Technische Universität Bergakademie Freiberg

Computational Materials Science
Fakultät IV

Personal Programming Project

Peridigm code for Peridynamics Hydrogen Embrittlement Model

Prasanna Ramesh Hegde

Matriculation Number: 69030

May 4, 2025

Supervised by
Mr. Akshay Balachandran Jeeja

Professor Incharge
Prof. Dr.-Ing. habil. Bernhard Eidel
Dr.-Ing. Stefan Prüger
Dr.-Ing. Arun Prakash

Contents

1 Abstract

Hydrogen embrittlement represents a critical failure mechanism in metallic materials, significantly hindering the broader implementation of hydrogen-based energy technologies. To address this challenge, the current project introduces a peridynamic modeling approach specifically developed for simulating hydrogen embrittlement within the C++ based, open-source software called Peridigm, complemented by Python-based post-processing. This approach utilizes a two-parameter bond-based peridynamic theory, effectively overcoming inherent limitations associated with Poisson's ratio present in classical continuum mechanics.

The implemented model in Peridigm software is tested rigorously for its compatibility and stability in achieving the desired results. This covers unit-test, boundary condition test, mesh-convergence test, decoupling mechanical and diffusion kinetics, bond-failure analysis, time-step sensitivity analysis and crack-path prediction and crack propagation.

The developed model incorporates a tailored damage criterion designed explicitly to capture the critical influences of hydrogen embrittlement mechanisms. Through this formulation, the simulation effectively integrates hydrogen diffusion processes with resulting embrittlement effects. To validate the robustness and accuracy of the proposed peridynamic model, several rigorous test scenarios, particularly focusing on mode I crack propagation, are executed. The effectiveness and predictive capability of the model are determined based on the final contour plots of mode I crack propagation.

Keywords: Peridigm, Hydrogen embrittlement, Peridynamics, Crack propagation

2 Introduction

Hydrogen has emerged as a promising alternative to traditional fossil fuels due to its renewability, high energy potential, and environmentally friendly by-products [1]. It also serves as an essential energy carrier, capable of storing the intermittent energy generated by renewable sources. Large-scale hydrogen applications face significant challenges, especially in transportation and storage sectors, primarily due to hydrogen embrittlement. Metallic materials such as steel and cobalt, commonly used in hydrogen storage vessels, are particularly susceptible to this phenomenon [2, 3]. The small covalent radius ($3.70e-11$ m) and high permeability of hydrogen molecules allow easy diffusion into metallic lattices, causing embrittlement. Hydrogen embrittlement deteriorates critical material properties, notably strength and ductility. Accumulated hydrogen around crack tips can shift the failure mode from ductile to brittle, significantly reducing fracture toughness to about one-tenth of the material's original strength. This degradation can cause material failure even under relatively low loads, increasing the risk of hydrogen leakage. Hydrogens wide flammability range and low ignition energy further exacerbate the risks associated with leaks, potentially resulting in explosions or combustion events [4, 5].

Peridynamics (PD) is an alternative formulation of traditional continuum mechanics, characterized by the spatial integral equilibrium equation [1]. Unlike classical methods, the PD approach effectively addresses challenges associated with spatial derivatives near discontinuities, making it particularly suitable for fracture analysis. In the PD framework, the material domain is discretized into discrete material points interacting with each other within a finite interaction domain. Instead of the classical stress concept, interactions between these points are governed by peridynamic forces (PD forces). These PD forces are transmitted through bonds, which connect pairs of material points. Damage within this model is identified by bond breakage, allowing complex crack behaviors, including crack initiation, propagation, and branching, to emerge spontaneously from a straightforward bond-breaking criterion. Due to these advantages, PD has been widely employed to examine fracture phenomena in various materials [6, 7].

Regarding corrosion damage simulation in metals, Li and Guo have investigated chloride diffusion and corrosion initiation in concrete-embedded rebars [8]. Hermann et al. developed a coupled model combining finite element methods with PD to analyze corrosion in bone implants [9]. Despite these developments, research specifically addressing the PD simulation of hydrogen embrittlement remains limited. De Meo et al. proposed a peridynamic hydrogen embrittlement (PDHE) model, based on the adsorption-induced dislocation emission (AIDE) mechanism, to study hydrogen-assisted failures in AISI 4340 steel [10]. These existing PDHE models, however, are based on bond-based peridynamics (BBPD), inherently constrained by limitations related to Poisson's ratio since a single PD parameter is linked to the two Lamé constants of isotropic elasticity.

Peridigm is a mesh-free computational tool based on peridynamic theory, developed primarily in C++ and optimized for execution on large-scale parallel computing systems, initially created by Sandia National Laboratories [11]. The primary objective of this Personal Programming Project is to propose a Peridigm code for a generalized PDHE model capable of accurately simulating hydrogen-assisted fractures for materials with arbitrary Poisson's ratios, as well as analyzing complex hydrogen-driven crack propagation. The proposed code employs a two-parameter bond-based peridynamics theory to overcome the limitations associated with Poisson's ratio, explicitly accounting for hydrogen diffusion and its embrittling effects. Section 3 presents a review on theory of PD and the two-parameter model, development of the peridynamic hydrogen diffusion model and peridynamic hydrogen embrittlement mechanism. Section 4 gives the numerical implementation of the proposed PDHE model. Section 5 explains the implementation of a custom material model into Peridigm, reading necessary input and geometrical parameters into the compute class. Section 6 presents an idea of the program flow. Section 8 verifies and validates the effectiveness and versatility of the proposed code, by conducting mesh-convergence, sensitivity analysis and mode I crack propagation, with/without hydrogen embrittlement. Sensitivity analysis, a key test on explicit numerical schemes, describes the effect of time step size on hydrogen-induced crack propagation. Numerical outcomes from all test cases are assessed for the expected behaviour of the implemented material model in Peridigm, demonstrating the codes reliability. Section 11 gives the detailed explanation of pre-processing methods that was practiced, input-deck creation for simulation and post-processing using python script.

3 Theory

3.1 Overview: Peridynamics

Peridynamics (PD) is a non-local formulation of continuum mechanics [1]. In this approach, the material body is represented by discrete points, known as material points, which interact with surrounding points within a specific finite region called the horizon, as illustrated in Fig.1. In three-dimensional analyses, the horizon takes the shape of a sphere, while in two-dimensional cases, it is a circle. A given material point, such as point i in Fig.1, interacts exclusively with other points located within this horizon, whereas no interactions occur with points lying beyond this boundary, exemplified by point k in Fig.1. The radius of this horizon, denoted by δ, defines the spatial extent of these interactions and is typically selected as a multiple of the discretization spacing (dx). A commonly used value for δ is approximately 3.015 times the grid spacing ($\delta = 3.015dx$) [12]. The force arising from interactions between these material points is termed the peridynamic force (PD force).

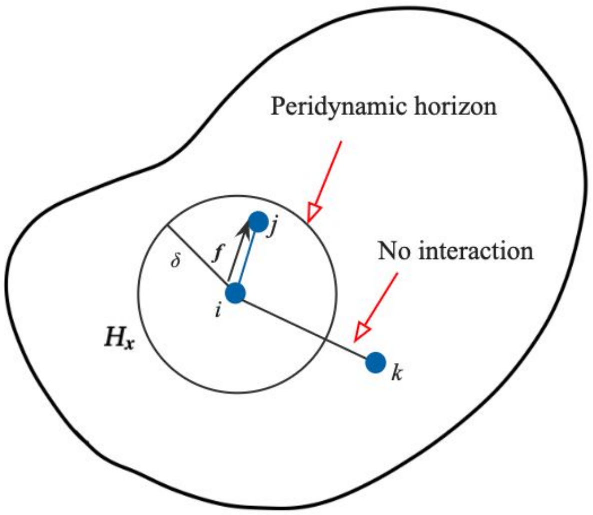


Figure 1: Peridynamic horizon and the peridynamic force acting on a bond.

According to the Newton's second law of motion, PD dynamic equilibrium equation is given as follows:

$$\rho\,\ddot{\boldsymbol{u}}(\boldsymbol{x},t) = \int_{H_{\boldsymbol{x}}} \boldsymbol{f}\Big(\boldsymbol{u}' - \boldsymbol{u},\ \boldsymbol{x}' - \boldsymbol{x},\ t\Big)\,\mathrm{d}V_{\boldsymbol{x}'} + \boldsymbol{b}(\boldsymbol{x},t) \tag{1}$$

where ρ is the density, $\ddot{\boldsymbol{u}}$ is the acceleration, t is time, $H_{\boldsymbol{x}}$ is the horizon of point i, $\boldsymbol{f}$ is the PD force acting between point i and j, $\boldsymbol{u'}$ and $\boldsymbol{u}$ are the displacements of point j and i respectively, $\boldsymbol{x'}$ and $\boldsymbol{x}$ are location vectors of point j and i in the reference configuration, $V_{\boldsymbol{x'}}$ is the volume of point j and $\boldsymbol{b}$ is the body force density.

3.2 Bond-Based Peridynamics

Peridynamic theory is categorized into two primary types: bond-based peridynamics (BBPD) and state-based peridynamics (SBPD). In the BBPD approach, the peridynamic force depends solely on the deformation of the individual bond that transmits it, as depicted in Fig.(1). Conversely, the SBPD framework considers the peridynamic force as dependent on the collective deformation of all bonds connecting two interacting material points.

In traditional BBPD, the PD force is linear to the bond stretch, see Fig.(2). Bond stretch is defined as:

$$\begin{aligned} s &= \frac{|\boldsymbol{\xi}+\boldsymbol{\eta}| - |\boldsymbol{\xi}|}{|\boldsymbol{\xi}|}\,, \\ \boldsymbol{\xi} &= \boldsymbol{x}' - \boldsymbol{x}\,, \\ \boldsymbol{\eta} &= \boldsymbol{u}' - \boldsymbol{u}\,. \end{aligned} \tag{2}$$

where $\boldsymbol{\xi}$ is the initial relative position and $|\boldsymbol{\xi}|$ is the initial bond length, $\boldsymbol{\eta}$ is the relative displacement of the two material points.

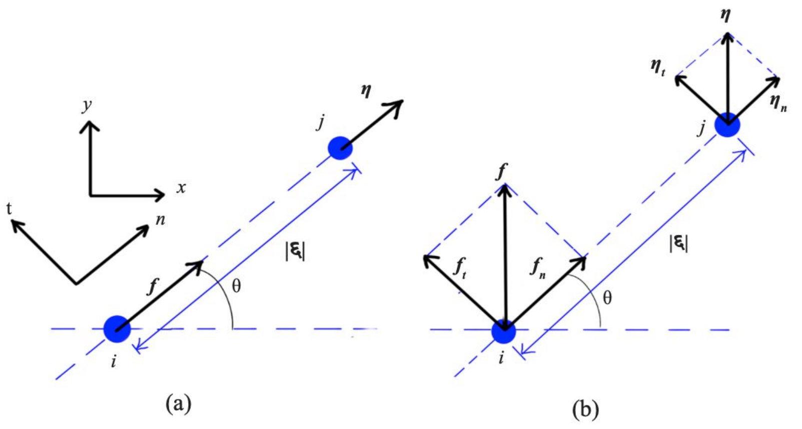


Figure 2: Two BBPD models: (a) the traditional BBPD, (b) the two-parameter model of BBPD.

In BBPD, the bond performs like a spring that connects the two material points and the PD force is calculated as:

$$\begin{aligned} \boldsymbol{f} &= \omega\big(|\boldsymbol{\xi}|\big)\, cs\, \frac{\boldsymbol{y}' - \boldsymbol{y}}{|\boldsymbol{y}' - \boldsymbol{y}|}\,, \\ \boldsymbol{y} &= \boldsymbol{x} + \boldsymbol{u}\,, \\ \boldsymbol{y}' &= \boldsymbol{x}' + \boldsymbol{u}' \end{aligned} \tag{3}$$

where $\omega\big(|\boldsymbol{\xi}|\big)$ is the influence function, $\boldsymbol{y}$ and $\boldsymbol{y'}$ are the location vectors of the point i

and j in the current configuration. c is the PD parameter and is related to Youngs modulus and Poisson's ratio. The PD parameter is calculated as [12, 13]:

$$c = \begin{cases} \dfrac{6E}{\pi\, h\, \delta^3 (1-\mu)}, & \text{plane stress} \\ \dfrac{6E}{\pi\, h\, \delta^3 (1-2\mu)\,(1-\mu)}, & \text{plane strain} \\ \dfrac{6E}{\pi\, \delta^4 (1-2\mu)}, & \text{3D} \end{cases} \tag{4}$$

where E is the Youngs modulus, μ is the Poisson's ratio, h is the thickness, and δ is the horizon size. Different from the two independent material parameters for isotropic material in continuum mechanics, there is only one PD parameter in BBPD, which leads to the limitation of Poisson's ratio (μ=1/3 in plane stress, μ=1/4 in plane strain and 3D).

3.3 Two-parameter model: Extended Bond-Based Peridynamics

Poisson's ratio plays a crucial role in determining crack behavior, particularly in complex fracture scenarios such as crack branching and intersections [14, 15]. Many steels used in hydrogen storage and transportation systems exhibit Poisson's ratios that deviate from the restrictive values assumed in basic models. For instance, AISI 4340 has a Poisson's ratio of 0.29, while both X80 and STS316L steels have ratios around 0.3 [16, 17]. Hydrogen-induced fractures in these materials often follow intricate paths, as demonstrated in both experimental studies (Hirose and Mura, 1984; [18]) and numerical simulations [10]. The resulting fracture surfaces frequently exhibit small pits, a characteristic effect of hydrogen diffusion, and the crack trajectories tend to be curved and nonlinear.

Despite these observations, most existing peridynamic models for hydrogen embrittlement rely on bond-based peridynamics (BBPD), which imposes a fixed Poisson's ratio. This limitation reduces the models ability to accurately simulate the deformation behavior and fracture patterns of steel. To overcome this issue, a peridynamic hydrogen embrittlement (PDHE) model built upon a two-parameter extension of the BBPD framework [1] is used in this project, hereafter referred to as the two-parameter model. This formulation allows for greater flexibility in representing materials with varying Poisson's ratios and improves the predictive capability for complex fracture phenomena.

The two-parameter model is an extended BBPD model that adds a tangential spring to the bond interaction [19]. The bond can transmit both the axial and tangential PD forces (see Fig.2(b)), where $\boldsymbol{x}$-$\boldsymbol{y}$ represents the global coordinate system and $\boldsymbol{n}$-$\boldsymbol{t}$ represents the local coordinate system. The PD forces are linearly related to the axial and tangential components of the bond stretch:

$$\begin{pmatrix} \boldsymbol{f_n} \\ \boldsymbol{f_t} \end{pmatrix} = \begin{bmatrix} k_n & 0 \\ 0 & k_t \end{bmatrix} \begin{pmatrix} \dfrac{\boldsymbol{\eta_n}}{|\boldsymbol{\xi}|} \\ \dfrac{\boldsymbol{\eta_t}}{|\boldsymbol{\xi}|} \end{pmatrix} \tag{5}$$

where k_n and k_t are the PD parameters in axial and tangential direction, respectively, the subscript n and t represent the axial and tangential component. Since we are considering plane stress problem, k_n and k_t can be given as:

$$\begin{cases} k_n = \dfrac{6E}{\pi\, h\, \delta^3\,(1-\mu)}, \\ k_t = \dfrac{6E\,(1-3\mu)}{\pi\, h\, \delta^3\,(1-\mu^2)}. \end{cases} \tag{6}$$

If we substitute the limited Poisson's ratio of 1/3 into Eq. (5), k_n equals the PD parameter c of Eq.(4) and k_t equals 0. Under this condition, the two-parameter model can be reduced to the BBPD when applying the limited Poisson's ratio, demonstrating that the BBPD is the special case of the two-parameter model.

In the peridynamic (PD) framework, material degradation is represented through the failure of bonds between material points. To effectively model hydrogen embrittlement, this study employs the prototype microelastic brittle (PMB) material model as described in [13]. Once a bond fails, it is no longer capable of transmitting peridynamic forces. Hydrogen diffusion into the steel lattice leads to a reduction in ductility, and the resulting hydrogen-assisted failure manifests typical characteristics of brittle fracture.

The failure of a bond is governed by a critical stretch criterion: a bond is considered broken when its stretch exceeds a predefined critical threshold, denoted as s_c. To mathematically represent bond failure, a scalar damage function $b_d(t, s)$ is introduced, which tracks the status of bond breakage over time and space.

$$b_d(t,s) = \begin{cases} 1, & s < s_c \\ 0, & s \geq s_c \end{cases} \tag{7}$$

The relation between the critical bond stretch s_c and the critical energy release rate G_c is as follows:

$$s_c = \begin{cases} \sqrt{\dfrac{10\, G_c}{c\, \pi\, \delta^5}}, & \text{3D} \\ \sqrt{\dfrac{4\, G_c}{c\, h\, \delta^4}}, & \text{plane problems} \end{cases} \tag{8}$$

where c is the PD parameter in BBPD, δ is the horizon size, h is the thickness.

The damage value $\varphi(\boldsymbol{x}, t)$ is defined to represent the local damage of the material point:

$$\varphi(\boldsymbol{x}, t) = 1 - \frac{\displaystyle\int_{H_x} b_d(t, s)\, \mathrm{d}V_{x'}}{\displaystyle\int_{H_x} \mathrm{d}V_{x'}}. \tag{9}$$

where $\varphi(\boldsymbol{x}, t)$ ranges from 0 to 1, and larger damage value means more broken bonds of the material point.

