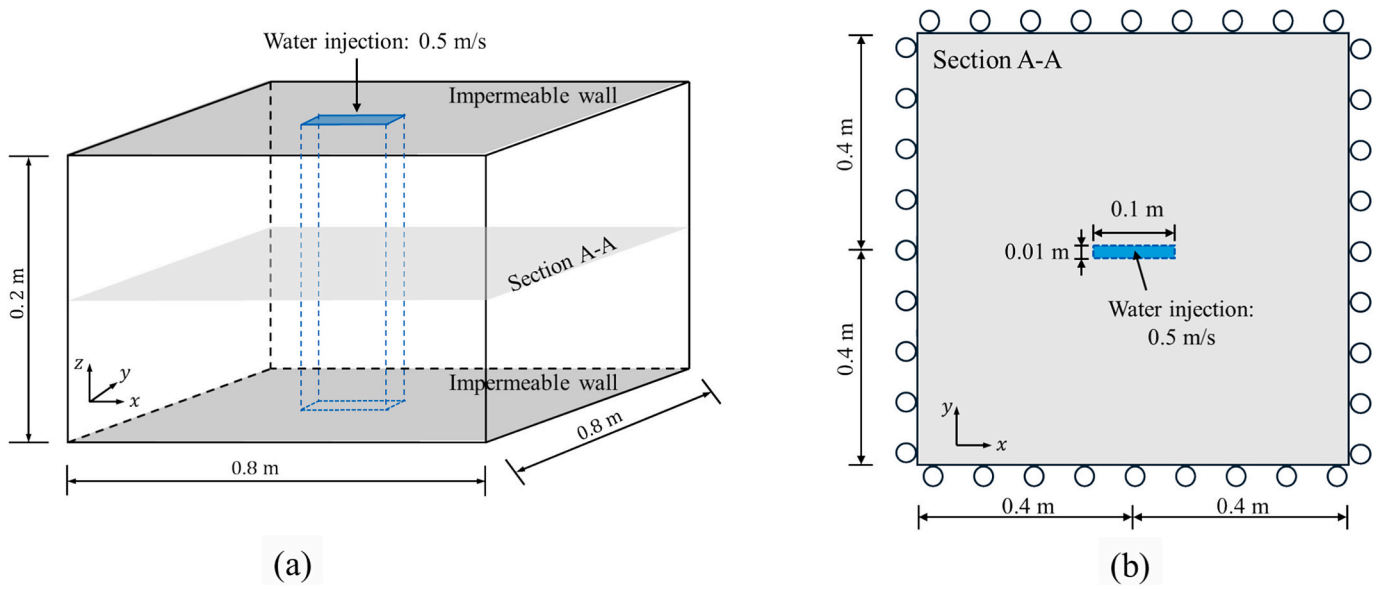


Fig. 23. Schematic illustration of the two-phase hydraulic fracturing example: (a) three-dimensional model; (B) cross section A-A in (a).

