Software Requirement Specifications for Multi-Physics Peridynamics Simulation Code

Michael L. Parks, David J. Littlewood, Stewart A. Silling, John A. Mitchell

January 8, 2010

1 Introduction

1.1 Purpose

This document states the requirements of a rapid production massively parallel multiphysics simulation code based upon peridynamics. These requirements stated serve as an acceptance procedure for this software system. This document contains design elements and is also intended as a starting point for a preliminary implementation.

1.2 Scope

This software system will provide a multiphysics simulation capability for peridynamic models. The working name for the code is ***Peridigm***.

What *Peridigm* is: The software product will perform multiphysics peridynamics simulations, be optimization-enabled, have born-in uncertainty quantification (UQ), exhibit good scalability, be usable as a stand-alone code, will contain modules suitable for incorporation into Sierra Mechanics where appropriate, and will allow for the potential integration of existing material models (e.g. the Lamé material library).

It will run in both in a serial (single processor) mode and in a parallel mode, using MPI. The implementation will allow for easy modification and extension with new functionality.

It will be open-source under a suitable public license. As per current 1400 policy, the goal is not to generate revenue through product sales, but instead to capture *mindshare* and recognition as *thought leaders* through development and innovation.

This software will be be maintained by the developers and new features will be added, pending continued funding, as driven by customer demands.

What *Peridigm* is not: This software system does not employ a graphical user interface, nor does it provide means for plotting or visualization of data. It does not automagically provide constitutive models for arbitrary materials or physics, nor a means to automagically couple arbitrary physics. It does not perform complex mesh/grid generation but will mesh simple geometric solids, including rectangular solids and cylindrical solids.

Objectives. This software system will constitute the first open source production quality massively parallel multiphysics peridynamics code, using the state-based formulation of peridynamics [6].

Benefit. This software system will provide a unified mathematical and computational framework for multiphysics simulations.

Goals. This software system will realize an instantiation of the plan and approach being developed in Andy Salinger's *Agile Components* work (e.g., demonstrate the approach and process of rapidly developing a production-quality application code from pre-existing components). In addition to illustrating how components can be assembled into a production code, the infrastructure needed to rapidly prototype a wider variety of simulation codes will be further established by producing any needed additional components.

1.3 Definitions, acronyms, and abbreviations

BLAS: Basic Linear Algebra Subroutines

Boost: The Boost C++ libraries are a collection of peer-reviewed, open source libraries that extend the functionality of C++.

CMake: Cross-Platform Make, a cross-platform, open-source make system. CMake is used to control the software compilation process using simple platform and compiler independent configuration files.

CTest: CTest is a testing tool distributed as a part of CMake, and is used to automate testing.

DAKOTA: The DAKOTA (Design Analysis Kit for Optimization and Terascale Applications) toolkit [1] provides a flexible, extensible interface between analysis codes and iteration methods.

PD: Peridynamics [4, 5, 6].

LAPACK: Linear Algebra PACKage

MPL: Metaprogramming library.

SRS: Software Requirement Specifications.

Subversion: Open-source revision control system.

TPL: Third Party Library

Trac: Project management and bug/issue tracking system with interface to Subversion and integrated wiki.

Trilinos: The Trilinos Project [2] is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems.

UQ: Uncertainty quantification.

VOTD: Version Of The Day.

1.4 References

References

[1] M. S. Eldred, B. M. Adams, K. Haskell, W. J. Bohnhoff, J. P. Eddy, D. M. Gay, W. E. Hart, P. D. Hough, T. G. Kolda, L. P. Swiler, and J.-P. Watson, *DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 4.2 reference manual*, Tech. Report SAND2006-4055, Sandia National Laboratories, Albuquerque NM, Updated November 2008. Available online from `http://www.cs.sandia.gov/dakota/documentation.html`.

[2] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the Trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.

[3] S. Silling, M. Zimmermann, and R. Abeyaratne, *Deformation of a peridynamic bar*, Journal of Elasticity, 73 (2003), pp. 173–190.

[4] S. A. Silling, *Reformulation of elasticity theory for discontinuities and long-range forces*, J. Mech. Phys. Solids, 48 (2000), pp. 175–209.