3.4 Peridynamic hydrogen diffusion model

De Meo et al. introduced a nonlocal peridynamic hydrogen diffusion model grounded in the Fisher-type diffusion formulation [10], which was later employed by Ran et al. to investigate hydrogen-related fracture phenomena [20]. To model hydrogen transport within the steel lattice in this study, we adopt the same C-type grain boundary diffusion mechanism.

$$\dot{C}(\boldsymbol{x}, t) = \int_{H_x} d_h \, \frac{C(x', t) - C(x, t)}{|\boldsymbol{\xi}|^2} \, \mathrm{d}V_{x'} \tag{10}$$

The governing peridynamic hydrogen diffusion equation, corresponding to Eq. (10), closely resembles the formulation used in peridynamic corrosion damage models, as discussed in [20, 21]. In this framework, hydrogen migrates from regions of higher concentration to those of lower concentration. Here, C denotes the hydrogen concentration, $\dot{C}$ its time derivative, and H_x represents the peridynamic horizon. The initial bond length is given by $|\boldsymbol{\xi}|$, and d_h is the peridynamic diffusion bond constant, which is linked to the grain boundary diffusion coefficient.

For plane-stress conditions, the diffusion bond constant d_h is computed as detailed in [10], allowing the model to accurately represent hydrogen transport across material microstructures:

$$d_h = \frac{6\, D_{\mathrm{gb}}}{\pi\, h\, \delta^3} \tag{11}$$

In this context, h denotes the material thickness and δ represents the horizon radius. The grain-boundary diffusion coefficient D_{gb}, governs the rate at which hydrogen penetrates the steel lattice and is influenced by various factors such as temperature and pressure. An increase in temperature, for instance, enhances the likelihood of hydrogen diffusion into the material.

The proposed peridynamic hydrogen embrittlement (PDHE) model incorporates two distinct types of bonds: mechanical bonds and diffusion bonds. Mechanical bonds are responsible for transmitting peridynamic forces between material points, whereas diffusion bonds facilitate hydrogen transport. As outlined in Section 3.3, a mechanical bond is considered broken once its stretch exceeds a predefined critical threshold, after which it can no longer carry force. In contrast, diffusion bonds remain unaffected by mechanical damage in accordance with the modeling assumption adopted in [21].

3.5 Coupled hydrogen embrittlement and the PD damage model

Hydrogen atoms generated from corrosion can diffuse into the material lattice, leading to the degradation of the material's properties. To describe the hydrogen concentration inside the material, a normalized factor named hydrogen coverage ψ is introduced:

$$\psi = \frac{C}{C_{sv}} \tag{12}$$

where C_{sv} is the saturated value of hydrogen concentration. The value of hydrogen coverage ranges from $0 \sim 1$ and larger hydrogen coverage means more hydrogen atoms are concentrated in the steel lattice.

Hydrogen embrittlement is initiated when hydrogen accumulates near the crack tip, and in this project, the hydrogen adsorption-induced dislocation emission (AIDE) mechanism is incorporated into the peridynamic hydrogen embrittlement (PDHE) model. The AIDE mechanism integrates features of both hydrogen-enhanced decohesion (HEDE) and hydrogen-enhanced localized plasticity (HELP). As hydrogen diffuses into the material, it promotes the formation of micro-voids, leading to a reduction in both strength and ductility. The accumulation of hydrogen at the crack tip enhances lattice dislocation emission, which in turn accelerates micro-void coalescence and facilitates crack growth.

To effectively couple hydrogen embrittlement behavior with the peridynamic damage formulation, the critical bond stretch is modeled as a function of hydrogen surface coverage. The dependence of the critical bond stretch on hydrogen coverage, as represented in Eq. (8), provides a means to link the microstructural effects of hydrogen with the macroscopic fracture behavior within the peridynamic framework.

$$s_c(\psi) = s_c(0)\left(1 - 1.0467\,\psi + 0.1687\,\psi^2\right) \tag{13}$$

Here, $s_c(0)$ represents the initial critical bond stretch as defined by Eq. (8). As illustrated in Fig. (3), the critical bond stretch decreases progressively with increasing hydrogen concentration. Specifically, when the hydrogen coverage reaches saturation ($\psi = 1$), the critical bond stretch reduces to approximately 12.2% of its original value.

Since a bond connects two material points, the hydrogen coverage influencing the bond's failure is determined by averaging the coverage values at both connected points.

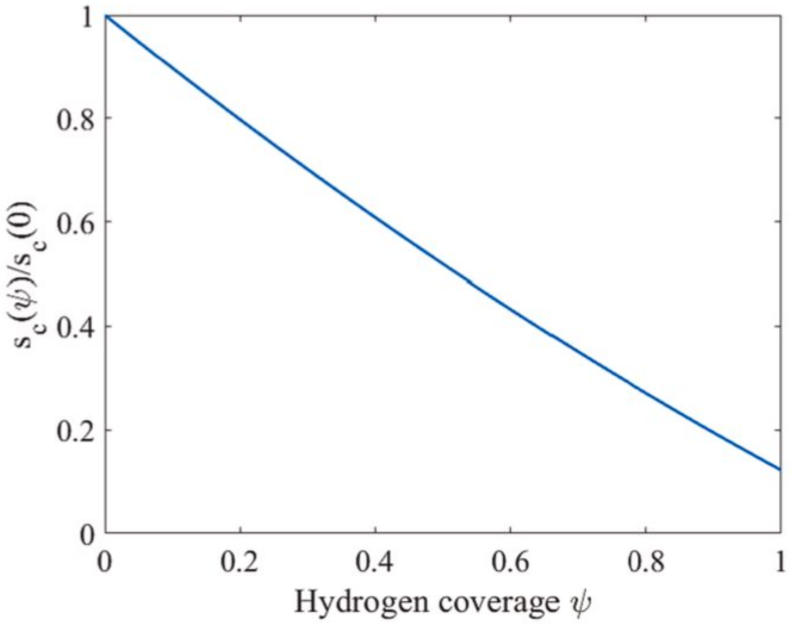


Figure 3: The relation between the critical bond stretch and the hydrogen coverage. [1]

Thus, the hydrogen coverage used in Eq. (13) corresponds to the mean value of the hydrogen coverage at material points i and j:

$$\psi = \frac{\psi(\boldsymbol{x}) + \psi(\boldsymbol{x'})}{2} \tag{14}$$

As new damage develops within the material, the rate of hydrogen diffusion increases, thereby enhancing the likelihood of hydrogen embrittlement. To capture this effect in the simulation, a saturated diffusion boundary condition is imposed on the freshly formed crack surfaces. Within the numerical framework, a material point is considered to have reached hydrogen saturation if its damage value exceeds 0.36. This threshold not only governs the application of the saturation condition but is also used to identify the position of the crack tip, as outlined in [13].

4 Numerical Methods and Implementation

4.1 Discretization

In PD numerical simulation, the domain is discretized into N number of material points, each with an associated location and volume. A uniform mesh is applied in this study. Furthermore, the integration symbols of the peridznamic equilibrium equation and diffusion equation are replaced with the summation symbols:

$$\rho\,\ddot{\boldsymbol{u}}(\boldsymbol{x},t) = \sum_{j=1}^{N_i} \boldsymbol{f}(\boldsymbol{\xi},\boldsymbol{\eta},\theta,\theta',t)V_j + \boldsymbol{b}(\boldsymbol{x},t) \tag{15}$$

$$\dot{C}(\boldsymbol{x},t) = \sum_{j=1}^{N_i} d_h \frac{C(x_j,t) - C(x,t)}{|\boldsymbol{\xi}|^2} V_j \tag{16}$$

In the peridynamic (PD) numerical simulation, N_i represents the number of neighboring material points within the horizon.

4.2 Boundary effects

As with other meshfree methods, boundary effects are encountered in PD simulations. To mitigate the numerical errors introduced by these boundary effects, surface and volume corrections are applied, as described in previous works [12, 22].

The horizons of the material points located near the surface are incomplete, and the PD parameters of these points require surface correction:

$$k_i^n = \lambda_c k_i^o \tag{17}$$

$$\lambda_c = \frac{2V_0}{V_x + V_{x'}} \tag{18}$$

Here, k_i^n and k_i^o (where $i = n, t$) represent the modified and original PD parameters, respectively, with the original PD parameters calculated using Eq. (6). The surface correction factor is denoted by λ_c, and V_0 is the volume of the entire PD horizon, which is equal to $\frac{h}{\pi\delta^2}$ in 2D and $\frac{4}{3}\pi\delta^3$ in 3D. V_x and $V_{x'}$ are the volumes of the PD horizons at points x and x', respectively.

Neighboring points located at the edge of the PD horizon are not fully included, and therefore, the PD forces between the central point and these boundary points require volume correction.

$$\boldsymbol{f_n} = v_c \boldsymbol{f_o} \tag{19}$$

$$v_c = \begin{cases} 1, & |\boldsymbol{\xi}| < \delta - \frac{dx}{2}, \\ \frac{\delta - |\boldsymbol{\xi}| + \frac{dx}{2}}{dx}, & |\boldsymbol{\xi}| > \delta - \frac{dx}{2}. \end{cases} \tag{20}$$

where $\boldsymbol{f_n}$ and $\boldsymbol{f_o}$ are the modified PD forces and original PD forces, respectively. v_c is the volume correction factor. $|\boldsymbol{\xi}|$ is the bond length. δ is the horizon size. dx is the grid space.

4.3 Time integration scheme

Unlike implicit solvers, the explicit method avoids large-matrix operations, which results in lower computational costs. In this study, an explicit dynamic relaxation method with a quasi-static assumption is employed, in which each time step is carefully chosen. In dynamic methods, a nonlinear problem is addressed using artificial damping, which ensures the solution becomes stable after a sufficient number of iterations. The equation of motion (Eq. 15) can be rewritten in vector form as follows [23]:

$$\ddot{\boldsymbol{u}}(\boldsymbol{x}, t) + c\dot{\boldsymbol{u}}(\boldsymbol{x}, t) = \boldsymbol{f}(\boldsymbol{u}, \boldsymbol{x}, t) \tag{21}$$

Here, c represents the damping ratio coefficient, and Λ is the fictitious diagonal density matrix. Using the adaptive dynamic relaxation method, the optimal diagonal density matrix and damping coefficient can be determined through Greschgorin's theorem and Rayleigh's quotient, respectively [24]. Let $\boldsymbol{u_n}$, $\dot{\boldsymbol{u_n}}$, $\ddot{\boldsymbol{u_n}}$, and $\boldsymbol{f_n}$ denote the displacement, velocity, acceleration, and force vector at $t = n$, respectively, where Δt is the time-step size, assumed constant. In the central difference scheme, the velocity and acceleration vectors are approximated as follows [23]:

$$\boldsymbol{u}^n \approx \frac{1}{2\Delta t}\left(\boldsymbol{u}^{n+1} - \boldsymbol{u}^{n-1}\right), \tag{22}$$

$$\ddot{\boldsymbol{u}}^n \approx \frac{1}{\Delta t^2}\left(\boldsymbol{u}^{n+1} - 2\boldsymbol{u}^n + \boldsymbol{u}^{n-1}\right), \tag{23}$$

Then, substitute (22) and (23) into (21), and rearrange terms for $\boldsymbol{u}^{n+1}$:

$$\boldsymbol{u}^{n+1} = \frac{2\Delta t^2 \boldsymbol{f}^n + 4\boldsymbol{u}^n + (c\Delta t - 2)\boldsymbol{u}^{n-1}}{2 + c\Delta t}, \tag{24}$$

which is the update scheme for the displacement field. Equation (25) is employed to approximate $\boldsymbol{u}^{-1}$ to initialize the displacement iteration:

$$\boldsymbol{u}^{-1} = \boldsymbol{u}^0 - \Delta t\, \dot{\boldsymbol{u}}^0 + \frac{\Delta t^2}{2}\ddot{\boldsymbol{u}}^0, \tag{25}$$

where $\boldsymbol{u}^0$, $\dot{\boldsymbol{u}}^0$, and $\ddot{\boldsymbol{u}}^0$ are the initial displacement, velocity, and acceleration vectors, respectively. The velocity and acceleration vectors can be updated afterwards by (22) and (23), though not necessary.

In two-dimensional cases, the CFL condition is more stringent. Assuming that the wave speeds along x and y directions are the same and the use of a uniform grid where $\Delta x = \Delta y$, the critical time step size becomes [23]

$$\Delta t \leq \Delta x \cdot \sqrt{\frac{\Lambda_{ii}}{E_{\max}}}, \tag{26}$$

in which $E_{\max}$ is the maximum component of the elastic stiffness matrix used to approximate the maximum possible wave speed. Eq.(26) is the time step size employed in this paper.

The damping ratio c^n is then selected carefully by the lowest frequency of the system using Rayleigh's quotient [23]:

$$c^n = 2\sqrt{\frac{(\boldsymbol{u}^n)^\top \boldsymbol{k}^n \boldsymbol{u}^n}{(\boldsymbol{u}^n)^\top \boldsymbol{u}^n}}, \tag{27}$$

where $\boldsymbol{k}^n$ is the diagonal local stiffness matrix, which is given by [23]

$$\boldsymbol{k}^n_{ii} = -\left(\frac{\boldsymbol{f}^n_i}{\Lambda_{ii}} - \frac{\boldsymbol{f}^{n-1}_i}{\Lambda_{ii}}\right) / \left(\boldsymbol{u}^n_i - \boldsymbol{u}^{n-1}_i\right), \tag{28}$$

where $\boldsymbol{f}^n_i$ is the i-th component of the force vector $\mathbf{f}$ at time $t = n$ and Λ_{ii} is set to 1. Since the local stiffness matrix calculation involves division by the difference between current and old displacement components, it is highly possible to encounter a zero-component in the displacement field where the criterion fails [24]. Therefore, the local stiffness $\boldsymbol{k}^n_{ii}$ is set to zero when the difference between displacement fields vanishes. Finally, when the c^n equals 1.9 when c^n calculated by Eq.(27), is exceeds 2.0. The limitation of damping coefficient guarantees numerical stability and minimizes the dynamic response to obtain the quasi-static solution.

The forward Euler method is applied in computing the hydrogen concentration:

$$C_{n+1}(\boldsymbol{x}, t_n) = C_n(\boldsymbol{x}, t_n) + \dot{C}_n(\boldsymbol{x}, t_n)\Delta t_h \tag{29}$$

where Δt_h is the hydrogen diffusion time step size, chosen according to [25], the hydrogen concentration at time step t_n ($t_n = t_{n-1} + \Delta t_h$) is obtained by Eq. (29) before computing the PD force.

5 External libraries, software (packages) and Interfaces

5.1 Overview: Peridigm

The Peridigm code was developed for high-fidelity peridynamic simulations across two-dimensional and three-dimensional domains using the mesh-free discretization method proposed by Silling and Askari [11]. This open-source C++ software leverages libraries from the Trilinos [26] project to enable large-scale parallel simulations. Peridigm is designed to support engineering simulations that involve solid mechanics and material failure, while also offering a software framework for peridynamic method developers. Its key features include a variety of constitutive models, bond failure criteria, contact models, and capabilities for both explicit and implicit time integration. The code's MPI-based architecture allows it to run efficiently on platforms ranging from personal laptops to large-scale supercomputing systems.

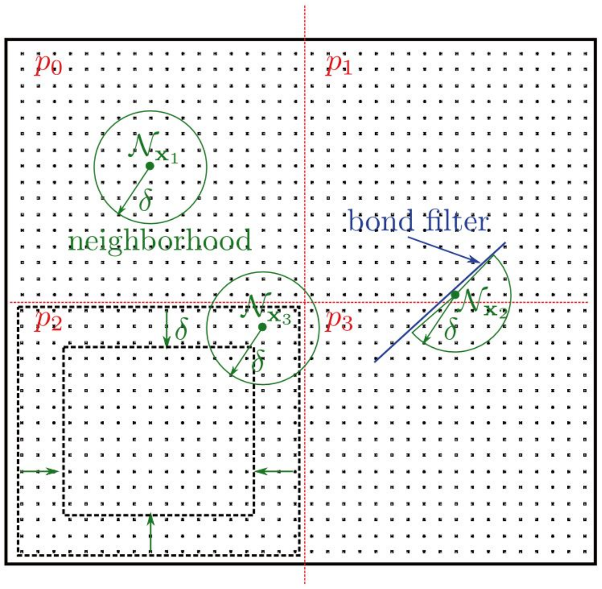
Step-by-step instructions on installing,creating input-deck and running a simulation using Peridigm, is given in section 11.

5.2 Data Structures for Nonlocal Calculations

A key aspect of performing peridynamic calculations is the construction and management of neighborhood lists. These lists serve as the primary data structure for iterating over sets of nodal volumes [11]. For instance, the summation in Eq. (21) typically involves a neighborhood size $\mathcal{N}_x$ on the order of $\mathcal{O}(100)$ for three-dimensional simulations and can grow as large as $\mathcal{O}(1000)$. An example of a mesh-free discretization used by Peridigm is shown in Fig. 4. In this method, neighborhoods, depicted in green for three points (x_1, x_2, x_3) in the domain, are determined through a spatial proximity search. A point q is considered part of the neighborhood $\mathcal{N}_x$ if the distance between points x and q in the undeformed configuration is less than the horizon δ.

