Received: 29 May 2024 | Accepted: 5 November 2024

DOI: 10.1002/pamm.202400160

RESEARCH ARTICLE

Reducing the entry barrier for users and developers of peridynamic high performance computing

Jan-Timo Hesse[1] | **Christian Willberg**[2] | **Anna Pernatii**[3]

[1]German Aerospace Center (DLR), Braunschweig, Germany

[2]Magdeburg-Stendal University of Applied Sciences, Magdeburg, Germany

[3]Otto von Guericke University Magdeburg, Magdeburg, Germany

Correspondence
Jan-Timo Hesse, German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Braunschweig, Germany.
Email: jan-timo.hesse@dlr.de

Christian Willberg, Magdeburg-Stendal University of Applied Sciences, Breitscheidstr. 2, 39114 Magdeburg, Germany. Email: christian.willberg@h2.de

Anna Pernatii, Otto von Guericke University Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany. Email: anna.pernatii@ovgu.de

Funding information
Deutsche Forschungsgemeinschaft; Gekoppelte Peridynamik-Finite-Elemente-Simulationen zur Schädigungsanalyse von Faserverbundstrukturen, Grant/Award Number: WI 4835/5-1; M-ERA.NET; Saxon State Parliament, Grant/Award Number: 3028223

Abstract

Over the past two decades, Peridynamics (PD) research has exhibited remarkable diversity, with contributions spanning more than 180 journals and involving over 1000 researchers. Despite its broad applicability, a persistent challenge remains—how to foster widespread adoption. In the engineering domain, classical continuum mechanics, predominantly facilitated by the finite element method, enjoys extensive utilization, supported by a plethora of commercial and open-source software tools. PD simulation tools often find themselves competing with these well-established counterparts. In research, custom codes tailored for specific applications are frequently created and applied, with little consideration for their future reuse or third-party utilization. The complexity is further exacerbated in High-performance computing (HPC) applications, where a deep understanding of solvers, parallel computing, and PD is imperative for a successful software development. Given the constraints of a typical doctoral thesis, large-scale problems are often left unexplored, and research tends to be confined to simplistic geometries. This paper aims to elucidate various pathways for the seamless integration and utilization of a PD framework. We will showcase examples that illustrate how the barriers for both users and developers can be significantly lowered, ultimately propelling PD simulation tools to a higher standard of maturity in the medium term.

1 | INTRODUCTION

In engineering applications, the mechanical behavior of materials and structures is typically modeled using classical continuum mechanics. This approach utilizes the Partial Differential Equation (PDE) $\mathrm{div}\boldsymbol{\sigma} + \mathbf{b} = \rho\ddot{\mathbf{u}}$; to describe the relationship between external loads $\mathbf{b}$, inertia loads $\rho\ddot{\mathbf{u}}$, and the internal reaction of the material $\mathrm{div}\boldsymbol{\sigma}$, where the mechanical stresses $\boldsymbol{\sigma}$ are determined by spatial derivatives of the displacements $\mathbf{u}$. While this approach is commonly solved using

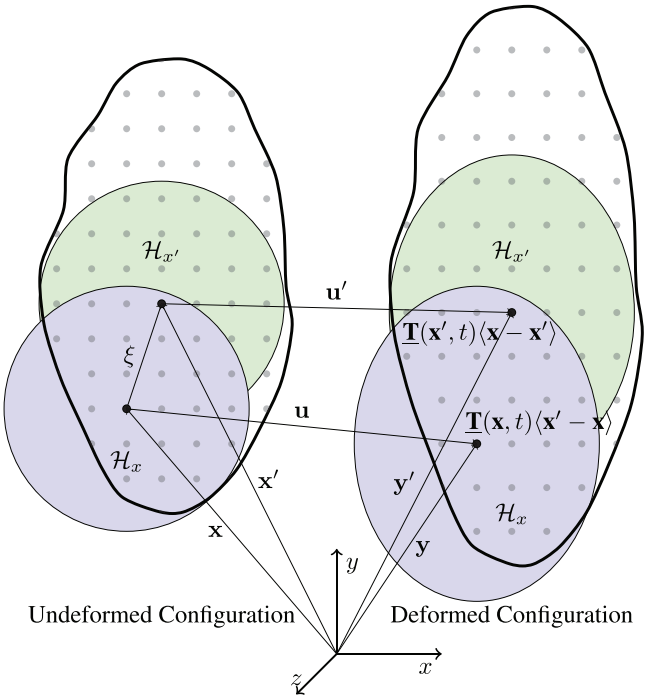

FIGURE 1 Deformation and interaction of material points $\mathbf{x}$ and $\mathbf{x}'$ in the NOSB theory and their respective neighborhoods $\mathcal{H}_{\mathbf{x}}$ and $\mathcal{H}_{\mathbf{x}'}$. NOSB, non-ordinary state-based.

the Finite Element Method (FEM), it has limitations when dealing with fractures, as the derivative at the crack tip is not determinable. In such cases and if a critical stress value is exceeded, a switch to fracture mechanics is typically required [1].