[5] S. A. Silling and E. Askari, *A meshfree method based on the peridynamic model of solid mechanics*, Comp. Struct., 83 (2005), pp. 1526–1535.

[6] S. A. Silling, M. Epton, O. Weckner, J. Xu, and E. Askari, *Peridynamic states and constitutive modeling*, J. Elasticity, 88 (2007), pp. 151–184.

[7] O. Weckner and R. Abeyaratne, *The effect of long-range forces on the dynamics of a bar*, Journal of the Mechanics and Physics of Solids, 53 (2005), pp. 705–728.

[8] O. Weckner, G. Brunk, M. A. Epton, S. A. Silling, and E. Askari, *Greens functions in non-local three-dimensional linear elasticity*, Proc. R. Soc. A, 465 (2009), pp. 3463–3487.

1.5 Overview

Section 2 provides an overview of the software's purpose and intended functionality. Included in Section 2 are a description of general user characteristics and the code's relationship to external libraries. Section 3 consists of detailed requirements descriptions for hardware and software interfaces, functionality and performance, development tools, and software testing. The appendix contains an outline for the order that features should be implemented.

2 Overall Description

2.1 Product perspective

This project is intended to realize an instantiation of the ongoing *Agile Components* work and also to provide the first open source production quality massively parallel multiphysics peridynamics code. As it matures, this code is intended to support multiphysics and also multiscale peridynamic simulations at Sandia and in the larger academic and industrial community.

The software deliverable will function as a stand-alone analysis code and will contain modules suitable for integration into Sierra Mechanics and/or other analysis codes where appropriate. Examples of functionality that will be modularized for incorporation into other analysis codes include:

- The calculation of peridynamic deformation states;
- The calculation of peridynamic force states;
- An interface allowing for the adaptation of classical material models for use in peridynamic analyses;
- Data structures for storing peridynamics-specific information (e.g., neighbor list).

The software system will be constructed utilizing Trilinos components. Figure 2 shows some of the functionality provided by Trilinos, and its expected order of integration into the software system.

To provide for UQ and optimization, the software system will interface with the DAKOTA project via the *TriKota* Trilinos package. TriKota automates the interface between DAKOTA and Trilinos-based codes, shown in Figure 3.

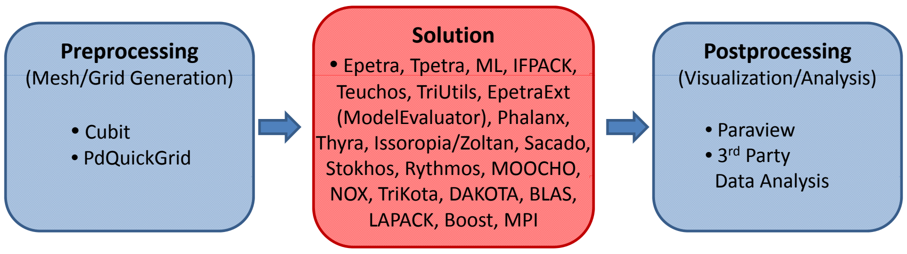
Preprocessing
(Mesh/Grid Generation)
• Cubit
• PdQuickGrid
Solution
• Epetra, Tpetra, ML, IFPACK, Teuchos, TriUtils, EpetraExt (ModelEvaluator), Phalanx, Thyra, Issoropia/Zoltan, Sacado, Stokhos, Rythmos, MOOCHO, NOX, TriKota, DAKOTA, BLAS, LAPACK, Boost, MPI
Postprocessing
(Visualization/Analysis)
• Paraview
• 3rd Party Data Analysis

Figure 1: Interaction with other software components/packages, distributed by stage of solution process.