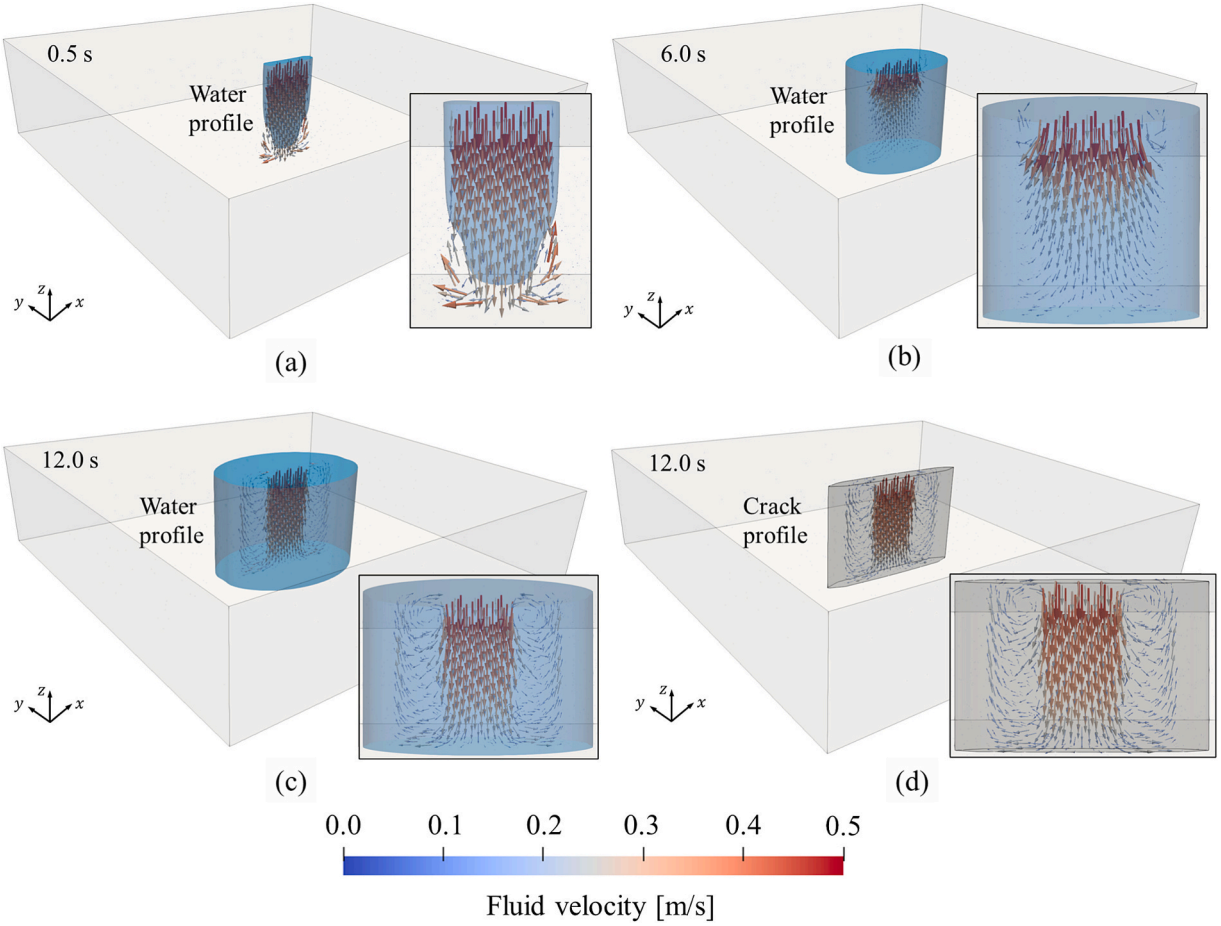


Fig. 24. 3D evolution of the fluid profile (a-c) and crack profile (d), with arrows indicating the fluid velocity vectors. The water profile and crack profile are defined by the contours at a water fraction of 0.5 and a phase field value of 0.9, respectively.

the coupled two-phase CFD-PFM algorithm can overcome this limitation while significantly enhancing computational efficiency.

Fig. 24 illustrates the variations in fluid phase distribution and velocity during the 3D hydraulic fracturing process. As the fracture propagates, significant changes in the fluid velocity field are observed. At 0.5 s, water has not yet fully filled the initial fracture, resulting in the formation of noticeable vortices within the trapped air. By 6 s, the fracture is filled with water, which begins to permeate into the

