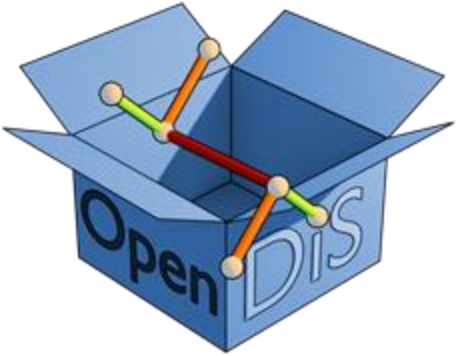


https://opendis.github.io/OpenDiS/

OpenDiS Workshop

Open Dislocation **S**imulator

Nicolas Bertin, Lawrence Livermore National Laboratory
Wei Cai, Stanford University

Welcome! Here is the Zoom protocol:

1. Please mute your microphone
2. Feel free to turn on your camera (optional/encouraged)
3. Feel free to "raise hand" if you have a question/comment
4. Feel free to send your question/comment to chat

LLNL-PRES-2006898

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 Lawrence Livermore National Security, LLC.

Objectives of this workshop

- Understand the framework
- Walk through the basic concepts
- Run some examples
- Implement some modules
- Contribute code
- Long-term objective: build a community of contributors

Agenda

- **Day 1**: 4 hours, Scope: user's tutorial
  - **Session 1**: 1.5 hour, overview of the project, download code, run a python example, code structure, basic classes
  - Break: 0.5 hour
  - **Session 2**: 1.5 hour, interactive session, build and run case studies, examine results
  - Q/A: 0.5 hour
- **Day 2**: 4 hours, Scope: developer's tutorial
  - **Session 3**: 1.5 hour, how to implement a new module (example: mobility law, output vtk files)
  - Break: 0.5 hour
  - **Session 4**: 1.5 hour, how to add user-defined enhancements (examples: stop simulation based on user-defined criteria, different loading conditions, impenetrable obstacles to dislocations, etc.)
  - Q/A: 0.5 hour

https://opendis.github.io/OpenDiS/

OpenDiS Workshop

Open Dislocation **S**imulator

Session 1

Nicolas Bertin, Lawrence Livermore National Laboratory
Wei Cai, Stanford University

LLNL-PRES-2006898

Welcome! Here is the Zoom protocol:

1. Please mute your microphone
2. Feel free to turn on your camera (optional/encouraged)
3. Feel free to "raise hand" if you have a question/comment
4. Feel free to send your question/comment to chat

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 Lawrence Livermore National Security, LLC.

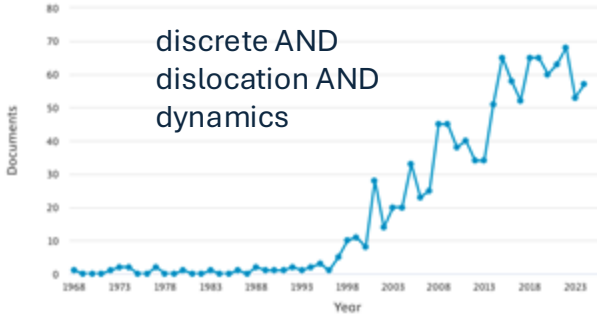
Outline of Session 1

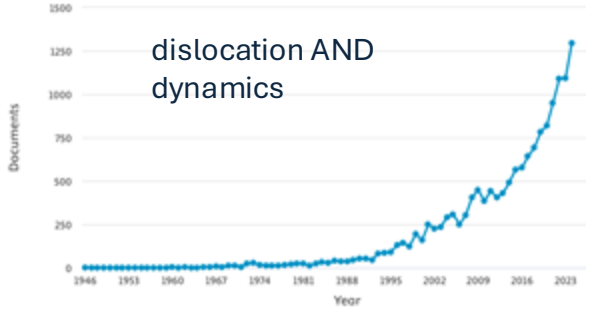
- **Project Overview**
  - History of DDD, significance, and challenges
  - General overview of OpenDiS
- **A Sneak-peek of OpenDiS**
  - Download OpenDiS
  - Run a simple Python example
  - Explore DisNet data structure
- **Code Structure**
  - OpenDiS core libraries: pydis and exadis
  - Module concepts
  - Data structures (state dictionary, DisNetManager)
  - Available modules

Introduction and history of DDD

- **1987:** DDD first developed for 2D systems by Lepinoux and Kubin
- **1992:** Approach and extended to three dimensions by Kubin and Canova
- **~2000s:** Zbib et al. (1998), Swartz (1999), Ghoniem et al. (2000), Weygand et al. (2003), etc.
- **The DDD method has gained wide use in modeling mechanical deformation of materials**
- Few notable DDD papers:
  - Bulatov et al., *Nature* (1998)
  - Madec et al., *Science* (2003)
  - Bulatov et al., *Nature* (2006)
  - Devincre et al., *Science* (2008)

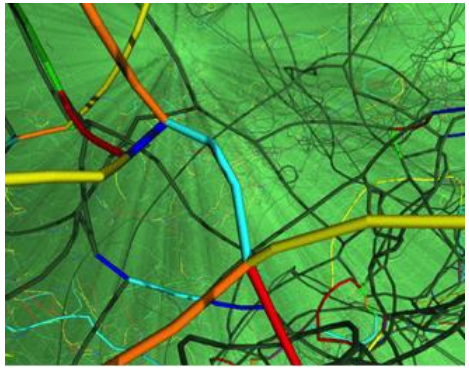
Publications

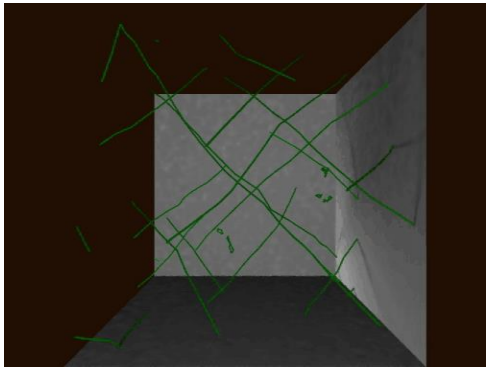




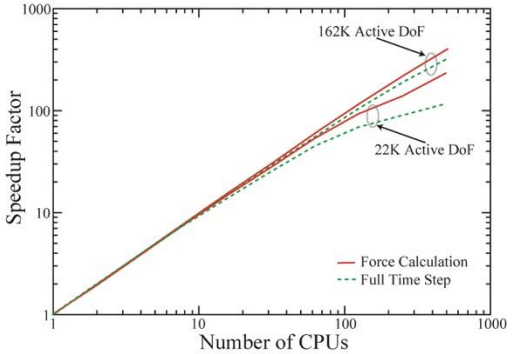
The ParaDiS project

- **Started development circa 2000**
  - ~20 core contributors
  - Written in C
  - 100,000+ lines of codes
  - CPU code
  - Massively parallel
  - Simulations on 100k+ CPUs
  - De facto standard for DDD simulations (widely adopted)





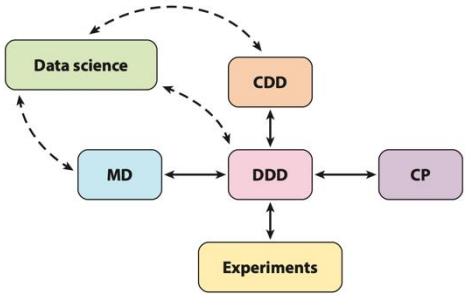
- **Stopped development in 2024**
  - Lack of flexibility / not easy to extend
  - Lack of compactness
  - Data structure
  - Designed for CPU
  - Branching LLNL/Stanford
  - Licensing

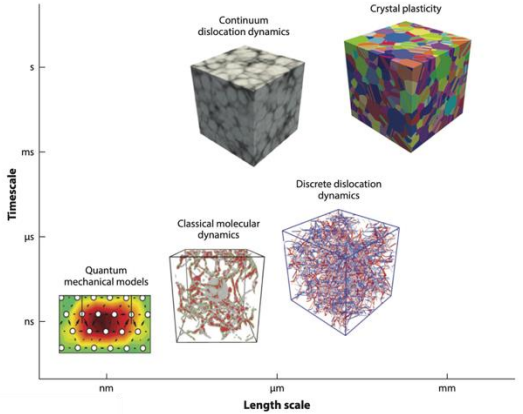


Arsenlis et al. "Enabling strain hardening simulations with dislocation dynamics." MSMSE 15(6) (2007)

Why DDD is important, what can we hope to solve with it

- Collective behavior of dislocations governs mechanical behavior of metals under most conditions
- DDD is at the center of the multiscale modeling framework of plasticity!

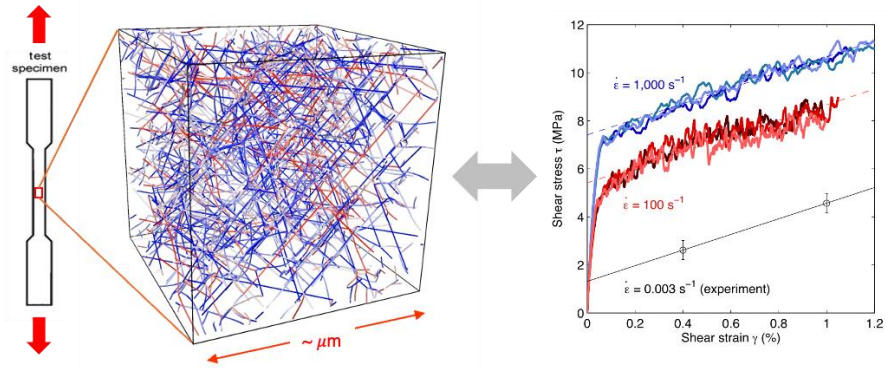




Annual Review of Materials Research, 50, 437 (2020).

DDD can provide:

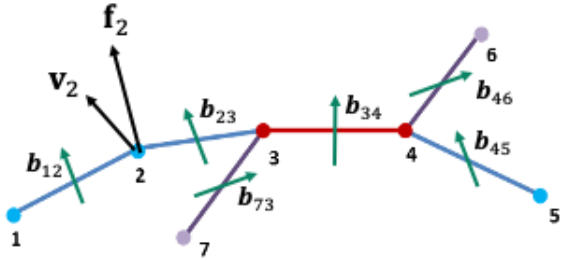
- Understanding fundamental dislocation mechanisms
- Connection between microstructural evolution to macroscopic behavior
- Constitutive models parametrization
- Explicit dislocation network configurations



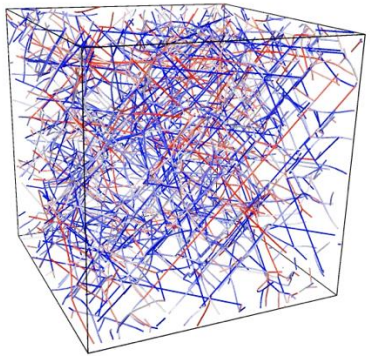
Physical Review Letters, 121, 085501 (2018).

Recap of the (nodal) DDD method

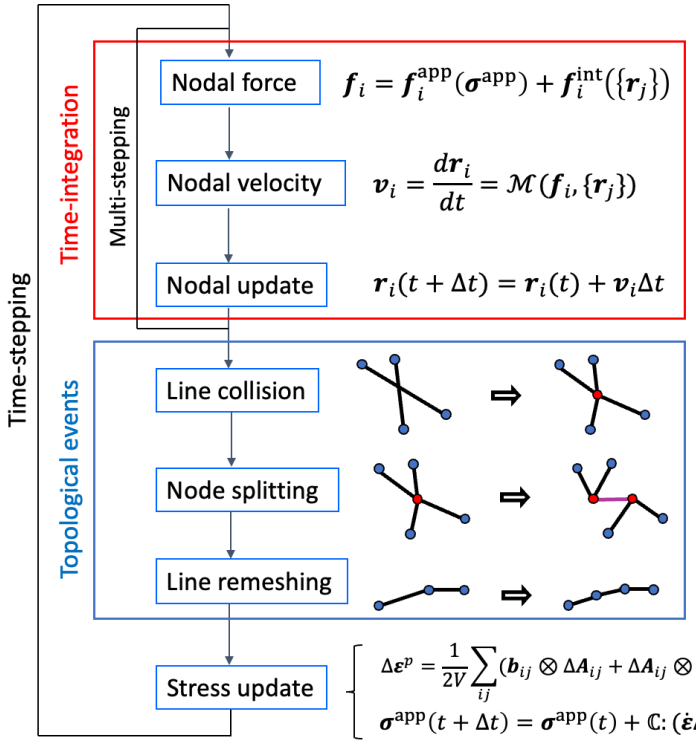
- **Microstructure representation**

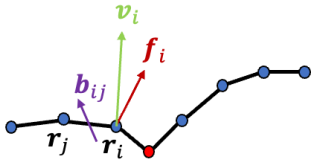


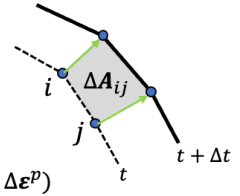
Degrees of freedom:
- nodal positions $\{\boldsymbol{r}_i\}$
- Burgers vectors $\{\boldsymbol{b}_{ij}\}$



- **DDD cycle**







Challenges and need for a community-driven project

- **DDD is a fairly mature method, BUT**
  - Remains computationally expensive
  - Needs new algorithmic developments
  - Needs to include more/new physics
  - Material systems become increasingly complex (e.g. HEAs)

- **DDD development cannot remain a set of isolated efforts if we want to make serious progress**

- **Previous ParaDiS effort is not the right way to go anymore…**
  - Lack of community effort because of initially restrictive license
  - Code branches out (ParaDiS 2.7, ParaDiS 4.0, all other in-house versions)
  - Everybody spends time re-implementing the same things over and over again…

Why OpenDiS?

To address design limitations of existing DDD codes:

- **Easy to use:** python interface
- **Easy to explore:** python prototyping
- **Easy to extend:** modular design
- **Easy to couple:** interface to core and extern libraries
- **Efficient:** natively supports GPU execution
- **Open-source license:** community-driven development

Manifesto

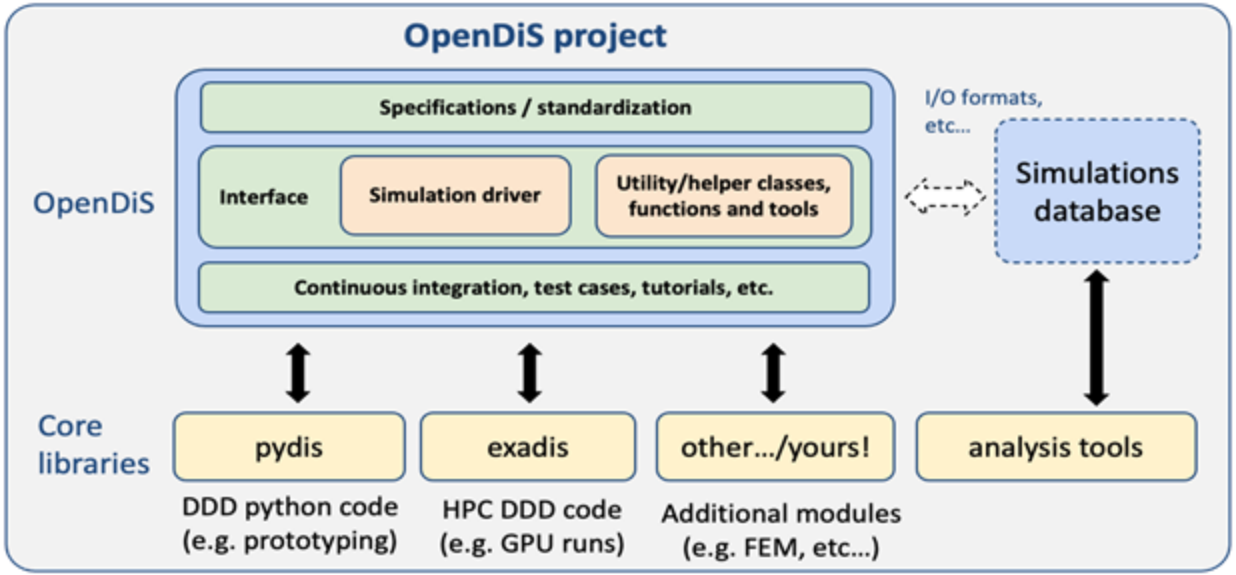
Our goals

- **Open Access**: We maintain a public Git repository that welcomes all to download and use the code.
- **Community Development**: We prioritize ease of development for all, making it accessible for anyone to create extension modules and contribute to the expanding OpenDiS repository.
- **Embracing Innovation**: We harness the power of existing and emerging computing architectures (including GPU and massively parallel computing) thus ensuring that OpenDiS stays at the forefront of computational efficiency.