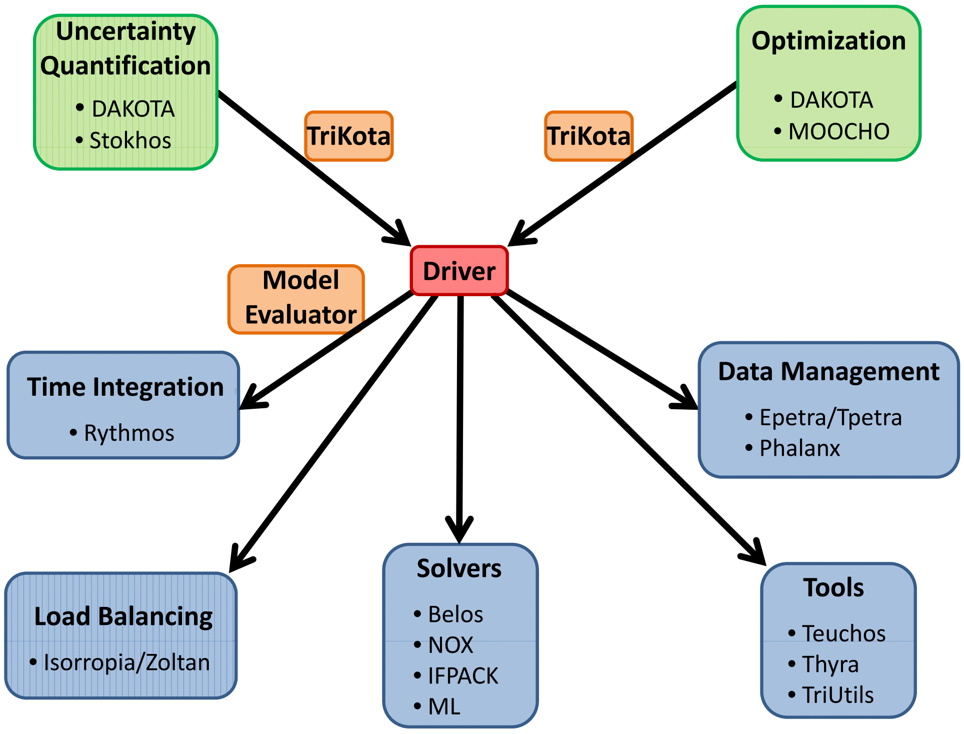

Figure 2: Interaction with other software components/packages used in solution stage, distributed by functional role.

2.2 Product functions

This software system will accept an input script and (possibly) other input data files describing materials and geometry, and perform a serial or parallel single- or multi-physics simulation based upon commands given in the input script. Simulations are either *dynamic* or *static*. *Explicit* and *implicit* time integrators are provided. Data will be output to disk (and optionally to the screen) as the simulation proceeds, followed by summary information when the simulation terminates. A more detailed description of the functions performed by this software system is given in Section 3.

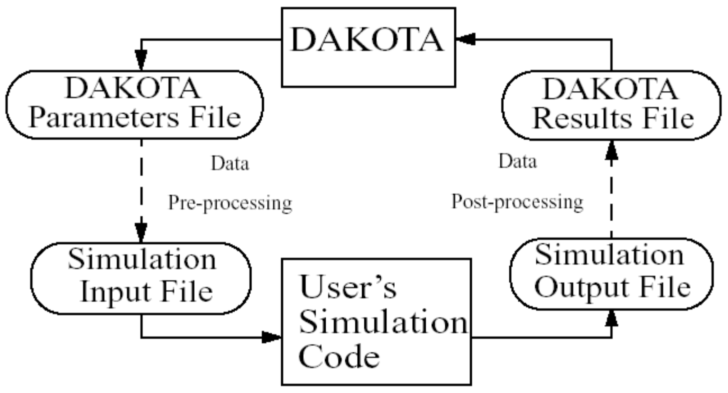

Figure 3: The loosely-coupled or "black-box" interface between DAKOTA and a user-supplied simulation code. For Trilinos software products, this functionality is achieved with the *TriKota* package. Image courtesy of [1].

2.3 User characteristics

It is assumed that users are knowledgeable in the physics they desire to simulate, and can specify peridynamic kernels appropriate to those physics (selecting from provided kernels if appropriate). It is also assumed users understand how to couple between these physical models (selecting from provided couplings if appropriate). We assume users have the skill to implement their own kernels and multiphysics couplings using provided hooks in the software if the kernels and multiphysics coupling provided in the code are not suitable for their simulations.

It is also assumed that users have at least intermediate level knowledge of the numerical algorithms employed in this software system and the other software components with which it interfaces and utilizes. For example, users are expected not to make foolish choices with regard to selection of discretizations, solvers, etc.