The construction and traversal of neighborhood lists significantly influence the computational expense of peridynamic simulations. This task becomes more complex when dealing with parallel domain decomposition, where nodes are distributed across multiple processors. Figure 4 illustrates how parallel decomposition relates to the neighborhood list structure. Building neighborhood lists across processor boundaries requires search methods that query neighboring points between processors. This is achieved in Peridigm by creating subsets of points on each processor, called frame-sets. In combination with load balancing through the Trilinos Zoltan library, framesets on each processor are used to optimize neighborhood list construction.

Nodes on adjacent processors within the search distance δ are collected. Using this cross-processor search, which only considers points within a frame-set, communication lists are



p_0
p_1
$\mathcal{N}_{\mathbf{x}_1}$
δ
neighborhood
bond filter
p_2
δ
$\mathcal{N}_{\mathbf{x}_3}$
δ
p_3
$\mathcal{N}_{\mathbf{x}_2}$
δ

Figure 4: A schematic of a discretized computational domain is shown. The boundary of the domain is represented by a solid black line. A decomposition into four MPI ranks, p_0, p_1, p_2, and p_3, is illustrated in red. The neighborhoods $\mathcal{N}_x$ for selected material points are represented by green circles with a radius of δ. A bond filter, which restricts the creation of bonds across a user-defined plane, is indicated in blue. Additionally, a frame-set used for proximity searches across processor boundaries is depicted with dotted black lines.

formed to enable parallel communication throughout the simulation. Specifically, each processor maintains a list of owned points and a list of shared points. Shared points are those off-processor points that lie within the distance δ from owned (on-processor) points.

Neighborhood lists are built using k-d tree search algorithms [11]. On each processor, trees are constructed from the union of owned and shared points. The construction and query processes approximately scale as $\mathcal{O}(N\log(N))$ and $\mathcal{O}(N^{1-1/3}+k)$, respectively, where N is the number of nodes in the tree, and k is the number of nodes in a neighborhood. The size of a neighborhood depends on the ratio of the horizon δ to the mesh spacing. Peridigm's proximity search for neighborhood list construction can be enhanced using bond filters. These filters are user-defined planes that specify fine-scale geometric features, such as thin notches. Bonds that cross these planes are excluded from neighborhood list construction, as shown in Fig. 4.

After parallel decomposition and neighborhood list construction, Peridigm organizes on-processor points into blocks based on material model type and instance. This grouping facilitates efficient evaluation of large node groups within a block, avoiding excessive branching or overload of functionals based on material model type. All on-processor nodes with the same material type and properties are grouped, and material model evaluation is carried out by passing the neighborhood list for these points to the appropriate subroutines, along with other material parameters.

These data structures are managed by `NeighborhoodData` and `DataManager` objects in

Peridigm. While global operations, such as time integration, are performed using parallel data structures spanning the entire domain, computations for individual blocks are handled with subsets of data stored in the `DataManager`. The `DataManager` handles communication to and from global data structures and manages all aspects of parallel communication. Routines that iterate over neighborhoods rely on information extracted from the corresponding `NeighborhoodData` objects. This design allows material models, bond damage models, contact models, and computation classes to be written as serial code, simplifying the process of modifying and extending these routines.

5.3 Mesh-free Discretizations

Peridigm operates on meshfree discretizations consisting of nodal volumes, each defined by spatial coordinates $\boldsymbol{x}$ and volume ΔV. The geometry of the domain, typically conceptualized by the user in terms of geometric primitives or similar representations, is captured by the spatial distribution of these nodal volumes. The total volume ΔV sums to match the volume of the overall domain geometry. Unlike traditional finite element methods, there is no topology associated with the meshfree discretization; specifically, no link arrays that describe connectivity between elements are present. Quadrature for meshfree discretizations of this type is significantly simplified, relying solely on the coordinates of individual nodes and their corresponding volumes.

Peridigm manages the association of material models, boundary conditions, and other simulation aspects with specific regions of the computational domain through the use of blocks and node sets. Each nodal volume in the discretization is assigned to a single block. In the Peridigm input script, material specifications are linked to each block, enabling the evaluation of internal forces using the appropriate material model and its corresponding properties. Node sets are collections of nodal volumes used to apply initial conditions, boundary conditions, and body forces. A single nodal volume may be part of multiple node sets. It is important to note that nonlocal boundary conditions for peridynamic models should be applied to a three-dimensional volumetric region rather than a two-dimensional surface. Additionally, special care should be taken when specifying initial and boundary conditions, ensuring proper associations with node sets during the creation of a *Peridigm* simulation.

Peridigm supports two primary file formats for meshfree discretizations: *exodus* files and text files. In practice, it is often convenient to use a finite element mesh generator to create a hexahedral or tetrahedral mesh of the domain. In this project, the open-source software *Gmsh* is used to create a mesh, and *meshio* software is used to write it to *exodus* and text file formats. A Detailed explanation is given in section 11. The text file format offers a more simplistic alternative that may be suitable for users who wish to create a meshfree discretization directly.

5.4 Compute classes

Peridigm is designed with extensibility in mind. Compute classes enable the calculation of quantities that are of interest, which are stored as state variables and written to the Peridigm output file. Peridigm comes pre-packaged with a variety of compute classes, and the provided programming interface allows users to define their own classes without needing to manage parallel communication or output routines. Whenever an output field is linked to a specific compute class in the Peridigm input file, an instance of the compute class is created. The `compute()` method of the class is then called to generate the output for writing to disk.

These compute classes are derived from the `Compute` base class, which contains the programming interface outlined in Table 1.

`Compute( ... )`	Constructor
`FieldIds( ... )`	Return field IDs for variables used by the class
`initialize( ... )`	Initialize relevant data
`compute( ... )`	Compute quantities of interest

Table 1: Key methods in the `Compute` base class.

The constructor's arguments are particularly useful for compute classes that need to handle multiple material blocks or directly access the communicator object. The method `FieldIds()` returns a list of field IDs for the variables that the class operates on, ensuring that all required fields are correctly allocated and accessible in the `compute()` method. The `initialize()` method allows the compute class to prepare any necessary data before the simulation starts. Finally, the `compute()` method is where the actual computation of the quantities of interest takes place.

In this project `computeForce()` class is used, which is derived from the `Compute` base class and includes the private data shown in Table 2.

`modelCoordinatesFieldId`	Original/reference coordinates
`FieldIds( ... )`	Return field IDs for variables used by the class
`coordinatesFieldId`	Current coordinates
`volumeFieldId`	Per-node volume
`concentrationFieldId`	Used for hydrogen concentration
`damageFieldId`	Used to store Damage value

Table 2: Private data of `computeForce` class

The constructor of the `computeForce` class is outlined in table 3. The constructor interacts with the `FieldManager` class, which tracks information for the allocation, storage, and parallel communication of field data. The `FieldManager` class and associated `DataManager` class leverage the Trilinos framework. As show in Table 3, the `fieldManager` object returns the field IDs for the necessary vectors, allowing for efficient access to these data structures in subsequent calculations in the `compute()` routine.

```
computeForce(.....)
   modelCoordinatesFieldId = fieldManager.getFieldId("ModelCoordinates")
   coordinatesFieldId = fieldManager.getFieldId("Coordinates")
   volumeFieldId = fieldManager.getFieldId("Volume")
   concentrationFieldId = fieldManager.getFieldId("Temperature")
   damageFieldId = fieldManager.getFieldId("Damage")
   fieldIds.push_back(modelCoordinatesFieldId)
   fieldIds.push_back(coordinatesFieldId)
   fieldIds.push_back(volumeFieldId)
   fieldIds.push_back(concentrationFieldId)
   fieldIds.push_back(damageFieldId)
end
```

Table 3: Contents of the `computeForce` constructor

5.5 Implementation of `MySimpleMaterial` in Peridigm

In this section, a detailed explanation of how to implement a simple material model `MySimpleMaterial` in the Peridigm framework. This material model involves defining the material properties, setting up necessary field IDs to manage material data, and implementing the `computeForce` method for necessary calculations.

5.5.1 Including Required Header Files

At the beginning of the code, several header files are included to provide necessary functionality for the material model:

```cpp
#include "Peridigm_MySimpleMaterial.hpp"
#include "Peridigm_Field.hpp"
#include "material_utilities.h"
#include "Peridigm_DataManager.hpp"
#include <Teuchos_ParameterList.hpp>
#include <iostream>
#include <cmath>
#include <fstream>
#include <filesystem>
```

- `"Peridigm_MySimpleMaterial.hpp"`: The main header file for the Peridigm framework, containing base classes and function declarations.

- `"Peridigm_Field.hpp"`: Defines the field management system for handling material data.
- `"material_utilities.h"`: Contains utility functions specific to material handling.
- `"Peridigm_DataManager.hpp"`: Manages field data and provides access to them.
- `<Teuchos_ParameterList.hpp>`: Part of Trilinos, used to manage material parameters.
- `<iostream>`, `<cmath>`, `<fstream>`, `<filesystem>`: Standard libraries for input/output and file handling.

5.5.2 Constructor: `MySimpleMaterial::MySimpleMaterial`

The constructor of the material model is where the material properties are initialized and where field IDs are registered. This allows Peridigm to track the fields associated with the material and use them in the simulation.

```
MySimpleMaterial::MySimpleMaterial(const Teuchos::ParameterList& params)
    : Material(params),
    m_bulkModulus(0.0), m_shearModulus(0.0), m_density(0.0), m_horizon(0.0)
{
    m_density = params.get<double>("Density");

    PeridigmNS::FieldManager& fieldManager = PeridigmNS::FieldManager::self();

    m_modelCoordinatesFieldId = fieldManager.getFieldId(
      PeridigmField::NODE, PeridigmField::VECTOR,
      PeridigmField::CONSTANT, "Model_Coordinates");

    m_coordinatesFieldId = fieldManager.getFieldId(
      PeridigmField::NODE, PeridigmField::VECTOR,
      PeridigmField::TWO_STEP, "Coordinates");

    m_volumeFieldId = fieldManager.getFieldId(
      PeridigmField::ELEMENT, PeridigmField::SCALAR,
      PeridigmField::CONSTANT, "Volume");

    m_concentrationFieldId = fieldManager.getFieldId(PeridigmField::NODE, PeridigmField::SCALAR,
      PeridigmField::TWO_STEP, "Temperature");

    m_damageFieldId = fieldManager.getFieldId(
      PeridigmField::NODE, PeridigmField::SCALAR,
      PeridigmField::TWO_STEP, "Damage");

    m_fieldIds.push_back(m_modelCoordinatesFieldId);
    m_fieldIds.push_back(m_coordinatesFieldId);
    m_fieldIds.push_back(m_volumeFieldId);
    m_fieldIds.push_back(m_concentrationFieldId);
    m_fieldIds.push_back(m_damageFieldId);
}
```

- The constructor initializes the material properties like `m_bulkModulus`, `m_shearModulus`, `m_density`, and `m_horizon`, which are later populated with data.
- The `m_density` is fetched from the input `ParameterList`.
- `m_modelCoordinatesFieldId`, `m_coordinatesFieldId`, etc., are field IDs used to track the different material properties (like coordinates, volume, concentration, and damage).
- The field IDs are then added to the `m_fieldIds` vector, which allows Peridigm to access these fields during the simulation.

5.5.3 Pure Virtual Methods Implementation

Several pure virtual methods from the base `Material` class are implemented, such as `Name()`, `Density()`, `BulkModulus()`, and `ShearModulus()`.

```
std::string MySimpleMaterial::Name() const {
    return "MySimpleMaterial";
}

double MySimpleMaterial::Density() const {
    return m_density;
}

double MySimpleMaterial::BulkModulus() const {
    return m_bulkModulus;
}

double MySimpleMaterial::ShearModulus() const {
    return m_shearModulus;
}
```

These methods are used to retrieve the material properties. For example, `Density()` returns the density of the material, which was initialized in the constructor.

5.5.4 Field IDs Accessor: `FieldIds()`

The `FieldIds()` method returns the vector of field IDs that the material model uses. This allows Peridigm to access and update the data fields associated with the material during the simulation.

```
std::vector<int> MySimpleMaterial::FieldIds() const {
    return m_fieldIds;
}
```

5.5.5 Force Computation: `computeForce()`

The `computeForce()` method is where the material model computes the internal forces for each node. This method is called by Peridigm at each timestep to update the forces acting on each node based on the current state of the system.

```cpp
void MySimpleMaterial::computeForce(const double dt,
                                    const int numOwnedPoints,
                                    const int* ownedIDs,
                                    const int* neighborhoodList,
                                    PeridigmNS::DataManager& dataManager) const
{
    double *modelCoord = nullptr;
    double *currentCoord = nullptr;
    double *volume = nullptr;
    double *concentration = nullptr;
    double* damage = nullptr;

    dataManager.getData(m_modelCoordinatesFieldId, PeridigmField::STEP_NONE)->ExtractView(&modelCoord);
    dataManager.getData(m_coordinatesFieldId,      PeridigmField::STEP_NP1)->ExtractView(&currentCoord);
    dataManager.getData(m_volumeFieldId,           PeridigmField::STEP_NONE)->ExtractView(&volume);
    dataManager.getData(m_concentrationFieldId,    PeridigmField::STEP_NP1)->ExtractView(&concentration);
    dataManager.getData(m_damageFieldId, PeridigmField::STEP_NP1)->ExtractView(&damage);

    for (int iID = 0; iID < numOwnedPoints; ++iID) {
        int nodeID = ownedIDs[iID];
        double C = concentration[nodeID];

        // Example of processing based on node ID
        if (nodeID < 10000) {
            damage[nodeID] = 0.0; // Set damage value for nodes within range
        }

        // Export data to file (e.g., coordinates and damage)
        outFile << currentCoord[3*nodeID] << " " << currentCoord[3*nodeID+1] << " "
                << currentCoord[3*nodeID+2] << " " << damage[nodeID] << std::endl;
    }
}
```

- `dataManager.getData(...)`, `ExtractView(...)`: These methods are used to access and extract the views of the fields for coordinates, volume, concentration, and damage.
- `computeForce()` computes forces for each node by iterating over `numOwnedPoints`, processes them based on their state, and exports relevant data to a file.
- The method also handles file output for exporting the coordinates and damage values for nodes, providing a way to visualize and analyze the results.

5.5.6 Data Export

The PDHE code implemented into *Peridigm* includes functionality for exporting data to a text file. The code creates an output file and stores into the current working directory:

```cpp
fs::path currentDir = fs::current_path();
fs::path outputPath = currentDir / outputFileName;
```

The results are written to the file in the following format:

```
outFile << currentCoord[3*nodeID] << " " << currentCoord[3*nodeID+1] << " "  << currentCoord[3*nodeID+2] << "
    ↪ " << damage[nodeID] << std::endl;
```

This section of code ensures that the results are saved to a file for later analysis or visualization, including node positions and their respective damage values or any other desired data.

5.6 Registering `MySimpleMaterial` in Peridigm

To register your material model with Peridigm, you need to modify the appropriate Peridigm source files. This involves adding the material model to the list of available materials and making it accessible for simulations.

5.6.1 Modifying `Peridigm_MaterialFactory.cpp`

In the `Peridigm_MaterialFactory.cpp` file, you need to add an entry for your new material model. This allows Peridigm to recognize your material during runtime.

```
#include "MySimpleMaterial.hpp"

// In the MaterialFactory class
std::shared_ptr<PeridigmNS::Material>
MaterialFactory::createMaterial(const std::string& materialName,
                                const Teuchos::ParameterList& params)
{
  if(materialName == "MySimpleMaterial") {
    return std::make_shared<PeridigmNS::MySimpleMaterial>(params);
  }
  // Other materials...
  return nullptr;
}
```

This modification enables Peridigm to create an instance of `MySimpleMaterial` when the material name ”MySimpleMaterial” is specified in the input file.

5.6.2 Modifying the CMakeLists.txt File

To include our custom material model in the Peridigm build process, we must modify the `CMakeLists.txt` file. This ensures that the necessary files for our custom material model are compiled and linked correctly. First, we need to add the source file for your material model to the `CMakeLists.txt`. This involves including the file that contains the implementation of your material model, such as `MySimpleMaterial.cpp`.

Locate the section in the `CMakeLists.txt` file where the source files are listed. We need to add our material model source file to this list:

```
set(SOURCE_FILES
    src/Peridigm_Test.cpp
    src/Peridigm_Material.cpp
    src/MySimpleMaterial.cpp  % Add this line
    ...
)
```

This will ensure that the source file for your custom material model is included during the compilation process.

Next, we need to make sure that the header file for our material model is properly linked. Locate the section where the include directories are specified:

```
include_directories(
    ${CMAKE_SOURCE_DIR}/src
    ...
)
```

Ensure that the directory containing `MySimpleMaterial.hpp` is included. If its in the `src` directory, the above entry should already cover it. Otherwise, you can manually add the appropriate path.