Key features

- **Modular Design with Extensive Extensions**
- **Built-in Python Interface:**
- **Replaceable HPC modules**
- **Testing and Continuous Integration**

General overview of the OpenDiS framework



- **Central layer:** provides higher level specifications, utilities, and helper classes
- **Core libraries:** implement fundamental functions for performing DDD simulations
- **Standardization:** simulation reproducibility and data analysis efficiency

General overview of the OpenDiS framework

- **OpenDiS layer**
  - Backbone of the OpenDiS framework and **provides specifications, interfaces, drivers, utilities**, and continuous integration features
  - **Written in python** to simplify the interaction between the different core modules and/or external libraries, and **to facilitate user-defined enhancements**
  - The specifications **define the overall rules and standards for module implementations** (e.g. as a set of abstract python classes), **to ensure compatibility** and integration across the different modules

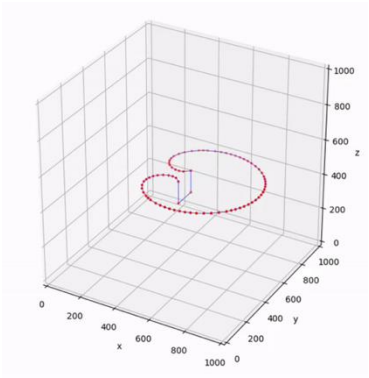
- **Core libraries**
  - The core libraries **provide the computational engine for DDD simulations** and implement functions, algorithms, and solvers in the form of modules (e.g. mobility law module, time-integration module, etc.)
  - OpenDiS provides the following two native core libraries:
    - **pydis**: a Python-based library for learning, prototyping, and running smaller-scale simulations.
    - **exadis**: a C++-based library with Python interface (pyexadis) developed for high-performance and GPU computing, intended for large-scale and production simulations.
  - The modular design of OpenDiS allows users to develop, couple, and contribute their own core libraries to enhance the capabilities of the existing framework.

OpenDiS core libraries

PyDiS

(**Py**thon **Di**slocation **S**imulator)

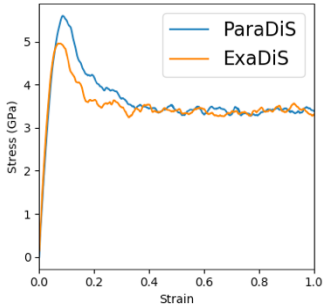
- DDD code written in python
- Serves for educational purposes
- Enables rapid prototyping / testing of new modules
- Manipulate and analyze dislocation structures
- Run small-scale simulations
- Modular design



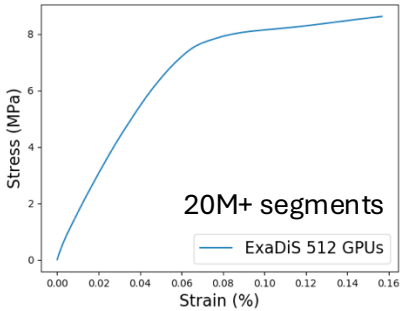
ExaDiS

(**Exa**scale **Di**slocation **S**imulator)

- Core HPC DDD library that is replacing ParaDiS
- Based on Kokkos for portability: runs on CPU, OpenMP, NVIDIA GPU, AMD GPU, etc.
- Execution and memory patterns abstractions
- C++ backend / python interface
- Modular design



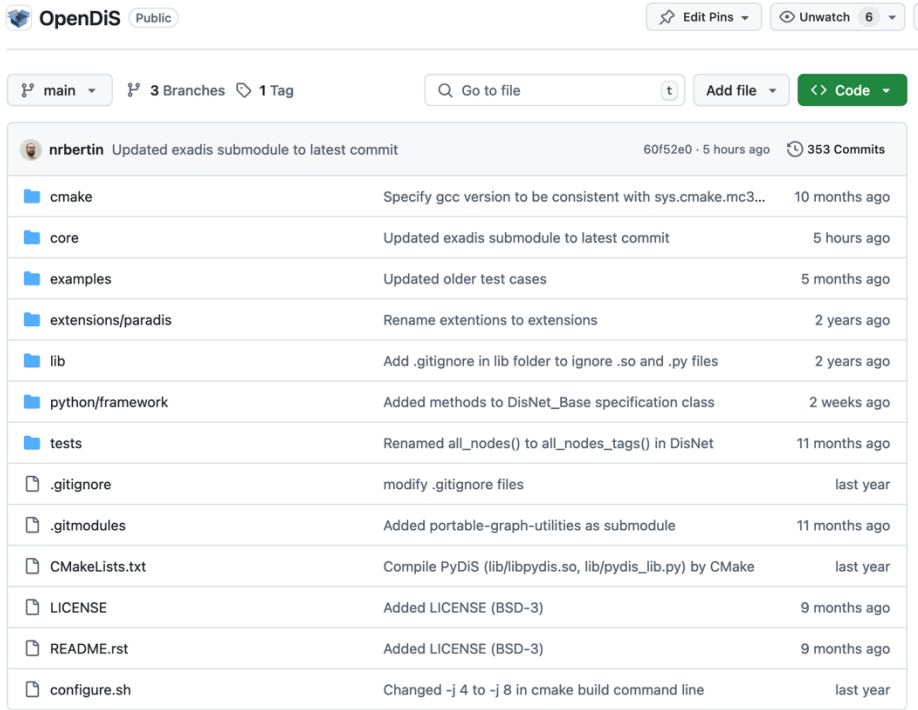
ParaDiS 8CPUs ~2.5 days
ExaDiS 1GPU ~20 min



ExaDiS running on
512 GPUs (V100s)

Structure of the code repository

https://github.com/OpenDiS/OpenDiS



Outline of Session 1

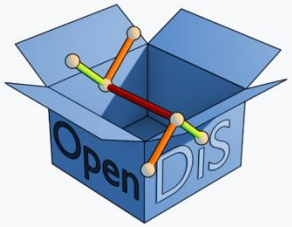
- **Project Overview**
  - History of DDD, significance, and challenges
  - General overview of OpenDiS

- **A Sneak-peek of OpenDiS**
  - Download OpenDiS
  - Run a simple Python example
  - Explore DisNet data structure

- **Code Structure**
  - OpenDiS core libraries: pydis and exadis
  - Module concepts
  - Data structures (state dictionary, DisNetManager)
  - Available modules

Download OpenDiS

Documentation website
https://opendis.github.io/OpenDiS



OpenDiS 0.1.0 documentation

Search

- OpenDiS Manifesto
- Installation
- Tutorials
- Code Structure
- Core Libraries

Welcome to OpenDiS!

Revolutionizing Dislocation Dynamics Simulations Through Open Source Collaboration

The OpenDiS Project is a community-driven initiative aimed at developing a robust open-source code framework and a code development platform for dislocation dynamics (DD) simulations. Our mission is to provide a high-performance, accessible, configurable, and extensible tool enabling researchers to explore the intricate world of dislocation lines and their impact on materials behavior.

We invite you to join the OpenDiS project to shape the future of dislocation dynamics. Together, we can build an open-source code and develop an open collaboration platform for accelerating scientific innovation.

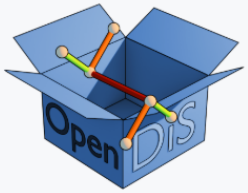
Read the full OpenDiS Manifesto

Git repository

The **OpenDiS** code is hosted on Github at OpenDiS Github repo. For questions or issues, open an issue.

Download OpenDiS

Documentation website
https://opendis.github.io/OpenDiS



OpenDiS 0.1.0 documentation

Search

- OpenDiS Manifesto
- Installation
  - How to get the code
  - System Requirements
  - Compilation
- Tutorials
- Code Structure
- Core Libraries

How to get the code

Important

For clarity, throughout this documentation variable `OPENDIS_DIR` is used to designate the directory that contains your downloaded OpenDiS code. For instance, you may set this directory via an environment variable `OPENDIS_DIR` using

```
export OPENDIS_DIR=${HOME}/Codes/OpenDiS.git
```

export OPENDIS_DIR=${HOME}/Codes/OpenDiS_demo.git

You may specify a different path that you prefer. If you use `bash`, you can include the above line in your `~/.bash_profile` file so that this line is automatically executed every time you log in.

Download OpenDiS together with submodules

Use the following steps to download OpenDiS into the location specified by your `${OPENDIS_DIR}` variable.

```
mkdir -p ${OPENDIS_DIR}
git clone --recurse-submodule https://github.com/OpenDiS/OpenDiS.git ${OPENDIS_DIR}
cd ${OPENDIS_DIR}
```

Download OpenDiS

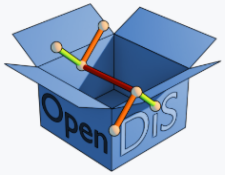
Documentation website
https://opendis.github.io/OpenDiS

Demo on MacBook

```
(base) Cai-Mac3:~ caiwei$ export OPENDIS_DIR=${HOME}/Codes/OpenDiS_demo.git


(base) Cai-Mac3:Codes caiwei$ mkdir -p ${OPENDIS_DIR}
(base) Cai-Mac3:Codes caiwei$ git clone --recurse-submodule https://github.com/OpenDiS/OpenDiS.git ${OPENDIS_DIR}
Cloning into '/Users/caiwei/Codes/OpenDiS_demo.git'...
remote: Enumerating objects: 7374, done.
remote: Counting objects: 100% (610/610), done.
remote: Compressing objects: 100% (115/115), done.
remote: Total 7374 (delta 529), reused 530 (delta 495), pack-reused 6764 (from 2)
Receiving objects: 100% (7374/7374), 13.52 MiB | 15.17 MiB/s, done.
Resolving deltas: 100% (4691/4691), done.


Submodule path 'core/exadis/kokkos': checked out '0d4a2d38ba14db2a741575639b40c31f3232185e'
Submodule path 'core/exadis/python/pybind11': checked out '19a6b9f4efb569129c878b7f8db09132248fbaa1'
Submodule path 'core/pydis/python/ctypesgen': checked out '0f0c5d69e56f66e0ed7788058c31f423c6b08aab'
Submodule path 'core/pydis/python/pydis/graph': checked out '118aff2c5bde4db24c419fb5019f2275e78c1206'
(base) Cai-Mac3:Codes caiwei$ cd ${OPENDIS_DIR}
(base) Cai-Mac3:OpenDiS_demo.git caiwei$ ls
cmake          configure.sh   examples       lib            python         tests
CMakeLists.txt core           extensions     LICENSE        README.rst
```

Run OpenDiS Example

Documentation website
https://opendis.github.io/OpenDiS



OpenDiS 0.1.0 documentation

Search

- OpenDiS Manifesto
- Installation
- Tutorials
  - Frank-Read Source
    - **Frank-Read Source by Pure Python**
    - Frank-Read Source by Python calling ExaDiS
    - Graph Data Conversion
    - Inter-operability between PyDiS and ExaDiS Modules
  - Binary Junction
  - Strain Hardening
  - Creating initial dislocation configurations

Frank-Read Source by Pure Python

We can run the following test case without having to compile OpenDiS.

To run the simulation, simply execute:

```
cd ${OPENDIS_DIR}
cd examples/02_frank_read_src
python3 -i test_frank_read_src_pydis.py
```

Initial Condition

The initial configuration of this simulation is a rectangular (prismatic) dislocation loop, where all four sides are of pure edge type. Their Burgers vectors are normal to the plane of the rectangular loop. All four corner nodes have their constraint = pinned (PINNED_NODE). This is sufficient to pin the bottom and two side arms of the prismatic loop. However, the top arm contains a node (in the middle) that is free to move. This allows the top arm to bow out (under the action of applied stress) and act as a Frank-Read source.

The data structure of the initial condition is explained in Section DisNet Class.

Boundary Condition

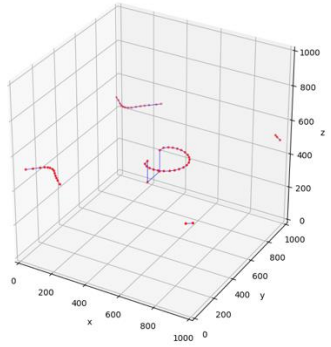
Periodic boundary condition (PBC) is applied in all three directions, as specified in the following line of the test script.

```
net = init_frank_read_src_loop(box_length=Lbox, arm_length=0.125*Lbox, pbc=True)
```

Simulation Behavior

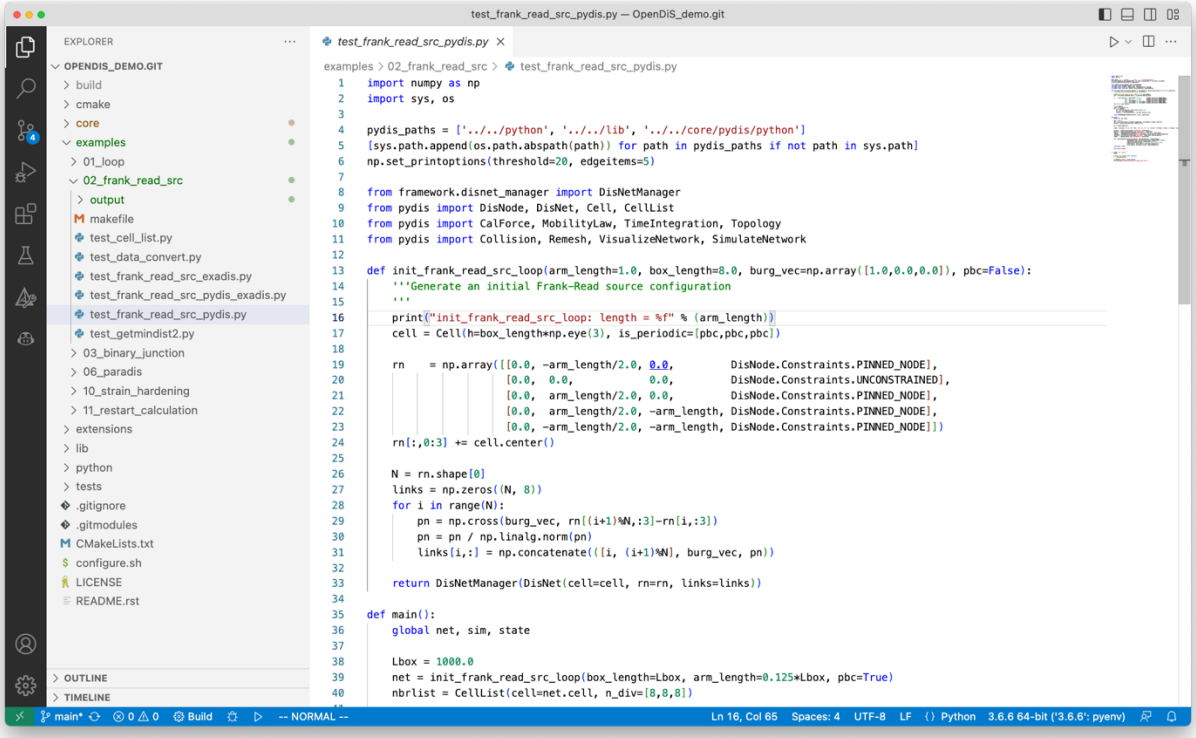
Audience participation! - select on zoom poll:

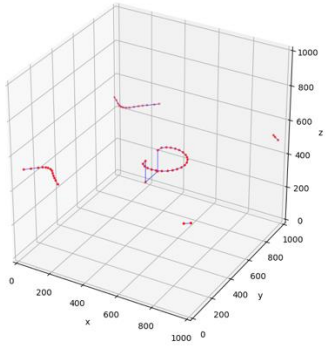
1. I see the Frank-Read source moving !
2. I can run Python code but no window opens
3. I am having problem running the code