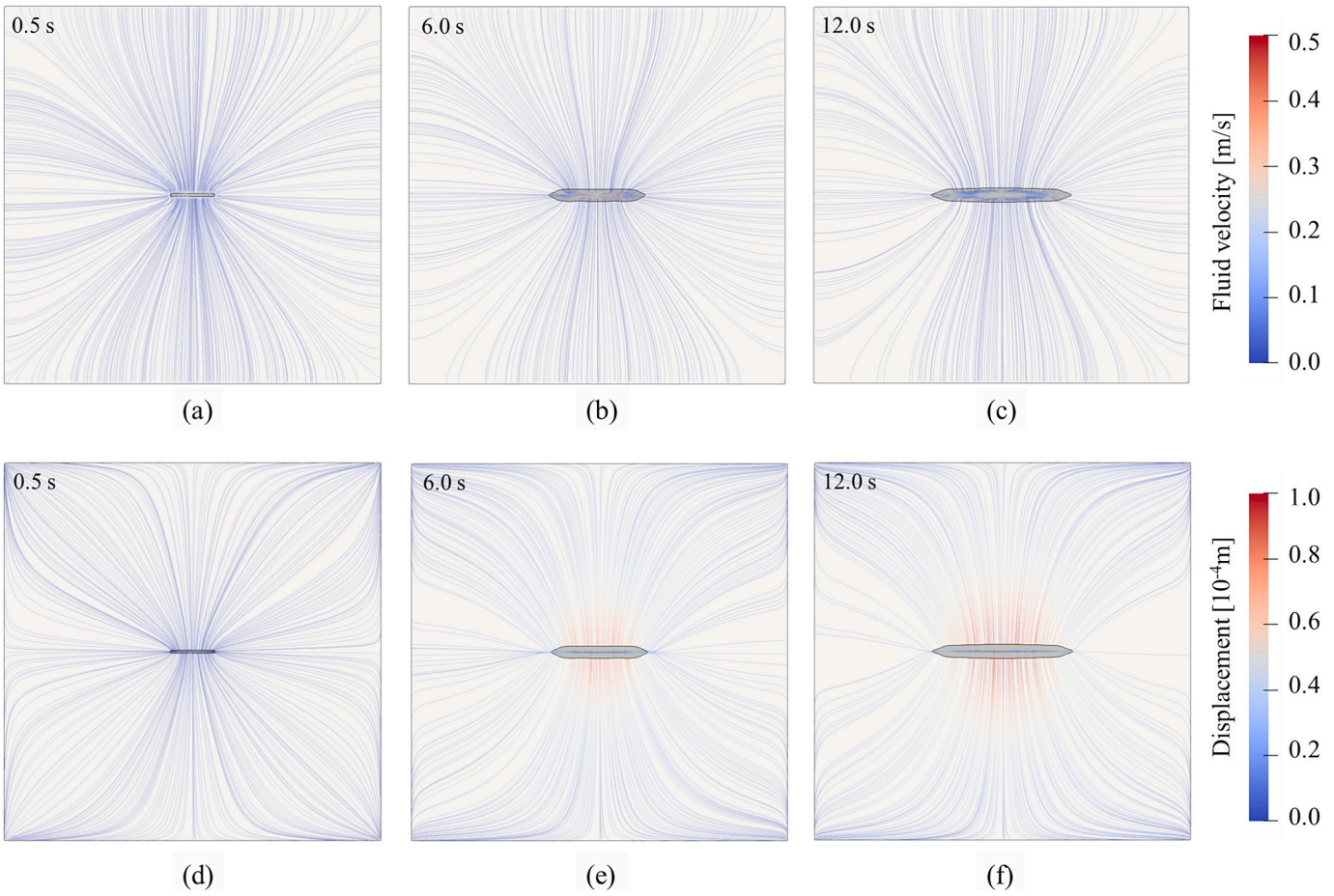


Fig. 25. Streamlines of fluid velocity (a-c) and solid displacement (d-f) on top surface.

surrounding solid. As water infiltration progresses, the fracture gradually expands. At 12 s, the fracture has visibly widened, and new vortices emerge within it.

Fig. 25 further presents the streamlines of fluid velocity and solid displacement. It is evident that in the region surrounding the fracture, the displacement streamlines closely align with the fluid velocity streamlines. However, near the boundaries, the direction of the displacement streamlines is more significantly influenced by the constraints.

Combining Fig. 24 and Fig. 25, this evolving fluid velocity field provides a critical foundation for further exploration of the complex mechanisms underlying hydraulic fracturing.

Fig. 26 and Fig. 27 illustrate the phase field, water volume fraction field, and pressure field at 0.5 s, 6 s, and 12 s on the top surface and on the half section, respectively. As time progresses, the fluid initially infiltrates the fracture until it is filled and subsequently penetrates into the solid. The resulting permeability-induced drag force causes the solid to displace, and once a critical state is reached, the fracture begins to propagate. Due to space constraints, details of the coupled CFD-PFM algorithm and more verification examples will be presented in future work. Here, this example serves solely to demonstrate the potential of the proposed GPU-PFM algorithm in three-dimensional models and multiphysics simulations.

4. Conclusions

This work presents an original full-process GPU-parallel FVM framework for solving phase field methods (PFM) to simulate cracks. First, the theory of phase field methods and the FVM-based phase field solution framework are introduced in detail. Based on this framework, we implemented GPU parallelization for mesh post-processing, discretization and assembly of the governing equations, linear system solvers, and iterative algorithms, and provided a detailed description of the algorithm's implementation process. The full GPU parallelization was achieved for the entire process, excluding mesh reading and result exporting, to fully utilize the computational power of the GPU, avoiding frequent data exchange between the CPU and GPU during the computation. Three classical test cases were used to verify the accuracy of the GPU-parallel FVM framework in solving phase field methods, including the single-edge notched tension test, the L-shaped panel test, and the notched plate with a hole. Two two-phase hydraulic fracturing examples are further conducted to showcase its potential for three-dimensional models and coupling with two-phase CFD. In particular, the performance of GPU and CPU in the single-edge notched tension test was compared in detail, and the acceleration ratio of the GPU was tested for cases with different numbers of mesh. The results show that GPU parallelization is more suitable for large-scale cases with over 2 million grids. For cases with fewer than 2 million grids, the acceleration ratio