1.1 | Peridynamics (PD)

A novel modeling approach was proposed in the early 2000s in the form of PD [2] and is an alternative to the widely used phase-field model [3]. PD may be thought of as a continuum version of molecular dynamics [4], replacing the continuum mechanics PDE with an integral equation. In PD, a material point interacts with all other material points within a finite domain $\mathcal{H}$, called a neighborhood, as illustrated by the colored circles around points $\mathbf{x}$ and $\mathbf{x}'$ in Figure 1. Over the last two decades, three main modeling approaches within PD have been developed, namely the Bond-based (BB), Ordinary state-based (OSB) and Non-ordinary state-based (NOSB) PD formulations, each differing in the way they fulfill conservation of momentum and angular momentum. This has implications for material modeling, while the BB model is limited to fixed Poisson's numbers but has a simple and computationally efficient implementation, the non-ordinary state-based model allows any material model to be implemented.

To utilize material models from classical continuum mechanics, a nonlocal deformation gradient is typically calculated, which allows for the determination of nonlocal strain measures and stress-strain relations [5].

PD research has been very diverse in the last 20 years, with publications in over 180 journals and more than 1000 involved researchers, as shown in Figure 2. While this section does not claim to be comprehensive, it aims to provide an overview of the topics in the field, given that PD covers a wide range of applications.

1.2 | PD software

Mesh-free particle-based methods are mostly used for the numerical approximation of PD equations. Table 1 provides an overview of current PD software developments. In addition to this list, many local research groups are also developing their own in-house software. Most of the listed tools are problem specific or developed by a single person. `Peridigm` and

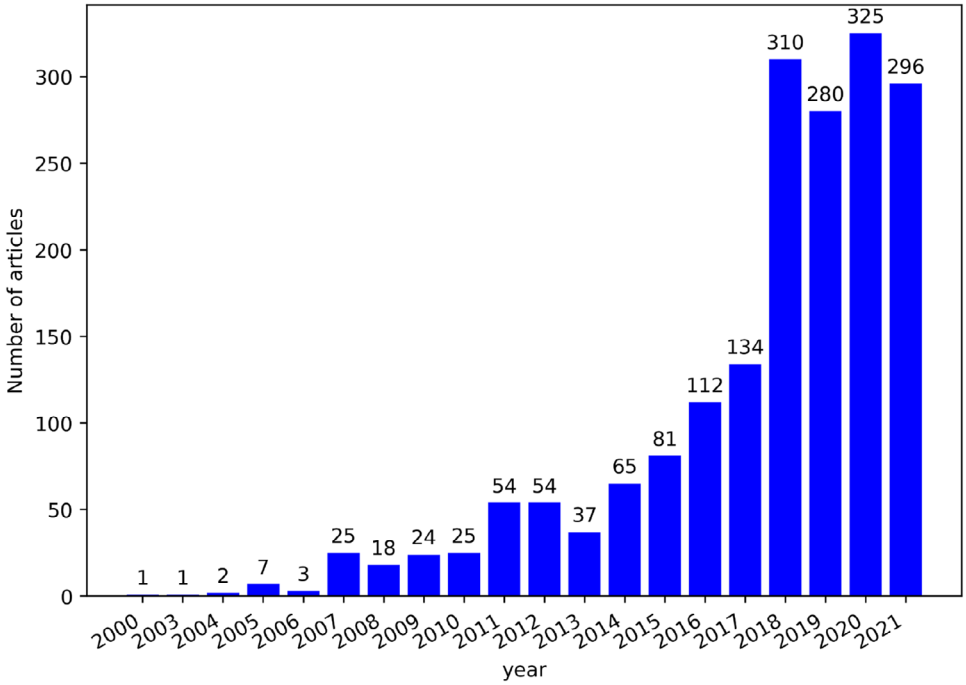

FIGURE 2 PD related articles per year in database (Dec. 2021). PD, peridynamics.

TABLE 1 Overview about PD software tools.