Run OpenDiS Example

A sneak-peak at test_frank_read_src_pydis.py



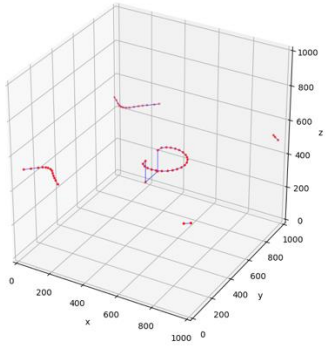


Run OpenDiS Example

Examine DisNet object

```
OpenDiS
OpenDiS (python3)
(base) Cai-Mac3:02_frank_read_src caiwei$ python3 -i test_frank_read_src_pydis.py
init_frank_read_src_loop: length = 125.000000
step = 0 dt = 1.000000e-08
step = 10 dt = 1.000000e-08
step = 20 dt = 1.000000e-08
step = 30 dt = 1.000000e-08
step = 40 dt = 1.000000e-08
step = 50 dt = 1.000000e-08
step = 60 dt = 1.000000e-08
step = 70 dt = 1.000000e-08
step = 80 dt = 1.000000e-08
step = 90 dt = 1.000000e-08
step = 100 dt = 1.000000e-08
step = 110 dt = 1.000000e-08
step = 120 dt = 1.000000e-08
step = 130 dt = 1.000000e-08
step = 140 dt = 1.000000e-08
step = 150 dt = 1.000000e-08
step = 160 dt = 1.000000e-08
step = 170 dt = 1.000000e-08
step = 180 dt = 1.000000e-08
step = 190 dt = 1.000000e-08
>>> G.
G.Edge_with_attr(                   G.export_data()                     G.neighbors_dict(
G.Node_with_attr(                   G.find_precise_glide_plane(         G.neighbors_tags(
G.add_nodes_segments_from_list(     G.from_networkx(                    G.node_prop_list()
G.all_nodes_dict()                  G.get_new_tag(                      G.nodes(
G.all_nodes_mapping()               G.get_nodes_data()                  G.num_nodes()
G.all_nodes_tags()                  G.get_segs_data(                    G.num_segments()
G.all_segments_dict()               G.get_segs_data_with_positions()    G.out_degree(
G.all_segments_mapping()            G.has_node(                         G.pos_array()
G.all_segments_tags()               G.has_segment(                      G.remove_empty_arms(
G.cell                              G.import_data(                      G.remove_two_arm_node(
G.clear_graph()                     G.insert_node(                      G.seg_prop_list()
G.convert_nodeforce_array_to_dict(  G.insert_node_between(              G.segments(
G.convert_nodeforce_dict_to_array(  G.is_equivalent(                    G.split_node(
G.convert_nodevel_array_to_dict(    G.is_sane(                          G.tags_to_nodes
G.convert_nodevel_dict_to_array(    G.merge_node(                       G.to_networkx()
G.copy()                            G.neighbor_segments_dict(
>>> G.all_nodes_tags()
dict_keys([(0, 0), (0, 2), (0, 3), (0, 4), (0, 11), (0, 12), (0, 17), (0, 18), (0, 19), (0, 27), (0, 28), (0, 32), (0, 33), (0, 34), (0, 41), (0, 42), (0, 64), (0, 77), (0, 78), (0, 79), (0, 80), (0, 83), (0, 84), (0, 85), (0, 86), (0, 87), (0, 88), (0, 90), (0, 43), (0, 101), (0, 102), (0, 107), (0, 110), (0, 124), (0, 128), (0, 51), (0, 111), (0, 118), (0, 14), (0, 103), (0, 121), (0, 47), (0, 21), (0, 99), (0, 104), (0, 71), (0, 9), (0, 70), (0, 112), (0, 73), (0, 10), (0, 96), (0, 74)])
>>>
```

type G.
then press [tab]
to see all available functions




```
>>> help(G)
```

Outline of Session 1

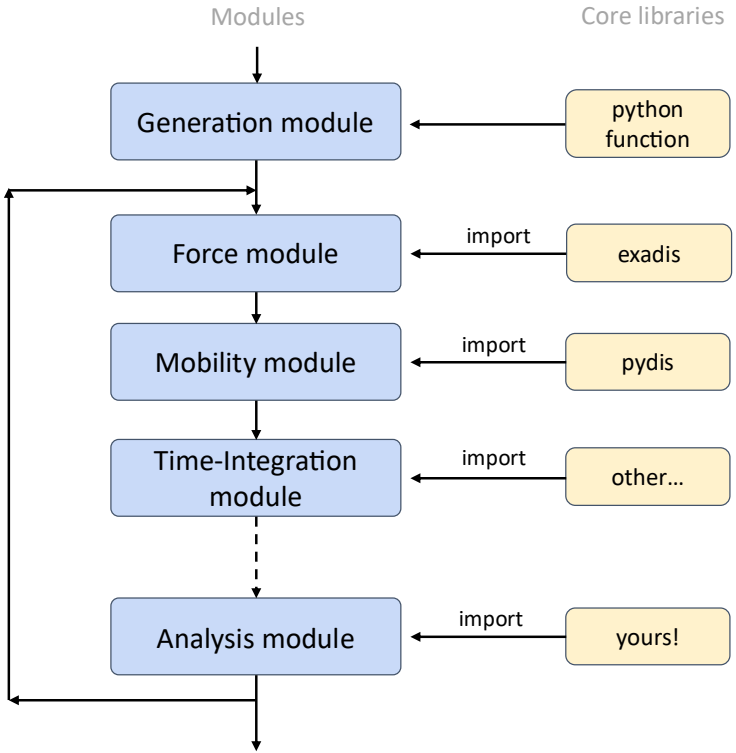
- **Project Overview**
  - History of DDD, significance, and challenges
  - General overview of OpenDiS
- **A Sneak-peek of OpenDiS**
  - Download OpenDiS
  - Run a simple Python example
  - Explore DisNet data structure
- **Code Structure**
  - OpenDiS core libraries: pydis and exadis
  - Module concepts
  - Data structures (state dictionary, DisNetManager)
  - Available modules

OpenDiS simulations: python scripting


```
import pyexadis
from pyexadis_base import ExaDisNet, NodeConstraints, DisNetManager, SimulateNetwork, VisualizeNetwork
from pyexadis_base import CalForce, MobilityLaw, TimeIntegration, Collision, Remesh

Lbox = 1000.0
N = init_frank_read_src_loop(box_length=Lbox, arm_length=0.125*Lbox, pbc=False)

vis = VisualizeNetwork()

state = {"burgmag": 3e-10, "mu": 50e9, "nu": 0.3, "a": 1.0, "maxseg": 0.04*Lbox, "minseg": 0.01*Lbox, "rann": 2.0}

calforce  = CalForce(force_mode='LineTension', state=state)
mobility  = MobilityLaw(mobility_law='SimpleGlide', state=state)
timeint   = TimeIntegration(integrator='EulerForward', state=state, dt=1.0e-8)
collision = Collision(collision_mode='Retroactive', state=state)
topology  = None
remesh    = Remesh(remesh_rule='LengthBased', state=state)

sim = SimulateNetwork(calforce=calforce, mobility=mobility, timeint=timeint,
                      collision=collision, topology=topology, remesh=remesh, vis=vis,
                      state=state, max_step=200, loading_mode='stress',
                      applied_stress=np.array([0.0, 0.0, 0.0, 0.0, -4.0e8, 0.0]),
                      print_freq=10, plot_freq=10, plot_pause_seconds=0.0001,
                      write_freq=10, write_dir='output')
sim.run(N, state)
```

OpenDiS workflow: modular architecture

Simulation pipeline



Python script to set up and drive simulations

```python
from framework.disnet_manager import DisNetManager
def generate_configuration():
    # create dislocation network cell, nodes, segments...
    G = DisNet(cell, nodes, segs)
    return DisNetManager(G)
```

```python
from pyexadis_base import CalForce
force = CalForce(force_mode='LineTension',...)
```

```python
from pydis import MobilityLaw
mobility = MobilityLaw(mobility_law='BCC_0b',...)
```

```python
from other_ddd_lib import TimeIntegration
timeint = TimeIntegration(integrator='Trapezoid',...)
```

…

OpenDiS modules: specifications

Prototype of a module

```python
class MyModule:
    """ Prototype of an OpenDiS module """
    def __init__(self, state: dict, **kwargs) -> None:
        # do some initialization
        pass

    def Compute(self, N: DisNetManager, state: dict) -> dict:

        # Get dislocation network in module-specific format
        G = N.get_disnet(MyLibraryDisNet)

        # Perform some computations on the network
        nodevalues = compute_values(G, state)

        # Make some modifications to the network
        G, mymoduleflags = modify_network(G, state)

        # Update state dictionary
        state["nodevalues"] = nodevalues
        state["mymoduleflags"] = mymoduleflags

        return state
```

Specifications

- Each method implemented in a module must:
  - Receive a **`DisNetManager`** and the **`state`** dictionary as inputs
  - Can perform computation(s) on the network
  - Can modify the network (e.g. topological operations)
  - Returns an updated state dictionary
    - Global values
    - Nodal values

- Special base modules (e.g. Force, Mobility law) have additional specifications to guarantee consistency of the implementations:
  - Force calculations modules: see the CalForce_Base specification class.
  - Mobility law modules: see the MobilityLaw_Base specification class.

OpenDiS modules: data structure

Prototype of a module

```python
class MyModule:
    """ Prototype of an OpenDiS module """
    def __init__(self, state: dict, **kwargs) -> None:
        # do some initialization
        pass

    def Compute(self, N: DisNetManager, state: dict) -> dict:

        # Get dislocation network in module-specific format
        G = N.get_disnet(MyLibraryDisNet)

        # Perform some computations on the network
        nodevalues = compute_values(G, state)

        # Make some modifications to the network
        G, mymoduleflags = modify_network(G, state)

        # Update state dictionary
        state["nodevalues"] = nodevalues
        state["mymoduleflags"] = mymoduleflags

        return state
```

Compatibility and inter-operability between modules is maintained via the two following objects:

- **`DisNetManager`:** container class that allows for co-existence and conversion between various dislocation network data structures. This allows for different modules to use their own data structures to manipulate the networks. See DisNetManager class.
- **`state`:** dictionary containing the global parameters (e.g. materials parameters, simulation settings, nodal forces, etc.) pertaining to the state of a simulation / analysis. Modules can communicate by writing/reading items into the state dictionary. See State Dictionary.

Data structures: state dictionary

- Used to hold a collection of variables and arrays defining the state of a simulation
- Propagated through the modules during a simulation

The state dictionary thus serves two purposes:

- Define the global simulation parameters (e.g. materials parameters)
- Serve as a memory buffer for modules to store / retrieve / communicate information

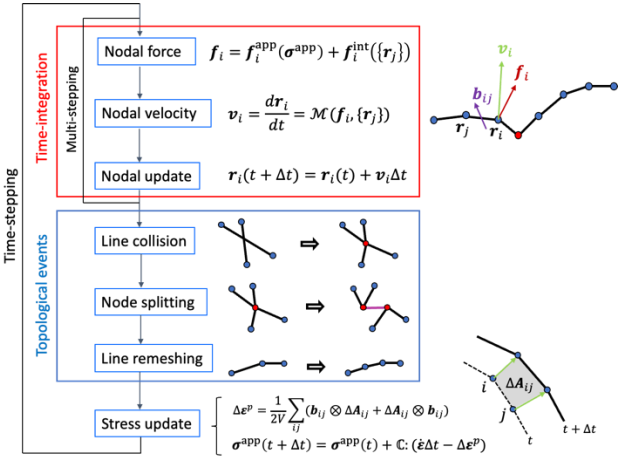
```
calforce = CalForce(state=state, ...)
mobility = MobilityLaw(state=state, ...)

# Compute nodal forces and store them into the state dictionary
state = calforce.NodeForce(N, state)
print('nodal forces', state["nodeforces"])

# Compute nodal velocities from the nodal forces stored into the state dictionary
state = mobility.Mobility(N, state)
print('nodal velocities', state["nodevels"])
```

```
state = {
    "crystal": 'fcc',
    "burgmag": 2.55e-10,
    "mu": 54.6e9,
    "nu": 0.324,
    "a": 6.0,
    "maxseg": 2000.0,
    "minseg": 300.0,
    "rtol": 10.0,
    "rann": 10.0,
    "nextdt": 1e-10,
    "maxdt": 1e-9,
}
```

https://opendis.github.io/OpenDiS/code_structure/data_structure/state_dictionary.html

OpenDiS typical simulation script

```
import pyexadis
from pyexadis_base import ExaDisNet, NodeConstraints, DisNetManager, SimulateNetwork, VisualizeNetwork
from pyexadis_base import CalForce, MobilityLaw, TimeIntegration, Collision, Remesh

Lbox = 1000.0
N = init_frank_read_src_loop(box_length=Lbox, arm_length=0.125*Lbox, pbc=False)

vis = VisualizeNetwork()

state = {"burgmag": 3e-10, "mu": 50e9, "nu": 0.3, "a": 1.0, "maxseg": 0.04*Lbox, "minseg": 0.01*Lbox, "rann": 2.0}

calforce  = CalForce(force_mode='LineTension', state=state)
mobility  = MobilityLaw(mobility_law='SimpleGlide', state=state)
timeint   = TimeIntegration(integrator='EulerForward', state=state, dt=1.0e-8)
collision = Collision(collision_mode='Retroactive', state=state)
topology  = None
remesh    = Remesh(remesh_rule='LengthBased', state=state)

sim = SimulateNetwork(calforce=calforce, mobility=mobility, timeint=timeint,
                      collision=collision, topology=topology, remesh=remesh, vis=vis,
                      state=state, max_step=200, loading_mode='stress',
                      applied_stress=np.array([0.0, 0.0, 0.0, 0.0, -4.0e8, 0.0]),
                      print_freq=10, plot_freq=10, plot_pause_seconds=0.0001,
                      write_freq=10, write_dir='output')
sim.run(N, state)
```



OpenDiS available modules

Modules	pydis	pyexadis	
CalForce	LineTension Elasticity_SBA Elasticity_SBN1_SBA	LINE_TENSION_MODEL CUTOFF_MODEL	DDD_FFT_MODEL SUBCYCLING_MODEL
MobilityLaw	Relax SimpleGlide	SimpleGlide BCC_0B BCC_NL	FCC_0 FCC_0_FRIC FCC_0B
TimeIntegration	EulerForward	EulerForward Trapezoid	RKF Subcycling
Collision	Proximity	Proximity Retroactive	
Topology	MaxDiss	TopologySerial TopologyParallel	
Remesh	LengthBased	LengthBased	
CrossSlip		ForceBasedSerial ForceBasedParallel	
Driver	SimulateNetwork	SimulateNetwork SimulateNetworkPerf	
Visualization	VisualizeNetwork	VisualizeNetwork	

Wrap-up of Session 1

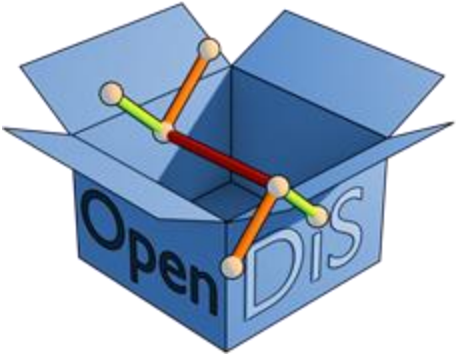
- **What we covered**
  - History of DDD, significance, and challenges
  - General overview of OpenDiS
  - Download OpenDiS
  - Run a simple Python example
  - Explore DisNet data structure
  - OpenDiS core libraries: pydis and exadis
  - Module concepts
  - Data structures (state dictionary, DisNetManager)
  - Available modules

We shall resume soon

- **What we will cover next**
  - Building (compiling) the code
  - How to set up an OpenDiS simulation
  - Running more examples
  - Analyzing the results

Feel free to send questions in zoom chat

Email us with questions during the break:
bertin1@llnl.gov, caiwei@stanford.edu



https://opendis.github.io/OpenDiS/

OpenDiS Workshop

Open Dislocation **S**imulator

Session 2

Nicolas Bertin, Lawrence Livermore National Laboratory
Wei Cai, Stanford University

LLNL-PRES-2006898

Welcome! Here is the Zoom protocol:

1. Please mute your microphone
2. Feel free to turn on your camera (optional/encouraged)
3. Feel free to "raise hand" if you have a question/comment
4. Feel free to send your question/comment to chat

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 Lawrence Livermore National Security, LLC.

Scope of Session 2

- **What we covered in Session 1**
  - History of DDD, significance, and challenges
  - General overview of OpenDiS
  - Download OpenDiS
  - Run a simple Python example
  - Explore DisNet data structure
  - OpenDiS core libraries: pydis and exadis
  - Module concepts
  - Data structures (state dictionary, DisNetManager)
  - Available modules

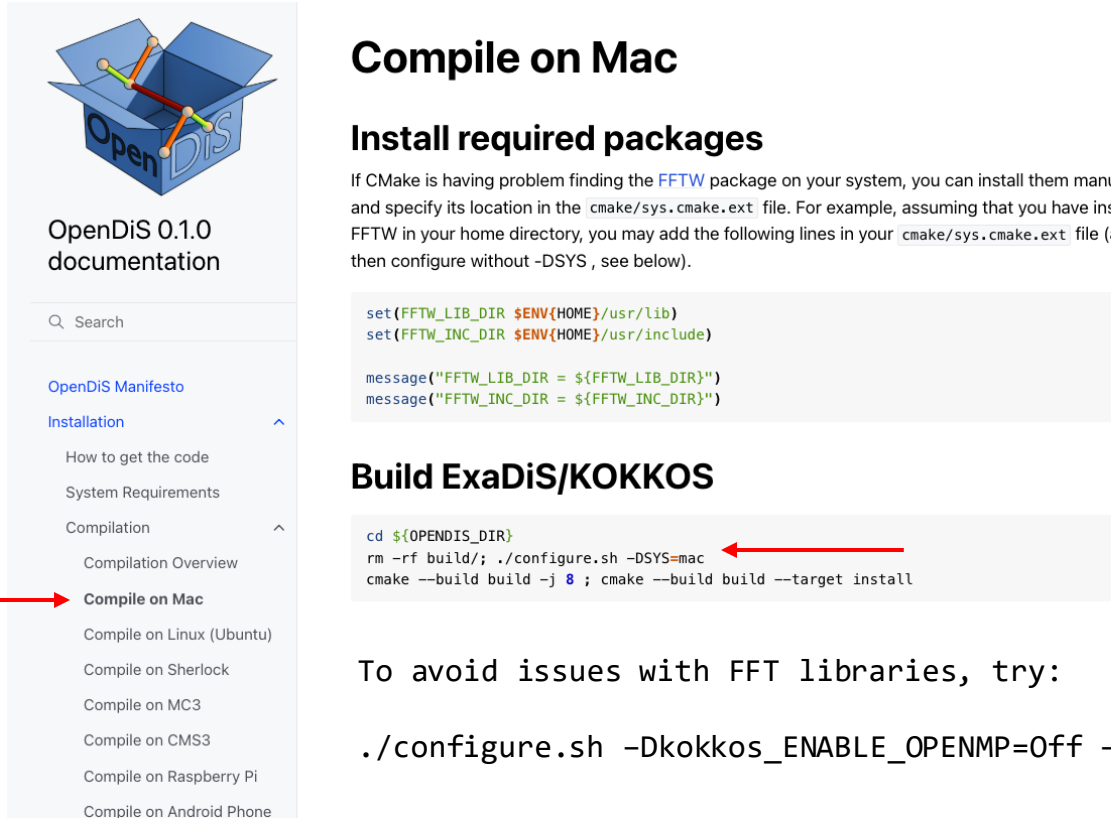
- **What we will cover in Session 2**
  - Building (compiling) the code
  - How to set up an OpenDiS simulation
  - Running more examples
  - Analyzing the results