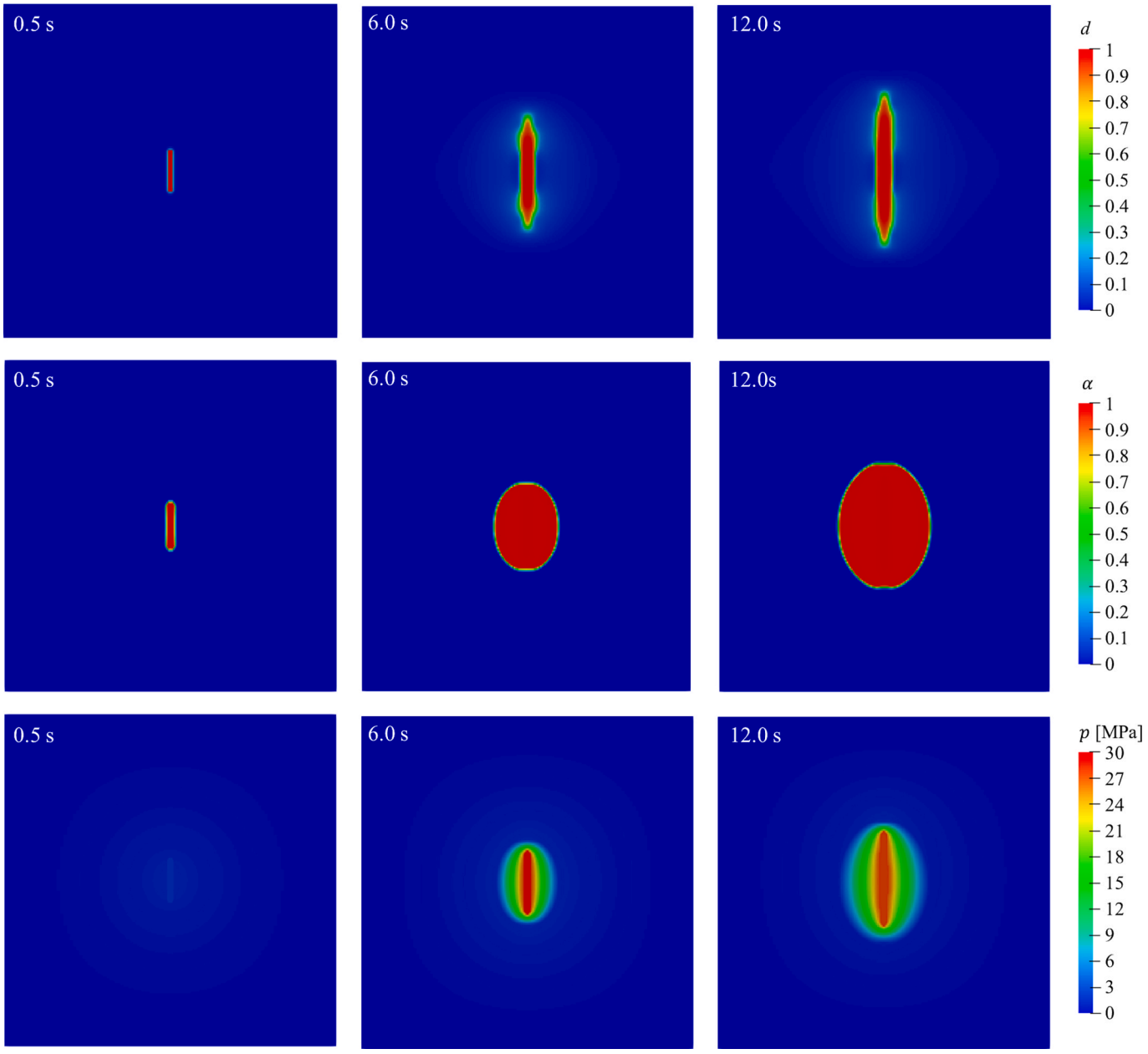


Fig. 26. Phase field, volume fraction field of water, and pressure field at 0.5 s, 6 s, and 12 s on the top surface.

decreases with the number of grids, and especially in cases with only hundreds of thousands of grids, the GPU's computational power cannot be fully utilized. When the grid number exceeds 2 million, the tested RTX 3060Ti GPU is equivalent to 15 AMD® Ryzen® CPU R7-5800X cores, or 120 CPU cores in total. Based on this equivalence, the hardware cost of GPU parallelization is only 6 % of that for CPU parallelization, given the same computational power.

Therefore, utilizing full-process GPU parallelization can significantly reduce the computational cost of simulations and expand the application scope of the algorithm, such as enabling larger-scale simulations, more complex physical models, and even coupling various algorithms (e.g., computational fluid dynamics and phase field methods) within a unified GPU-parallel framework for more accurate and realistic multiphase hydraulic fracturing simulations.

The high efficiency and scalability of the full-process GPU-parallelized FVM framework provide strong support for its application in industrial scenarios, such as hydraulic fracturing and CO_2 fracturing in the energy sector, while also paving a feasible path for future real-time simulations. Moreover, since artificial intelligence (AI) relies on GPU computation, the proposed framework can leverage the same hardware to rapidly generate large volumes of training data for AI models. This not only reduces the cost of data collection from experiments and engineering applications but also minimizes the expenses associated with acquiring CPU-based supercomputing resources.