Name	Language	Maturity level	License	References
PeriPy	Python	Medium	MIT	[11]
PeriDem	C++, Python	Low		[12]
PeriPyDIC	Python	Low	GPLv3.0	[13]
PeriPyVFM	Python	Low	GPLv3.0	[14]
BB_PD	Matlab, C	Low-medium	no defined	[15]
LAMMPS	Python	Low	GPLv2	[16]
PeriFlakes	Python, C	Low	GPLv3.0	[17]
NLMech	C++	Medium		[12]
Relation-based software	C++	Medium	MIT	[18]
EMU	Fortran	High	Closed source	[19]
`Peridigm`	C, C++	High	BSD	[20–22]
PeriHub	C, C++	High	BSD	[23, 24]

Abbreviations: BB, bond base; BSD, Berkeley software distributions; PD, peridynamics; LAMMPS, Large-scale Atomic/Molecular Massively Parallel Simulator; MIT, Massachusetts Institute of Technology.

EMU are the most sophisticated software tools. EMU is not open source, but researchers can request the code. However, due to the code developments at the Sandia National labs and the rules of development, the code is officially not usable, as it violates the Treaty on the NonProliferation of Nuclear Weapon. `Peridigm` is an open-source tool that can be used for large-scale problems. It includes multiple material laws, simple damage models, and BB, OSB, and NOSB PD modeling approaches. `Peridigm` is usable under a Berkeley Software Distribution (BSD) license. Currently, there is some irregular development ongoing. It provides multiple ways of model input and provides ParaView readable output. The entire code allows the analysis of large-scale problems. A lot of research has been performed utilizing the software [6–10].

One significant challenge still remains: How can PD gain widespread adoption? In the engineering field, classical continuum mechanics is extensively utilized through the FEM. Numerous software tools are readily available for this purpose. However, when it comes to PD, the situation is quite different. While `Peridigm` stands as one of the most advanced options

and it offers an open-source code with a rich set of functionalities, unfortunately the process of implementing it is quite demanding, and the installation can be both arduous and time-consuming.

To implement even the simplest material law in `Peridigm` a minimum of five files are required to be modified. Additionally, if the user wishes to introduce new files and directories to the project, he must contend with CMake scripts and the associated complexities.

Furthermore, it is worth noting that `Peridigm` is only partially maintained. This results in valuable research discoveries being underutilized. Consequently, PD struggles to find applications beyond specific niche problems. There is an evident and pressing need for software that seamlessly combines functionality, user-friendly installation, and easy integration.

This paper introduces a solution called `PeriLab`, which is built using Julia and incorporates Message Passing Interface (MPI) to manage a large number of degrees of freedom (dof). The objectives for `PeriLab` encompass:

- Installation: The installation process must be straightforward and not pose a barrier to using the tool.
- Usability: The tool should offer simplicity and flexibility to cater to various user needs.
- Implementation and Extension: It should be designed in a way that facilitates efficient research code development and extension.
- Functionality: All existing functionalities must be well-documented and thoroughly tested.

2 | APPROACH

`PeriLab` is divided into several core areas:

- IO—responsible for reading input and writing results.
- Physics—encompasses all physical models and analyses.
- Core—houses all solvers, currently including the Verlet solver.
- Compute—calculates resultant quantities that are of interest to the user but not necessary for problem-solving.
- Support—contains utility routines used in various parts of the program.
- MPI_Communication—handles High performance computing (HPC) communication.

2.1 | Installation

To install `PeriLab`, first of all Julia must be downloaded and installed. To utilize `PeriLab` two simple options occur. The first utilizes the built in Julia package manager (Pkg) and the official latest release of `PeriLab`:

1. start Julia in batch shell
2. type "]", in order to start the Pkg
3. write "add `PeriLab`"

After installation of the module and its dependencies you can use `PeriLab` via "using `PeriLab`" in your Julia code. Alternatively, if you want to contribute or be able to adapt the software, you can compile and install `PeriLab`'s current development version via the source code:

1. Clone the repository: "git clone https://github.com/PeriHub/PeriLab.jl"
2. Move into the directory "cd `PeriLab`.jl"
3. type "]", in order to start the Pkg
4. activate the project "activate."
5. compile `PeriLab` "precompile"

In comparison to the Peridigm installation the time needed is reduced from hours to minutes. It is much simpler, because the modern package distribution system of Julia resolves all the package dependencies.

2.2 | Usability

The usability has been improved compared to `Peridigm`. Inputs are allowed as in mesh text files. Internally these meshes are stored as data frames, Listing 1. Besides the minimum definition (x, y, [optional z], volume and block_id) the variables defined in the header of the mesh file can be referenced throughout `PeriLab`. The types of these fields are determined automatically. In the example, Listing 1, coordinates will have the type **Int64** and Volume the type **Float64**. The type is defined by the entries, meaning if a Float value occurs the whole field will be float. The list can also be defined as a Boolean if needed.