Outline

- **Compile OpenDiS**
- **Run OpenDiS Example**
  - Frank-Read Source in ExaDiS
  - DisNetManager Class
- **How to Set up a Simulation**
  - Import modules
  - Create Initial Dislocation Configurations
  - Running the Simulation
  - Performance Considerations
- **Run more Examples**
  - Binary Junction
  - Strain Hardening

Compile OpenDiS (ExaDiS)

Documentation website
https://opendis.github.io/OpenDiS



To avoid issues with FFT libraries, try:

```
./configure.sh -Dkokkos_ENABLE_OPENMP=Off -DEXADIS_FFT=Off
```

Compile OpenDiS

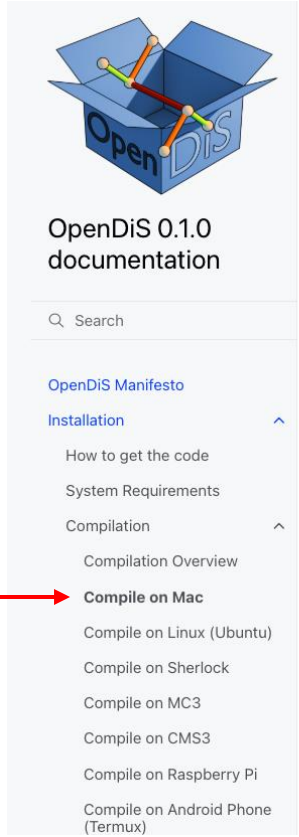
Documentation website
https://opendis.github.io/OpenDiS

Demo on MacBook

```
(base) Cai-Mac3:OpenDiS_demo.git caiwei$ cd ${OPENDIS_DIR}
(base) Cai-Mac3:OpenDiS_demo.git caiwei$ rm -rf build/; ./configure.sh -DSYS=mac
cd build ; cmake -DSYS=mac -S ..

****************************************
 options set by cmake/sys.cmake.mac
****************************************
-- The CXX compiler identification is AppleClang 15.0.0.15000040


to build and install:
  cmake --build build -j 8 ; cmake --build build --target install

(base) Cai-Mac3:OpenDiS_demo.git caiwei$ cmake --build build -j 8 ; cmake --build build --target install
[  1%] Building C object core/pydis/c/CMakeFiles/pydis.dir/calforce/SegSegForce.c.o


[100%] Built target pyexadis
Install the project...
-- Install configuration: ""
-- Installing: /Users/caiwei/Codes/OpenDiS_demo.git/lib/libpydis.so
-- Installing: /Users/caiwei/Codes/OpenDiS_demo.git/lib/pydis_lib.py
-- Installing: /Users/caiwei/Documents/Codes/OpenDiS_demo.git/build/install/lib/libexadis.a
```

Compile OpenDiS (ExaDiS)

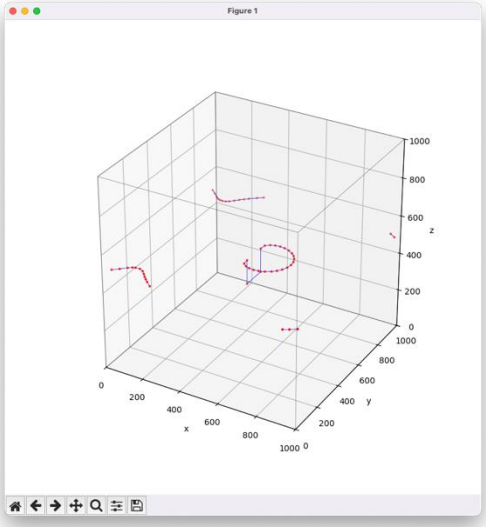
Documentation website
https://opendis.github.io/OpenDiS



Compile on Mac

Run test case (OMP version)

```
export OMP_NUM_THREADS=8
cd ${OPENDIS_DIR}
cd examples/02_frank_read_src
python3 -i test_frank_read_src_exadis.py
```

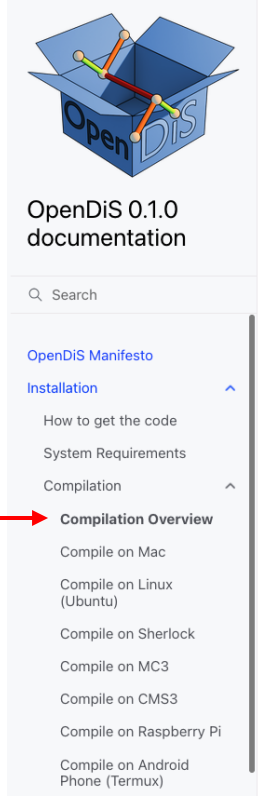


Audience participation! - select on zoom poll:

1. I see the Frank-Read source moving !
2. I can run exadis but no window opens
3. ExaDiS compiles but I cannot run it
4. I am having problem compiling ExaDiS

Compile OpenDiS (ExaDiS)

Documentation website
https://opendis.github.io/OpenDiS



Troubleshooting

This section reports common errors encountered by OpenDiS users. If you encounter other issues or bugs, please also browse the Issues section and/or open a new issue.

pyexadis import errors

Issue: Python is not being able to import `pyexadis` (python binding to the ExaDiS core library), with errors such as

```
import pyexadis
ModuleNotFoundError: No module named 'pyexadis'
```

or

```
raise ImportError('Cannot import pyexadis')
ImportError: Cannot import pyexadis
```

This typically happens when the python installation used for compilation is not the same as the python installation used to run the code.

When running the `./configure.sh` script, the python executable to be used for compilation should be indicated, e.g.:

```
-- Found PythonInterp: /path/to/python3
```

Then after compilation one should be able to run the scripts using the same python executable, e.g.

```
/path/to/python3 test_frank_read_src_exadis.py
```

Alternatively, one can explicitly specify the python executable to be used by passing build option `-DPYTHON_EXECUTABLE=...` to the `configure.sh` script, e.g.

```
./configure.sh \
```

Run OpenDiS Example

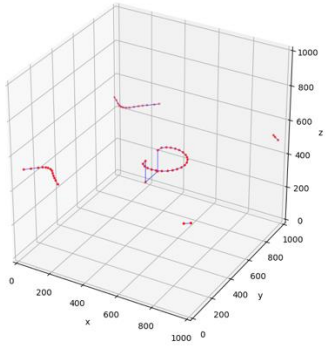
Examine ExaDisNet object

OpenDiS_demo

OpenDiS_demo (python3)

```
 Output file: output/config.110.data
step = 120, nodes = 75, dt = 1.000000e-08, time = 1.200000e-06, elapsed = 0.9 sec
Exporting configuration in ParaDiS format
 Output file: output/config.120.data
step = 130, nodes = 89, dt = 1.000000e-08, time = 1.300000e-06, elapsed = 0.9 sec
Exporting configuration in ParaDiS format
 Output file: output/config.130.data
step = 140, nodes = 93, dt = 1.000000e-08, time = 1.400000e-06, elapsed = 1.0 sec
Exporting configuration in ParaDiS format
 Output file: output/config.140.data
step = 150, nodes = 97, dt = 1.000000e-08, time = 1.500000e-06, elapsed = 1.2 sec
Exporting configuration in ParaDiS format
 Output file: output/config.150.data
step = 160, nodes = 106, dt = 1.000000e-08, time = 1.600000e-06, elapsed = 1.2 sec
Exporting configuration in ParaDiS format
 Output file: output/config.160.data
step = 170, nodes = 127, dt = 1.000000e-08, time = 1.700000e-06, elapsed = 1.3 sec
Exporting configuration in ParaDiS format
 Output file: output/config.170.data
step = 180, nodes = 102, dt = 1.000000e-08, time = 1.800000e-06, elapsed = 1.4 sec
Exporting configuration in ParaDiS format
 Output file: output/config.180.data
step = 190, nodes = 69, dt = 1.000000e-08, time = 1.900000e-06, elapsed = 1.5 sec
Exporting configuration in ParaDiS format
 Output file: output/config.190.data
step = 200, nodes = 56, dt = 1.000000e-08, time = 2.000000e-06, elapsed = 1.5 sec
Exporting configuration in ParaDiS format
 Output file: output/config.200.data
RUN TIME: 1.662191 sec
>>> G
<pyexadis_base.ExaDisNet object at 0x1047f43b0>
>>> G.
G.cell                      G.get_nodes_data()          G.import_data(              G.read_data(
G.export_data()             G.get_positions()           G.is_sane()                 G.read_paradis(
G.generate_line_config(     G.get_segs_data()           G.net                       G.set_positions(
G.generate_prismatic_config(  G.get_tags()              G.num_nodes()               G.write_data(
G.get_forces()              G.get_velocities()          G.num_segments()
>>> G.
```

type G.
then press [tab]
to see all available functions




```
>>> help(G)
```

Data structures: DisNetManager class

WHY?

- Each core library / module has its own internal data structure
  - `pydis`: python-based data structure (`DisNet`)
  - `exadis`: C++-based data structure with CPU/GPU memory space (`ExaDisNet`)

HOW?

- `DisNetManager`: helper class that allows different network data structures to co-exist and inter-operate
  - Provides a way to request the dislocation network in a given format (e.g. `DisNet` object or `ExaDisNet` object)
  - Provides a method to access the network raw data
  - Provides additional utility methods
  - Works by calling import / export functions from base network data structure (`DisNet` / `ExaDisNet`)

Data structures: DisNetManager class example

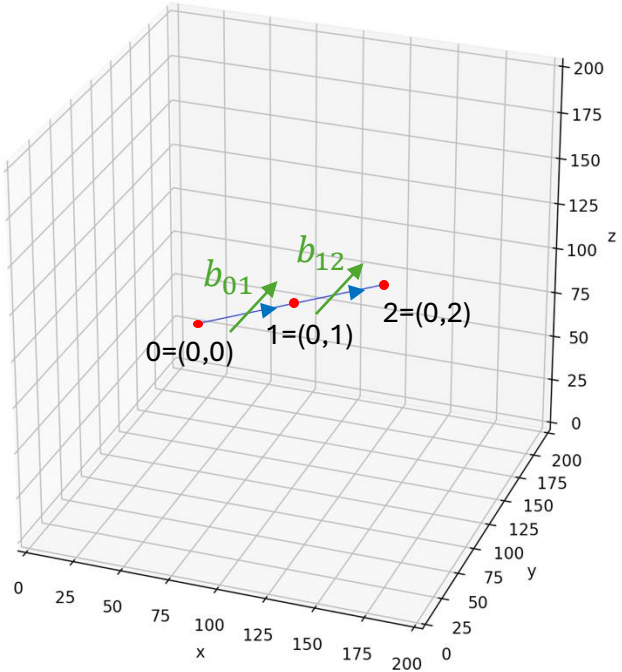

```python
from framework.disnet_manager import DisNetManager

class MyPyDisModule():
    def Foo(self, N: DisNetManager, state: dict):
        G = N.get_disnet(DisNet) # convert network to DisNet object
        # perform some operations on the DisNet object
        return state

class MyExaDisModule():
    def Bar(self, N: DisNetManager, state: dict):
        G = N.get_disnet(ExaDisNet) # convert network to ExaDisNet object
        # perform some operations on the ExaDisNet object
        return state

state = {...} # initialize state dictionary
G = ExaDisNet(...) # initialize network using exadis
N = DisNetManager(G) # wrap network into DisNetManager

pydis_module = MyPyDisModule(...) # initialize pydis module
exadis_module = MyExaDisModule(...) # initialize exadis module

pydis_module.Foo(N, state) # execute pydis Foo method
exadis_module.Bar(N, state) # execute exadis Bar method
exadis_module.Bar(N, state) # execute exadis Bar method again
```

Data structures: DisNetManager class utilities

- `DisNetManager.get_disnet(disnet_type):` Converts the dislocation network into the requested disnet_type object.
- `DisNetManager.get_active_type():` Returns the type of network object that is currently active.
- `DisNetManager.export_data():` Exports the raw network data into a data dictionary.
- `DisNetManager.import_data(data):` Set the content of the network by importing it from a data dictionary. Argument data must be the output of an export_data() method.
- `DisNetManager.write_json(filename):` Writes the DisNetManager data to a JSON file.
- `DisNetManager.read_json(filename):` Reads the DisNetManager data from a JSON file.
- `DisNetManager.num_nodes():` Returns the number of nodes in the network.
- `DisNetManager.num_segments():` Returns the number of segments in the network.
- `DisNetManager.is_sane():` Checks if the network is sane.

https://opendis.github.io/OpenDiS/code_structure/data_structure/disnetmanager_class.html

Data structures: raw dislocation network representation



Raw network data structure is a directed graph

```
data = N.export_data()

>> data = {
        'cell': {
                'h': array([[200.,   0.,   0.], [  0., 200.,   0.], [  0.,   0., 200.]]),
                'origin': array([0., 0., 0.]),
                'is_periodic': [1, 1, 1]
        },
        'nodes': {
                'tags': array([[0, 0], [0, 1], [0, 2]]),
                'positions': array([
                        [ 64.64466094,  64.64466094, 100.        ],
                        [100.        , 100.        , 100.        ],
                        [135.35533906, 135.35533906, 100.        ]
                ]),
                'constraints': array([[7], [0], [7]])
        },
        'segs': {
                'nodeids': array([[0, 1], [1, 2]]),
                'burgers': array([
                        [0.70710678, 0.70710678, 0.        ],
                        [0.70710678, 0.70710678, 0.        ]
                ]),
                'planes': array([
                        [-1.,  1.,  1.],
                        [-1.,  1.,  1.]
                ])
        }
}
```

All lengths are in units of burgmag!

Outline

- Compile OpenDiS
- Run OpenDiS Example
  - Frank-Read Source in ExaDiS
  - DisNetManager Class
- **How to Set up a Simulation**
  - Import modules
  - Create Initial Dislocation Configurations
  - Running the Simulation
  - Performance Considerations
- Run more Examples
  - Binary Junction
  - Strain Hardening

How to set up and run an OpenDiS simulation

Step	Example
0. Download and build the code	`./configure.sh # see Session 2`

1. Importing modules and core libraries	`opendis_paths = ['/path/to/OpenDiS/python', '/path/to/OpenDiS/lib',`
	`'/path/to/OpenDiS/core/pydis/python', '/path/to/OpenDiS/core/exadis/python']`
	`[sys.path.append(path) for path in opendis_paths if not path in sys.path]`
	`from framework.disnet_manager import DisNetManager`
	`from pydis import ...`
	`import pyexadis`
	`from pyexadis_base import ...`

2. Initializing pyexadis (required if using pyexadis modules)	`pyexadis.initialize() # num_threads, device_id`

3. Defining the global state dictionary	`state = {`
	`"crystal": 'fcc', "burgmag": 2.55e-10, "mu": 54.6e9, "nu": 0.324, "a": 6.0,`
	`"maxseg": 2000.0, "minseg": 300.0, "rtol": 10.0, "rann": 10.0, "maxdt": 1e-9,`
	`}`

4. Creating the initial dislocation configuration	`G = ExaDisNet().read_paradis('my_config.data')` `N = DisNetManager(G)`

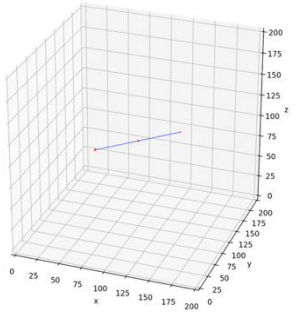
5. Defining simulation modules	`calforce = CalForce(force_mode='DDD_FFT_MODEL', state=state, Ngrid=32, cell=N.cell)`
	`mobility = MobilityLaw(mobility_law='FCC_0', state=state, Medge=64103.0, Mscrew=64103.0)`
	`...`
	`sim = SimulateNetwork(...)`
	`sim.run(N, state)`

6. Running the simulation	`python my_script.py` `python -i my_script.py`

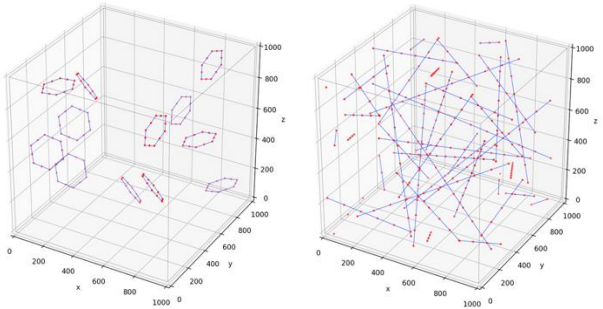
Creating initial dislocation configurations