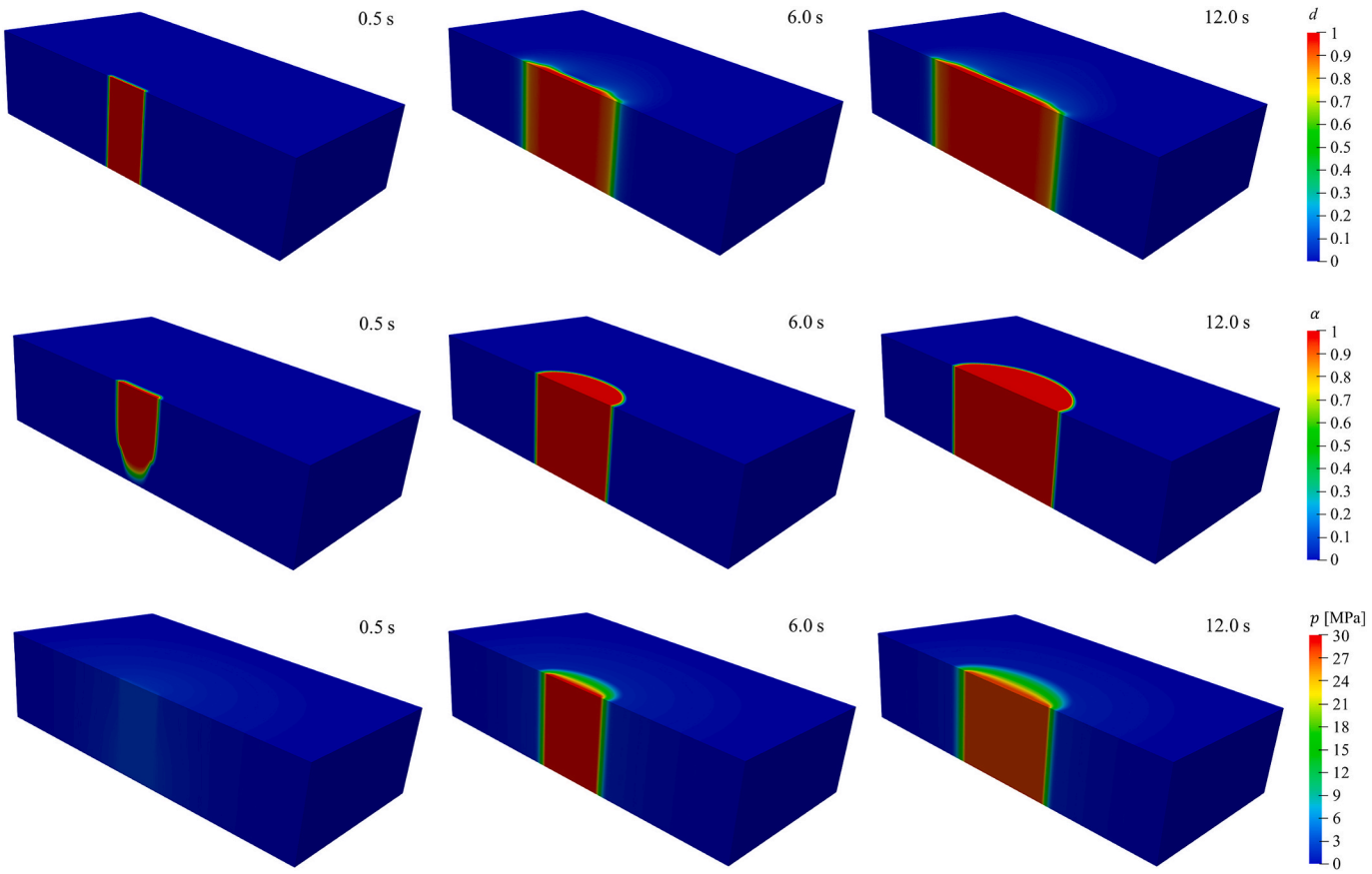


Fig. 27. Phase field, volume fraction field of water, and pressure field at 0.5 s, 6 s, and 12 s on the half section.

However, this method also faces the common limitation of other GPU-parallel methods, namely, memory constraints. Currently, a GPU-PFM simulation with 1 million grids requires 1.2 GB of GPU memory, so even with the RTX 4090 GPU, only up to 20 million grids can be computed. Future work will focus on developing multi-GPU parallel algorithms to overcome this limitation. Further extensions of this method, such as coupling with CFD methods, will be examined and discussed in greater detail in future studies.

CRediT authorship contribution statement

Tao Yu: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology. **Yuntian Zhao:** Writing – original draft, Methodology. **Jidong Zhao:** Writing – review & editing, Supervision, Methodology, Funding acquisition.

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.

Acknowledgments

The authors acknowledge the support provided by the Research Grants Council of Hong Kong (through TRS #T22-607/24N, CRF #C7082-22G, GRF #16203123, #16212724) and the National Natural Science Foundation of China (#52439001).

Data availability

Data will be made available on request.