Listing 1: Mesh input example.

```
data = Dataframe(Dict("x" => [1, 1, 3],
  "y" => [25, 30, 22],
  "z" => [25, 30, 22],
  "volume" => [1.2, 0.8, 1],
  "block_id" => [1, 2, 1],
  "active_x" => [true, true, false],
  "active_y" => [true, true, false],
  "active_z" => [true, true, false],
  "field" => [1.0, 3.3, 2.3]
))
```

The advantage of this approach is that point-associated information can easily be mapped from the pre-processor into `PeriLab`. An example would be predefined temperatures or coordinate system definitions. These fields are defined as constant fields and are not time-dependent. The data type is defined by the input and can be of type Int64, Float64, or Boolean. As stated all these fields are accessible through the data manager everywhere in the software, using the given names.

Besides the mesh file a YAML input-deck is specified to define material properties, nodes sets, blocks, and so forth. Parameter names can be addressed directly in the models. For boundary conditions all parameters which are defined in `PeriLab` can be addressed and utilized by a function evaluator, meaning that time-, location- or material-depending conditions can be provided.

For the export of results, the Exodus.jl package is utilized. It provides an interface to a C++ library. Also a Comma-separated values (CSV) export is included to provide global variables in an already human readable way.

Additionally `PeriLab` provides the option to utilize already existing input data, in the form of user materials or input files from Abaqus, making it easier for new users. Also experienced HPC users will benefit, as functionalities such as logging processing core information or a debugging mode are included. In total the usability of `PeriLab` is drastically increased for users as well as developers.

2.3 | Implementation and extension

Introducing custom models in `Peridigm` or Abaqus (as a nonPD example) can be challenging. You need to have a deep understanding of how the compiler works and where to call your functions. In the context of `PeriLab`, creating a new material model involves copying your module file into the "material" folder, effectively creating a new material model using the template shown in Listing 2. You have the flexibility to choose your module name. The material name serves as an identifier within your input deck. When you use this name as a material model option in your input deck, this module is activated.

For implementing a new material model in `Peridigm`, at least five files need to be created or modified. This number can vary and increase, especially when dealing with CMake files. In `PeriLab`, only one file is created with a provided template and placed in the corresponding directory. This is done by using the meta programing feature provided by Julia. Figure 3 highlights this workflow. The software will search for modules within the project folder. Within the YAML file the model name is specified and can be referenced. This name has to be defined within the new material module in a function. If the name of a module and the definition are equal, the module will be included and compiled. The whole procedure is done automatically and works for the basic models (additive, damage, material, and thermal). Extensions are possible by using the same strategy.

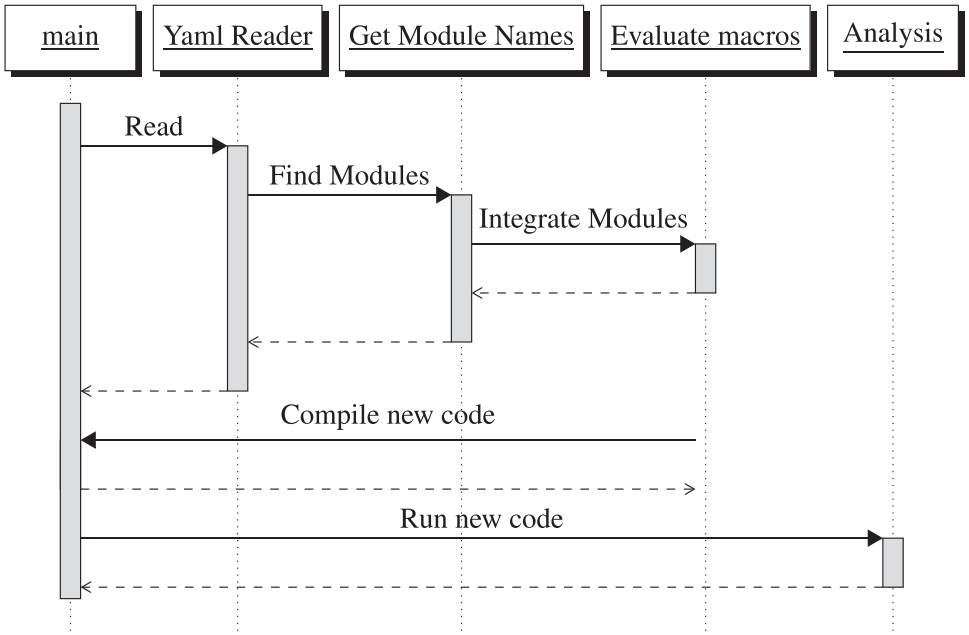

FIGURE 3 Concept of macro usage to include modules in PeriLab.