- **Manual generation:** provide list of nodes and segments

```
from framework.disnet_manager import DisNetManager
from pydis import DisNode, DisNet, Cell
from pyexadis_base import ExaDisNet, NodeConstraints

G1 = DisNet(cell=cell, rn=rn, links=links)
N1 = DisNetManager(G1)

G2 = ExaDisNet(cell, nodes, segs)
N2 = DisNetManager(G2)
```

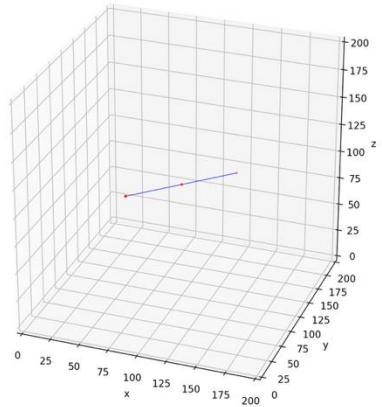


- **Utility functions:** (pyexadis_utils.py)



```
import pyexadis
from pyexadis_base import ExaDisNet
from pyexadis_utils import *
pyexadis.initialize()

G1 = ExaDisNet().read_paradis('config.data')
N1 = DisNetManager(G1)

G2 = ExaDisNet().generate_line_config(crystal, Lbox, num_lines, theta, maxseg,
Rorient, seed, verbose)
N2 = DisNetManager(G2)

G3 = ExaDisNet().generate_prismatic_config(crystal, Lbox, num_loops, radius, maxseg,
Rorient, seed, uniform)
N3 = DisNetManager(G3)

N = combine_networks([N1, N2, N3])
```

Example: creating a simple dislocation source



pydis: DisNet class

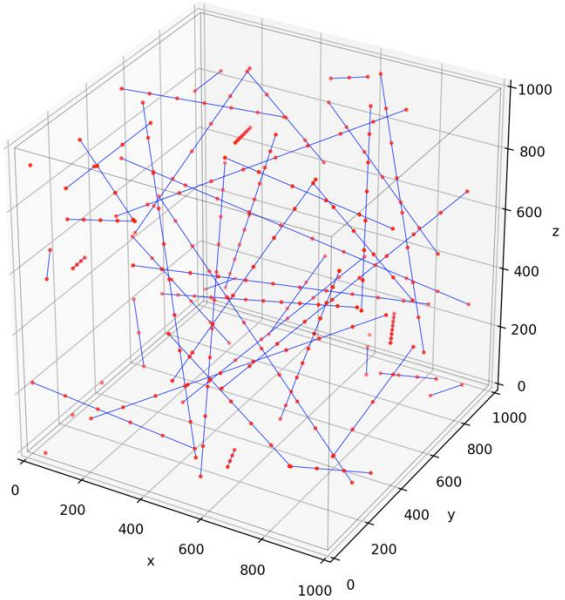
```python
from framework.disnet_manager import DisNetManager
from pydis import DisNode, DisNet, Cell

Ldis = 100.0 # dislocation length
Lbox = 2*Ldis # simulation box size
burg = 1.0/np.sqrt(2.0)*np.array([1.,1.,0.]) # Burgers vector
plane = np.array([-1.,1.,1.]) # plane normal

# Simulation cell object
cell = Cell(h=Lbox*np.eye(3), is_periodic=[True,True,True])

# List of nodes, nodes = [x,y,z,constraint]
linevec = Ldis*burg # line vector
rn = np.array([[*(-0.5*linevec), DisNode.Constraints.PINNED_NODE],
               [*( 0.0*linevec), DisNode.Constraints.UNCONSTRAINED],
               [*( 0.5*linevec), DisNode.Constraints.PINNED_NODE]])
rn[:,0:3] += cell.center() # translate line to the center of the box

# List of segments, links = [node1,node2,burg,plane]
links = np.array([[0, 1, *burg, *plane],
                  [1, 2, *burg, *plane]])

G = DisNet(cell=cell, rn=rn, links=links)
N = DisNetManager(G)
```

pyexadis: ExaDisNet class

```python
from framework.disnet_manager import DisNetManager
import pyexadis
from pyexadis_base import ExaDisNet, NodeConstraints
pyexadis.initialize()

Ldis = 100.0 # dislocation length
Lbox = 2*Ldis # simulation box size
burg = 1.0/np.sqrt(2.0)*np.array([1.,1.,0.]) # Burgers vector
plane = np.array([-1.,1.,1.]) # plane normal

# Simulation cell object
cell = pyexadis.Cell(h=Lbox*np.eye(3), is_periodic=[True,True,True])

# List of nodes, nodes = [x,y,z,constraint]
linevec = Ldis*burg # line vector
nodes = np.array([[*(-0.5*linevec), NodeConstraints.PINNED_NODE],
                  [*( 0.0*linevec), NodeConstraints.UNCONSTRAINED],
                  [*( 0.5*linevec), NodeConstraints.PINNED_NODE]])
nodes[:,0:3] += cell.center() # translate line to the center of the box

# List of segments, segs = [node1,node2,burg,plane]
segs = np.array([[0, 1, *burg, *plane],
                 [1, 2, *burg, *plane]])

G = ExaDisNet(cell, nodes, segs)
N = DisNetManager(G)
```

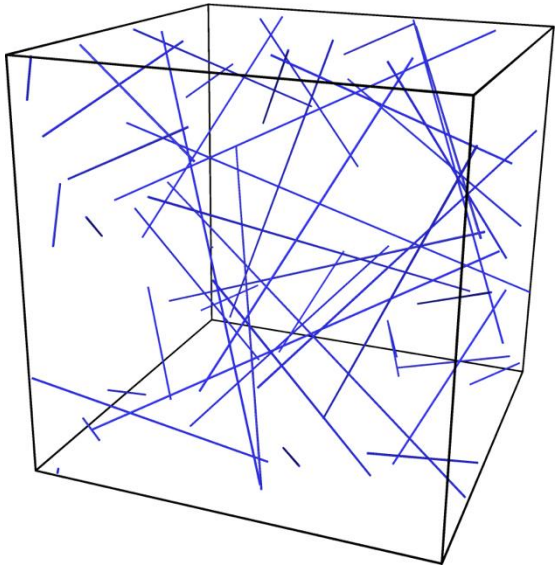
Configuration visualization

VisualizeNetwork()
(matplotlib)



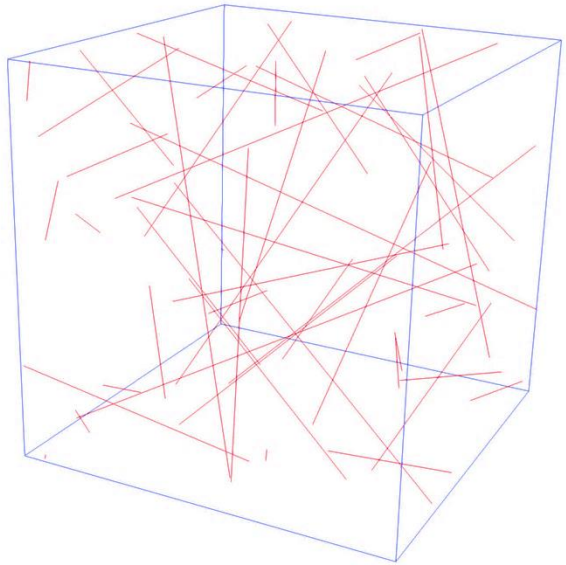

```
vis = VisualizeNetwork()
vis.plot_disnet(N)
```

Ovito (Pro)
(.data files)



```
write_data(N, 'config.data')
```

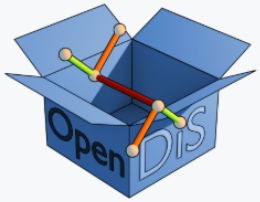
ParaView
(.vtk files)



```
write_vtk(N, 'config.vtk')
```

Set-up OpenDiS Simulations

Documentation website
https://opendis.github.io/OpenDiS



OpenDiS 0.1.0 documentation

Search

- OpenDiS Manifesto
- Installation
- Tutorials
  - Frank-Read Source
  - Binary Junction
  - Strain Hardening
  - Creating initial dislocation configurations
  - **Setting up a simulation**
- Code Structure
- Core Libraries

Setting up a simulation

This section details the various steps to set up and run a DDD simulation with OpenDiS using a python script.

Example: test_binary_junction_pydis.py

Example: test_binary_junction_exadis.py

1. Importing modules and core libraries

First, import the OpenDiS framework modules, e.g.

```
import os, sys

opendis_path = '/path/to/OpenDiS/python/'
if not opendis_path in sys.path: sys.path.append(opendis_path)
from framework.disnet_manager import DisNetManager
```

Then, for importing `pydis` modules, e.g.:

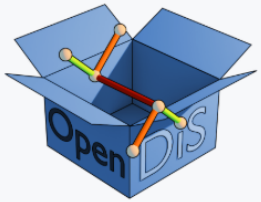
```
# Importing pydis
pydis_paths = ['/path/to/OpenDiS/core/pydis/python', '/path/to/OpenDiS/lib']
[sys.path.append(os.path.abspath(path)) for path in pydis_paths if not path in sys.path]
from pydis import DisNet, Cell, DisNode
from pydis import CalForce, MobilityLaw, TimeIntegration, Topology, Collision, Remesh
```

For importing `pyexadis` modules, e.g.:

```
# Importing pyexadis
pyexadis_path = '/path/to/OpenDiS/core/exadis/python/'
if not pyexadis_path in sys.path: sys.path.append(pyexadis_path)
```

Set-up OpenDiS Simulations

Documentation website
https://opendis.github.io/OpenDiS



OpenDiS 0.1.0 documentation

Search

- OpenDiS Manifesto
- Installation
- Tutorials
  - Frank-Read Source
  - Binary Junction
  - Strain Hardening
  - Creating initial dislocation configurations
  - **Setting up a simulation**
- Code Structure
- Core Libraries

2. Initializing pyexadis

When using `pyexadis` modules, `pyexadis` must be explicitly initialized before it can be used. For this, all the simulation script must be inserted in between calls to the `pyexadis.initialize()` and `pyexadis.finalize()` functions:

Example: test_binary_junction_pydis.py

Example: test_binary_junction_exadis.py

```
# Importing stuff

pyexadis.initialize()

# Code to setup and run the simulation here...

pyexadis.finalize()
```

This is to make sure that Kokkos is initialized and terminated properly on the chosen execution spaces. The `pyexadis.initialize()` can take the following arguments:

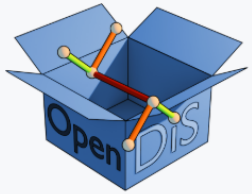
- `num_threads` : specifies the number of OpenMP threads to be used. For instance, use `pyexadis.initialize(num_threads=8)` to run a simulation using 8 threads. If not specified (default), ExaDiS will use the maximum number of threads available on the machine or defined from the `OMP_NUM_THREADS` environment variable if set.
- `device_id` : specifies the id of the device (GPU) to be used. If not specified (default), ExaDiS will select the first entry in the list of available devices.

3. Defining the global state dictionary

The global `state` dictionary is used to hold a collection of variables and arrays defining the state of a simulation, see State dictionary section. It is propagated through the modules during a simulation:

Set-up OpenDiS Simulations

Documentation website
https://opendis.github.io/OpenDiS



OpenDiS 0.1.0 documentation

Search

4. Creating the initial dislocation configuration

Ways to create initial dislocation configurations are detailed in the Creating initial dislocation configurations section. For instance, this can be done by reading a dislocation configuration from a ParaDiS format file:

```
G = ExaDisNet()
G.read_paradis('my_config.data')
```

Example: test_binary_junction_pydis.py

The network object then needs to be wrapped into a `DisNetManager` object:

```
N = DisNetManager(G)
```

Example: test_binary_junction_exadis.py

This step is required to ensure inter-operability between the different core implementations and data structures.

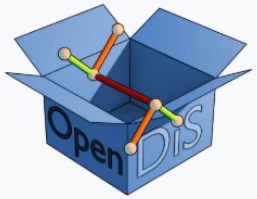
5. Defining simulation modules

Next we define the simulation modules to be used in the simulation. A full DDD simulation typically uses the following modules corresponding to the different elementary stages of the DDD simulation cycle:

- `CalForce`: module that computes forces at dislocation nodes
- `MobilityLaw`: module that computes velocities of dislocation nodes
- `TimeIntegration`: module that performs time-integration of the system
- `Collision`: module that implements dislocation collisions
- `Topology`: module that implements dislocation topological operations
- `CrossSlip`: module that performs cross-slip events
- `Remesh`: module that performs remeshing of the dislocation lines

Set-up OpenDiS Simulations

Documentation website
https://opendis.github.io/OpenDiS



OpenDiS 0.1.0 documentation

Search

- OpenDiS Manifesto
- Installation
- Tutorials
  - Frank-Read Source
  - Binary Junction
  - Strain Hardening
  - Creating initial dislocation configurations
  - **Setting up a simulation**
- Code Structure
- Core Libraries

6. Running the simula

↑ Back to top

After the above python setup is saved in script, e.g. `my_script.py`, it is run by executing the python script

```
python my_script.py
```

Example: test_binary_junction_pydis.py

For debugging or interacting with the simulation, we can also use the interactive mode of python

```
python -i my_script.py
```

Example: test_binary_junction_exadis.py

In this mode, after the script has been executed, one can manipulate the data, e.g. to investigate the dislocation network or perform additional analyses, see Dislocation network examination for an example.

Hint

When using `pyexadis`, one must not call the `pyexadis.finalize()` instruction when using the interactive mode, otherwise the ExaDiS memory will be freed and thus inaccessible. A good option can be to call the `pyexadis.finalize()` function only when a non-interactive python mode is detected, e.g.

```
if not sys.flags.interactive:
    pyexadis.finalize()
```

7. Performance considerations

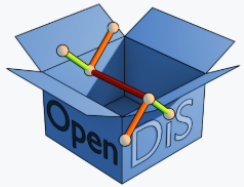
- `pydis` vs. `pyexadis` modules performance:
  - As a pure python code, `pydis` modules are generally slow. They are usually used for learning, prototyping, or running small-scale simulations.

Outline

- **Compile OpenDiS**
- **Run OpenDiS Example**
  - Frank-Read Source in ExaDiS
  - DisNetManager Class
- **How to Set up a Simulation**
  - Import modules
  - Create Initial Dislocation Configurations
  - Running the Simulation
  - Performance Considerations
- **Run more Examples**
  - Binary Junction
  - Strain Hardening

Run OpenDiS Examples

Documentation website
https://opendis.github.io/OpenDiS



OpenDiS 0.1.0 documentation

Search

Binary Junction by Pure Python

We can run the following test case without having to compile OpenDiS.

To run the simulation, simply execute:

```
cd ${OPENDIS_DIR}
cd examples/03_binary_junction
python3 -i test_binary_junction_pydis.py
```

Initial Condition

The initial configuration for this simulation is made of two dislocation lines intersecting at a single point. Each dislocation line consists of three nodes, and their end nodes are pinned (constraint == PINNED_NODE). On the other hand, nodes in the middle of the lines are unconstrained. This allows the two dislocation lines to form a binary junction under zero stress condition.

