SAND2021-14998C

Sandia National Laboratories

Exceptional service in the national interest

An Introduction to the Peridigm Code

One Nonlocal World: Nonlocal Codes

David Littlewood

December 2nd, 2021

SAND2021-XXXX

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

The Peridigm Peridynamics Code

Meshfree peridynamics code for solid mechanics and multi-physics simulations

Open-source software originally developed at Sandia National Laboratories

C++ code that utilizes Sandia's *Trilinos* project

Designed for extensibility

Major capabilities:

- Bond-based, state-based, and NOSB constitutive models
- Implicit and explicit time integration
- Short-range force contact model
- MPI-based design enables large-scale parallel simulations

Compatible with pre- and post-processing tools

- Cubit mesh generator
- Paraview visualization code
- SEACAS utilities

Collaborators

David Littlewood

Michael Parks

John Mitchell

John Foster

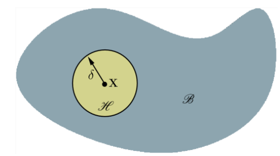
Patrick Diehl

Others …

Peridynamic Model of Solid Mechanics & Its Meshfree Discretization

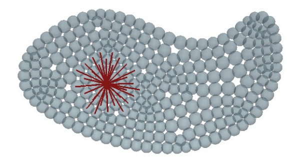
Balance of linear momentum

$$\rho(\mathbf{x})\ddot{\mathbf{u}}(\mathbf{x},t) = \int_{\mathscr{B}} \left\{ \underline{\mathbf{T}}[\mathbf{x},t]\left\langle \mathbf{x}' - \mathbf{x} \right\rangle - \underline{\mathbf{T}}'[\mathbf{x}',t]\left\langle \mathbf{x} - \mathbf{x}' \right\rangle \right\} \,\mathrm{d}V_{\mathbf{x}'} + \mathbf{b}(\mathbf{x},t)$$



Meshfree discretization of Silling and Askari

$$\rho(\mathbf{x})\ddot{\mathbf{u}}_h(\mathbf{x},t) = \sum_{i=0}^{N} \left\{ \underline{\mathbf{T}}[\mathbf{x},t]\left\langle \mathbf{x}_i' - \mathbf{x} \right\rangle - \underline{\mathbf{T}}'[\mathbf{x}_i',t]\left\langle \mathbf{x} - \mathbf{x}_i' \right\rangle \right\} \Delta V_{\mathbf{x}_i'} + \mathbf{b}(\mathbf{x},t)$$



State-based formulation supports wide variety of constitutive models

$$\underbrace{\underline{\mathbf{T}}[\mathbf{x},t]}_{\text{Force State}} \; \underbrace{\left\langle \mathbf{x}_i' - \mathbf{x} \right\rangle}_{\text{Bond}}$$

Material failure is modeled through the breaking/weakening of peridynamics bonds

Silling, Reformulation of elasticity theory for discontinuities and long-range forces. *Journal of the Mechanics and Physics of Solids* 48, 2000.

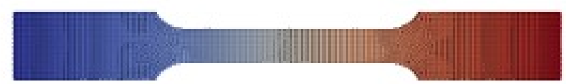
Silling and Askari, A meshfree method based on the peridynamic model of solid mechanics. *Computers and Structures* 83, 2005.

Silling, Epton, Weckner, Xu, and Askari, Peridynamic states and constitutive modeling, *Journal of Elasticity* 88, 2007.