5.6.3 Rebuild Peridigm

After registering the material model, the next step is to rebuild Peridigm to ensure that the new model is included in the compilation. Ensure that you have all the necessary dependencies and the Peridigm build environment set up correctly. This includes the Trilinos library and other required dependencies. Navigate to the Peridigm source directory and create a build directory. From there, run `cmake` to configure the build system.

```
cmake ../source/
make -j4
make install
```

This will configure the build system based on your system's environment and any modifications you made to the code The `-j4` option tells `make` to use 4 parallel cores for faster compilation. You can adjust the number based on the resources available on your system. After rebuilding Peridigm, one can test the new material model by running simulations that use it. Specify `MySimpleMaterial` as the material in the Peridigm input script.

```
Material = MySimpleMaterial
```

6 Program flowchart

There exists a distinct temporal disparity between the processes of hydrogen embrittlement and crack propagation [1]. Specifically, the rates of hydrogen embrittlement and diffusion are considerably slower than the rate of crack propagation. Consequently, if both processes are modeled using the same time step, the computational cost of the numerical simulation increases substantially. To address this issue, a staggered coupling approach is employed in the numerical simulations. In this approach, the forward Euler method is used to solve the PD diffusion equation (Eq. 16), while the adaptive dynamic relaxation method is utilized to solve the PD equilibrium equation (Eq. 15). The procedural flowchart for the PDHE model is presented in Fig. 5. The computational steps involved in the model are as follows:

1. **Pre-processing:** This step involves the initialization of material properties and discretization details, followed by the construction of the neighbor-point matrix. Additionally, the crack is initialized at this stage.

2. **Boundary condition application:** The simulation enters a loop over the loading steps (from $n = 1$ to N_t), during which the mechanical loading and hydrogen diffusion boundary conditions are applied.

3. **Hydrogen diffusion:** The diffusion loop begins, iterating through each time step (from $n_h = 1$ to N_h). In this loop, the hydrogen concentration at each material point is computed using the forward Euler method, considering all neighboring points.

4. **Calculation of PD forces:** In this phase, the forces corresponding to the peridynamic (PD) model are computed. This involves iterating over each material point and its neighbors, calculating the forces as specified by Eq. (5).

5. **Damage evaluation:** The critical bond stretch, adjusted for hydrogen coverage, is updated according to Eq. (13). This update determines whether bonds are broken, calculates the damage value at each material point, and modifies the saturated diffusion boundary condition along the crack path.

6. **Displacement computation:** The displacements of material points are then determined using the adaptive dynamic relaxation method. The time-stepping procedure continues until the final load step (N_t) is reached.

7. **Post-processing:** The final displacements, damage values, and other relevant data are output in text file format for further analysis.

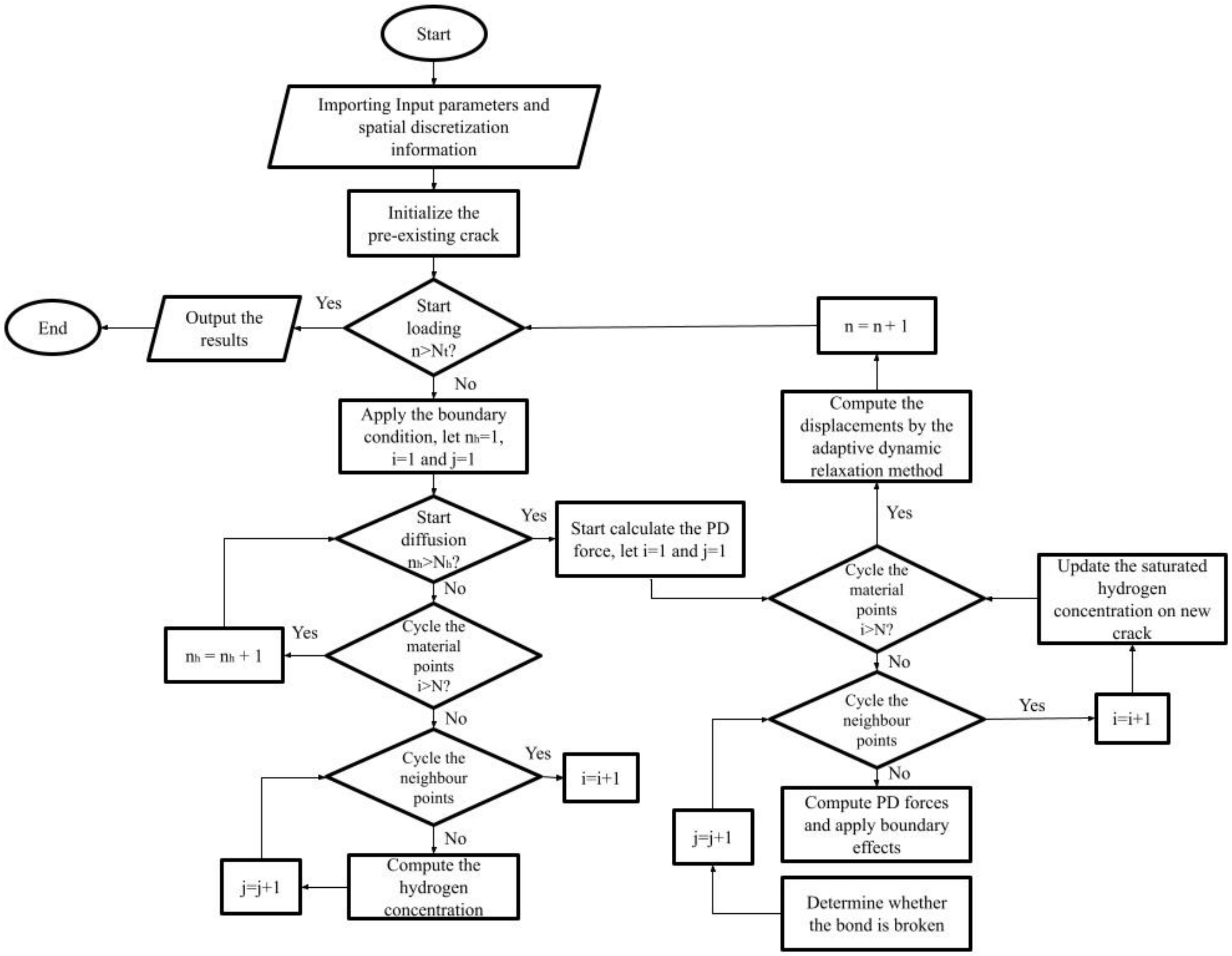

Start
Importing Input parameters and spatial discretization information
Initialize the pre-existing crack
Start loading n>Nt?
Yes
Output the results
End
No
Apply the boundary condition, let nh=1, i=1 and j=1
Start diffusion nh>Nh?
Yes
Start calculate the PD force, let i=1 and j=1
No
Cycle the material points i>N?
Yes
nh = nh + 1
No
Cycle the neighbour points
Yes
i=i+1
No
Compute the hydrogen concentration
j=j+1
n = n + 1
Compute the displacements by the adaptive dynamic relaxation method
Yes
Cycle the material points i>N?
Update the saturated hydrogen concentration on new crack
No
Cycle the neighbour points
Yes
i=i+1
No
Compute PD forces and apply boundary effects
j=j+1
Determine whether the bond is broken

Figure 5: Flowchart of the algorithm for PDHE model.

7 Usage of Chatbots

Interactive chatbots powered by large language models can streamline many aspects of implementing and documenting the PDHE model in Peridigm. Below are several key use cases where a chatbot came in handy.

7.1 Understanding Peridigm

A chatbot can serve as a realtime tutor on Peridigm's architecture and workflow. For example, it:

- Explained the core components of Peridigm (e.g. mesh representation, data manager, field evaluator).
- Clarified the sequence of operations from input deck parsing to solver execution.
- Provided the meaning of common keywords (e.g. `Material`, `Block`, `Solver`).
- Answered questions about existing Peridigm modules and their interfaces.

7.2 Understanding How to Include a Material Model into Peridigm

When integrating a new hydrogen embrittlement material model, a chatbot:

- Outlined the required class inheritance and registration macros for a custom material in Peridigm.
- Generated a template C++ header and source file with the correct namespace, constructor, and `computeForce` function signature.
- Tutored on linking the new model into the build system (CMakeLists) and updating YAML input deck syntax.
- Suggested unit tests to verify that the model responds correctly to prescribed displacements and hydrogenconcentration fields.

7.3 Literature Survey and Reference Management

For keeping abreast of recent developments, a chatbot:

- Summarized recent journal articles on hydrogen embrittlement and peridynamic modeling.
- Extracted key parameter values (diffusivity, critical stretch) from publications.
- Suggested BibTeX entries and helped to maintain a consistent reference database.

- Generated briefly annotated bibliographies to guide further reading.
- Helped generate suitable LaTex formats of complex equations, which was further used in reports.

8 Tests for Verification

To ensure the accuracy, robustness, and predictive capability of the proposed peridynamic hydrogen-embrittlement (PDHE) model, a comprehensive suite of numerical verification tests is performed. These include unit tests and boundary-condition tests to validate individual code components, mesh-convergence studies to confirm spatial discretization independence, decoupling analyses of mechanical and diffusion kinetics, bond-damage analyses, time-step sensitivity analyses to assess the stability of the explicit solver, and crack-path prediction in mode I crack-propagation simulation under hydrogen loading. Together, these tests demonstrate that the implementation reliably captures both mechanical response and hydrogen-driven degradation across a range of loading and diffusion.

8.1 Unit testing

Unit testing underpins reliable software development by isolating and verifying individual components. For the Peridynamic Hydrogen Embrittlement (PDHE) model integrated within the Peridigm framework, systematic tests of mathematical routines ensure that vector operations, stiffness corrections, and force scaling behave as intended under controlled inputs. This report presents the framework, execution, and outcomes of these targeted tests, aiding early detection of implementation errors and bolstering confidence ahead of fullscale simulations.

All tests are executed via the `Unit_test_PDHE.cpp` file, which writes detailed results to `Unit_test_log.txt`. You must keep this `.cpp` file in the same folder path of other files which are being tested. Compilation and runtime commands are standardized as follows:

Compilation:

```
g++ -std=c++17 -O2 Unit_test_PDHE.cpp \
    PDHE_element_routine.cxx PDHE_material_routine.cxx PDHE_boundary_effects.cxx \
    -o Unit_test_PDHE -lstdc++fs
```

Execution:

```
./Unit_test_PDHE
```

The text file is represented in the following manner:

```
//---------------------------------------------------------//
        Unit test for formation of xi vector
//---------------------------------------------------------//
xi vector:
```

```
x-component = 2
y-component = 2
--------------------------------------------------------------------------
... (additional entries)
```

8.1.1 Test Case 1: Displacement Vector (ξ)

- Aim: Compute the relative position between $P_i = [1.0, 2.0]$ and $P_j = [3.0, 4.0]$.
- Setup: Coordinates passed to the `xi_vec` function.
- Expected: $\boldsymbol{\xi} = [2.0, 2.0]$.
- Observed: Log reports [2.0 2.0], exact match.

8.1.2 Test Case 2: Vector Norm ($|\xi|$)

- Aim: Calculate the Euclidean length of $\boldsymbol{\xi} = [2.0,\ 2.0]$.
- Expected: $\sqrt{8} \approx 2.828427$.
- Observed: 2.82843, within numerical tolerance.

8.1.3 Test Case 3: Secondary Displacement (η)

- Aim: Verify `eta_vec([1,2],[3,4])` yields $[2.0, 2.0]$.
- Observed: Matches expectation.

8.1.4 Test Case 4: Surface Correction

- Aim: Adjust stiffness k_n, k_t based on volumes $V_i = 4, V_j = 5$ and horizon $\delta = 3$.
- Expected: $k_n = 37.6991$, $k_t = 75.3982$.
- Observed: Log confirms $k_n = 37.6991$, $k_t = 75.3982$.

8.1.5 Test Case 5: Volume Correction

- Aim: Scale force vector $[1.0, 2.0]$ by a volume correction factor having $\delta = 3, h = 4, |\boldsymbol{\xi}| \approx 2.8284$ as inputs.
- Expected: Factor ≈ 0.4359, resulting force $[0.436, 0.872]$.
- Observed: $(0.43589, 0.87178)$, agreement within tolerance.

8.1.6 Test Case 6: Generic Vector Norm

- Aim: Norm of $[1.0, 2.0]$.
- Expected: $\sqrt{5} \approx 2.23607$.
- Observed: 2.23607.

8.1.7 Test Case 7: Vector Addition

- Aim: Sum $[1.0, 2.0] + [3.0, 4.0]$.
- Expected/Observed: $[4.0, 6.0]$.

Directory Layout

```
/project-root/
 Unit_test_PDHE              # Driver executable
 Unit_test_PDHE.cpp          # Test harness
 PDHE_element_routine.cxx    # Vector operations
 PDHE_material_routine.cxx   # Material parameter routines
 PDHE_boundary_effects.cxx   # Surface/volume corrections
 Unit_test_log.txt           # Log of outcomes
```

8.2 Boundary condition test

This sub-section documents the procedure and results for verifying prescribed boundary conditions in the PDHE model using the Peridigm code. A simple rectangular domain with discrete node sets is used to confirm that displacement and hydrogen concentration constraints are correctly applied and reported. The steps include modifying the YAML input, populating nodeset files, executing Peridigm, and interpreting the generated log output.

The aim of this test is to confirm that prescribed Dirichlet boundary conditions for displacement and hydrogen concentration are enforced and that the PDHE `computeForce` routine reports the applied values.

Test procedure includes:

- Enable the `Boundary condition test` flag to `true` in the Peridigm YAML input.
- Provide nodeset text file names under `File name for displacement in +ve x/y-direction` and `File name for concentration` for the displacement and concentration BCs.
- Verify that during execution, Peridigm outputs the node IDs and their corresponding imposed values.

Edited PDHE material block in the YAML file should look as follows:

```
Materials:
  MyMaterial:
    Material Model: "PDHE"
    # ... other parameters ...
    File name for displacement in +ve x/y-direction: "nodeset\_top.txt"
    File name for concentration: "nodeset\_concentration.txt"
    Boundary condition test: true
```

Important: Do not place these nodeset file names under any other parameter. Only the two fields above control the BC test behavior. When you run Peridigm with this YAML, it will attempt to open `nodeset_top.txt` and `nodeset_concentration.txt`. If the files are missing or misnamed, you will see warnings of the form:

```
Error: Could not open node set file: <file_name>
```

These messages can be safely ignored as long as the correct files are provided in the YAML.

Run the program:

```
Peridigm Boundary_condition_test.yaml
```

Sample Output: Excerpt from the generated `Output.txt`:

```
#Header
nodeID value

Check for Concentration boundary condition in mol/m^2
1 2.65e-05
2 2.65e-05

Check for Displacement boundary condition in m
3 2e-08
... (additional entries)
```

This confirms that node IDs 1 and 2 received the correct concentration BC, and node 3 received the displacement BC as prescribed.

8.3 Mesh convergence testing

A systematic mesh convergence investigation was carried out for the PDHE model integrated into Peridigm, assessing both mechanical and diffusion responses. A 200×100 mm rectangular block was discretized with progressively finer node counts ($4,000$ to $12,000$). Under uniaxial tension and an imposed surface hydrogen concentration, mean vertical displacements and mean hydrogen concentrations were recorded at load step 3000 (with 20 diffusion sub-steps). Nodes with corresponding values of the order below 10^{-10} were excluded from averaging. By examining these metrics across five different mesh densities, we identify the point of diminishing returns where further refinement yields negligible changes. Convergence trends indicate that simulations with $10,000$ to $12,000$ nodes yield stable results, guiding the optimal mesh selection for subsequent studies.

8.3.1 Model Setup

Geometry: Rectangular block of size $200 \times 100\,\mathrm{mm}$.

Boundary Conditions:
Mechanical: Uniform vertical displacement imposed on both top and bottom surfaces (uniaxial tension).
Diffusion: Constant hydrogen concentration prescribed on the leftmost surface; diffusion allowed into the bulk.

Time Stepping: 3 000 mechanical load steps, each with 20 hydrogen diffusion substeps.

Mesh Densities: $4,000; 6,000; 8,000; 10,000; 12,000$ nodes.

Data Processing: At load step 3000, extract displacements and concentrations. Exclude nodes whose values are below 1×10^{-10} from mean calculations to avoid floatingpoint noise.

8.3.2 Displacement Convergence

Figure 6 shows that the mean displacement increases from approximately $9.75 \times 10^{-6}\,\mathrm{m}$ at $4,000$ nodes to about $1.005 \times 10^{-5}\,\mathrm{m}$ at $8,000$ nodes. Beyond $8,000$ nodes, the curve flattens, with only minor variation ($< 0.5\%$) between $10,000$ and $12,000$ nodes, indicating numerical convergence of the mechanical response.

8.3.3 Hydrogen Concentration Convergence

Figure 7 illustrates the mean concentration trend. Starting at $1.36 \times 10^{-5}\,\mathrm{mol/m^2}$ for $4,000$ nodes, the value rises to $1.77 \times 10^{-5}\,\mathrm{mol/m^2}$ at $8,000$ nodes and then plateaus near $1.88 \times 10^{-5}\,\mathrm{mol/m^2}$ for the finest meshes. The relative change between $10,000$ and $12,000$ nodes is under 0.2%, confirming convergence of the diffusion field.

8.3.4 Discussion

- **Coarse Mesh Behavior ($4,000$–$6,000$ nodes):** Both displacement and concentration show significant sensitivity, with under-prediction due to insufficient resolution of gradients near boundaries.