References

Afzal, A., Saleel, C.A., Prashantha, K., Bhattacharyya, S., Sadhikh, M., 2021. Parallel finite volume method-based fluid flow computations using OpenMP and CUDA applying different schemes. J. Therm. Anal. Calorim. 145, 1891–1909.
Aissa, M., Verstraete, T., Vuik, C., 2017. Toward a GPU-aware comparison of explicit and implicit CFD simulations on structured meshes. Comput. Math. Appl. 74, 201–217.
Aliha, M., Bahmani, A., Akhondi, S., 2016. Mixed mode fracture toughness testing of PMMA with different three-point bend type specimens. Euro. J. Mech.-A/Solids 58, 148–162.
Ambati, M., Gerasimov, T., De Lorenzis, L., 2015. A review on phase-field models of brittle fracture and a new fast hybrid formulation. Comput. Mech. 55, 383–405.
Amor, H., Marigo, J.-J., Maurini, C., 2009. Regularized formulation of the variational brittle fracture with unilateral contact: numerical experiments. J. Mech. Phys. Solids 57, 1209–1229.
Barenblatt, G.I., 1962. The mathematical theory of equilibrium cracks in brittle fracture. Adv. Appl. Mech. 7, 55–129.
Belytschko, T., Lin, J.I., 1987. A three-dimensional impact-penetration algorithm with erosion. Int. J. Impact Eng 5, 111–127.
Bordas, S., Rabczuk, T., Zi, G., 2008. Three-dimensional crack initiation, propagation, branching and junction in non-linear materials by an extended meshfree method without asymptotic enrichment. Eng. Fract. Mech. 75, 943–960.
Bourdin, B., Francfort, G.A., Marigo, J.-J., 2008. The variational approach to fracture. J. Elast. 91, 5–148.
Bunger, A.P., Detournay, E., Garagash, D.I., 2005. Toughness-dominated hydraulic fracture with leak-off. Int. J. Fract. 134, 175–190.
Cardiff, P., Demirdžić, I., 2021. Thirty years of the finite volume method for solid mechanics. Arch. Comput. Meth. Eng. 28, 3721–3780.
Cardiff, P., Tuković, Ž., Jaeger, P.D., Clancy, M., Ivanković, A., 2017. A Lagrangian cell-centred finite volume method for metal forming simulation. Int. J. Numer. Meth. Eng. 109, 1777–1803.
Carrillo, F.J., Bourg, I.C., 2021. Modeling multiphase flow within and around deformable porous materials: a Darcy-Brinkman-Biot approach. Water Resour. Res. 57, e2020WR028734.
Carrillo, F.J., Bourg, I.C., Soulaine, C., 2020. Multiphase flow modeling in multiscale porous media: an open-source micro-continuum approach. J. Comput. Phys.: X 8, 100073.
Dugdale, D.S., 1960. Yielding of steel sheets containing slits. J. Mech. Phys. Solids 8, 100–104.