Key Elements of a Peridynamics Code

- Discretization
- Proximity search
- Constitutive model
- Bond damage law
- Contact model
- Time integrator for explicit transient dynamics
  - Stable time step
- Implicit time integrator (quasi-statics)
  - Tangent stiffness matrix
  - Linear solver
- Pre- and post-processing

disk_impact
example problem



tensile_test
example problem

A Brief Tour of Peridigm Capabilities

Discretization

Meshless discretization
Conversion of hex/tet mesh
Bond filters
Exodus/genesis file format

Time integrator

Explicit dynamics (velocity Verlet)
Quasi-statics, implicit dynamics
Tangent stiffness matrix
Linear solvers (Trilinos)

Multi-physics

Mechanics, thermal /diffusion
General interface for one-way coupling
Capabilities for strongly coupled physics

Proximity search

Construction of neighbor lists
K-d tree, Zoltan (Trilinos)

Compute classes

Track quantities of interest
Neighborhood statistics
Many others

Constitutive models

General state-based formulation
Supports bond-based models
General interface for NOSB models
LPS, PMB, viscoelasticity, plasticity, others

Bond damage laws

Critical stretch
Damage for NOSB models

Contact

Global proximity search
Short-range force contact model

A Brief Tour of Peridigm Capabilities

Discretization

Meshless discretization
Conversion of hex/tet mesh
Bond filters
Exodus/genesis file format

Proximity search

Construction of neighbor lists
K-d tree, Zoltan (Trilinos)

Constitutive models

General state-based formulation
Supports bond-based models
General interface for NOSB models
LPS, PMB, viscoelasticity, plasticity, others

Time integrator

Explicit dynamics (velocity Verlet)
Quasi-statics, implicit dynamics
Tangent stiffness matrix
Linear solvers (Trilinos)

Compute classes

Track quantities of interest
Neighborhood statistics
Many others

Bond damage laws

Critical stretch
Damage for NOSB models

Multi-physics

Mechanics, thermal /diffusion
General interface for one-way coupling
Capabilities for strongly coupled physics

Contact

Global proximity search
Short-range force contact model

Leverage computer science domain expertise → performance, scaling

Obtaining and Building Peridigm

1. Install required compilers, if necessary
   - A modern, mainstream linux complier (gcc, clang, etc.)
   - A MPI compiler (mpich, openmpi, etc.)
2. Download and build the required dependencies
   - Boost, netcdf, hdf5, SuperLU, YAML
   - Trilinos https://trilinos.github.io/
3. Clone the Peridigm git repository
   - https://github.com/peridigm/peridigm
4. Build Peridigm

⚠CAUTION

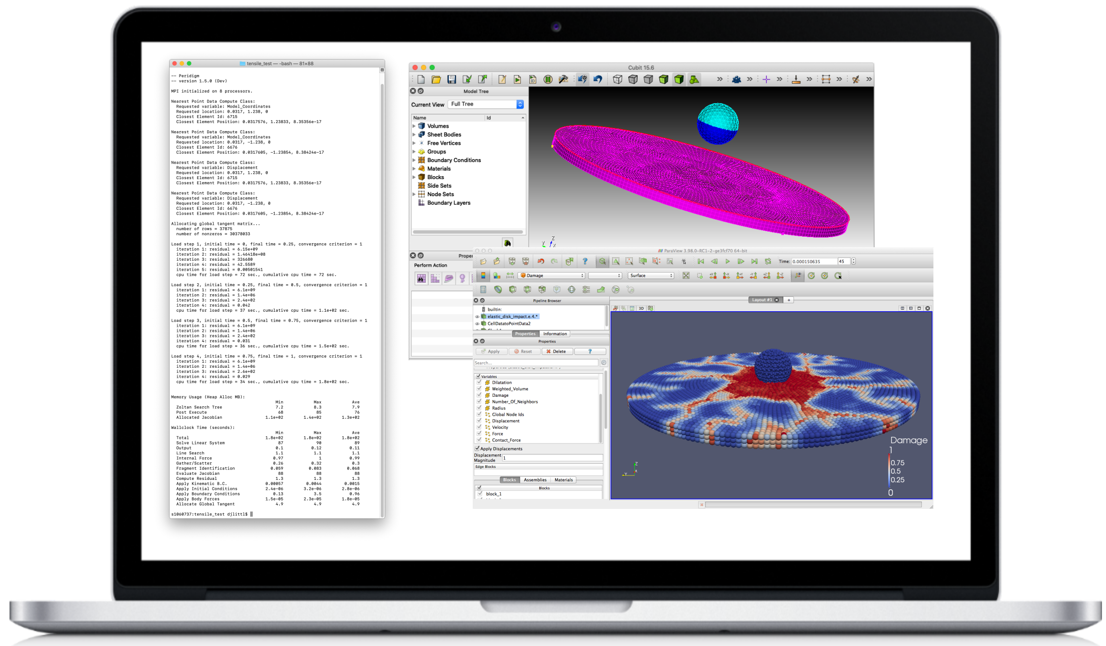
The Trilinos build process is complex, with many configuration options. See Peridigm documentation for guidance.

```
cmake \
-D CMAKE_BUILD_TYPE:STRING=Release \
-D TRILINOS_DIR:PATH=/YourPath/CLANG_8.0_GCC_10.2_MPICH/lib/cmake/Trilinos \
-D CMAKE_C_COMPILER:STRING=/opt/local/bin/mpicc \
-D CMAKE_CXX_COMPILER:STRING=/opt/local/bin/mpicxx \
-D CMAKE_CXX_FLAGS:STRING="-O2 -Wno-long-long -ftrapv -std=c++11" \
../peridigm
```