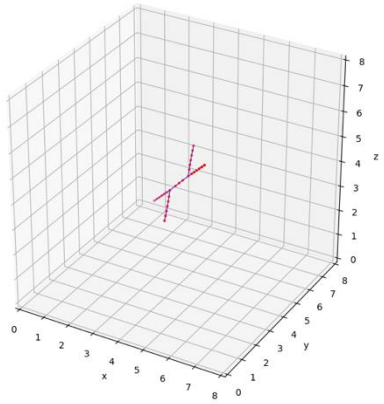
Boundary Condition

Periodic boundary condition (PBC) is turned off, as specified in the following line of the test script.

```
net = init_two_disl_lines(z0=z0, box_length=Lbox, pbc=False)
```

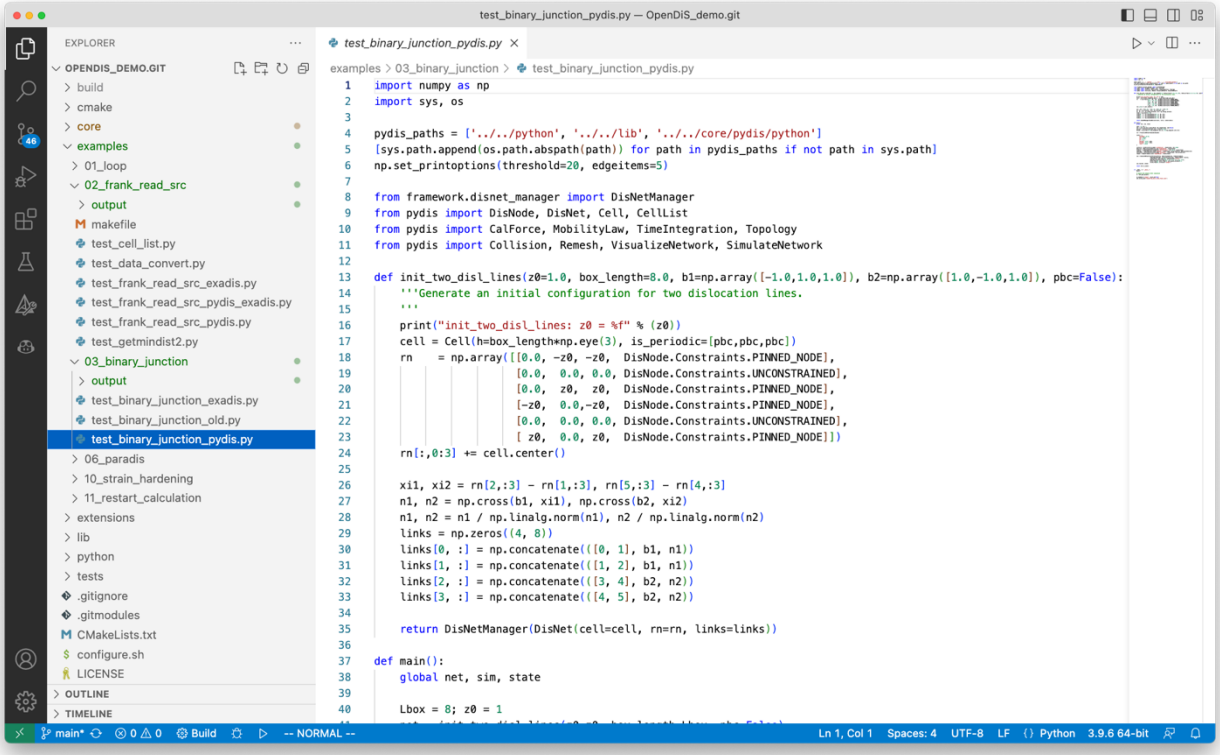
Simulation Behavior

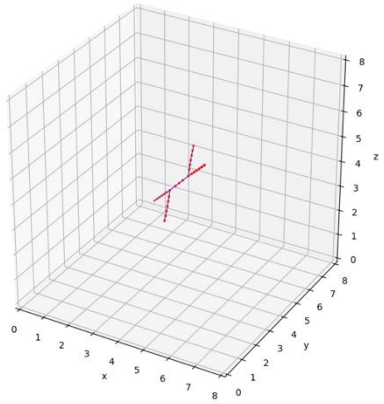
If the simulation runs successfully, a (Python Matplotlib) window should open and the final dislocation configuration at the end of the simulation should look something like this.



Run OpenDiS Examples

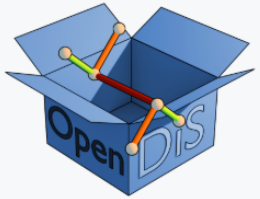
Examine test_binary_junction_pydis.py





Run OpenDiS Examples

Documentation website
https://opendis.github.io/OpenDiS



OpenDiS 0.1.0 documentation

Search

- OpenDiS Manifesto
- Installation
- Tutorials
  - Frank-Read Source
  - Binary Junction
  - Strain Hardening
    - **Strain Hardening Simulation on CPU**
    - Strain Hardening Simulation on GPU
  - Creating initial dislocation configurations

Strain Hardening Simulation on CPU

We can run a tensile test simulation of a single-crystal Cu using the following commands (running ExaDiS on CPU). This type of simulation is also called the strain-hardening simulation because it predicts stress-strain curves in the plastic regime where the flow stress increases with strain.

```
cd ${OPENDIS_DIR}
cd examples/10_strain_hardening/
export OMP_NUM_THREADS=8
python3 test_strain_hardening_exadis.py
```

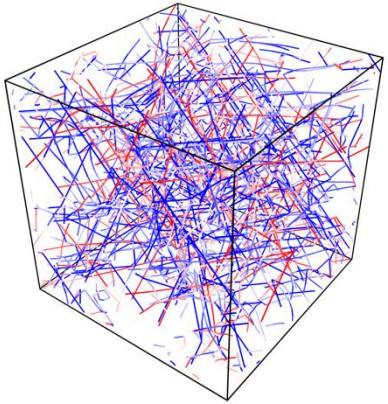
Initial Condition

The initial dislocation configuration for this simulation is read from a data file `180chains_16.10e.data` (in ParaDiS data format) by the following Python commands

```
G = ExaDisNet()
G.read_paradis('180chains_16.10e.data')
```

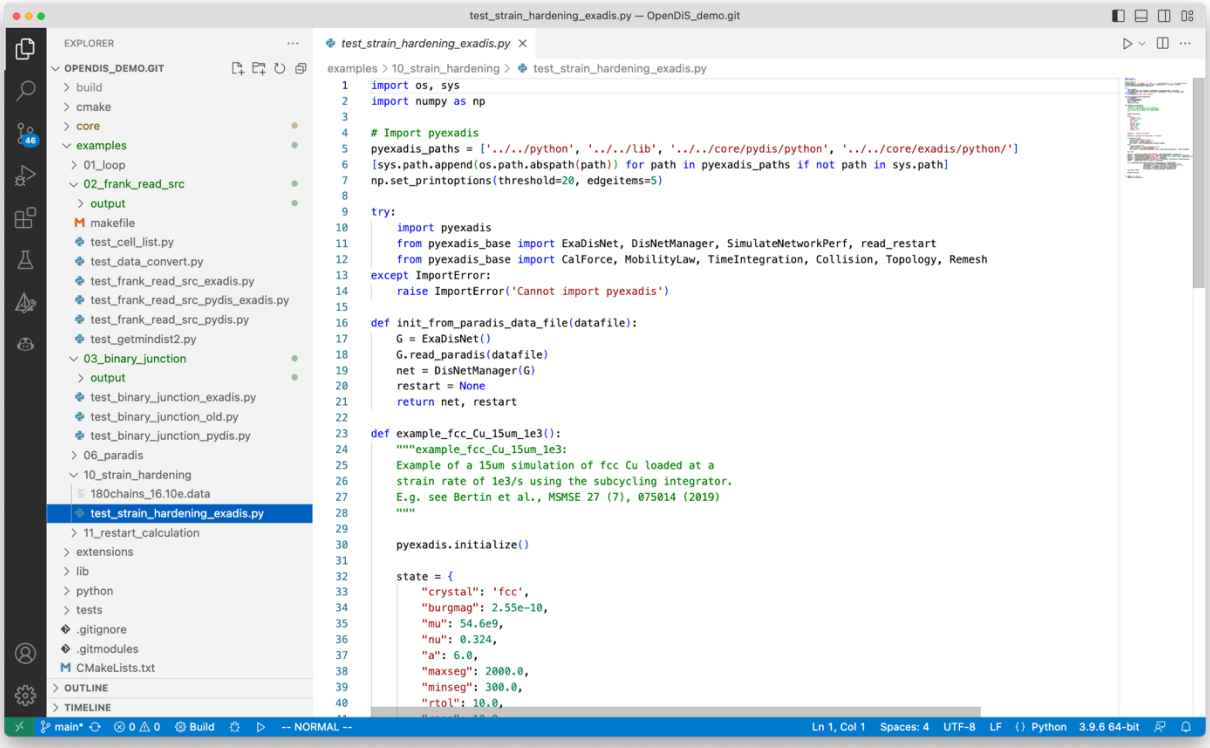
The simulation cell size is 15 μm × 15 μm × 15 μm, subjected to periodic boundary conditions in all three directions. The initial dislocation density is $\rho_0 \approx 1.2 \times 10^{12}$ m^{-2}. The initial dislocation structure can be visualized by Ovito as shown below. The dislocation lines are colored according to the dislocation character angle.

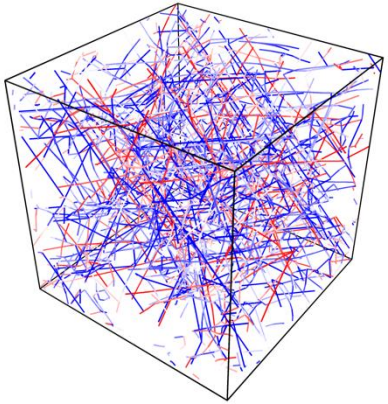
(Note: requires the FFT library)



Run OpenDiS Examples

Examine test_strain_hardening_exadis.py





Compiling and running on GPU

NVIDIA GPU (using CUDA)

```
./configure.sh \
        -DKokkos_ENABLE_CUDA=On \
        -DKokkos_ENABLE_CUDA_LAMBDA=On \
        -DKokkos_ARCH_AMPERE80=On
```

AMD GPU (using HIP)

```
./configure.sh \
        -DKokkos_ENABLE_HIP=On \
        -DKokkos_ARCH_NATIVE=On \
        -DKokkos_ARCH_AMD_GFX90A=On

cmake --build build -j 8 ; cmake --build build --target install

(srun –n1 –g1) python script.py
```

Wrap-up of Session 2

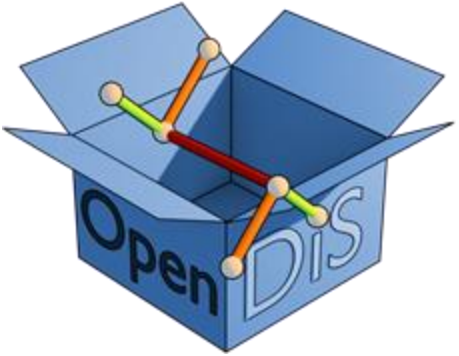
- **What we covered**
  - How to compile OpenDiS (ExaDiS)
  - Run OpenDiS examples
  - DisNetManager class
  - How to set up and run simulations
  - Analyze and visualize results

- **What we will cover next**
  - Let's code!
  - How to implement a new module
  - How to add user-defined enhancements

We shall resume tomorrow at 7 am PDT

Feel free to send questions in zoom chat

Email us with questions during the break:
bertin1@llnl.gov, caiwei@stanford.edu



https://opendis.github.io/OpenDiS/

OpenDiS Workshop

Open Dislocation **S**imulator

Session 3

Nicolas Bertin, Lawrence Livermore National Laboratory
Wei Cai, Stanford University

LLNL-PRES-2006898

Welcome! Here is the Zoom protocol:

1. Please mute your microphone
2. Feel free to turn on your camera (optional/encouraged)
3. Feel free to "raise hand" if you have a question/comment
4. Feel free to send your question/comment to chat

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 Lawrence Livermore National Security, LLC.

Scope of Session 3

- **What we covered in Session 2**
  - Downloading and building the code
  - Running some examples
  - Analyzing the results

- **What we will cover in Session 3**
  - Diving into the code
  - How to implement a new module
  - Example: a new mobility law – Python implementation
  - Example: a new mobility law – ExaDiS implementation
  - Example: custom output files (vtk)

Outline

- **OpenDiS Code Structure**
  - DisNetManager (DisNet and ExaDisNet)
  - state dictionary
  - Modules (relationships and dependencies)

- **A User-Defined Module – Python Implementation**
  - **Force / mobility specification classes**
  - **Mobility example**

- **A User-Defined Module – ExaDiS Implementation**
  - **Introduction to ExaDiS**
  - **Density / Force / Mobility examples**

- **A User-Defined Type of Output Files**

DisNetManager

Run test_frank_read_src_pydis_exadis.py

OpenDiS_demo

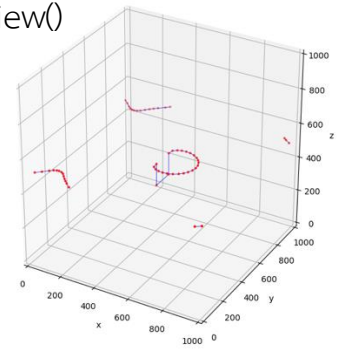
OpenDiS_demo (python3)

```
(base) Cai-Mac3:02_frank_read_src caiwei$ python3 -i test_frank_read_src_pydis_exadis.py
Host Serial Execution Space:
  KOKKOS_ENABLE_SERIAL: yes
Kokkos atomics disabled

Serial Runtime Configuration:
init_frank_read_src_loop: length = 125.000000
Setting rtol to 0.250000
Setting rtol to 0.250000
Setting rtol to 0.250000
Setting rtol to 0.250000
  set up sim with option 1
step = 0 dt = 1.000000e-08
step = 10 dt = 1.000000e-08
step = 20 dt = 1.000000e-08
step = 30 dt = 1.000000e-08
step = 40 dt = 1.000000e-08
step = 50 dt = 1.000000e-08
step = 60 dt = 1.000000e-08
step = 70 dt = 1.000000e-08
step = 80 dt = 1.000000e-08
step = 90 dt = 1.000000e-08
step = 100 dt = 1.000000e-08
step = 110 dt = 1.000000e-08
step = 120 dt = 1.000000e-08
step = 130 dt = 1.000000e-08
step = 140 dt = 1.000000e-08
step = 150 dt = 1.000000e-08
step = 160 dt = 1.000000e-08
step = 170 dt = 1.000000e-08
step = 180 dt = 1.000000e-08
step = 190 dt = 1.000000e-08
>>> dir()
['Cell', 'CellList', 'DisNet', 'DisNetManager', 'DisNode', 'ExaDiS_CalForce', 'ExaDiS_Collision', 'ExaDiS_MobilityLaw', 'ExaDiS_Remesh', 'ExaDiS_TimeIntegration', 'ExaDiS_Topology', 'ExaDisNet', 'G', 'PyDiS_CalForce', 'PyDiS_Collision', 'PyDiS_MobilityLaw', 'PyDiS_Remesh', 'PyDiS_TimeIntegration', 'PyDiS_Topology', 'SimulateNetwork', 'VisualizeNetwork', '__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'init_frank_read_src_loop', 'main', 'net', 'np', 'os', 'pydis_paths', 'pyexadis', 'sim', 'state', 'sys']
>>> net
framework.DisNetManager(num_nodes=59, num_segments=59)
>>> G
<pydis.disnet.DisNet object at 0x10b5098e0>
>>> G0 = DisNetManager(G).get_disnet(ExaDisNet)
>>> G0
<pyexadis_base.ExaDisNet object at 0x10eedf680>
>>> G1 = DisNetManager(G0).get_disnet(DisNet)
>>> G.is_equivalent(G1)
True
>>>
```

Explore DisNet Object

G.all_nodes_tags()

G.nodes((0,0)).view()



list(G.all_segments_tags())

G.segments(((0, 21), (0, 63))).view()

G.segments(((0, 21), (0, 63))).burg_vec_from((0,21))

G.segments(((0, 21), (0, 63))).burg_vec_from((0,63))

state Dictionary

Run test_frank_read_src_pydis_exadis.py

OpenDiS_demo

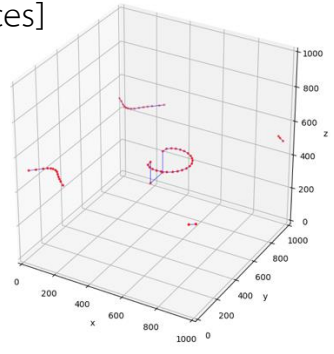
OpenDiS_demo (python3)

```
>>>
>>>
>>>
>>>
>>> state.keys()
dict_keys(['burgmag', 'mu', 'nu', 'a', 'maxseg', 'minseg', 'rann', 'applied_stress', 'istep', 'nodeforces', 'nodeforcetags', 'nodeforce_dict', 'vel_dict', 'nodevels',
 'nodeveltags', 'dt', 'nodeflag_dict'])
>>> state['burgmag']
3e-10
>>> state['mu']
50000000000.0
>>> state['applied_stress']
array([ 0.e+00,  0.e+00,  0.e+00,  0.e+00, -4.e+08,  0.e+00])
>>>
>>> state['nodeforces']
array([[-5.54990229e+09, -1.40007723e+10, -1.30881357e+10],
       [-6.19557934e+09, -6.37531930e+09,  0.00000000e+00],
       [-5.54990229e+09,  1.40007723e+10, -1.30881357e+10],
       [-2.50000000e+10, -1.30881357e+10,  1.30881357e+10],
       [-2.50000000e+10,  1.30881357e+10,  1.30881357e+10],
       ...,
       [ 7.89498087e+09,  4.16163672e+09,  0.00000000e+00],
       [ 6.97344410e+09,  6.74859930e+09,  0.00000000e+00],
       [ 7.64900806e+09, -1.02422331e+10,  0.00000000e+00],
       [-6.18738003e+09,  6.37598591e+09,  0.00000000e+00],
       [ 1.94559346e+10, -1.20295698e+09,  0.00000000e+00]])
>>>
>>> state['nodevels']
array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [-2.63128283e+08, -2.70761898e+08,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       ...,
       [ 2.94424834e+08,  1.55198502e+08,  0.00000000e+00],
       [ 3.11375170e+08,  3.01335499e+08,  0.00000000e+00],
       [ 3.63014813e+08, -4.86086865e+08,  0.00000000e+00],
       [-2.62641069e+08,  2.70646986e+08,  0.00000000e+00],
       [ 6.62707175e+08, -4.09750668e+07,  0.00000000e+00]])
>>>
```

Explore state dictionary

state.keys()