Dutto, L.C., Lepage, C.Y., Habashi, W.G., 2000. Effect of the storage format of sparse linear systems on parallel CFD computations. Comput. Methods Appl. Mech. Eng. 188, 441–453.
Economides, M.J., Nolte, K.G., 1989. Reservoir Stimulation. Prentice Hall Englewood Cliffs, NJ.
Fan, Z., Jidong, Z., Ballarini, R., Shitao, P., Suwen, C., 2022. Peridynamic modeling of stochastic fractures in bolted glass plates. Mech. Res. Commun. 122, 103890.
Fei, F., Choo, J., 2020. A phase-field method for modeling cracks with frictional contact. Int. J. Numer. Meth. Eng. 121, 740–762.
Francfort, G.A., Marigo, J.-J., 1998. Revisiting brittle fracture as an energy minimization problem. J. Mech. Phys. Solids 46, 1319–1342.
Gao, M., Wang, X., Wu, K., Pradhana, A., Sifakis, E., Yuksel, C., Jiang, C., 2018. GPU optimization of material point methods. ACM Trans. Graph. (TOG) 37, 1–12.
Geertsma, J., De Klerk, F., 1969. A rapid method of predicting width and extent of hydraulically induced fractures. J. Petrol. Tech. 21, 1571–1581.
Griffith VI, A.A., 1921. The phenomena of rupture and flow in solids. Philos. Trans. R. Soc. Lond. Ser. A, Contain. Pap. Math. Phys. Charact. 221, 163–198.
Irwin, G.R., 1957. Analysis of stresses and strains near the end of a crack traversing a plate.
Issa, R.I., 1986. Solution of the implicitly discretised fluid flow equations by operator-splitting. J. Comput. Phys. 62, 40–65.
Jasak, H., Weller, H., 2000. Application of the finite volume method and unstructured meshes to linear elasticity. Int. J. Numer. Meth. Eng. 48, 267–287.
Jespersen, D.C., 2010. Acceleration of a CFD code with a GPU. Sci. Program. 18, 193–201.
Kang, Y., Zheng, Y., Li, S., Zhang, J., Tang, J., Yang, C., Luo, Y., 2024. GPU-accelerated approach for 2D fracture analysis of structures combining finite particle method and cohesive zone model. Eng. Fract. Mech. 306, 110198.
Kiran, R., Nguyen-Thanh, N., Yu, H., Zhou, K., 2023. Adaptive isogeometric analysis–based phase-field modeling of interfacial fracture in piezoelectric composites. Eng. Fract. Mech. 288, 109181.
Kuo, F., Chiang, C., Lo, M., Wu, J., 2020. Development of a parallel explicit finite-volume Euler equation solver using the immersed boundary method with hybrid MPI-CUDA paradigm. J. Mech. 36, 87–102.
Lei, J., Li, D.-L., Zhou, Y.-L., Liu, W., 2019. Optimization and acceleration of flow simulations for CFD on CPU/GPU architecture. J. Braz. Soc. Mech. Sci. Eng. 41, 290.
Li, W., Nguyen-Thanh, N., Du, H., Zhou, K., 2023. Adaptive phase-field modeling of dynamic brittle fracture in composite materials. Compos. Struct. 306, 116589.
Liu, G.-Y., Xu, W.-J., Govender, N., Wilke, D.N., 2021. Simulation of rock fracture process based on GPU-accelerated discrete element method. Powder Technol. 377, 640–656.
Lu, Z., Zhu, F., Higo, Y., Zhao, J., 2025. Coupled semi-Lagrangian and poroelastic peridynamics for modeling hydraulic fracturing in porous media. Comput. Meth. Appl. Mech. Eng. 437, 117794.
Michalakes, J., Vachharajani, M., 2008. GPU acceleration of numerical weather prediction. In: IEEE International Symposium on Parallel and Distributed Processing. IEEE, pp. 1–7.
Miehe, C., Hofacker, M., Welschinger, F., 2010. A phase field model for rate-independent crack propagation: Robust algorithmic implementation based on operator splits. Comput. Meth. Appl. Mech. Eng. 199, 2765–2778.
Moës, N., Dolbow, J., Belytschko, T., 1999. A finite element method for crack growth without remeshing. Int. J. Numer. Meth. Eng. 46, 131–150.
Mualem, Y., 1976. A new model for predicting the hydraulic conductivity of unsaturated porous media. Water Resour. Res. 12, 513–522.
Nguyen-Thanh, N., Li, W., Huang, J., Zhou, K., 2020. Adaptive higher-order phase-field modeling of anisotropic brittle fracture in 3D polycrystalline materials. Comput. Meth. Appl. Mech. Eng. 372, 113434.
Peng, C., Wang, S., Wu, W., Yu, H.-S., Wang, C., Chen, J.-Y., 2019. LOQUAT: an open-source GPU-accelerated SPH solver for geotechnical modeling. Acta Geotech. 14, 1269–1287.
Rashid, Y.R., 1968. Ultimate strength analysis of prestressed concrete pressure vessels. Nucl. Eng. Des. 7, 334–344.
Rhie, C.M., Chow, W.-L., 1983. Numerical study of the turbulent flow past an airfoil with trailing edge separation. AIAA J. 21, 1525–1532.
Roenby, J., Larsen, B.E., Bredmose, H., Jasak, H., 2017. A new volume-of-fluid method in OpenFOAM. In: Marine VI: Proceedings of the VI International Conference on Computational Methods in Marine Engineering. CIMNE, pp. 266–277.
Rusche, H., 2002. Computational Fluid Dynamics of Dispersed Two-Phase Flow at High Phase Fractions. Ph. D. thesis, University of London.
Seleš, K., Lesičar, T., Tonković, Z., Sorić, J., 2019. A residual control staggered solution scheme for the phase-field modeling of brittle fracture. Eng. Fract. Mech. 205, 370–386.
Shin, H., Thamburaja, P., Srinivasa, A., Reddy, J., 2023. Modeling impact fracture in a quasi-brittle solids using a 3D nonlocal graph-based finite element analysis: theory, finite element simulations, and experimental verification. J. Mech. Phys. Solids 170, 105097.
Shu, S., Zhang, J., Yang, N., 2020. GPU-accelerated transient lattice Boltzmann simulation of bubble column reactors. Chem. Eng. Sci. 214, 115436.