Example of a Peridigm build script

Performing an Analysis with Peridigm

1. Create a discretization (mesh file)
2. Create an input deck
3. Run the Peridigm executable
4. Post-process the results



Performing an Analysis with Peridigm

1. Create a discretization (mesh file)

Option 1) Genesis file

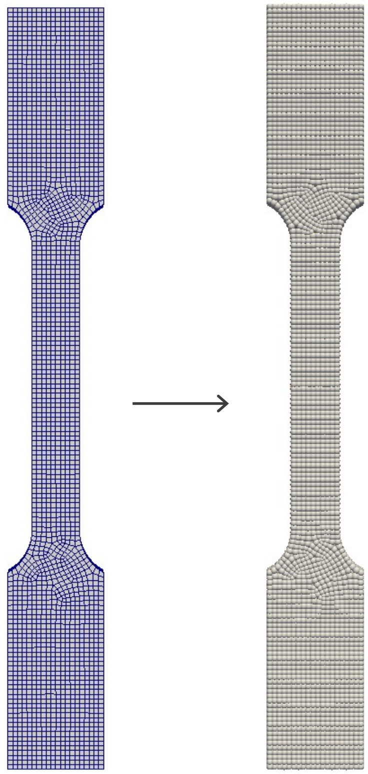
- Cubit mesh generator (hexahedron or tetrahedron mesh)
- Designate blocks and node sets
- Genesis sphere meshes also supported

Option 2) Text file

- Discretization defined by (coordinates, volume, block id) at each node
- User-supplied node sets (lists of node ids)

Option 3) Internal mesh generator

- Rectangular or cylindrical solid
- Restricted to single block
- Primarily used for code testing



Peridigm can convert a hex/tet mesh into a meshless discretization

Performing an Analysis with Peridigm

2. Create an input deck

- The input deck is an text file
  - Yaml format is preferred
  - Xml formal also available (depreciated)
- Simulation parameters are logically organized
  - Material models
  - Contact models
  - Boundary conditions
  - Time integrator / solver
  - File I/O
- Material models are associated with regions of the discretization using **blocks**
- Boundary conditions are associated with regions of the discretization using **node sets**

```
Materials:
    My Material:
        Material Model: "Elastic"
        Density: 8.0
        Bulk Modulus: 1.500e12
        Shear Modulus: 6.923e11
Blocks:
    My Block:
        Block Names: "block_1 block_2"
        Material: "My Material"
        Horizon: 0.16
Boundary Conditions:
    My Prescribed Displacement:
        Type: "Prescribed Displacement"
        Node Set: "nodelist_1"
        Coordinate: "y"
        Value: "0.005*t”
Solver:
    Final Time: 1.0
        QuasiStatic:
            Number of Load Steps: 4
            Absolute Tolerance: 1.0
            Maximum Solver Iterations: 10
Output:
    Output Filename: ”my_simulation_results"
    Output Frequency: 2
    Output Variables:
        Number_Of_Neighbors: true
        Displacement: true
```