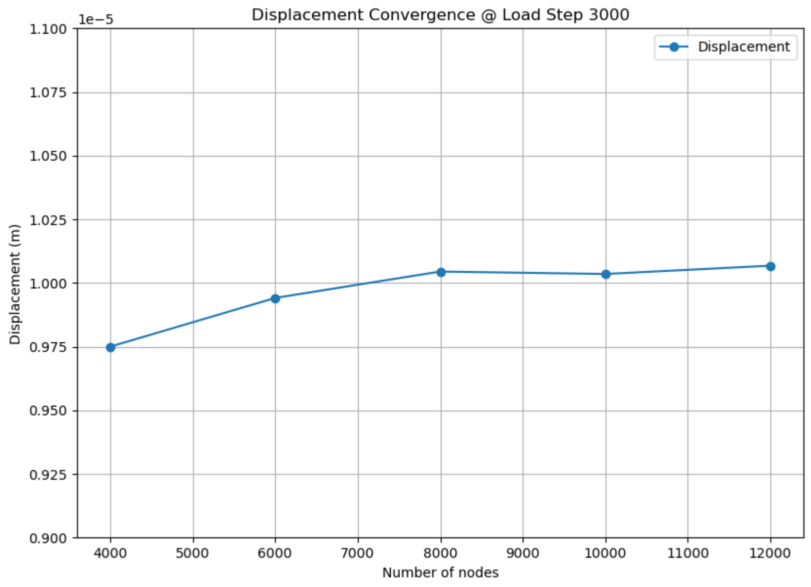


Figure 6: Mesh convergence of mean vertical displacement at load step 3000.

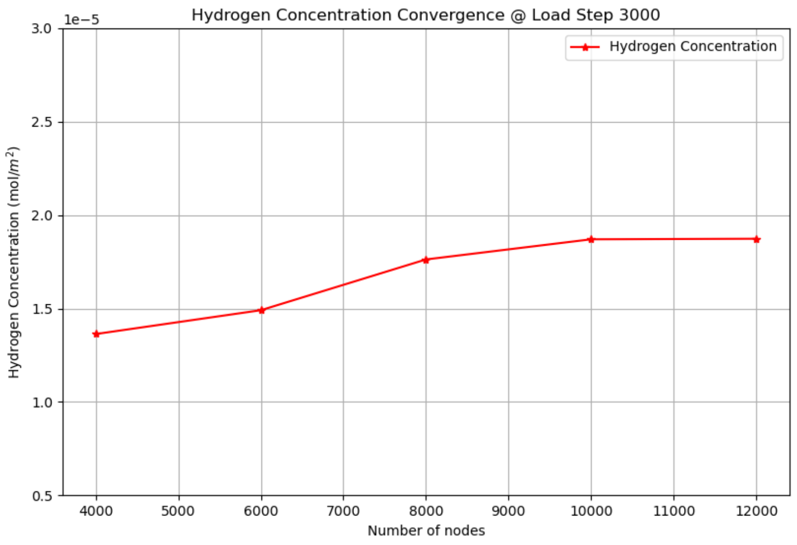


Figure 7: Mesh convergence of mean hydrogen concentration at load step 3000.

- **Intermediate Mesh ($8,000$ nodes):** A Major portion of convergence is achieved; mechanical and diffusion fields exhibit diminishing changes.
- **Fine Mesh ($10,000$–$12,000$ nodes):** Response curves are virtually identical, indicating that mesh-induced errors are below engineering tolerances.

The criterion of a less than 0.5% change in key outputs guides the selection of an efficient yet accurate mesh. For the PDHE simulations in a 200×100 mm block, $10,000$ nodes strike a balance between computational cost and numerical fidelity. This mesh convergence study demonstrates that both the mean displacement and the hydrogen concentration reach stable values once the mesh exceeds approximately $8,000$ nodes. To ensure reliable PDHE simulation results without excessive computational burden, it is recommended to have a mesh density of at least $10,000$ nodes for the given problem geometry and loading protocol.

8.4 Decoupling analyses Mechanical and Difusion kinetics

To verify the independent implementation of mechanical and diffusion processes in the Peridynamic Hydrogen Embrittlement (PDHE) model, two separate simulations were conducted on a 200×100 mm rectangular domain. In the mechanical-only test, uniaxial tension was applied via a prescribed displacement history up to 3000 load steps. In the diffusion-only test, a constant hydrogen concentration was imposed on the left face and allowed to diffuse for 20 sub-steps per load step. Contour plots of vertical displacement and hydrogen concentration at the final load step illustrate the decoupled response fields. Detailed qualitative observations are discussed for each case.
Test Procedure:

8.4.1 Geometry and Discretization

The computational domain is a rectangle of dimensions 200×100 mm, discretized uniformly (mesh density as per standard PDHE settings).

8.4.2 Mechanical-Only Simulation

- **Active Physics:** Peridynamic mechanical balance (diffusion turned off).
- **Boundary Conditions:**
  - *Top and Bottom Surfaces:* Prescribed vertical displacement $u(n)$ following the piece-wise linear loading history shown in Figure 10.
  - *Left and Right Surfaces:* Homogeneous zero traction (free).
- **Loading History:** Ramp from $u = 0$ to 0.02 mm over 1000 steps, held constant to 3000 steps.
- **Output:** Vertical displacement field at load step 3000.

8.4.3 Mechanical Response

Figure 8 shows the vertical displacement field after 3000 steps. Several key observations were drawn as follows:

- **Linear Gradient:** Displacement increases almost linearly from bottom (blue, $\approx -2 \times 10^{-5}$ m) to top (red, $\approx +2 \times 10^{-5}$ m), indicating uniform strain in the domain.
- **Edge Effects:** Slight flattening of contours near the vertical boundaries suggests minor peridynamic horizon effects at free surfaces.
- **Symmetry:** The field is symmetric about the mid-height line, confirming consistent loading on top and bottom.

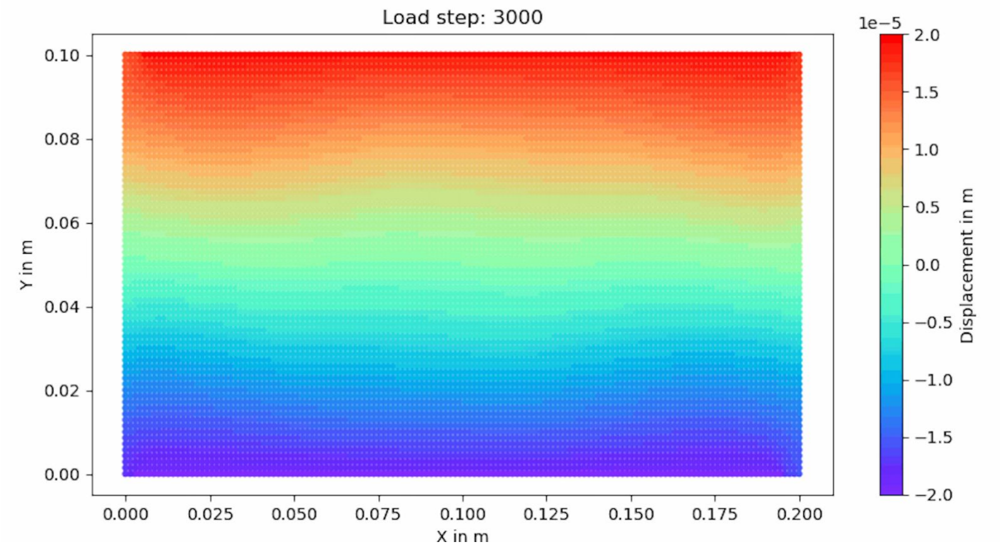


Figure 8: Contour of vertical displacement (m) at load step 3000 in the mechanicalonly simulation.

8.4.4 Diffusion-Only Simulation

- **Active Physics:** Hydrogen diffusion equation (with mechanics turned off).
- **Boundary Conditions:**
  - *Left Face:* Fixed hydrogen concentration $C = C_{\text{sat}} = 2.65 \times 10^{-5}\,\text{mol/m}^2$ (prescribed Dirichlet).
  - *Other Faces:* Insulated (zero flux).
- **Time Stepping:** 3000 loadstep equivalents, each with 20 diffusion substeps.
- **Output:** The concentration field at the end of load step 3000.

8.4.5 Diffusion Response

Figure 9 presents the hydrogen concentration field at step 3000. Further observations were made:

- **Exponential Decay:** Concentration drops smoothly from the left face (red, $2.65 \times 10^{-5}\,\mathrm{mol/m^2}$) into the interior (blue, $\approx 1.5 \times 10^{-5}\,\mathrm{mol/m^2}$), reflecting the diffusion-governed gradient.
- **Uniform Vertical Profile:** Along any horizontal line (constant y), the concentration is nearly uniform, indicating negligible vertical flux under the given conditions.

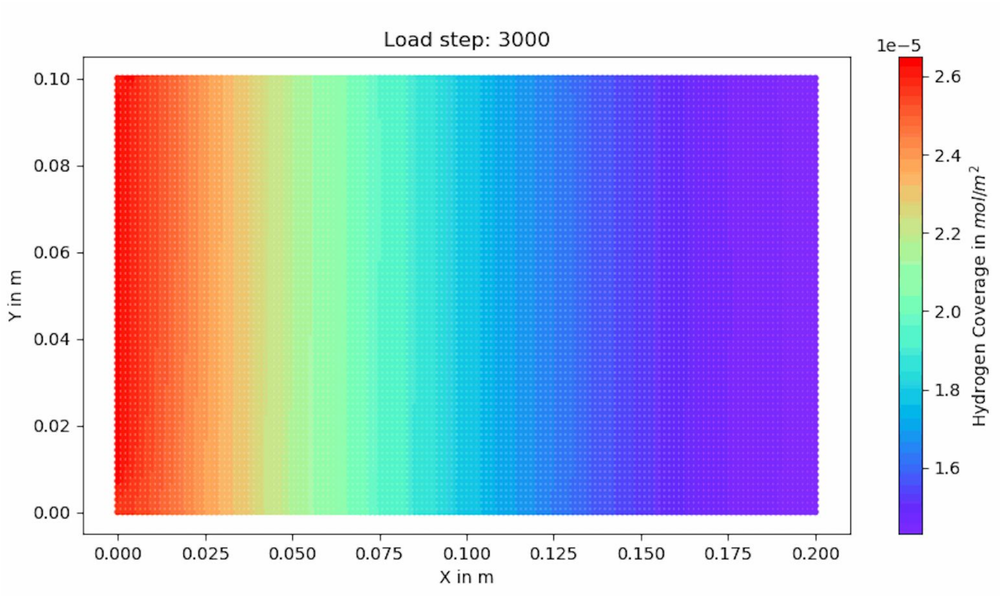


Figure 9: Contour of hydrogen concentration ($\mathrm{mol/m^2}$).

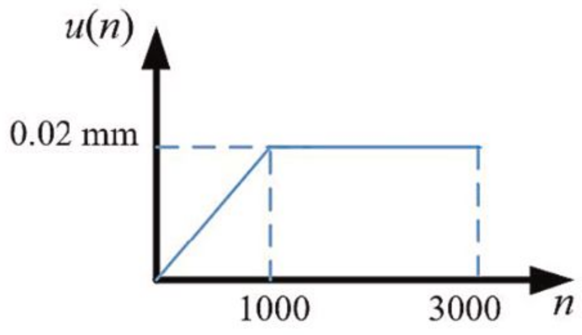


Figure 10: Prescribed displacement history.

Conclusions:

1. The mechanical-only test produces an almost perfectly linear displacement field, validating the peridynamic force calculation under uniaxial tension.
2. Minor edge perturbations confirm that horizon truncation is well controlled and does not compromise overall accuracy.
3. The diffusion-only test yields a smooth, physically meaningful gradient, with a clearly identifiable boundary layer.
4. Vertical uniformity of concentration profiles demonstrates correct implementation of zero-flux side boundaries.

Together, these decoupled simulations affirm that the PDHE models mechanical and diffusion kernels operate correctly in isolation, paving the way for fully coupled hydrogen embrittlement studies.

8.5 Bond failure testing

A comparative bond-failure study was conducted for two numerical domains under identical uniaxial loading protocols: one without hydrogen (pure mechanical response) and one with hydrogen diffusion activated and an initial crack. The mean damage, averaged over all bonds in each domain, is tracked up to 3000 load steps. As expected, the hydrogen-free domain remains undamaged (zero mean), while the hydrogenated domain with a crack exhibits progressive damage accumulation emanating from a pre-existing crack. The results, summarized in Figure 11, demonstrate the accelerating effect of hydrogen on bond degradation in the cracked region.

8.5.1 Test Setup

Geometry: A 200×100 mm rectangular block discretized with between 10,000 and 12,000 nodes.

Initial Crack (Domain with hydrogen only): A 50 mm long notch centered on the left face, represented by broken bonds along that line.

Loading: Uniaxial tension applied via prescribed displacement history (ramp to 0.02 mm over 1000 steps, held until 3000).

Diffusion: Constant hydrogen concentration prescribed on the left face; 20 diffusion substeps per mechanical load step.

Damage Metric: At each step, bond damage is computed per node and averaged over the entire domain.

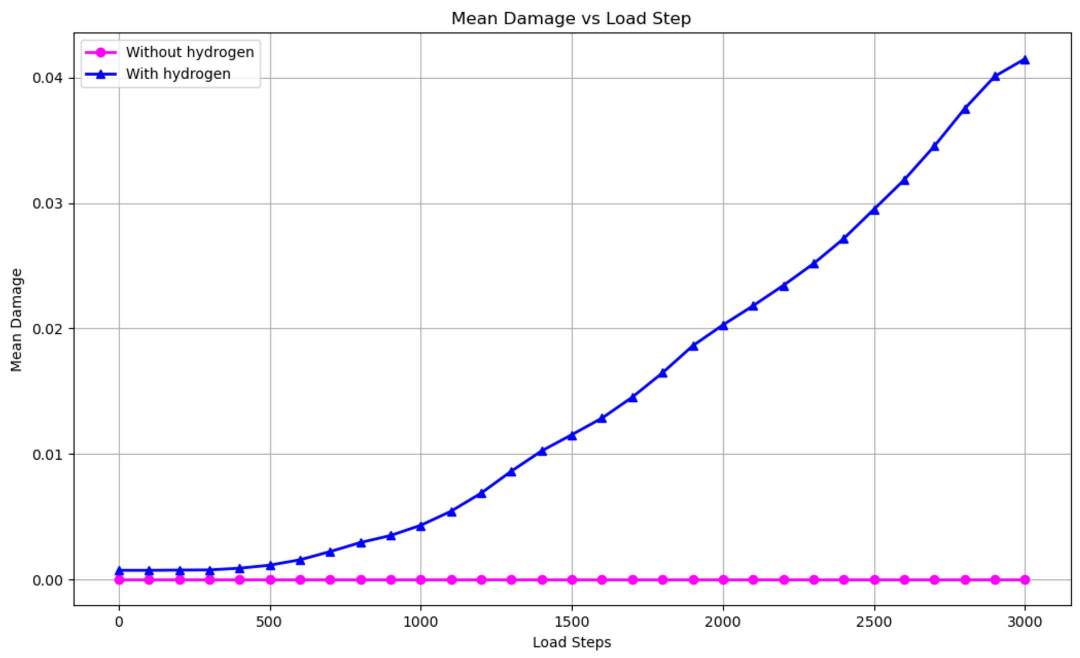


Figure 11: Mean damage v/s load steps for the hydrogenfree domain and the cracked, hydrogen-exposed domain.

8.5.2 Graph Behavior

8.5.2.1 Hydrogen Free Domain: The magenta curve remains pinned at zero damage throughout all 3000 load steps. In the absence of hydrogen or initial cracks, purely mechanical bond forces never exceed failure thresholds under the prescribed displacement history.

8.5.2.2 Cracked, Hydrogenated Domain: The blue curve starts near zero at step 0 and remains nearly flat for the first few hundred steps, indicating an incubation period during which hydrogen diffuses into the region around the pre-existing 50 mm crack. Once diffusion sufficiently embrittles the crack tip bonds (around 400-500 steps), the mean damage begins a gradual upturn. This onset reflects initial bond failures nucleating at the crack front.

As loading continues, damage growth accelerates markedly. Between 1000 and 2000 steps, the slope increases, as embrittled bonds along the crack propagate additional microfailures further into the bulk. In the final third of the simulation (20003000 steps), damage growth is almost exponential, reaching a mean of ≈ 0.04 at step 3000 indicating that crack has propagated till the end of the domain.

8.5.3 Key Observations

- **CrackDriven Initiation:** Early damage localizes around the 50 mm crack on the left surface, where hydrogen concentration is highest.

- **Incubation Period:** The flat region up to ~ 500 steps shows the time needed for hydrogen to embrittle bonds adjacent to the crack.
- **Accelerated Damage Propagation:** Past 1000 steps, damage spreads rapidly from the crack tip into the interior.
- **Averaged Magnitude:** Because damage is averaged over the whole mesh-including undamaged areas-the mean remains low (a few percent) even as local bond failures become significant near the crack.

Conclusions:
This study confirms that an initial crack, when coupled with hydrogen diffusion, dramatically accelerates bond failure under uniaxial tension. The stark contrast between the undamaged, hydrogen-free domain and the rapidly degrading cracked, hydrogenated domain underscores the critical interplay of pre-existing flaws and embrittlement in PDHE simulations.

8.6 Time-step sensitivity testing

This work presents a systematic convergence study of the explicit Euler time integration scheme as applied to hydrogen diffusion in the PDHE model within Peridigm. We use a range of time step sizes determined by the CFL stability criterion and compute the L_2 error against a highly resolved reference solution. The observed convergence rate of $p \approx 1.32$ falls below the theoretical first-order accuracy.