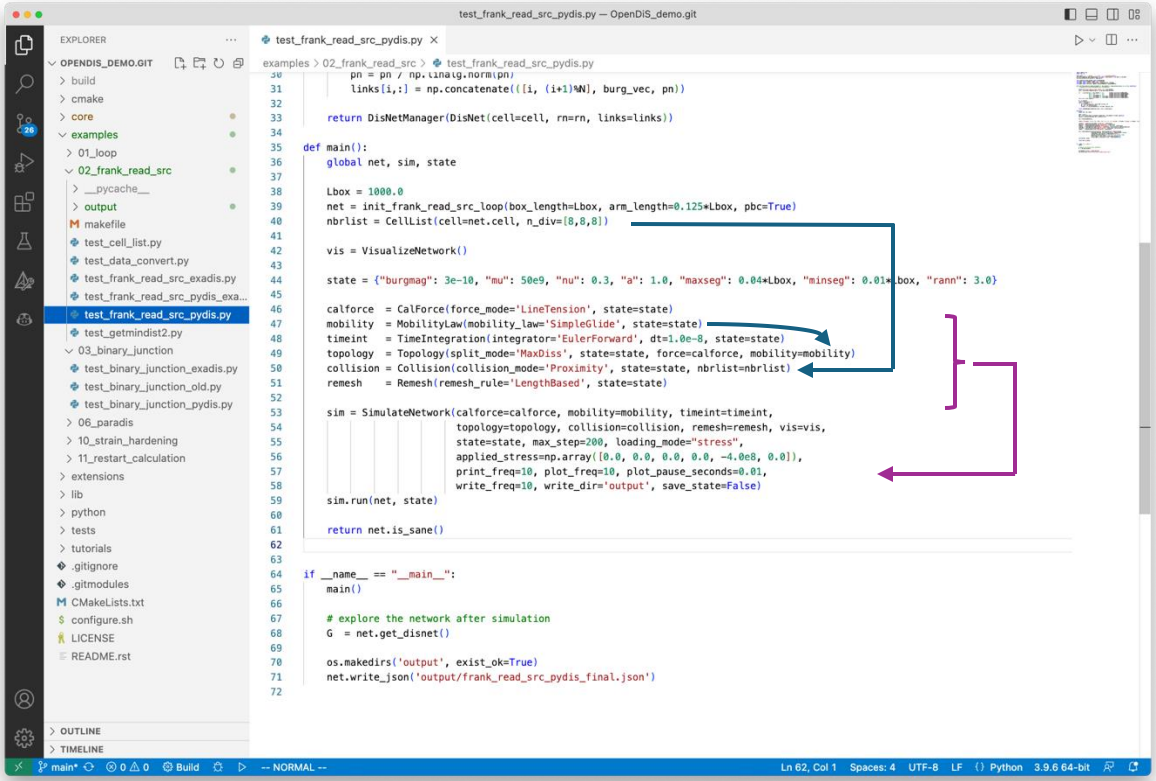
state[nodeforces]



state[nodevels]

Modules (relationships and dependencies)

Example test_frank_read_src_pydis.py



Outline

- OpenDiS Code Structure
  - DisNetManager (DisNet and ExaDisNet)
  - state dictionary
  - Modules (relationships and dependencies)
- **A User-Defined Module – Python Implementation**
  - **Force / mobility specification classes**
  - **Mobility example**
- A User-Defined Module – ExaDiS Implementation
  - Introduction to ExaDiS
  - Density / Force / Mobility examples
- A User-Defined Type of Output Files

OpenDiS: `CalForce` module

`CalForce_Base` specification class

```
class CalForce_Base(ABC):
    """CalForce_Base: base class for CalForce

    Defines the interface for the CalForce module
    """

    @abstractmethod
    def NodeForce(self, N: DisNetManager, state: dict, pre_compute: bool=True) -> dict:
        """NodeForce: compute all nodal forces and store them in the state dictionary
        """
        pass

    @abstractmethod
    def PreCompute(self, N: DisNetManager, state: dict) -> dict:
        """PreCompute: pre-compute some data for force calculation
        """
        pass

    @abstractmethod
    def OneNodeForce(self, N: DisNetManager, state: dict, tag: Tag, update_state: bool=True) -> np.array:
        """OneNodeForce: compute and return the force on one node specified by its tag
        """
        pass
```

https://github.com/OpenDiS/OpenDiS/blob/main/python/framework/calforce_base.py

OpenDiS: `MobilityLaw` module

`MobilityLaw_Base` specification class

```
class MobilityLaw_Base(ABC):
    """MobilityLaw_Base: base class for MobilityLaw

    Defines the interface for the MobilityLaw module
    """

    @abstractmethod
    def Mobility(self, N: DisNetManager, state: dict) -> dict:
        """Mobility: compute all nodal velocities and store them in the state dictionary
        """
        pass

    @abstractmethod
    def OneNodeMobility(self, N: DisNetManager, state: dict, tag: Tag, f: np.array, update_state: bool=True) -> np.array:
        """OneNodeMobility: compute and return the mobility of one node specified by its tag
        """
        pass
```

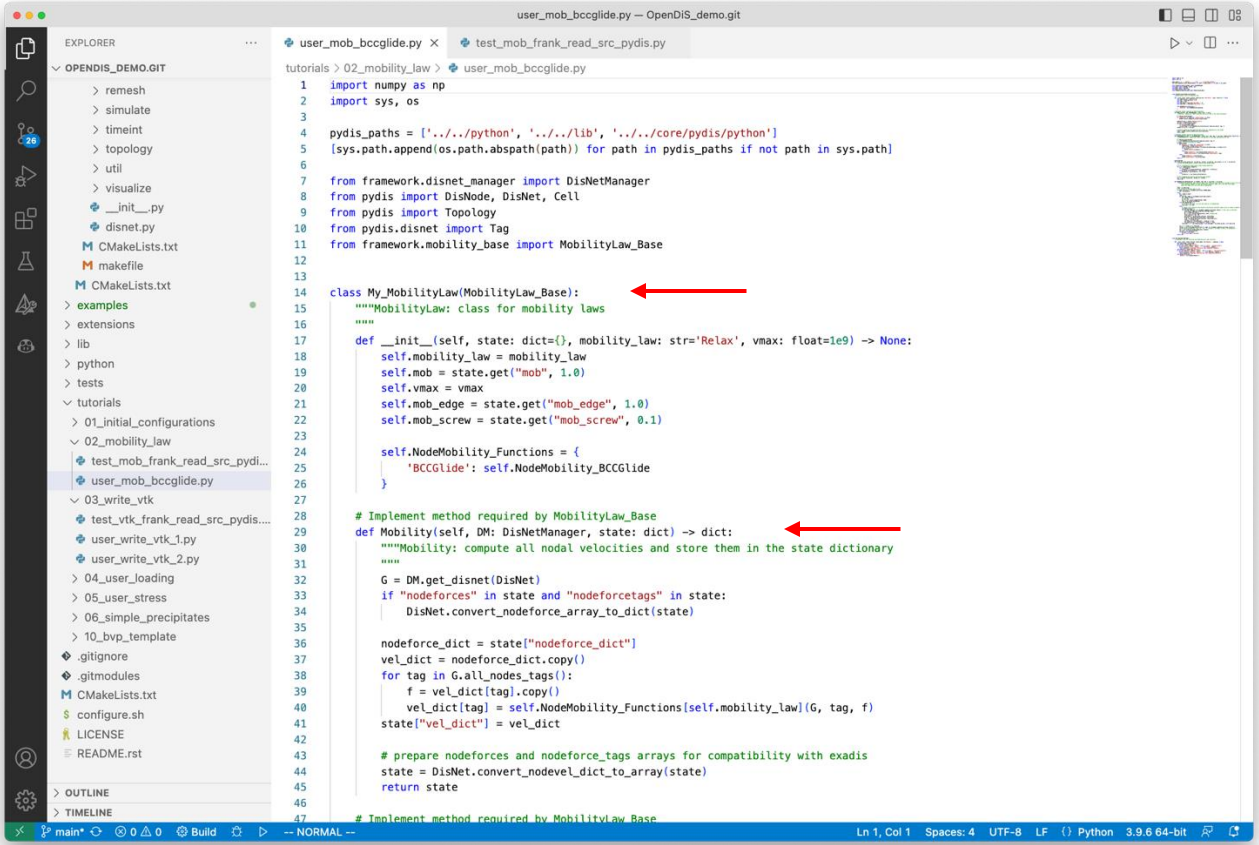
https://github.com/OpenDiS/OpenDiS/blob/main/python/framework/mobility_base.py

PyDiS: Example of a `MobilityLaw` module


```
class MobilityLaw:
    """MobilityLaw: class for mobility laws
    """
    def __init__(self, state: dict={}, mobility_law: str='Relax', vmax: float=1e9) -> None:
        self.mobility_law = mobility_law
        self.mob = state.get("mob", 1.0)
        self.vmax = vmax

        self.NodeMobility_Functions = {
                'SimpleGlide': self.NodeMobility_SimpleGlide
        }

    def Mobility(self, DM: DisNetManager, state: dict) -> dict:
        """Mobility: calculate node velocity according to mobility law function
        """
        G = DM.get_disnet(DisNet)
        if "nodeforces" in state and "nodeforcetags" in state:
            DisNet.convert_nodeforce_array_to_dict(state)

        nodeforce_dict = state["nodeforce_dict"]
        vel_dict = nodeforce_dict.copy()
        for tag in G.all_nodes_tags():
            f = vel_dict[tag].copy()
            vel_dict[tag] = self.NodeMobility_Functions[self.mobility_law](G, tag, f)
        state["vel_dict"] = vel_dict

        # prepare nodeforces and nodeforce_tags arrays for compatibility with exadis
        state = DisNet.convert_nodevel_dict_to_array(state)
        return state

    def OneNodeMobility(self, DM: DisNetManager, state: dict, tag: Tag, f: np.array, update_state: bool=True) -> np.array:
        """OneNodeMobility: compute and return the mobility of one node specified by its tag
        """
        G = DM.get_disnet(DisNet)
        v = self.NodeMobility_Functions[self.mobility_law](G, tag, f)
        # update velocity dictionary if needed
        if update_state:
            if "nodevels" in state and "nodeveltags" in state:
                nodeveltags = state["nodeveltags"]
                ind = np.where((nodeveltags[:,0]==tag[0])&(nodeveltags[:,1]==tag[1]))[0]
                if ind.size == 1:
                    state["nodevels"][ind[0]] = v
                else:
                    state["nodevels"] = np.vstack((state["nodevels"], v))
                    state["nodeveltags"] = np.vstack((state["nodeveltags"], tag))
            else:
                state["nodevels"] = np.array([v])
                state["nodeveltags"] = np.array([tag])
        return v
```

```
    @staticmethod
    def ortho_vel_glide_planes(vel: np.ndarray, normals: np.ndarray, eps_normal=1.0e-10) -> np.ndarray:
        """ortho_vel_glide_planes: project velocity onto glide planes
        """
        # first orthogonalize glide plane normals among themselves
        for i in range(normals.shape[0]):
            for j in range(i):
                normals[i] -= np.dot(normals[i], normals[j]) * normals[j]
            if np.linalg.norm(normals[i]) < eps_normal:
                normals[i] = np.array([0.0, 0.0, 0.0])
            else:
                normals[i] /= np.linalg.norm(normals[i])

        # then orthogonalize velocity with glide plane normals
        vel -= np.dot( np.dot(vel, normals.T), normals )
        return vel

    def NodeMobility_SimpleGlide(self, G: DisNet, tag: Tag, f: np.array) -> np.array:
        """NodeMobility_SimpleGlide: node velocity equal node force divided by sum of arm length / 2
           To do: add glide constraints
        """
        node1 = G.nodes(tag)
        # set velocity of pinned nodes to zero
        if node1.constraint == DisNode.Constraints.PINNED_NODE:
            vel = np.zeros(3)
        else:
            R1 = node1.R.copy()
            Lsum = 0.0
            for nbr_tag, node2 in G.neighbors_dict(tag).items():
                R2 = node2.R.copy()
                # apply PBC
                R2 = G.cell.closest_image(Rref=R1, R=R2)
                Lsum += np.linalg.norm(R2-R1)
            vel = f / (Lsum/2.0) * self.mob
            normals = np.array([edge.plane_normal for edge in G.neighbor_segments_dict(tag).values()])
            #print("Mobility_SimpleGlide: tag = %s, vel = %s, normals = %s"%(tag, str(vel), str(normals)))
            vel = self.ortho_vel_glide_planes(vel, normals)
            vel_norm = np.linalg.norm(vel)
            if vel_norm > self.vmax:
                vel *= self.vmax / vel_norm
        return vel
```

User-defined mobility law



Example:

tutorials/02_mobility_law

MobilityLaw:

user_mob_bccglide.py

Run simulation:

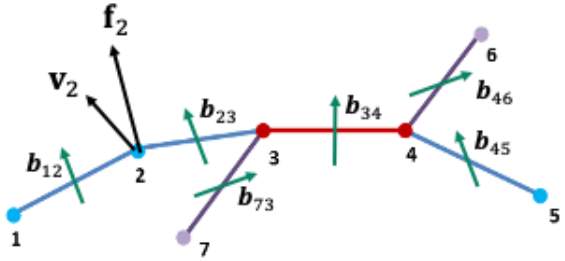
test_mob_bccglide.py

Outline

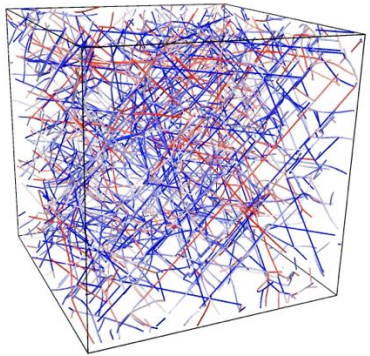
- OpenDiS Code Structure
  - DisNetManager (DisNet and ExaDisNet)
  - state dictionary
  - Modules (relationships and dependencies)
- A User-Defined Module – Python Implementation
  - Force / mobility specification classes
  - Mobility example
- **A User-Defined Module – ExaDiS Implementation**
  - **Introduction to ExaDiS**
  - **Density / Force / Mobility examples**
- A User-Defined Type of Output Files

Recap of the (nodal) DDD method

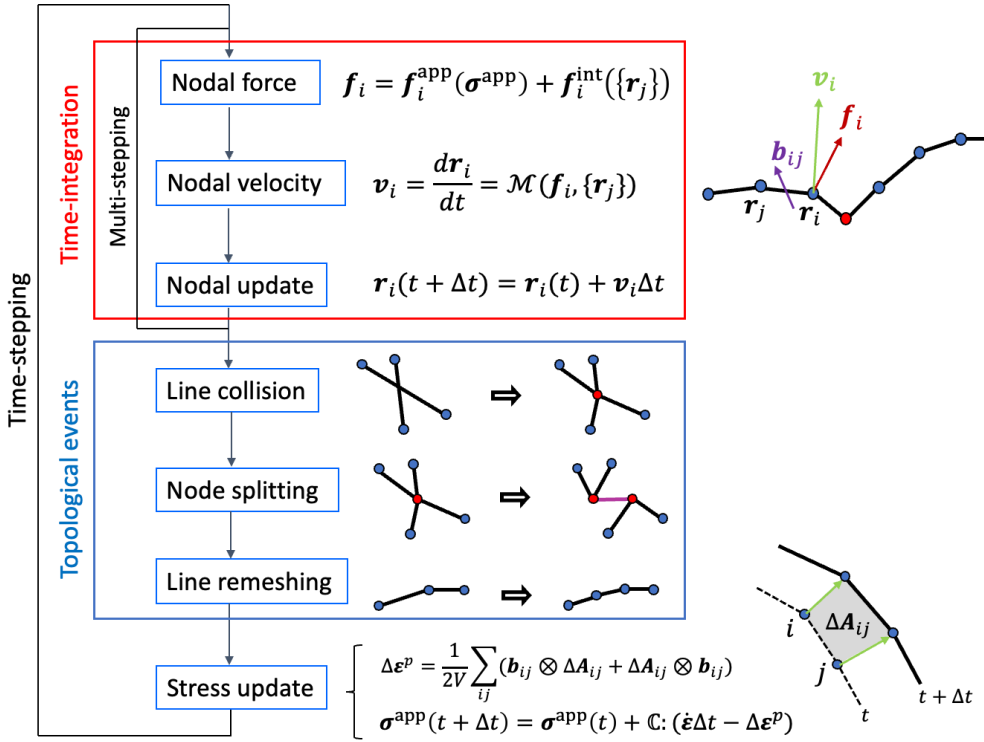
- **Microstructure representation**



Degrees of freedom:
- nodal positions $\{\boldsymbol{r}_i\}$
- Burgers vectors $\{\boldsymbol{b}_{ij}\}$



- **DDD cycle**



ExaDiS: Overview

- **Core Backend**:
  - Written in modern C++ and built on the Kokkos framework
  - Maintains a single source code version agnostic of hardware
  - Supports diverse systems: serial CPU, OpenMP, GPU (CUDA, HIP)