2.4 | Functionality

Compared to `Peridigm`, the current functionality is reduced. It includes only a subset of material and damage routines and a single solver. The program flow after compilation is shown in Figure 4. The `PeriLab` module is included in a main.jl file and the program is started via `PeriLab.main()`. The main file is used to check if the Project.toml and the resulting Manifest.toml exists and will activate the project.

After starting the main function with a batch command the input-deck is expected. This file is parsed in the IO module. This module is used to read all files needed for model creation and to distribute all the data in the data manager and processing cores if MPI is used. After that the solver is initialized. For the currently implemented solver the step width is determined and all fields and the physics models will be initialized. If they are not used, no initialization will occur. The Exodus output file and / or the CSV file is also created. Afterwards the solving process will be started and the physical models are evaluated. In parallel the simulation results are written. At the end of this process the result files are merged if needed and the program stops.

2.5 | Documentation and testing

When it comes to developing complex algorithms, such as those found in PD, proper documentation and testing are crucial components.

Proper documentation is essential for any software, especially one that involves complex algorithms like PD. Documentation serves as a guide for users and developers alike, providing insights into how the code works, what it does, and how to use it effectively. In the case of PD software, documentation can help bridge the gap between researchers and developers by making complex concepts more accessible.

In Julia, documentation is typically written using Markdown syntax. This allows for easy inclusion into project repositories on platforms like GitHub. Moreover, tools like Documenter.jl make it simple to generate HTML documentation for projects.

Testing is another critical aspect of software development. It ensures that the code behaves as expected, catching bugs and errors before they affect users. In Julia, testing frameworks like Test.jl provide an easy way to write tests for your code. These tests can be run using the julia command-line tool or integrated into CI/CD pipelines for automated testing. For `PeriLab` a main goal was to get the test code coverage as high as possible. That is why the software includes extensive unit- as well as full-scale tests, which will be executed every commit that is submitted to the repository.

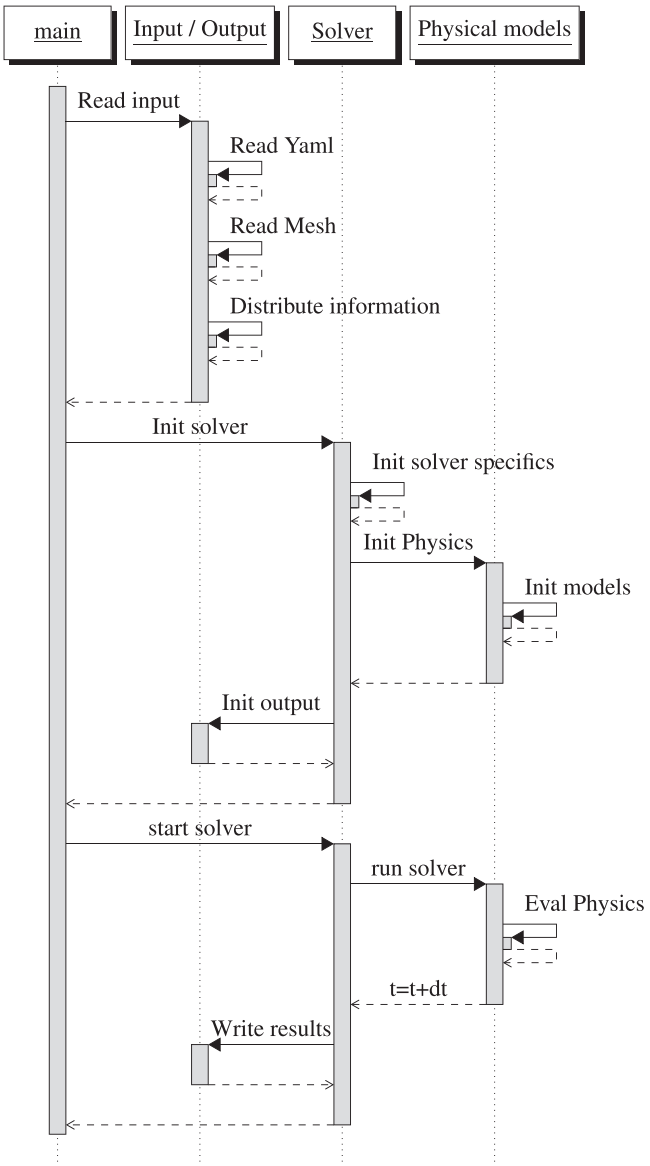

FIGURE 4 Program flow of PeriLab.