8.6.1 Background and Objectives

The explicit Euler method is formally first order accurate in time. In the context of peridynamic hydrogen diffusion, stable time step sizes Δt were computed via

$$\Delta t \leq \Delta x \cdot \sqrt{\frac{\rho_{\text{material}}}{E_{\max}}}, \tag{30}$$

Here $E_{\max}$ is the maximum component of the elastic stiffness matrix used to approximate the maximum possible wave speed, ρ_{material} is the density of the material. Eq.(30) is the time step size employed for hydrogen diffusion. The goal of this study is to:

1. Quantify the empirical convergence order by measuring the error

   $$e(\Delta t) = \| C_{\Delta t} - C_{\text{ref}} \|_{L_2} \tag{31}$$

   at final time T, across four decreasing time steps.
2. Compare the observed rate p against the expected $p = 1$.
3. Investigate any anomalies, specifically a slight kink in the error curve.

8.6.2 Methodology

- **Domain and Parameters:** Domain and other parameters pertaining to diffusion kinetics remain the same as in subsection **??**.
- **Reference Solution:** Computed with an extremely small time step (Δt_{ref}) to render temporal discretization errors negligible.
- **Time Steps Tested:** Four values of Δt spanning roughly 1.5×10^{-9} to 6×10^{-9} s, all satisfying the CFL bound.
- **Error Norm:** The discrete L_2 norm over all nodes at time T.
- **Order Estimation:** Fit the relation on a log-log scale to extract the slope p.

$$e(\Delta t) \approx C\,(\Delta t)^p \tag{32}$$

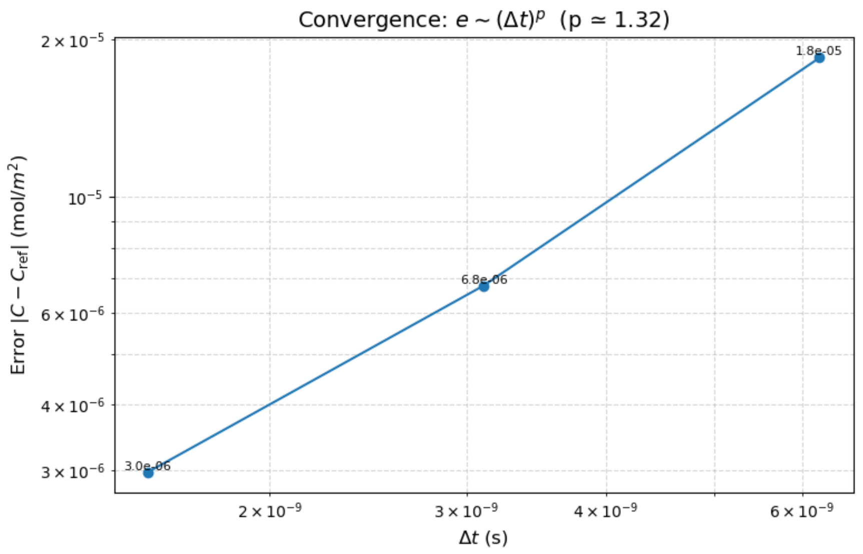


Figure 12: Error $e = \|C - C_{\text{ref}}\|_{L_2}$ v/s time-step size Δt (log-log). The best-fit slope gives $p \approx 1.32$.

8.6.3 Observed Convergence Rate

Regression on the four data points in Figure 12 yields

$$e(\Delta t) \propto (\Delta t)^{1.32}\,.$$

While this confirms that the error decreases as Δt is reduced, the rate $p = 1.32$ exceeds the formal $p = 1$ of the Euler scheme.

8.7 Crack path prediction and pattern validation

This section presents the PDHE peridynamic implementation by predicting the desired damage and hydrogen concentration fields in a compact tension like geometry. A rectangular plate (200×100 mm) with a 50 mm pre-crack on the left face is subjected to uniaxial tension and a constant surface hydrogen concentration of 2.65×10^{-5} mol/m^2. Mechanical loading is ramped to 0.02 mm over 1000 steps and held until 3000 steps, with 20 diffusion substeps per mechanical step. Damage and hydrogen-coverage field contours at four key load steps demonstrate crack initiation and propagation, confirming model fidelity and the role of hydrogen diffusion in crack propagation.

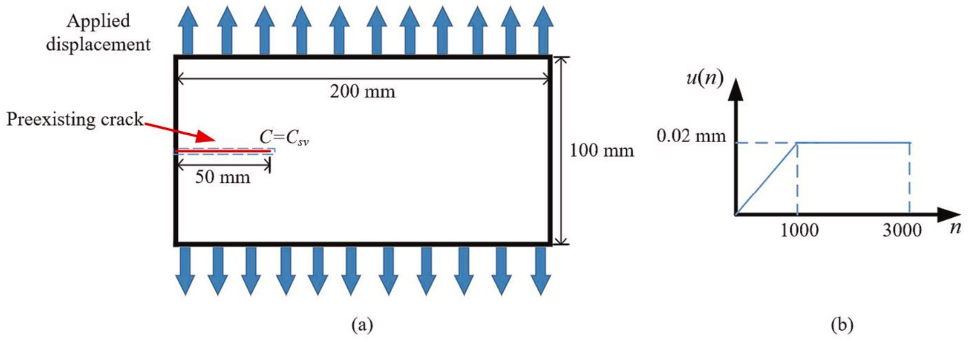


Figure 13: (a) Loading and diffusion boundary conditions: prescribed vertical displacement on top/bottom, hydrogen concentration $C = C_{\text{sv}}$ on the 50 mm pre-crack. (b) Displacement history $u(n)$: linear ramp to 0.02 mm over 1000 steps, held to 3000.
[1]

Table 4: Material and diffusion parameters used in validation.

Parameter	Unit	Value
Youngs modulus, E	GPa	207
Density, ρ	kg/m^3	8.0
Poisson's ratio, ν	-	0.29
Fracture toughness, K_{IC}	MPam$^{0.5}$	58.4
Grain-boundary diffusion, D_{gb}	m^2/s	0.84×10^{-9}
Saturated hydrogen conc., C_{sv}	mol/m^2	2.65×10^{-5}

8.7.1 Damage Contours and Crack Propagation

Figure 14 shows the evolution of the damage field at four representative load steps: 1300, 1800, 2300, and 3000.

Load Step 1300: At step 1300, localized damage initiates at the crack tip. A small elliptical zone of moderate damage (green-yellow, damage ≈ 0.4) extends a few millimeters ahead of the pre-existing notch. The rest of the plate remains pristine (purple).

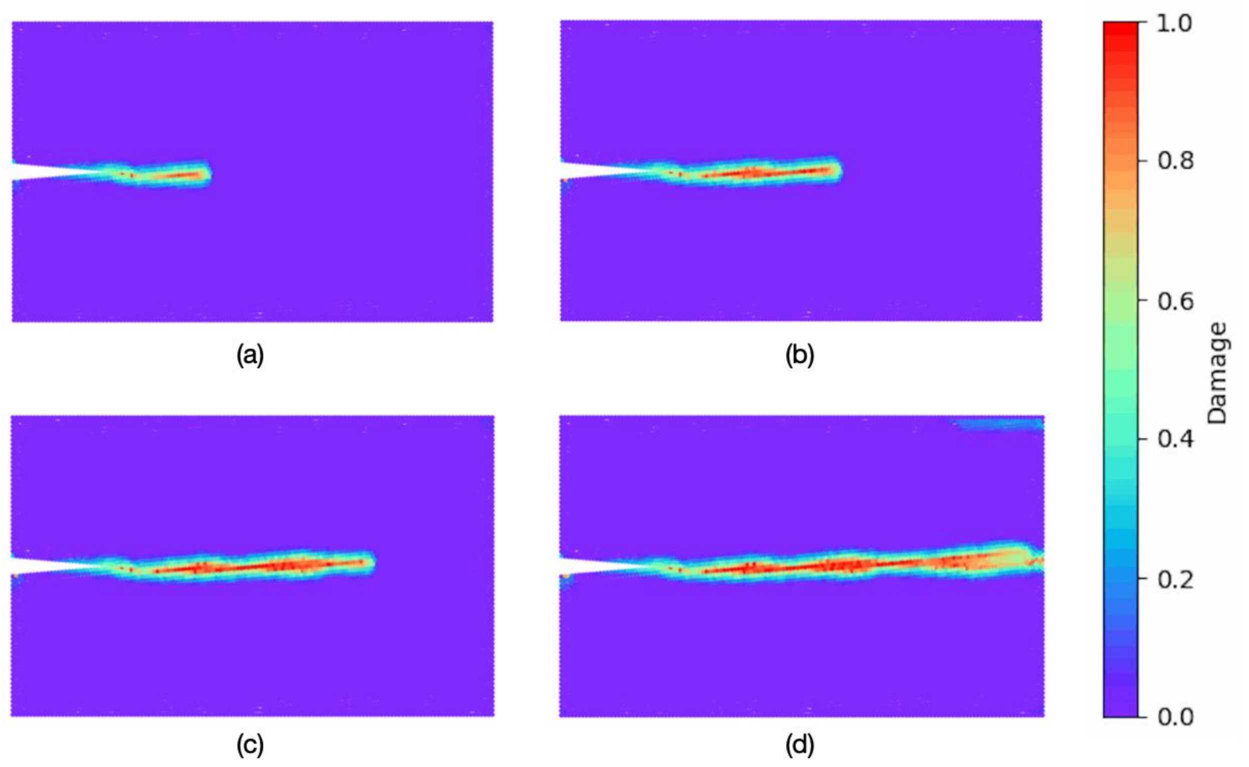


Figure 14: Crack path for Compact tension test: (a)Load step 1300, (b)Load step 1800, (c)Load step 2300, (d)Load step 3000

Load Step 1800: By step 1800, the damage region has lengthened significantly along the crack axis. The central band turns orange (damage ≈ 0.6), indicating progressive bond-failure under combined mechanical opening and hydrogen embrittlement. The damage profile remains narrow, highlighting a predominantly mode I crack propagation.

Load Step 2300: At load step 2300, the damaged zone advances further into the bulk, reaching toward the mid-plane.

Load Step 3000: By the final step, the primary crack has propagated approximately 100 mm from the left face toward the right boundary. A continuous red band indicates fully broken bonds (damage ≈ 1), while the zone behind the tip remains yellow-orange. The direction of propagation stays aligned with the initial crack orientation, confirming stable crack growth under uniaxial tension.

8.7.1.1 Discussion

- **Crack Initiation:** Damage nucleates precisely at the hydrogen-saturated crack tip, validating the coupling of diffusion and bond-failure.

- **Propagation Path:** The crack extends horizontally, following the pre-crack direction, with no significant deviation–consistent with mode I fracture mechanics.
- **Hydrogen Effect:** The early onset and accelerated damage growth reflect hydrogen-induced reduction in critical energy release rate.
- **Damage Distribution:** The narrow, high-damage band indicates peridynamic nonlocal interactions are sufficiently localized to reproduce sharp crack profiles.

8.7.2 Hydrogen Coverage Contours at Key Load Steps

Figure 15 presents the hydrogen coverage field (C in mol/m^2) at four representative mechanical load steps: 1300, 1800, 2600, and 3000. In all cases, a saturated concentration $C_{\text{sv}} = 2.65 \times 10^{-5}$ mol/m^2 is maintained along the 50 mm pre-crack on the left face, with 20 diffusion sub-steps per mechanical increment.

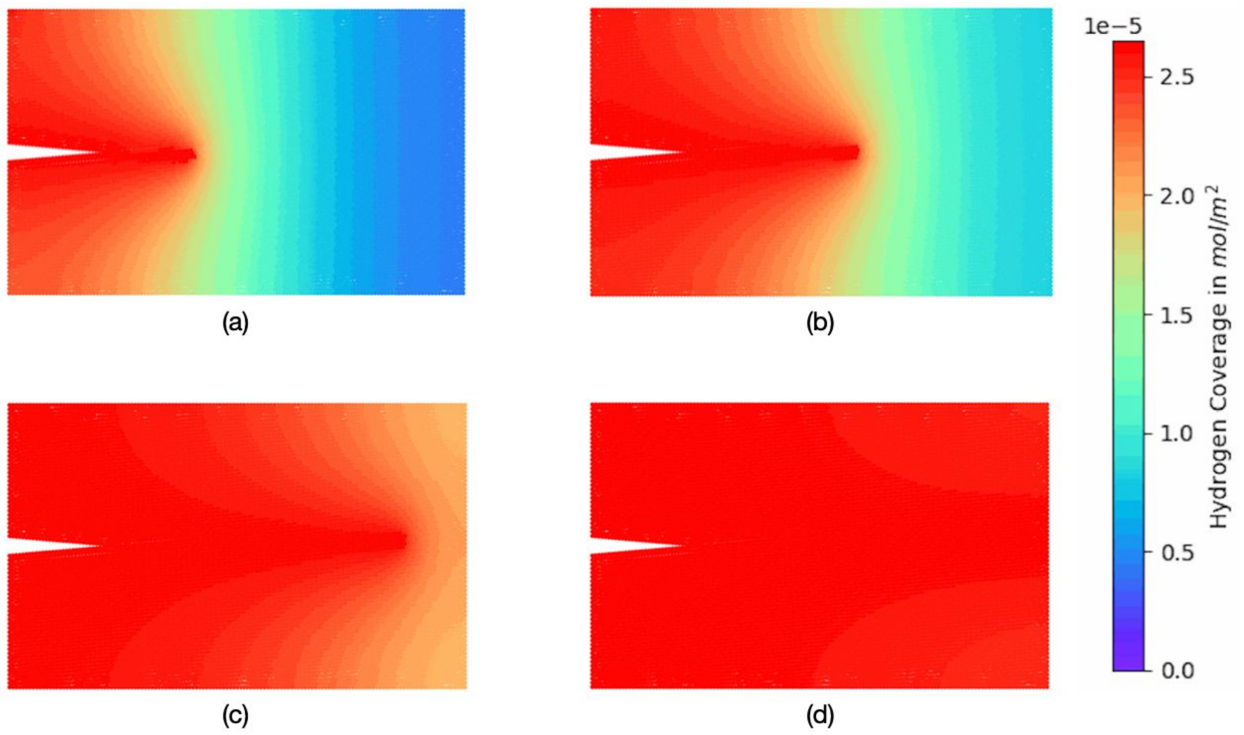


Figure 15: Hydrogen coverage at load steps (a) 1300, (b) 1800, (c) 2600, and (d) 3000. The colorbar at right spans 02.7 $\times 10^{-5}$ mol/m^2.

Load Step 1300 At step 1300, the coverage is highest (deep red) immediately adjacent to the pre-existing crack tip, forming a narrow plume extending about 510 mm into the material. Behind the tip, concentration falls off smoothly (orange to green) toward the right boundary, indicating the early diffusion front. The vertical uniformity of the red zone shows negligible vertical gradients, as expected under one-dimensional diffusion along the crack face.

Load Step 1800 By step 1800, the high-concentration region has thickened: the red plume now spans roughly 15 mm from the left face. The gradient zone (yellow to light blue) has moved further into the bulk, and the curvature of the contour lines near the crack tip becomes slightly convex, reflecting increased lateral diffusion into neighboring peridynamic horizons.

Load Step 2600 At step 2600, nearly the entire domain up to mid-width is saturated (predominantly red). Only a thin band (12 mm) of lower concentration remains near the right edge. The plume retains a pointed tip shape centered on the crack, with faint asymmetryslightly more penetration above the crack axisdue to local variations in bondbreak evolution affecting local diffusivity.

Load Step 3000 By the final step, hydrogen coverage is essentially uniform (solid red) across nearly the entire plate, with only the extreme right edge (last 5 mm) showing a hint of yellow. The crack tip region is fully saturated, and vertical symmetry is restored, indicating steady state diffusion dominance over mechanical influences.

8.7.2.1 Discussions

- **Spatial Gradient Evolution:** The smooth, continuous shift from green/blue at early steps to red at later steps confirms a stable, monotonic diffusion front.
- **TipCentric Saturation:** At all times, the highest coverage remains tightly localized around the crack tip, demonstrating correct Dirichlet enforcement.
- **Vertical Uniformity:** Minimal variation along the vertical direction indicates accurate implementation of zero-flux conditions on top and bottom boundaries.
- **Horizon Influence:** The slight bulging of contours at intermediate steps (18002600) reflects the nonlocal peridynamic coupling, which broadens the diffusion front beyond classical Fickian predictions.

Conclusions: These discussions concludes that PDHE model accurately captures the sequence of crack initiation at a hydrogenated notch, subsequent stable crack propagation along the prescribed direction, and eventual bond failure field. The detailed damage contours at steps 1300 to 3000 confirm both the mechanical and diffusion kernels operate as intended.

9 Computation Time Study

We instrumented the `computeForce()` driver in our Peridynamics Hydrogen Embrittlement (PDHE) code to measure, per loadstep, three disjoint timing components: the hydrogen-diffusion subcycle, the mechanical (PD) force loop, and the nodal displacement update. This section summarizes the timings, identifies the bottleneck, and suggests avenues for performance improvement.

9.1 Methodology

We inserted a RAII timer (`ScopedTimer`) into `computeForce()`, partitioning each loadstep into three nonoverlapping blocks:

- **Hydrogen subcycle:** the $N_h$step loop calling `element_routine_hydrogen()`.
- **Mechanical PD:** the loop over owned points calling `element_routine_PD()`.
- **Displacement:** the coordinate update and I/O logging.