2.4 Constraints

This software system does not provide for interactive simulations. All simulations are driven via an input script or though direct coupling with a driving application, such as DAKOTA.

The primary code will be developed in C++. Developers will adhere to good software development practices. Parallelism will be achieved through MPI or through already-parallelized Trilinos components built upon MPI. Reliability is highly desirable – the code will exit gracefully upon failure or error, returning an informative error message.

Developers will utilize automated daily regression testing and pre-checkin tests. The test suite will contain, among other things, problems whose analytic solutions are known. (See "verification tests" in §3.6.) Basic correctness of code will judged by the ability of the code to reproduce analytic solutions up to the error introduced by numerical methods.

2.5 Assumptions and dependencies

This software system depends upon the Trilinos project [2] and the DAKOTA project [1], and assumes the correctness, completeness, and stability of these software projects. This software system is also then dependent upon all TPLs that utilized Trilinos packages are dependent upon (for example, BLAS, LAPACK, Boost, etc.) Further, the developers assume cooperation from Trilinos and Dakota developers in implementing any functionality in their packages deemed critical to the success of this project.

It is assumed the users will generate their own geometries and meshes via other third-party packages (e.g., Cubit), although we will provide mesh-generation tools for simple regular geometric solids. It is assumed the users will utilize other third-party packages for post-processing of data, including visualization (e.g., Paraview.) We will provide the capability for the user to specify in the input script an output format format suitable for direct input to a visualization package. (For example, the user could specify in the input script that the output be written directly to the VTK or Exodus formats, which are both readable by Paraview.)

3 Specific Requirements

3.1 External interface requirements

3.1.1 User interfaces

The user will interface with the code through input scripts and other input files, and also through output data files generated by the software system. Documentation and syntax of input script commands will be provided on a webpage or wiki. Sample input scripts will also be provided with the source distribution. The software system will provide a checkpoint/restart capability, allowing users to restart a computation from a restart file.

3.1.2 Hardware interfaces

Initial support only for RedHat, MacOS, and operating systems run by massively parallel DOE machines.

3.1.3 Software interfaces

This software system will interface with and directly utilize many packages within Trilinos, as well as linking with DAKOTA. The VOTD of this software system will build against the last stable VOTD of Trilinos and DAKOTA. Software that utilizes Peridigm modules, e.g. Sierra Mechanics, is expected to maintain compatibility with the last stable VOTD of Trilinos.

Specifically, we are likely to utilize the following software packages:

Epetra: Parallel numerical linear algebra. Also provides wrappers for BLAS & LAPACK.

Tpetra: Next-generation, templated version of Petra.

ML: Multilevel solvers.

Belos: Next-generation iterative solvers.

IFPACK: Distributed algebraic preconditioner package.

Teuchos: Common tools package.

TriUtils: Further Trilinos utilities.

EpetraExt: Matrix/Vector read/write utilities. Also contains *ModelEvaluator*, a nonlinear model evaluator.

Phalanx: Local field evaluation kernel.

phdMesh: Parallel Heterogeneous Dynamic unstructured Mesh (phdMesh) data structure library.

Intrepid: Compatible discretization library.

Cubit: Geometry and mesh generation toolkit.

Thyra: Set of interfaces and supporting code that defines basic interoperability mechanisms between numerical software.

Issoropia/Zoltan Parallel dynamic load balancing and related services.

Sacado: Automatic differentiation package.

Stokhos: Intrusive stochastic Galerkin uncertainty quantification methods.

Rythmos: Time integration methods.

MOOCHO: Multifunctional Object-Oriented arCHitecture for Optimization, a package for nonconvex optimization.

NOX: Nonlinear solvers.

TriKota: Trilinos/DAKOTA interface.

DAKOTA: Algorithms for optimization with gradient and nongradient-based methods; uncertainty quantification with sampling, reliability, polynomial chaos, and epistemic methods; parameter estimation with nonlinear least squares methods; and sensitivity/variance analysis with design of experiments and parameter study methods.

BLAS: Basic linear algebra subroutines.

LAPACK: Dense serial linear algebra.

Boost: Template metaprogramming library (MPL).

MPI: Message passing library.

3.1.4 Communication interfaces

No communications interfaces are supported.