2.6 | Features

The system's design prioritizes ease of expansion and scalability. Consequently, we have avoided using structs in favor of a data manager, which plays a pivotal role in memory allocation and ensuring the assignment of unique names to variables or fields. This approach simplifies data access.

As already mentioned all input variables can be flexibly named, and they are automatically integrated into the data manager. If a field ends with "x," and the following one ends with "y," it is recognized as a multidimensional field. An example in the code is illustrated in Listing 1 (typically, a CSV file is read, yielding the same result). The reader utilizes the Dataframe.jl package to create a dataframe. This dataframe is analyzed, and fields are generated. The datafield entries x, y, and z are combined into "Coordinates" with a size equal to the number of nodes times three. Active fields such as active_x, active_y, and active_z have dimensions equal to the number of nodes times three, while "block_id" has a size equivalent to the number of nodes. The field size is determined by the "x," "y," or "z" in the field names. The coordinates x, y, and z define the spatial order of the problem. It is important to note that the mesh dof fields cannot exceed this dimension.

Time-dependent fields are accessible through "N" and "NP1," which correspond to the current and future time steps.

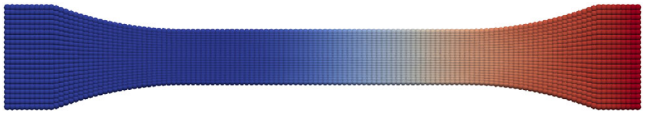

FIGURE 5 Displacement plot of a dogbone model example.

With configurable options, the data manager allows you to specify which parameters should be synchronized across all processing cores and in which directions. This approach streamlines access at the top layer, enabling a focus on model building and integration.

The central routine is the "compute_force" function, responsible for calculating force densities. You can choose to perform this calculation within this function or distribute it across multiple routines. Additionally, you can employ additional modules as needed. Material parameters are provided through the input deck and can be accessed using the specified parameter names. These names are consistent with the input deck. The data manager facilitates the creation of new fields or retrieval of data from existing fields. If you create a field with an existing name, you will obtain information about that field. Should you require information synchronization via overlap nodes, you can also define it using the data manager.

All these features collectively enable a straightforward and adaptable integration of new models into `PeriLab`. This approach remains consistent across various model types, including additive, damage, and thermal models.

Listing 1: Template for material modules.

```
module Material_template
export compute_forces
export material_name
function material_name()
 return "Material Template"
end
function compute_forces(datamanager, nodes, material_parameter, time, dt)
 return datamanager
end
end
```

The way it is done is the following. Julia allows the use of macros. These macros create code within code. The module folder are searched for the model modules and than they are include in the code. The code is run afterwards. For the compiler the code exists already. Therefore, there are no performance issues.

Listing 1: Automated integration of modules.

```
function create_module_specifics(name::String, module_list::Vector{Any},
    ↪ specifics::Dict{String,String}, values::Tuple)
for m in module_list
  parse_statement = module_name=" *m["Module Name"] * "." * specifics["Name"]* "
      ↪ ()"
  if eval(Meta.parse(parse_statement)) == name
    parse_statement = m["Module Name"] *"." * specifics["Call Function"]
    function_call = eval(Meta.parse(parse_statement))
    return function_call(values\ldots)
  end
end
end
```

3 | EXAMPLES

`PeriLab` OFFERS A RANGE OF BASIC examples to assist users in transitioning to more intricate geometries and applications. This paper highlights two specific models: a dogbone (refer to Figure 5) and a Compact Tension (CT) model (refer to Figure 6). It is important to note that these models showcase results generated from the finalized software, though ongoing work on verification and validation is still underway.

In the case of the dogbone model, which undergoes deformation when loaded on the left side, the results exhibit the anticipated uniformly increasing deformations. The model employs a linear elastic correspondence.