At the end of `computeForce()`, all accumulated times are printed. The timings printed below the summary are the timings logged by Peridigm which is not the project's scope. We run on a uniform 2D rectangular domain with $N_{\text{points}} = 10,000$, $N_t = 3000$ load-steps, $N_h = 20$ diffusion sub-steps.

9.2 Results

Table 5: Timing breakdown at 3000 load-steps.

Routine	**Time (s)**	**Fraction (%)**
Loadstep loop (total)	910.644	100.0
Hydrogen subcycle	673.067	73.91
Mechanical PD	231.322	25.40
Displacement update	6.25482	0.69

- **Bottleneck:** The hydrogen diffusion subcycle (`element_routine_hydrogen`) consumes virtually most of the load-step time (73.91%), even though its loops are simple explicit arithmetic.
- **Mechanical cost:** The peridynamic force loop is substantial (25.40% of total), but its neighbor-list traversal and bondforce computations are well-structured for parallelism and vectorization.
- **Displacement cost:** Nodal updates and I/O are minor (0.69%) but still nonnegligible when scaled to millions of material points.

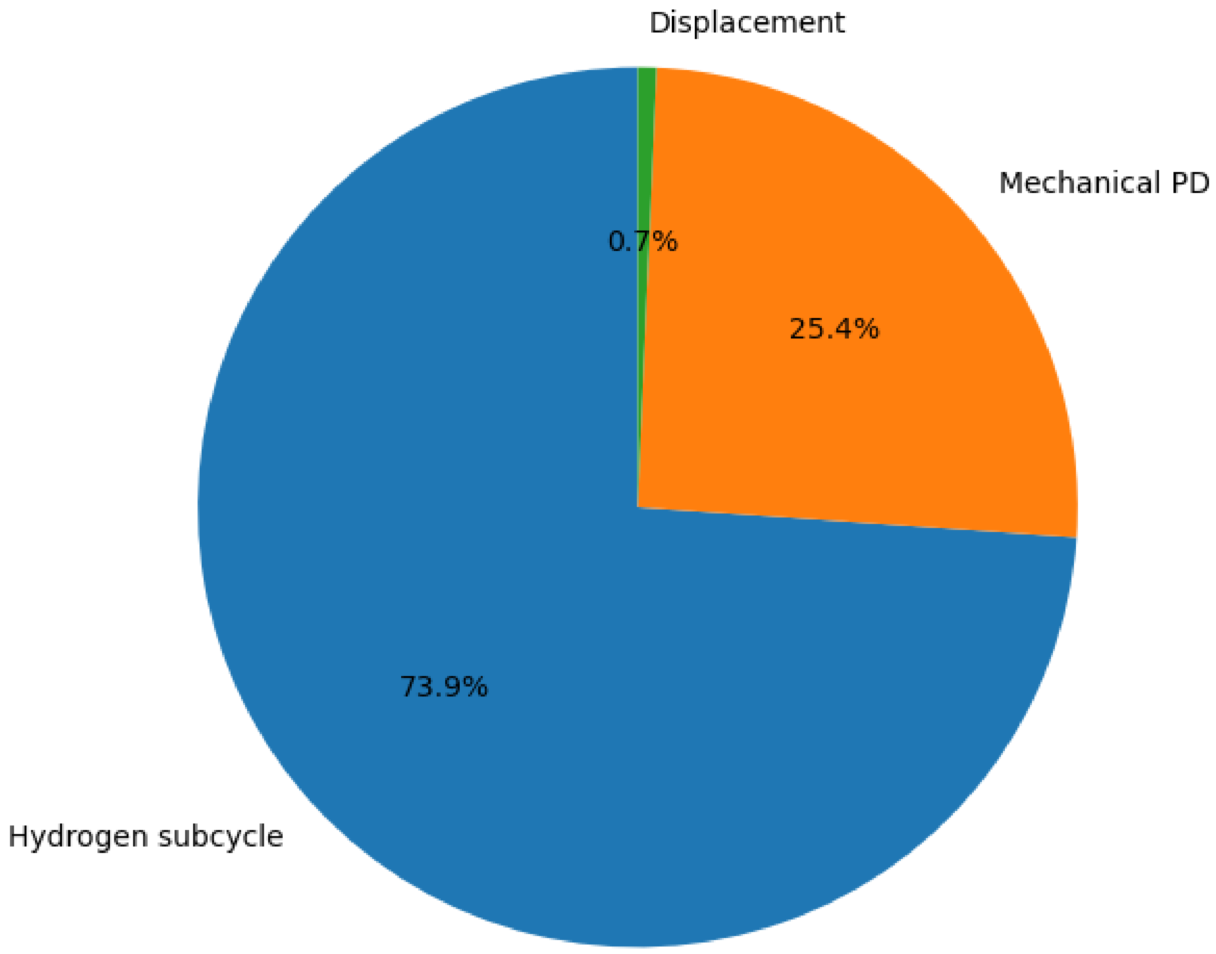


Figure 16: Pie chart showing the relative and absolute time spent in each phase per loadstep.

Potential Optimizations:

1. *Parallelize hydrogen subcycle:* Offload the $N_h \times N_{\text{points}}$ loop to OpenMP or GPU to expose fine-grained data parallelism.

2. *Accelerate neighbor searches:* Cache neighbor-counts and compress adjacency to improve memory locality in both diffusion and PD loops.

3. *Fuse loops:* Where possible, merge the hydrogen and PD loops to reduce repeated traversal of the same neighbor lists.

4. *Asynchronous I/O:* Buffer and aggregate logging to minimize stalls during the displacement/I/O phase.

This instrumentation has revealed that explicit hydrogen diffusion is the clear bottleneck in the PDHE simulation. By targeting that subcycle for parallel or GPU acceleration, we expect to reduce total runtime by up to an order of magnitude. Secondary gains are achievable through neighbor-list optimizations and loop fusion.

10 Conclusion

This work has presented the development, implementation, and verification of a peridynamic hydrogen-embrittlement (PDHE) model within the open-source Peridigm framework. A two-parameter bond-based formulation was employed to enable arbitrary Poisson's ratios, explicitly coupling hydrogen diffusion with mechanical bond failure. The implementation was rigorously tested through unit tests, boundary-condition checks, mesh-convergence studies, time-step convergence analyses, and decoupled "mechanical-only" and "diffusion-only" simulations.

Key findings include:

- **Verification of numerical routines:** Correct vector operations, stiffness corrections, and force computations were confirmed under controlled inputs.
- **Mesh-convergence:** Mean displacement and hydrogen-coverage metrics stabilized for meshes of 10,000-12,000 nodes, guiding the choice of optimal discretization.
- **Time-step convergence:** The explicit Euler diffusion scheme exhibited an empirical order of $p \approx 1.32$, slightly below the theoretical first order, indicating minor numerical artifacts.
- **Performance profiling:** The hydrogen-diffusion subcycle dominated the runtime ($\approx$ 74%), identifying clear targets for parallelization and GPU offloading.
- **Physical benchmark:** In a compact-tension-like geometry, damage contours traced a stable Mode I crack path under combined mechanical loading and hydrogen saturation, matching theoretical expectations.

Although four crack-path prediction scenarios were originally proposed, the present implementation has successfully realized only the Mode I crack-propagation case. This limitation is attributed to the intrinsic complexity of mixed-mode fracture in hydrogen-charged environments. Future work will extend the PDHE model to Mode II and mixed-mode configurations, incorporate advanced contact and crack-branching criteria, and leverage the identified performance optimizations to enable large-scale three-dimensional simulations.

Overall, the validated PDHE implementation in Peridigm lays a robust foundation for simulating hydrogen-assisted fracture in metallic materials. Its flexibility in Poisson's-ratio representation, combined with rigorous verification and clear performance pathways, positions it as a valuable tool for the design and analysis of hydrogen-compatible structures.

11 Manual

This section presents a detailed explanation for installing Peridigm, simulation setup and post-processing of the final results. The code for main implementation of PDHE model in *Peridigm* is placed under `Source_code` folder. Using instruction given section 5, implement the same into *Peridigm*. The whole project was conducted in the High-Performance cluster at the facility of TU Freiberg. To implement the PDHE model, access to the high-performance cluster is a must and Peridigm should be installed on your local node.

11.1 Peridigm installation

Peridigm is already installed by the service-desk team onto the HPC cluster. Since access to modify the main source directory is not granted to everyone, installation of Peridigm on your local node will make the custom model implementations easier. The required instructions to set up Peridigm onto our local node involve setting up directories for building and installing Peridigm, configuring the necessary paths, and running the installation script.

11.1.1 Directory setup

Change the permissions of `Directory_setup.sh` (necessary bash script is provided in the installation directory) to executable using `chmod u+x Directory_setup.sh` and run the script to have an appropriate directory setup. Inside the *Peridigm* folder, three subdirectories are created:

- `build` for building Peridigm.
- `source` contains cloned Peridigm from git repository.
- `install` for the final installation.

```bash
#!/bin/bash
cd ~/
git clone https://github.com/peridigm/peridigm.git
mv peridigm source
mkdir Peridigm
mv source ~/Peridigm
cd Peridigm
mkdir build
mkdir install
cd ~/
```

11.1.2 Installation using a bash script

Once the directories are created, place the bash script `make-peridigm.sh` into the build directory which is created from previous step, and run the script for local installation. Make the script executable if necessary using `chmod u+x make-peridigm.sh`. The script is a `CMake` script that includes the necessary paths to the working Trilinos installation and other dependencies.

Following is the `make-peridigm.sh` script (necessary bash script is provided in the installation directory):

```bash
#!/bin/bash

# Define paths to dependencies
Trilinos=/cluster/stages/2024.0/software/trilinos/install-peridigm-13-0
OPENMPI=/cluster/stages/2024.0/spack-0.22/opt/spack/linux-rocky8-cascadelake/gcc-11.4.0/openmpi-4.1.6-5
    ↪ voubczsrbixdifmpczll66tpm34th25
HDF5=/cluster/stages/2024.0/spack-0.22/opt/spack/linux-rocky8-cascadelake/gcc-11.4.0/hdf5-1.12.0-
    ↪ sxmurkczktjr357tjvp3bdoq5j5bhprs

# Set your home directory paths
PERIDIGM_SOURCE=~/peridigm/source
INSTALL_DIR=~/peridigm/install
BUILD_DIR=~/peridigm/build

# Remove any previous build cache
rm -f ${BUILD_DIR}/CMakeCache.txt

# Configure the Peridigm build
cmake \
  -D CMAKE_INSTALL_PREFIX:PATH=${INSTALL_DIR} \
  -D CMAKE_BUILD_TYPE:STRING=Release \
  -D Trilinos_DIR:PATH=${Trilinos}/lib/cmake/Trilinos \
  -D CMAKE_C_COMPILER:STRING=${OPENMPI}/bin/mpicc \
  -D CMAKE_CXX_COMPILER:STRING=${OPENMPI}/bin/mpicxx \
  -D CMAKE_Fortran_COMPILER:FILEPATH=$(which mpif90) \
  -D BOOST_ROOT=${BOOST_ROOT}/ \
  -D HDF5_DIR=${HDF5} \
  -D CMAKE_CXX_FLAGS:STRING="-O2 -Wall -pedantic -Wno-long-long -ftrapv -Wno-deprecated" \
  ${PERIDIGM_SOURCE}

make -j4
make install
```

The script proceeds by running `cmake`, which configures the build environment, including setting the compilers, flags, and linking paths for the necessary libraries. The `CMake` command also points to the source directory of Peridigm.

11.1.3 Loading Required Modules

To load and use a local installation of Peridigm, place the contents given below into your .bashrc(for Ubuntu) or .zshrc(for MacOS) file (necessary text file is provided in the installation directory):

```bash
export SOFTWARE_DIR=$HOME/software
export PATH=$SOFTWARE_DIR/bin:$PATH
export LD_LIBRARY_PATH=$SOFTWARE_DIR/lib:$LD_LIBRARY_PATH
export PATH=$SOFTWARE_DIR/bin:$PATH
export LD_LIBRARY_PATH=$SOFTWARE_DIR/lib:$LD_LIBRARY_PATH
module add peridigm/openmpi/gcc/11.4.0/master-12-24
module load libyaml/gcc/11.4.0/0.2.5
module load yaml-cpp/yaml-cpp/gcc/11.4.0/0.7.0
export PATH=~/peridigm/install/bin:$PATH
export LD_LIBRARY_PATH=~/Peridigm/install/lib:$LD_LIBRARY_PATH
```

`INSTALL_DIR` variable in the script points to a directory on your local node that allows you to create an isolated version of Peridigm that can be used independently of the system-wide installation.

By following all these instructions, one will be able to set up a local installation of Peridigm on your HPC system. This process involves setting up the necessary directories, configuring the build environment, compiling the source code, and loading the required modules for successful installation.

11.2 Pre-processing

Peridigm operates using mesh-free discretizations, where each nodal volume is characterized by their spatial coordinates $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$ and volume ΔV. These nodal volumes are grouped into blocks, and regions for applying initial conditions, boundary conditions, and body forces are defined through node sets.

The text file format used for discretizations in Peridigm directly represents the following structure. Each line in the file corresponds to a single nodal volume, listing the three spatial coordinates, the block number, and the associated volume ΔV. Node sets are defined in separate text files, where each file contains the node IDs that belong to a particular node set.

Gmsh, *Paraview* and *Meshio* software packages are installed using the following links and bash script:

For Gmsh software:
https://gmsh.info/Download
For Paraview software:
https://www.paraview.org/download/
For Meshio software (necessary bash script is provided in the installation directory):

```bash
#!/bin/bash
cd ~/
python -m pip install netCDF4
git clone https://github.com/nschloe/meshio.git
cd meshio
```

```
pip install
```

To create the mesh, *Gmsh* is used wherein required 2D domain is modelled and triangular mesh is used and exported to .msh format. Then the meshed domain is converted using "`meshio convert <file_name>.msh <file_name>.exo`" command on the terminal. The resulting `.exo` converted is imported into *Paraview*, using the `Extract Selection` feature, respective node ids for node sets are exported in text-file format. Further, using a python script (provided in the Pre-processing directory), the original `.msh` file is converted to text file format which is used as discretization file. Total number of nodes in the domain are displayed in *Paraview*, and this value is used to calculate the average nodal volume.

```python
import meshio
import sys

# 1) Read the Gmsh file
input_msh_file = sys.argv[1]
output_exo_file = sys.argv[2]
mesh = meshio.read(input_msh_file)

# 2) Create and open a text file for writing
with open(output_exo_file, "w") as f:
    # Optional: You can write a header if you like, but Peridigm's TextFile discretization
    # typically does not need one. If you do include it, remove it or comment it out:
    # f.write("x y z Volume BlockID\n")

    # 3) For each node, write out x,y,z plus placeholders for Volume & BlockID
    for i, point in enumerate(mesh.points):
        x, y, z = point*1e-3
        volume = 1.64365549*1e-6  # average nodal value computed
        blockID = 1      # placeholder
        f.write(f"{x} {y} 0.0 {volume} {blockID}\n")
```

11.3 Preparing for a simulation

The input for a Peridigm simulation consists of a discretization and an input deck. As discussed in subsection 11.2, the text file format is used in this project for discretizations.

The Peridigm input deck is a text file where users define the parameters for a simulation. This deck is organized into multiple sections, including those for the discretization, material models, damage models, time integrators, and simulation output. Often, the structure of the input deck corresponds directly to the blocks and node sets defined in the discretization file.

Peridigm supports two input deck formats: YAML and XML. The XML format was initially adopted during the development of Peridigm due to its seamless integration with various Trilinos packages, such as the `Teuchos::ParameterList` data structure. Later, support for YAML was introduced to provide a more user-friendly alternative. Both formats can be found in the examples and test subdirectories within the Peridigm code repository. The examples in the `examples` sub-directory were specifically created to offer new users a starting point.

Several sections are mandatory for every Peridigm simulation. These essential sections include: Discretization, Materials, Blocks, Solver and Output. For a simulation to be complete, the Boundary Conditions section must also be specified. In addition, there are optional sections like Damage and Contact, which are necessary for simulating bond failure and contact interactions, respectively. Examples of these optional sections are included in the example problems provided with Peridigm.
A basic Discretization section is given in table 6, in which the user specifies the name of the discretization file and the file format.

```
Discretization:

Type: "Text File"

Input Mesh File: "2D_single_crack_propagation.txt"
```

Table 6: Example of a Discretization section

Users may define any number of materials, each of which is given a unique label (`PDHE` in this case) for association with specific blocks in the discretization. Required material parameters, specifying the names for boundary condition files are also specified. Materials section is shown in table 7 and are self explanatory.

```
Materials:

  My Material:

    Material Model: "PDHE"

    Young's Modulus: 2.07e11

    Grain boundary diffusion coefficient: 0.84e-9

    Saturated value of hydrogen concentration: 2.65e-5

    Critical energy release rate: 16476.135

    Density: 8.0

    Poisson's ratio: 0.29

    No.  of load steps: 3000

    No.  of steps for hydrogen concentration: 20

    Thickness: 4.0e-3

    Minimum grid spacing: 1.0e-3

    Capture and save the simulation frame from N load steps: 3000

    File name for displacement in +ve x/y-direction: "nodeset_top.txt"

    File name for displacement in -ve x/y-direction: "nodeset_bottom.txt"

    File name for crack top lip: "nodeset_top_crack.txt"

    File name for crack bottom lip: "nodeset_bottom_crack.txt"

    File name for concentration: "nodeset_concentration.txt"

    Boundary condition test: false
```

Table 7: Example of a Materials section

Note that Peridigm does not explicitly track the units for any input parameters, and instead utilizes a consistent units approach in which users are free to select the system of units best suited for the given simulation. In this project, SI units are considered throughout the simulation and are mentioned in the header of the input file.