3.2 Functional Requirements

Peridynamic simulations will be driven via an input script. The input script will contain information related to initialization, simulation, output, and termination.

Validity checks on inputs. Minimal validity checking will be performed. Validity checking will be limited to ensuring all necessary input is present for a simulation. (For example, ensuring the user has specified a mesh and at least one peridynamic model before beginning simulation.) Validity checking of input will occur at the time that portion of the input script is processed.

Initialization. The user shall specify

- a discretization (either as a file of mesh and connectivity information to be read in) or as a discretization supported internally by Peridigm (e.g., a regular geometric solid.)
- a material type (from the material library)
- desired peridynamic physics (embodied by a specific peridynamic kernel function) as well as coupling operations between peridynamic physics
- a solver (e.g., explicit time integrator and supporting parameters)
- output to disk and/or screen (optional)

It is assumed for initial versions of code that only discretizations of the strong form of the peridynamic operator are used, essentially following the method proposed in [5].

Initialization will occur following the reading of input data files and processing of the initialization section of the input script. The initialization phase includes initializing all variables and fields, handling parallel distribution of data, and initial load balancing.

Simulation. The user shall specify a command in the input script to run the simulation. The chosen time integrator or other solver drives the simulation. Via the ModelEvaluator interface, the peridynamic simulation code returns response variables to the time integrator. The code also handles the inter-processor communcation necessary to evaluate integrals of peridynamic kernel functions. Periodic load balancing shall occur as required for efficient computation. Output generated by the simulation shall be written to disk and/or screen, as specified in the input script.

Termination. Following the analysis prescribed in the input script the software system cleans up memory and closes files, reporting summary data to screen and/or disk as specified.

3.3 Performance Requirements

The software system is expected to achieve reasonable serial and parallel performance. Scaling tests will be constructed for evaluating parallel performance over a variety of problem sizes to identify possible performance bottlenecks.

3.4 Design constraints

The ModelEvaluator framework (EpetraExt) and/or associated Thyra wrappers are required for directly compatibility with a number of Trilinos packages. Interfaces in the software system must be designed accordingly.

3.5 Software system attributes

Reliability. The software system shall pass all unit and regression tests. "Almost continuous integration" will be utilized, keeping code current with Trilinos and DAKOTA VOTDs.

Availability. The software system shall support checkpointing, recovery, and restart functionality.

Security. None.

Maintainability. The software system shall be highly modular, as it is built upon Trilinos components, and utilize accepted means of connecting together those components. The software system shall modularize peridynamic kernel functions, allowing them to be easily added or modified. In particular, interfaces will be provided to both peridynamic constitutive models and classical material models, discussed below.

Peridynamic Constitutive Models: These material models, such as the linear peridynamic solid (LPS) model, will reside in the `src/materials` directory, and will all inherit from a common base class, thus presenting the same interface to the Peridigm code. This interface will facilitate the direct passing of deformation state data to the material model and the direct processing of subsequent force evaluations computed within the material model. Further, this interface allows end users to easily add their own peridynamic materials to the material library.

Classical Constitutive Models: For these material models (e.g., those used in classical finite element codes), a special material model interface will be provided, as discussed in [6]. The classical material model interface will allow for the passing of the peridynamic approximation of the deformation gradient (or quantities derived therefrom) to the material model. Additionally, the classical material model interface will contain functionality for converting stresses computed by a classical material model into force state data applicable to peridynamics.

To ensure compatibility with various Trilinos components (e.g., `EpetraExt::ModelEvaluator`), the Peridigm material interfaces will require material models to be stateless, i.e. no data will be stored within a material model. To accommodate history-dependent material models, Peridigm will allow for the storage of history-dependent material state data in a way that is both external to the material model itself and compatible with the Rythmos and NOX solvers provided in Trilinos.

Portability. The software system will be developed initially for a limited range of platforms, but will utilize as much platform independent code as possible. The build system shall be CMAKE, a platform-independent build system.

3.6 Other requirements

This section is devoted to a discussion of developer tools and resources that will be utilized.