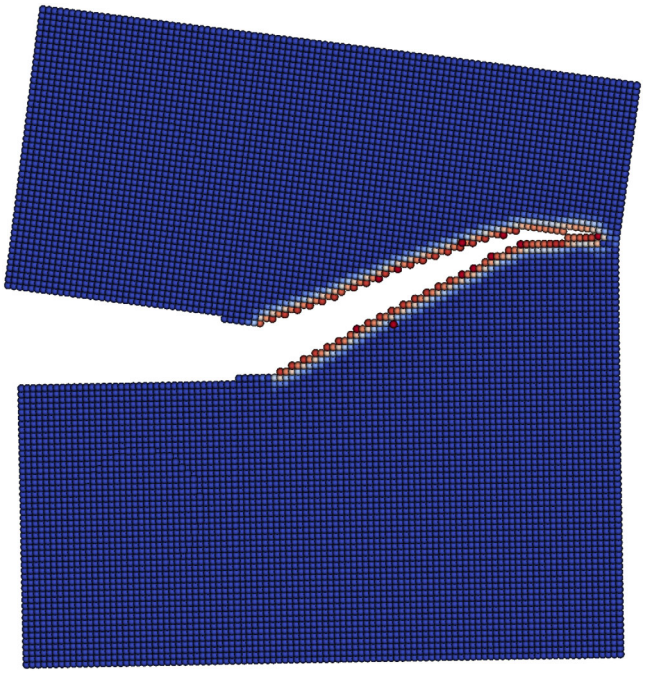

FIGURE 6 Scaled deformation and damage index plot of a CT specimen. CT, compact tension.

The second model incorporates an anisotropic energy-based damage model, initially developed by Foster et al. [25, 26]. The CT specimen features a pre-crack and is subjected to loading on the left-hand side, with the upper part pulled upwards and the lower part pulled downwards. This configuration initiates a deflected crack to the upper right corner, due to the anisotropic material behavior and a material orientation of 45°, as illustrated in the accompanying figure.

4 | CONCLUSION

The paper introduces `PeriLab`, an innovative simulation software tailored for PD applications. This software is meticulously crafted to enhance accessibility for both developers and users, with a primary emphasis on simplifying the integration of new capabilities. Notably, `PeriLab` supports the seamless incorporation and parallel computation of diverse models.

Julia was selected as the programing language for `PeriLab`, leveraging its optimal combination of speed and developer-friendly features. This strategic choice facilitates a smoother and more efficient development process.

In the paper, two models sourced from the software repository are computed and succinctly presented, offering a glimpse into the software's practical applications. The software is available as an open-source project. The overarching objective is to foster collaborative evolution, establishing `PeriLab`s a standard simulation tool within the PD domain. This collective effort aims to create a shared foundation for advancing simulation capabilities in the field.

ACKNOWLEDGMENTS

The work was funded by the Deutsche Forschungsgemeinschaft funded project: "Gekoppelte Peridynamik-Finite-Elemente-Simulationen zur Schädigungsanalyse von Faserverbundstrukturen" Grant number: WI 4835/5-1 and the M-ERA.NET funded project Exploring Multi-Method Analysis of composite structures and joints under consideration of uncertainties engineering and processing (EMMA). GRAPHIC 2 This measure is co-financed with tax funds on the basis of the budget passed by the Saxon State Parliament. Grant number: 3028223. The authors like to thank for the funding.

Open access funding enabled and organized by Projekt DEAL.

ORCID

Jan-Timo Hesse https://orcid.org/0000-0002-3006-1520
Anna Pernatii https://orcid.org/0000-0002-0004-0577