Performing an Analysis with Peridigm

3. Run the Peridigm executable

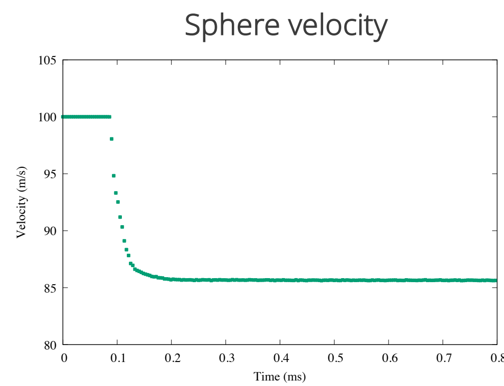
- Use a linux-like platform
  - Linux, macOS
- Hardware requirements depend on simulation size
  - Laptop, desktop, cluster
- Run Peridigm from the command line
- Serial or parallel execution (MPI)
  - Use SEACAS tools to decompose mesh

```
$ mpirun -np 8 Peridigm disk_impact.yaml

-- Peridigm
-- version 1.5.0 (Dev)

MPI initialized on 8 processors.

Time step (seconds):
  Stable time step    1.59402e-07
  User time step      not provided
  Safety factor       0.7
  Time step           1.11581e-07

Total number of time steps 7169

Explicit time integration [==================100% Complete]

Wallclock Time (seconds):
                                Min          Max          Ave
  Total                     3.9e+02      3.9e+02      3.9e+02
  Rebalance                 1.2e+02      1.2e+02      1.2e+02
  Output                        9.6           10          9.9
  Internal Force            1.3e+02      2.1e+02      1.8e+02
  Apply Kinematic B.C.       0.0028       0.0053       0.0036
```

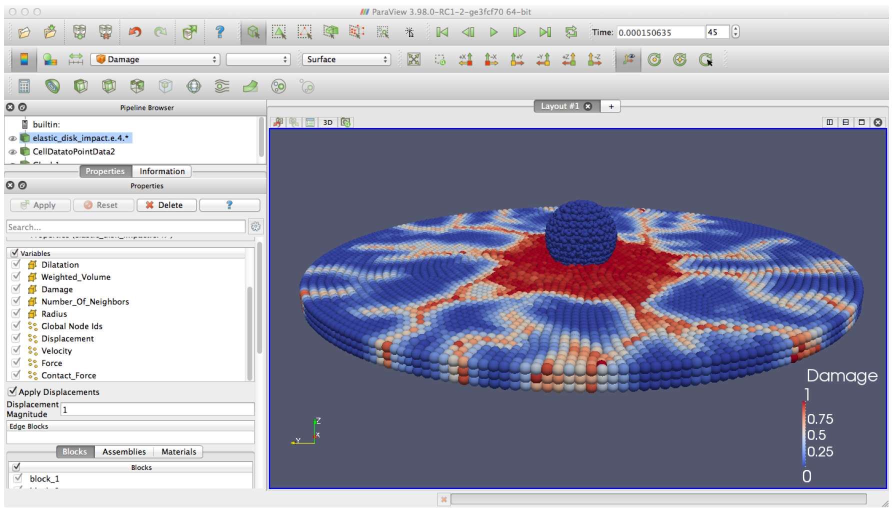
Performing an Analysis with Peridigm

4. Post-process the results
   - Paraview
   - Python (exodus.py module)
     - Process data directly in a python script
     - Extract data into your preferred format
   - SEACAS tools
     - epu, exodiff, blot
   - MATLAB
     - Exodus is built on netcdf