Project Management Software We will utilize *Trac*, an open source project management tool. Trac is a lightweight project management tool that is implemented as a web-based application, written in the Python programming language. It emphasizes ease of use and low ceremony, and is open source. Trac supports many features, including a ticket system, progress tracker, online repository browsing and management, user management, a wiki, and other features available through plugins. the trac site is located at `https://development.sandia.gov/trac/peridigm`.

Compilers The primary compilers utilized will be the gcc (GNU compiler collection) v.4.x.x. Additionally, supporting shell or python scripts will be used.

Build system. Configuration and building will be managed with CMAKE.

Version Control. Subversion will be used to manage the software system.

Issue Tracking Trac will be used for issue/ticket/bug tracking. This may require customized third-party plugins for Trac, including but not limited to TracIncludeMacro, TracNav, TracDownloader, TracSpamFilter, and the TracTicketModerator plugin (developed by Sandia) for moderated ticket submission.

Software Testing Multiple test suites will be developed and maintained for the software: unit tests, regression tests, performance tests, and verification tests. The software and test suites will be monitored for memory errors using appropriate utilities (e.g. valgrind, libgmalloc), and test coverage will be monitored using a suitable coverage tool (e.g. gcov).

Unit Tests. Unit tests will be constructed using the Boost unit-testing framework.

Regression Tests. A regression test suite will be developed and maintained to reduce the introduction of bugs as new code is added to the software system. The CTest utility will be used to manage the regression test suite.

Performance Tests. A performance test suite will be developed and maintained to monitor code performance and scalability when performing large-scale analyses. The CTest utility will be used to manage the performance test suite.

Verification Tests. A verification test suite will be developed and maintained to demonstrate the software's ability to correctly simulate material response in a set of problems with known solutions. The CTest utility will be used to manage the verification test suite.

Each verification test will include a short write-up describing the test, along with computational and theoretical results. The collection of verification tests will comprise a verification manual for Peridigm. Theoretical solutions to peridynamic integral equations can be derived following results in [3, 7, 8].

Wiki and Web Resources We will utilize the Trac wiki to provides access to software documentation, release history, and access to the Peridigm subversion repository.

Mailing List(s) We will utilize mailing lists (via *MailMan*) on development.sandia.gov.

A Feature Implementation Outline

A.1 Initial Implementation

Simple driver application to solve a state-based PD linear elastic solid mechanics problem with no short-range forces (essentially, a simplified exploding cylinder problem.) The solution will be driven by a Rythmos time integrator (explicit time integration only) using velocity initial conditions. The software will be constructed using Epetra parallel data structures/functionality and the ModelEvaluator interface.

Specific functionality includes the ability to:

- Generate mesh for cylinder problem [Status: In progress.]
- Distribute mesh/bond data into Epetra data structures [Status: Completed.]
- Return force state for state-based linear peridynamic solid material model [Status: Completed.]
- Call time integrator (Rythmos) [Status: Completed.]
- Write output to file [Status: In progress.]

A.2 Future Implementation

By quarter for FY10, the remaining milestones are:

Q2 Implement core functionality for single-physics simulation. This is complete when we can explode a (mechanical)cylinder (with bond breaking) in parallel. (No short range forces.)

Q3 Implement core functionality for multi-physics simulation. This is complete when we can explode a (thermo-plastic) cylinder (with bond breaking) in parallel. (No short range forces.)

Q4 Multi-physics demonstration problem: cylinder fragmentation. This is complete when we can explode a thermo-plastic cylinder (with bond breaking and short-range forces) in parallel, and connect to Dakota for optimization/UQ.

More specific implementation tasks are listed below. Not all of these are necessary for the FY10 year-end deliverable, but all are desirable features.

- Implement multiple physics models and methods coupling between different physics via Phalanx. Specifically, heat transfer and solid mechanics.
- Utilize Zoltan for periodic load balancing and parallel data movement.

- Implement short-range force calculations, including efficient construction and updating of a neighbor list.
- Allow for applied boundary conditions.
- Connect to DAKOTA for optimization and UQ. Target problem is the analysis and optimization of the fragment size distribution of an exploding cylinder.
- Implement additional material models, e.g. elastic-plastic PD solid. Create interface for linking with classical mechanics material models.
- Read mesh data file (including neighborlist information) from disk.
- Implement checkpoint/restart functionality for use with large-scale analyses.