REFERENCES

1. Bobaru, F., Foster, J. T., Geubelle, P. H., & Silling, S. A. (2016). *Handbook of peridynamic modeling, advances in applied mathematics*. CRC Press.
2. Silling, S. A. (2000). Reformulation of elasticity theory for discontinuities and long-range forces. *Journal of the Mechanics and Physics of Solids*, *48*(1), 175–209.
3. Diehl, P., Lipton, R., Wick, T., & Tyagi, M. (2022). A comparative review of peridynamics and phase-field models for engineering fracture mechanics. *Computational Mechanics Volume*, *69*, 1259–1293.
4. Silling, S. A., & Askari, E. (2005). A meshfree method based on the peridynamic model of solid mechanics. *Computers and Structures*, *83*(17–18), 1526–1535.
5. Silling, S. A., Epton, M., Weckner, O., Xu, J., & Askari, E. (2007). Peridynamic states and constitutive modeling. *Journal of Elasticity*, *88*, 151–184.
6. Bobaru, F., & Duangpanya, M. (2010). The peridynamic formulation for transient heat conduction. *International Journal of Heat and Mass Transfer*, *53*(19–20), 4047–4059.
7. Littlewood, D. J., Parks, M. L., Mitchell, J. A., & Silling, S. A. (2013). The peridigm framework for peridynamic simulations. In: *12th U.S. National Congress on Computational Mechanics*, 22-25 July 2013, Raleigh, North Carolina, USA, No. SAND2013-5927C (Albuquerque, New Mexico 87185 and Livermore, California 94550, USA).
8. Mitchell, J. A., Silling, S. A., & Littlewood, D. J. (2015). A position-aware linear solid constitutive model for peridynamics. *Mechanics of Materials and Structures*, *10*(5), 539–557.
9. Hoppe, L. (2020). Numerical simulation of fiber-matrix debonding in single fiber pull-out tests. *GAMM Archive for Students*, *2*(1), 21–35.
10. Li, X., Ye, H., & Zhang, J. (2021). Redesigning peridigm on SIMT accelerators for high-performance peridynamics simulations. In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 433–443. IEEE.
11. Boys, B., Dodwell, T., Hobbs, M., & Girolami, M. (2021). Peripy - a high performance OpenCL peridynamics package. *Computer Methods in Applied Mechanics and Engineering*, *386*, 114085.
12. Jha, P. K., Desai, P. S., Bhattacharya, D., & Lipton, R. (2021). Peridynamics-based discrete element method (peridem) model of granular systems involving breakage of arbitrarily shaped particles. *Journal of the Mechanics and Physics of Solids*, *151*, 104376.
13. Delorme, R., Tabiai, I., Laberge Lebel, L., & Lévesque, M. (2017). Generalization of the ordinary state-based peridynamic model for isotropic linear viscoelasticity. *Mechanics of Time-Dependent Materials*, *21*(4), 549–575.
14. Delorme, R., Diehl, P., Tabiai, I., Lebel, L. L., & Lévesque, M., (2020). Extracting constitutive mechanical parameters in linear elasticity using the virtual fields method within the ordinary state-based peridynamic framework. *Journal of Peridynamics and Nonlocal Modeling*, *2*(2), 111–135.
15. Hobbs, M., Hattori, G., & Orr, J. (2022). Predicting shear failure in reinforced concrete members using a three-dimensional peridynamic framework. *Computers & Structures*, *258*, 106682.
16. Lehoucq, R. B., & Silling, S. A. (2008). Force flux and the peridynamic stress tensor. *Journal of the Mechanics and Physics of Solids*, *56*(4), 1566–1577.
17. Queiruga, A. F., & Moridis, G. (2017). Numerical experiments on the convergence properties of state-based peridynamic laws and influence functions in two-dimensional problems. *Computer Methods in Applied Mechanics and Engineering*, *322*, 97–122.
18. Jenabidehkordi, A., Fu, X., & Rabczuk, T. (2022). An open source peridynamics code for dynamic fracture in homogeneous and heterogeneous materials. *Advances in Engineering Software*, *168*, 103124.
19. Silling, S. A. (2017). Stability of peridynamic correspondence material models and their particle discretizations. *Computer Methods in Applied Mechanics and Engineering*, *332*, 42–57.
20. Parks, M., Littlewood, D., Mitchell, J., & Silling, S. (2022). Peridigm users' guide. *Tech. rep*. (Report SAND2012-7800). Sandia National Laboratories.
21. Rädel, M., & Willberg, C. (2018). PeriDoX, GitHub repository, 03 2018.
22. Littlewood, D. J., Parks, M. L., Foster, J. T., Mitchell, J. A., & Diehl, P. (2023). The peridigm meshfree peridynamics code. *Journal of Peridynamics and Nonlocal Modeling*.
23. Willberg, C., Hesse, J. T., & Heinecke, F. (2022). Peridynamic simulation of a mixed-mode fracture experiment in PMMA utilizing an adaptive-time stepping for an explicit solver. *Journal of Peridynamics and Nonlocal Modeling*, *5*, 205–228.
24. Willberg, C., Hesse, J. T., Garbade, M., Rädel, M., Heinecke, F., Schuster, A., & Pernatii, A. (2023). A user material interface for the Peridynamic Peridigm framework. *SoftwareX*, *21*, 101322.
25. Foster, J. T., Silling, S. A., & Chen, W. (2011). An energy based failure criterion for use with peridynamic states. *International Journal for Multiscale Computational Engineering*, *9*(6), 675–688.
26. Willberg, C., Wiedemann, L., & Rädel, M. (2019). A mode-dependent energy-based damage model for peridynamics and its implementation. *Journal of Mechanics of Materials and Structures*, *14*(2), 193–217.

How to cite this article: Hesse, J.-T., Willberg, C., & Pernatii, A. (2024). Reducing the entry barrier for users and developers of peridynamic high performance computing. *Proceedings in Applied Mathematics and Mechanics*, *24*, e202400160. https://doi.org/10.1002/pamm.202400160