Silling, S.A., 2000. Reformulation of elasticity theory for discontinuities and long-range forces. J. Mech. Phys. Solids 48, 175–209.
Stevens, S.H., Kuuskraa, V.A., Spector, D., Riemer, P., 1999. CO_2 sequestration in deep coal seams: pilot results and worldwide potential. In: Greenhouse Gas Control Technologies. Elsevier Science, Oxford, pp. 175–180.
Sukumar, N., Dolbow, J., Moës, N., 2015. Extended finite element method in computational fracture mechanics: a retrospective examination. Int. J. Fract. 196, 189–206.
Tanné, E., Li, T., Bourdin, B., Marigo, J.-J., Maurini, C., 2018. Crack nucleation in variational phase-field models of brittle fracture. J. Mech. Phys. Solids 110, 80–99.
Tian, F., Tang, X., Xu, T., Li, L., 2020. An adaptive edge-based smoothed finite element method (ES-FEM) for phase-field modeling of fractures at large deformations. Comput. Methods Appl. Mech. Eng. 372, 113376.
Tubbs, K.R., Tsai, F.T.C., 2011. GPU accelerated lattice Boltzmann model for shallow water flow and mass transport. Int. J. Numer. Meth. Eng. 86, 316–334.
Van Genuchten, M.T., 1980. A closed-form equation for predicting the hydraulic conductivity of unsaturated soils. Soil Sci. Soc. Am. J. 44, 892–898.
Wang, D., Dong, Y., Sun, D., Yu, B., 2020a. A three-dimensional numerical study of hydraulic fracturing with degradable diverting materials via CZM-based FEM. Eng. Fract. Mech. 237, 107251.
Wang, D., Wang, B., Yuan, W., Liu, L., 2023. Investigation of rainfall intensity on the slope failure process using GPU-accelerated coupled MPM. Comput. Geotech. 163, 105718.
Wang, X., Qiu, Y., Slattery, S.R., Fang, Y., Li, M., Zhu, S.-C., Zhu, Y., Tang, M., Manocha, D., Jiang, C., 2020b. A massively parallel and scalable multi-GPU material point method. ACM Trans. Graph. (TOG) 39, 30: 1–30: 15.
Wells, G.N., Sluys, L., 2001. A new method for modelling cohesive cracks using finite elements. Int. J. Numer. Meth. Eng. 50, 2667–2682.
Wu, J.-Y., Li, F.-B., 2015. An improved stable XFEM (Is-XFEM) with a novel enrichment function for the computational modeling of cohesive cracks. Comput. Methods Appl. Mech. Eng. 295, 77–107.
Wu, J.-Y., Li, F.-B., Xu, S.-L., 2015. Extended embedded finite elements with continuous displacement jumps for the modeling of localized failure in solids. Comput. Methods Appl. Mech. Eng. 285, 346–378.
Wu, J.-Y., Nguyen, V.P., Nguyen, C.T., Sutula, D., Sinaie, S., Bordas, S.P., 2020. Phase-field modeling of fracture. Adv. Appl. Mech. 53, 1–183.
Xia, X., Liang, Q., 2016. A GPU-accelerated smoothed particle hydrodynamics (SPH) model for the shallow water equations. Environ. Model. Softw. 75, 28–43.
Xiong, Q., Li, B., Xu, J., 2013. GPU-accelerated adaptive particle splitting and merging in SPH. Comput. Phys. Commun. 184, 1701–1707.
Xu, X.-P., Needleman, A., 1994. Numerical simulations of fast crack growth in brittle solids. J. Mech. Phys. Solids 42, 1397–1434.
Xu, X., Wu, S., Jin, A., Gao, Y., 2018. Review of the relationships between crack initiation stress, mode I fracture toughness and tensile strength of geo-materials. Int. J. Geomech. 18, 04018136.
Yang, C., Zhu, F., Zhao, J., 2024a. Coupled total-and semi-Lagrangian peridynamics for modelling fluid-driven fracturing in solids. Comput. Methods Appl. Mech. Eng. 419, 116580.
Yang, X., Guo, N., Yang, Z., 2024b. A finite-volume implementation of the phase-field model for brittle fracture with adaptive mesh refinement. Comput. Geotech. 165, 105921.
Yang, X., Wang, E., Sun, W., Zhu, F., Guo, N., 2024c. Modeling fracture in multilayered teeth using the finite volume-based phase field method. J. Mech. Behav. Biomed. Mater. 157, 106655.
Yi, L., Zhang, D., Yang, Z., Li, X., Liao, Z., Chen, J., 2024. An improved phase-field model for oil–water two-phase flow and mixed-mode fracture propagation in hydraulic fracturing. Theor. Appl. Fract. Mech. 134, 104677.
Yue, Q., Wang, Q., Zhou, W., Rabczuk, T., Zhuang, X., Liu, B., Chang, X., 2023. An efficient adaptive length scale insensitive phase-field model for three-dimensional fracture of solids using trilinear multi-node elements. Int. J. Mech. Sci. 253, 108351.
Zhang, J., Dai, Z., Li, R., Deng, L., Liu, J., Zhou, N., 2023a. Acceleration of a production-level unstructured grid finite volume cfd code on gpu. Appl. Sci. 13, 6193.
Zhang, W., Wu, Z., Peng, C., Li, S., Dong, Y., Yuan, W., 2023b. Modelling large-scale landslide using a GPU-accelerated 3D MPM with an efficient terrain contact algorithm. Comput. Geotech. 158, 105411.
Zhang, X., Sloan, S.W., Vignes, C., Sheng, D., 2017. A modification of the phase-field model for mixed mode crack propagation in rock-like materials. Comput. Methods Appl. Mech. Eng. 322, 123–136.
Zhao, Z., Huang, K., Li, C., Wang, C., Qin, H., 2020. A novel plastic phase-field method for ductile fracture with GPU optimization. Comp. Graph. Forum, 105–117.
Zhong, J., Han, F., Zhang, L., 2024. Accelerated peridynamic computation on GPU for quasi-static fracture simulations. J. Peridyn. Nonlocal Model. 6, 206–229.
Zhou, S., Zhuang, X., Rabczuk, T., 2018. A phase-field modeling approach of fracture propagation in poroelastic media. Eng. Geol. 240, 189–203.