The Blocks section shown in table 8 maps the material model with label `My Material` to `block_0` in the discretization and specifies the horizon for that block. Likewise, the Blocks section may also be used to associate bond damage models, if any, with specific blocks.

```
Blocks:

  My Block:

    Block Names: "block_0"

    Material: "My Material"

    Horizon: 5e-3
```

Table 8: Example of a Blocks section

Initial and boundary conditions are specified in the Boundary Conditions section. The example given in table 9 specifies a prescribed temperature which acts as a proxy for applying saturated concentration boundary condition for the required node sets. This boundary condition is applied onto the crack surface which indicates hydrogen ions are deposited on the surface of the crack and are allowed for diffusion, and the same is applied to `nodeset_concentration.txt`. For the displacement boundary condition, a piece-wise type of displacement was required for the test-cases. So, the boundary condition is directly implemented into the `PDHE` model and required file-names are imported from materials section 7.

```
Boundary Conditions:

  Node Set One: "nodeset_concentration.txt"

  Hydrogen Concentration on Surface:

    Type: "Prescribed Temperature"

    Node Set: "Node Set One"

    Value: 2.65e-5
```

Table 9: Example of a Boundary condition section

Table 10 contains an example of a Solver section. But in this project, we will not be using any solver supported by Peridigm. Since mentioning solver section is mandatory, QuasiStatic solver is mentioned to run the simulation, and its specific attributes are defined.

```
Solver:

  Initial Time: 0.0

  Final Time: 1.0

  QuasiStatic:

    Number of Load Steps: 4

    Absolute Tolerence: 1.0

    Maximum Solver Iterations: 10
```

Table 10: Example of a Solver section

Finally, an Output section is presented in table 11. Parameters include the name of the output exodus file, the frequency at which data is written to file, and a list of the variables to be stored.

```
Output:

  Output File Type: "ExodusII"

  Output File Name: "Dummy_exodus_file_not_to_be_used"

  Output Frequency: 1

  Number of Load Steps: 4

  Output Variables:

    Displacement: true
```

Table 11: Example of a Output section

11.4 Running Peridigm

Place the necessary files into the simulation folder: the input-deck file, the discretized-domain text file, and the node-set text files. With the terminal's current directory set to the simulation folder, use the following command to run the `PDHE` simulation:

```
Peridigm <Input_deck_file.yaml>
```

11.5 Post-processing

Once the simulation completes, an `Output.txt` file is generated with a header at the top. The content of the file appears as follows:

```
#Header
x(m) y(m) Displacement(m) Hydrogen_concentration(mol/m^2) Damage

Load step: 0
0 0 0 1.62362e-24 0
0.2 0 0 1.11868e-89 0
0.2 0.1 0 2.88076e-89 0
... (additional entries)
```

This text file is further post-processed using the python script (which is provided under post-prosessing folder) and separate <`animation_file`>`.gif` is generated for *Damage*, *Displacements* and *Hydrogen coverage*.

12 Git log

The full commit history for this study is shown below. Each line gives the short hash, author, date and commit messages.

```
commit: 1ac27e14640f909f062070e85f041d7ab086bb76
author: Prasanna Ramesh Hegde
date:   2025-05-02 22:52:27 +0200
message:Final uploads of the codes

commit: 1ec8067152b8c151e7f20556268768d360ba0694
author: Prasanna Ramesh Hegde
date:   2025-05-02 21:19:34 +0200
message:Added timing summary for the code

commit: 13e1564bcc2760f8304da5abfcb812f5eeb86016
author: Prasanna Ramesh Hegde
date:   2025-05-01 02:11:25 +0200
message:Added changes suitable for testing

commit: 8ab63289e1570ac7d54275300b9415a7b72f4032
author: Prasanna Ramesh Hegde
date:   2025-04-30 01:30:00 +0200
message:Changes done according to boundary condition test

commit: 78bd4f5d94e2e87a8b17042968777d5c3ed942d3
author: Prasanna Ramesh Hegde
date:   2025-04-29 18:36:10 +0200
message:Added more unit tests to the file

commit: 8bcc575c20e9e73ac026745ddbc6855c12be02d5
author: Prasanna Ramesh Hegde
date:   2025-04-29 17:45:04 +0200
message:Added some more unit tests with some reader freindly printing headers

commit: 0838b76a2d1ada63acb82798f472291d5951cd12
author: Prasanna Ramesh Hegde
date:   2025-04-29 17:14:47 +0200
```

```
message:Created unit test file for PDHE material model

commit: d968d87dd81a5c3a8b349bb45d68f7184313434f
author: Prasanna Ramesh Hegde
date:   2025-04-29 09:48:20 +0200
message:Input deck for PDHE simulation

commit: 61b00a8067b29f3fd43f71d36b3c05e5d5478e14
author: Prasanna Ramesh Hegde
date:   2025-04-29 09:33:56 +0200
message:Added required comments for better understanding

commit: 2f8e5953cdfa3a43b17f59b75be7e2f0f26c61e6
author: Prasanna Ramesh Hegde
date:   2025-04-29 08:22:17 +0200
message:Cleared unecessary comments and alligned for better code viewing

commit: aec8e2c11b427f11ac14ba80074d638584125721
author: Prasanna Ramesh Hegde
date:   2025-04-28 10:04:11 +0200
message:Added a logic to not to have interactions
        between material points across the crack

commit: 69757f412b9b963cabda0783fe31a59ea8b8dc3f
author: Prasanna Ramesh Hegde
date:   2025-04-21 00:34:34 +0200
message:Final version 1 code for single crack propagation

commit: 6bfdbdb3c6556e7f64dab35467e752a19a4be345
author: Prasanna Ramesh Hegde
date:   2025-04-17 02:01:50 +0200
message:Changed volume equations to area

commit: fc0f02c5ec3d5968cb6cdbcb4d106a30e0cd537f
author: Prasanna Ramesh Hegde
date:   2025-04-17 01:53:08 +0200
message:Everything changed to 2D problem
```

```
commit: b5b6a255427b972a9b52dea92af82c458d7aaaf8
author: Prasanna Ramesh Hegde
date:   2025-04-14 22:15:11 +0200
message:Checked for all errors and fixed successfully

commit: 178d753d6486ec1393151e210bc60a0972e63c71
author: Prasanna Ramesh Hegde
date:   2025-04-13 09:50:08 +0200
message:Added critical bond stretch criteria
        and modified some parts of the code accordingly

commit: b0c401a98732fd43ff0e01f154e604f063e2e10f
author: Prasanna Ramesh Hegde
date:   2025-04-08 22:16:12 +0200
message:All runtime errors fixed

commit: 3d7813dd95e5042353f969f7c99eff3618fdcc2a
author: Prasanna Ramesh Hegde
date:   2025-04-08 11:26:30 +0200
message:Changed way of importing displacements, debugging Nan
        from U_dot_n_plus_half

commit: 4fe80d0fbe3c9d3026d81fc254ca8ce83d3bf001
author: Prasanna Ramesh Hegde
date:   2025-04-07 22:03:31 +0200
message:Added ampersand to address all vectors

commit: 69e548507558f3b747a26913ec5b9f2468f53f45
author: Prasanna Ramesh Hegde
date:   2025-04-07 21:15:12 +0200
message:Storing PD values in different way --> eleminating for loop

commit: 0a390cd90c7e5130c4d4d22663776f5d077cf02b
author: Prasanna Ramesh Hegde
date:   2025-04-07 21:14:00 +0200
message:Changed addressing the elements of vectors
```

```
commit: aeb2d33b5f7920ca048a0af9e2c0384961b9fffe
author: Prasanna Ramesh Hegde
date:   2025-04-07 18:29:56 +0200
message:Debuggin Nan values coming from PD forces

commit: 209be64e676bacf9206bc23ef0658345df3832be
author: Prasanna Ramesh Hegde
date:   2025-04-07 18:16:52 +0200
message:Peridigm PDHE material models for initial
        adding into the peridigm. Initial debugging
```

References

[1] Zhuang Chen, Diansen Yang, and Haoran Bian. Peridynamic modeling of crack propagation driven by hydrogen embrittlement. *Engineering Fracture Mechanics*, 293:109687, December 2023. doi: 10.1016/j.engfracmech.2023.109687. URL https://doi.org/10.1016/j.engfracmech.2023.109687.

[2] Nour-Eddine Laadel, Mohamed El Mansori, Nan Kang, Samuel Marlin, and Yves Boussant-Roux. Permeation barriers for hydrogen embrittlement prevention in metals–a review on mechanisms, materials suitability and efficiency. *International Journal of Hydrogen Energy*, 47(76):32707–32731, 2022. URL https://doi.org/10.1016/j.ijhydene.2022.07.164.

[3] Esteban R. Ugarte and Saeed Salehi. A review on well integrity issues for underground hydrogen storage. *Journal of Energy Resources Technology*, 144(4):042001, 10 2021. ISSN 0195-0738. doi: 10.1115/1.4052626. URL https://doi.org/10.1115/1.4052626.

[4] Maria Portarapillo and Almerinda Di Benedetto. Risk assessment of the large-scale hydrogen storage in salt caverns. *Energies*, 14(10), 2021. ISSN 1996-1073. doi: 10.3390/en14102856. URL https://www.mdpi.com/1996-1073/14/10/2856.

[5] Fuyuan Yang, Tianze Wang, Xintao Deng, Jian Dang, Zhaoyuan Huang, Song Hu, Yangyang Li, and Minggao Ouyang. Review on hydrogen safety issues: Incident statistics, hydrogen diffusion, and detonation process. *International journal of hydrogen energy*, 46(61):31467–31488, 2021. URL https://doi.org/10.1016/j.ijhydene.2021.07.005.

[6] Xizhuo Chen and Haitao Yu. A novel micropolar peridynamic model for rock masses with arbitrary joints. *Engineering Fracture Mechanics*, 281:109099, 2023. ISSN 0013-7944. doi: https://doi.org/10.1016/j.engfracmech.2023.109099. URL https://www.sciencedirect.com/science/article/pii/S0013794423000577.

[7] Haitao Yu, Xizhuo Chen, and Yuqi Sun. A generalized bond-based peridynamic model for quasi-brittle materials enriched with bond tensionrotationshear coupling effects. *Computer Methods in Applied Mechanics and Engineering*, 372:113405, 2020. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2020.113405. URL https://www.sciencedirect.com/science/article/pii/S0045782520305909.

[8] Wanjin Li and Li Guo. Peridynamic investigation of chloride diffusion in concrete under typical environmental factors. *Ocean Engineering*, 239:109770, 2021. ISSN 0029-8018. doi: https://doi.org/10.1016/j.oceaneng.2021.109770. URL https://www.sciencedirect.com/science/article/pii/S0029801821011367.

[9] Alexander Hermann, Arman Shojaei, Dirk Steglich, Daniel Hche, Berit Zeller-Plumhoff, and Christian J. Cyron. Combining peridynamic and finite element simulations to capture the corrosion of degradable bone implants and to predict their residual strength. *International Journal of Mechanical Sciences*, 220:107143, 2022. ISSN 0020-7403. doi: https://doi.org/10.1016/j.ijmecsci.2022.107143. URL https://www.sciencedirect.com/science/article/pii/S0020740322000741.

[10] Dennj De Meo, Cagan Diyaroglu, Ning Zhu, Erkan Oterkus, and M. Amir Siddiq. Modelling of stress-corrosion cracking by using peridynamics. *International Journal of Hydrogen Energy*, 41(15):6593–6609, 2016. ISSN 0360-3199. doi: https://doi.org/10.1016/j.ijhydene.2016.02.154. URL https://www.sciencedirect.com/science/article/pii/S0360319915301245.

[11] David J Littlewood, Michael L Parks, John T Foster, John A Mitchell, and Patrick Diehl. The peridigm meshfree peridynamics code. *Journal of Peridynamics and Nonlocal Modeling*, 6(1):118–148, 2024. URL https://doi.org/10.1007/s42102-023-00100-0.

[12] Erdogan Madenci and Erkan Oterkus. Peridynamic theory. In *Peridynamic theory and its applications*, pages 19–43. Springer, 2013. URL https://doi.org/10.1007/978-1-4614-8465-3_2.

[13] S.A. Silling and E. Askari. A meshfree method based on the peridynamic model of solid mechanics. *Computers Structures*, 83(17):1526–1535, 2005. ISSN 0045-7949. doi: https://doi.org/10.1016/j.compstruc.2004.11.026. URL https://www.sciencedirect.com/science/article/pii/S0045794905000805. Advances in Meshfree Methods.

[14] Xianyang Guo, Zhuang Chen, Xihua Chu, and Ji Wan. A plane stress model of bond-based cosserat peridynamics and the effects of material parameters on crack patterns. *Engineering Analysis with Boundary Elements*, 123:48–61, 2021. ISSN 0955-7997. doi: https://doi.org/10.1016/j.enganabound.2020.11.011. URL https://www.sciencedirect.com/science/article/pii/S0955799720302940.

[15] Xian Yang Guo and Xi Hua Chu. The performance of five extended bond-based peridynamic models on crack simulation. *International Journal of Applied Mechanics*, 13(09):2150101, 2021. doi: 10.1142/S1758825121501015. URL https://doi.org/10.1142/S1758825121501015.

[16] Hyung-Seop Shin, Nick Anthony Custodio, and Un-Bong Baek. Numerical analysis for characterizing hydrogen embrittlement behaviors induced in sts316l stainless steel using an in-situ small-punch test. *Theoretical and Applied Fracture Mechanics*, 116:103139, 2021. ISSN 0167-8442. doi: https://doi.org/10.1016/j.tafmec.2021.103139. URL https://www.sciencedirect.com/science/article/pii/S0167844221002391.

[17] Yi Shuai, Xinhua Wang, Junqiang Wang, Heng-Gang Yin, and Y. Frank Cheng. Modeling of mechanical behavior of corroded x80 steel pipeline reinforced with type-b repair sleeve. *Thin-Walled Structures*, 163:107708, 2021. ISSN 0263-8231. doi: https://doi.org/10.1016/j.tws.2021.107708. URL https://www.sciencedirect.com/science/article/pii/S0263823121001816.

[18] Michihiko Nagumo et al. *Fundamentals of hydrogen embrittlement*, volume 921. Springer, 2016. URL https://doi.org/10.1007/978-981-10-0161-1.

[19] Naveen Prakash and Gary D. Seidel. A novel two-parameter linear elastic constitutive model for bond based peridynamics. doi: 10.2514/6.2015-0461. URL https://arc.aiaa.org/doi/abs/10.2514/6.2015-0461.

[20] Xiu Ran, Songrong Qian, Ji Zhou, and Zhengyun Xu. Crack propagation analysis of hydrogen embrittlement based on peridynamics. *International Journal of Hydrogen Energy*, 47(14):9045–9057, 2022. ISSN 0360-3199. doi: https://doi.org/10.1016/j.ijhydene.2021.11.173. URL https://www.sciencedirect.com/science/article/pii/S0360319921045602.

[21] Ziguang Chen and Florin Bobaru. Peridynamic modeling of pitting corrosion damage. *Journal of the Mechanics and Physics of Solids*, 78:352–381, 2015. ISSN 0022-5096. doi: https://doi.org/10.1016/j.jmps.2015.02.015. URL https://www.sciencedirect.com/science/article/pii/S0022509615000526.

[22] Xuefeng Gu, Wei Zhou, Lichao Zhang, Xianren Sun, Wen Zhu, Ming Wu, Ming Xu, Jian Li, and He Yan. Experimental and numerical investigation of fracture behavior of low-alloy steel under hydrogen charging. *Computational Mechanics*, 60(5):809–823, 2017. doi: 10.1007/s00466-017-1469-1. URL https://doi.org/10.1007/s00466-017-1469-1.

[23] Jiangyi Luo, Ali Ramazani, and Veera Sundararaghavan. Simulation of micro-scale shear bands using peridynamics with an adaptive dynamic relaxation method. *International Journal of Solids and Structures*, 130:36–48, 2018. URL https://doi.org/10.1016/j.ijsolstr.2017.10.019.

[24] B. Kilic and E. Madenci. An adaptive dynamic relaxation method for quasi-static simulations using the peridynamic theory. *Theoretical and Applied Fracture Mechanics*, 53(3):194–204, 2010. ISSN 0167-8442. doi: https://doi.org/10.1016/j.tafmec.2010.08.001. URL https://www.sciencedirect.com/science/article/pii/S0167844210000479.

[25] Ziguang Chen, Siavash Jafarzadeh, Jiangming Zhao, and Florin Bobaru. A coupled mechano-chemical peridynamic model for pit-to-crack transition in stress-corrosion cracking. *Journal of the Mechanics and Physics of Solids*, 146:104203, 2021. ISSN 0022-5096. doi: https://doi.org/10.1016/j.jmps.2020.104203. URL https://www.sciencedirect.com/science/article/pii/S0022509620304245.

[26] The trilinos project website. URL https://trilinos.github.io.