Plot created with blot and gnuplot



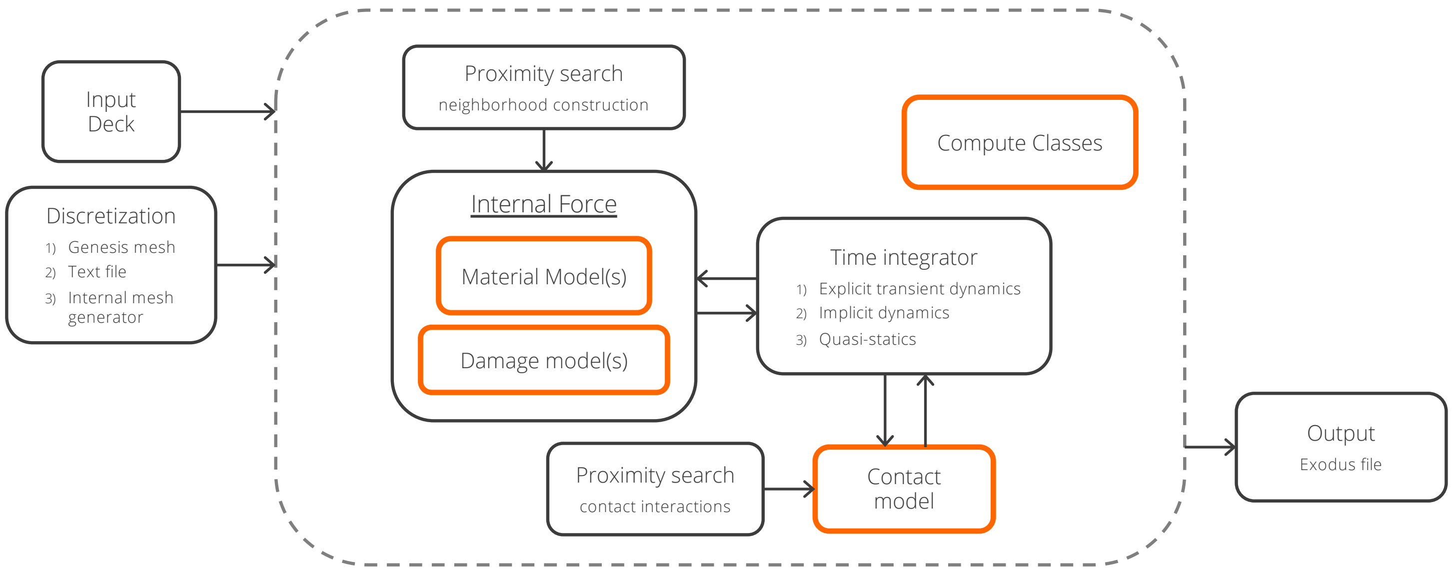
Paraview visualization



Developing in Peridigm

Peridigm software design

- State-based peridynamics
- Explicit and Implicit time integration
- Contact
- Massively parallel
- Performance
- Extensibility



Orange denotes extensible components

Extending Peridigm: Adding a Material Model

Peridigm material models derive from the `Material` base class

Developers specify the required data fields and implement functions that define model behavior

Required routines

```
void computeForce(const double dt,
                  const int numOwnedPoints,
                  const int* ownedIDs,
                  const int* neighborhoodList,
                  DataManager& dataManager) const;
```

Optional routines

```
void computeJacobian(const double dt,
                     const int numOwnedPoints,
                     const int* ownedIDs,
                     const int* neighborhoodList,
                     DataManager& dataManager,
                     SerialMatrix& jacobian,
                     JacobianType jacobianType)) const ;

Others …
```

Thank you!

https://github.com/peridigm/peridigm

GitHub