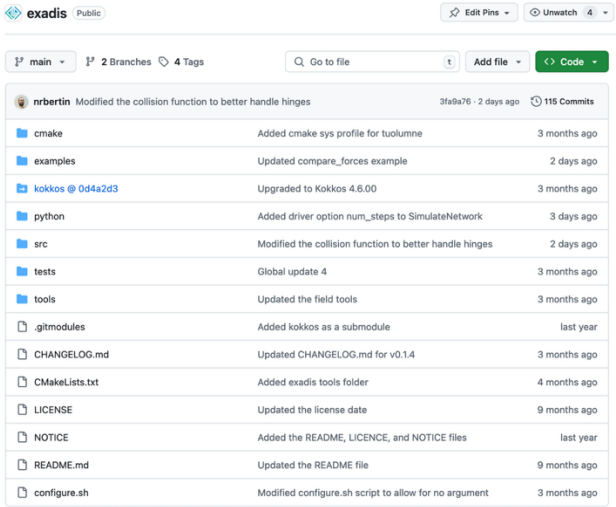
- **Modular Design**:
  - Each functionality is implemented as an independent module
  - Ensures extensibility and seamless coupling with other modules in the OpenDiS framework

- **High-Performance and Parallelism**:
  - Designed for high-performance kernel execution with a focus on parallelism
  - Abstracts complexity to lower the entry barrier for developers

- **Prototyping Support**:
  - Provides helper functions and data structures for easier initial implementation
  - Example: Topology class:
    - **`TopologySerial`**: Serial execution on CPU for prototyping
    - **`TopologyParallel`**: Parallel execution on GPU for high-performance computing

ExaDiS: Project structure

- `cmake/`: **cmake** related files, including pre-defined build system options
- `examples/`: examples of scripts and simulations
  - each example is numbered and placed in a dedicated subfolder
- `kokkos/`: directory containing the **Kokkos** submodule
- `python/`: files related to the python binding implementation
  - `pybind11/`: directory containing the **pybind11** submodule
  - `exadis_pybind.cpp`: implementation of the C++ classes / functions binding
  - `pyexadis_base.py`: interface enabling the use of **pyexadis** within OpenDiS
  - `pyexadis_utils.py`: **pyexadis** utility functions
- `src/` : C++ source and header files (*.cpp, *.h)
  - base class files and common functions are at the root of the src directory
  - individual module implementations are placed in subfolders
  - `collision_types/`: implementation of collision modules
  - `force_types/`: implementation of force modules
  - `integrator_types/`: implementation of integrator modules
  - `mobility_types/`: implementation of mobility modules
  - `neighbor_types/`: implementation of neighbor modules
  - `topology_types/`: implementation of topology modules
- `tests/`: files for testing and debugging
  - `benchmark/`: benchmark tests to evaluate the performance of the code
  - `debug.h`: debug utility functions
  - `test_exadis.cpp`: simple ExaDiS test simulations



https://github.com/LLNL/exadis

ExaDiS: Data structures

- **Dislocation network**
  - **`SerialDisNet`**
    - STL-based class that allows to easily create, manipulate, and modify dislocation networks on the host
    - Marked for a `Kokkos::Serial` execution space associated and `Kokkos::HostSpace` memory space
    - Implements all low-level topological operations, e.g. `split_seg()`, `split_node()`, and `merge_nodes()` methods
    - Designed to create initial dislocation networks and perform topological operations on existing networks
  - **`DeviceDisNet`**
    - Uses Kokkos views to store and access dislocation nodes, segments and connections for device execution (GPU)
    - Marked for a `Kokkos::DefaultExecutionSpace` execution space and corresponding default memory space
    - Defaults to the highest available execution/memory spaces available at compile time (e.g. device spaces when compiling for GPU)
    - Does not implement topological operations
  - **`DisNetManager`**
    - Used as a container to synchronize dislocation networks between `SerialDisNet` and `DeviceDisNet` for use in the different execution spaces
    - Instantiated by providing a `SerialDisNet` (resp. `DeviceDisNet`) object
    - A mirror `DeviceDisNet` (resp. `SerialDisNet`) object is automatically created
    - Provides methods `get_serial_network()` and `get_device_network()` to request network instances
    - A CPU/GPU memory copy is triggered only when needed
    - Used as the fundamental network class associated with the ExaDiS `System` object
- **`System`**
  - Base class in ExaDiS that contains all information about the simulated dislocation system, including the parameters, the crystal instance, and the dislocation network manager object
  - Fundamental data structure propagated from modules to modules
  - Must be allocated using the `exadis_new()` or the `make_system()` helper functions to ensure it is placed on a memory space accessible to all execution spaces

ExaDiS: Accessing the dislocation network

- `DisNetManager::get_serial_network()`:
  - Returns a pointer to a SerialDisNet instance of the network to be used for execution in the host execution space and whose data is allocated on the host memory space.

```
// Request a `SerialDisNet` instance of the dislocation network
SerialDisNet* net = system->get_serial_network();
// Perform operations in the serial execution space
...
```

- `DisNetManager::get_device_network()`:
  - Returns a pointer to a DeviceDisNet instance of the network to be used for execution in the device execution space and whose data is allocated on the device memory space or accessible from the device (e.g. via the use of unified memory).

```
// Request a `DeviceDisNet` instance of the dislocation network
DeviceDisNet* net = system->get_device_network();
// Perform operations in the device execution space
...
```

- Data movements:

```
void function1(System* system) {
        SerialDisNet* net = system->get_serial_network();
        // do some stuff on the host
}
void function2(System* system) {
        DeviceDisNet* net = system->get_device_network();
        // do some stuff on the device
}
// Main body
function1(system); // potential GPU to CPU memory copy
function2(system); // CPU to GPU memory copy
function2(system); // no memory copy
```

ExaDiS: Example – computing the total dislocation density

Serial implementation

```
double compute_dislocation_density(System* system) {
    // Request a `SerialDisNet` instance of the dislocation network
    SerialDisNet* net = system->get_serial_network();

    // Loop over the local segments, compute and sum their lengths
    double density = 0.0;
    for (int i = 0; i < net->Nsegs_local(); i++) {
        auto nodes = net->get_nodes(); // generic node accessor
        auto segs = net->get_segs(); // generic segment accessor
        auto cell = net->cell;

        // Get segment end nodes indices
        int n1 = segs[i].n1; // end-node 1 of segment i
        int n2 = segs[i].n2; // end-node 2 of segment i

        // Compute segment length
        Vec3 r1 = nodes[n1].pos;
        Vec3 r2 = cell.pbc_position(r1, nodes[n2].pos); // account for PBC
        double Lseg = (r2-r1).norm();

        // Increment density value by segment length
        density += Lseg;
    }

    // Normalize by the volume and convert to 1/m^2 units
    double burgmag = system->params.burgmag; // Burgers vector magnitude in m
    density *= 1.0/net->cell.volume()/burgmag/burgmag; // 1/m^2 units

    // Return the dislocation density value
    return density;
}
```

Parallel implementation

```
double compute_dislocation_density(System* system) {
    // Request a `DeviceDisNet` instance of the dislocation network
    DeviceDisNet* net = system->get_device_network();

    // Loop over the local segments, compute and sum their lengths
    double density = 0.0;
    Kokkos::parallel_reduce(net->Nsegs_local, KOKKOS_LAMBDA(const int& i, double& density_sum) {
        auto nodes = net->get_nodes(); // generic node accessor
        auto segs = net->get_segs(); // generic segment accessor
        auto cell = net->cell;

        // Get segment end nodes indices
        int n1 = segs[i].n1; // end-node 1 of segment i
        int n2 = segs[i].n2; // end-node 2 of segment i

        // Compute segment length
        Vec3 r1 = nodes[n1].pos;
        Vec3 r2 = cell.pbc_position(r1, nodes[n2].pos); // account for PBC
        double Lseg = (r2-r1).norm();

        // Increment density value by segment length
        density_sum += Lseg;
    }, density);

    // Normalize by the volume and convert to 1/m^2 units
    double burgmag = system->params.burgmag; // Burgers vector magnitude in m
    density *= 1.0/net->cell.volume()/burgmag/burgmag; // 1/m^2 units

    // Return the dislocation density value
    return density;
}
```

ExaDiS: Example – computing the total dislocation density

Templated implementation

```
template<class N>
double compute_dislocation_density(System* system, N* net) {
    // Define execution policy based on network instance type
    using policy = Kokkos::RangePolicy<typename N::ExecutionSpace>;

    // Loop over the local segments, compute and sum their lengths
    double density = 0.0;
    Kokkos::parallel_reduce(policy(0, net->Nsegs_local), KOKKOS_LAMBDA(const int& i, double& density_sum) {
        auto nodes = net->get_nodes(); // generic node accessor
        auto segs = net->get_segs(); // generic segment accessor
        auto cell = net->cell;

        // Get segment end nodes indices
        int n1 = segs[i].n1; // end-node 1 of segment i
        int n2 = segs[i].n2; // end-node 2 of segment i

        // Compute segment length
        Vec3 r1 = nodes[n1].pos;
        Vec3 r2 = cell.pbc_position(r1, nodes[n2].pos); // account for PBC
        double Lseg = (r2-r1).norm();

        // Increment density value by segment length
        density_sum += Lseg;
    }, density);

    // Normalize by the volume and convert to 1/m^2 units
    double burgmag = system->params.burgmag; // Burgers vector magnitude in m
    density *= 1.0/net->cell.volume()/burgmag/burgmag; // 1/m^2 units

    // Return the dislocation density value
    return density;
}
```

Now works both
for host (serial)

```
SerialDisNet* net = system->get_serial_network();
double density = compute_dislocation_density(system, net);
```

and device (parallel)

```
DeviceDisNet* net = system->get_device_network();
double density = compute_dislocation_density(system, net);
```

execution!

ExaDiS: Implementing a mobility law

Mobility law template

```
struct MobilityMOBNAME
{
    // Mandatory flag to instruct whether it is a linear or non-linear mobility law
    bool non_linear = false;

    // Mobility parameters
    struct Params {
        double drag = 1.0;
        Params() {}
        Params(double _drag) : drag(_drag) {}
    };
    Params params;

    // Constructor
    MobilityMOBNAME(System* system, Params _params) {
        // Initialize
        params = _params;
    }

    // Node velocity kernel
    template<class N>
    KOKKOS_INLINE_FUNCTION
    Vec3 node_velocity(System* system, N* net, const int& i, const Vec3& fi)
    {
        // Compute nodal velocity of node i under force fi
        Vec3 vi(0.0);
        // implements mobility here
        return vi;
    }

    static constexpr const char* name = "MobilityMOBNAME";
};

namespace MobilityType {
    typedef MobilityLocal<MobilityMOBNAME> MOBNAME;
}
```

Python binding to pyexadis

1. Register the mobility parameters in `exadis_pybind.cpp`

```
py::class_<MobilityType::MOBNAME::Params>(m, "Mobility_MOBNAME_Params")
    .def(py::init<double>(), py::arg("drag"));
```

2. Define the python mobility maker in `exadis_pybind.cpp`

```
m.def("make_mobility_mobname", &make_mobility<MobilityType::MOBNAME>,
      "Instantiate a MOBNAME mobility law",
      py::arg("params"), py::arg("mobparams"));
```

3. Register the mobility for the `TopologyParallel` module in `exadis_pybind.h` (optional, only required to make the mobility compatible with `TopologyParallel`)

```
} else if (strcmp(mobility->name(), "MobilityMOBNAME") == 0) {
    topology = new TopologyParallel<F,MobilityType::MOBNAME>(system, force, mobility, topolparams);
```

4. Add the mobility model to the `MobilityLaw` wrapper class in `pyexadis_base.py`

```
elif self.mobility_law == 'MOBNAME':
    drag = kwargs.get('drag', 1.0)
    mobparams = pyexadis.Mobility_MOBNAME_Params(drag)
    self.mobility = pyexadis.make_mobility_mobname(params=params, mobparams=mobparams)
```

ExaDiS: Implementing a force calculation module

Force template

```
class ForceTemplate : public Force {
private:
    // Member properties
    Vec3 fval;

public:
    // Force parameters
    struct Params {
        Vec3 fval = 0.0;
        Params() {}
        Params(Vec3 val) : fval(val) {}
    };

    // Constructor
    ForceTemplate(System* system, Params params) {
        // Initialize    }

    // Pre-compute operation
    void pre_compute(System* system) {} // nothing to pre-compute here

    // Global compute
    void compute(System* system, bool zero=true) {
        // Compute nodal forces for all nodes
        ...
    }

    // Individual compute
    Vec3 node_force(System* system, const int& i) {
        // Compute individual force at node i
        Vec3 fi = ...
        return fi;
    }

    // Team individual node force implementation (optional)
    template<class N>
    KOKKOS_INLINE_FUNCTION
    Vec3 node_force(System* system, N* net, const int& i, const team_handle& team) {
        // Compute individual force at node i using a team of workers
        Vec3 fi = ...
        return fi;
    }
};
```

SegForce template

```
struct SegForceTemplate
{
    // Mandatory flags to instruct what kernels are implemented in the struct
    static const bool has_pre_compute = false;
    static const bool has_compute_team = false;
    static const bool has_node_force = false;

    // Force parameters
    struct Params {
        Vec3 fval = 0.0;
        Params() {}
        Params(Vec3 val) : fval(val) {}
    };
    Params params;

    // Constructor
    SegForceTemplate(System* system, Params _params) {
        // Initialize
        ...
    }

    // Segment force kernel
    template<class N>
    KOKKOS_INLINE_FUNCTION
    SegForce segment_force(System* system, N* net, const int& i)
    {
        auto nodes = net->get_nodes(); // generic node accessor
        auto segs = net->get_segs(); // generic segment accessor
        auto cell = net->cell;

        // Get segment end nodes indices
        int n1 = segs[i].n1; // end-node 1 of segment i
        int n2 = segs[i].n2; // end-node 2 of segment i

        Vec3 f1 = ... // force on node 1
        Vec3 f2 = ... // force on node 2
        return SegForce(f1, f2);
    }
};

typedef ForceSeg<SegForceTemplate> ForceSegTemplate;
```

Outline

- OpenDiS Code Structure
  - DisNetManager (DisNet and ExaDisNet)
  - state dictionary
  - Modules (relationships and dependencies)
- A User-Defined Module – Python Implementation
  - Force / mobility specification classes
  - Mobility example
- A User-Defined Module – ExaDiS Implementation
  - Introduction to ExaDiS
  - Density / Force / Mobility examples
- **A User-Defined Type of Output Files**

Example: overriding the simulation driver

```
class SimulateNetwork:
    """SimulateNetwork: base simulation driver
    """

    def step(self, N: DisNetManager, state: dict):
        """step: take a time step of DD simulation on DisNetManager N
        """
        # Step begin
        self.step_begin(N, state)

        # Step time-integrate
        self.step_integrate(N, state)

        # Step post-integrate
        self.step_post_integrate(N, state)

        # Step topological operations
        self.step_topological_operations(N, state)

        # Step update response
        self.step_update_response(N, state)

        # Output and visualization
        self.step_write_files(DM, state)
        self.step_print_info(DM, state)
        self.step_visualize(DM, state)

        # Step end
        self.step_end(N, state)

    def run(self, N: DisNetManager, state: dict):
        """run: run a DDD simulation
        """
        ...
        for tstep in range(start_step, max_step):
            state["istep"] = tstep+1
            self.step(N, state)
        ...
```

pydis / pyexadis

simulate/sim_disnet.py

Can override any of the base steps from the DDD cycle invoked in the the simulation driver

e.g.

tutorials/03_write_vtk/user_write_vtk_1.py

Example: generating user-defined output files during a run

Define new class, override behavior

tutorials/03_write_vtk/user_write_vtk_1.py

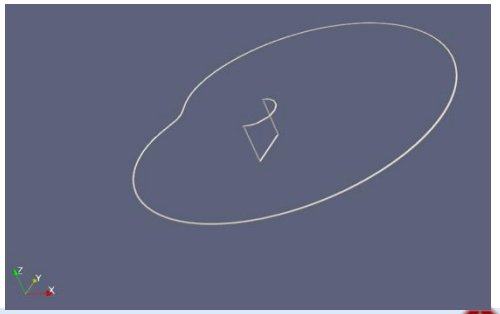
```
class My_SimulateNetwork(SimulateNetwork):
    def step_write_files(self, DM: DisNetManager, state: dict):
        if self.write_freq != None:
            istep = state['istep']
            if istep % self.write_freq == 0:
                #DM.write_json(os.path.join(self.write_dir, f'disnet_{istep}.json'))
                save_DisNet_to_vtp(DM, os.path.join(self.write_dir, f'disnet_{istep}.vtp'))
                if self.save_state:
                    with open(os.path.join(self.write_dir, f'state_{istep}.pickle'), 'wb') as file:
                        pickle.dump(state, file)
```

Run new simulation

tutorials/03_write_vtk/test_vtk_frank_read_src_pydis.py

```
sim = My_SimulateNetwork(calforce=calforce, mobility=mobility, timeint=timeint,
                         topology=topology, collision=collision, remesh=remesh, vis=vis,
                         state=state, max_step=200, loading_mode="stress",
                         applied_stress=np.array([0.0, 0.0, 0.0, 0.0, -4.0e8, 0.0]),
                         print_freq=10, plot_freq=10, plot_pause_seconds=0.01,
                         write_freq=10, write_dir='output', save_state=False)
sim.run(net, state)
```

Paraview



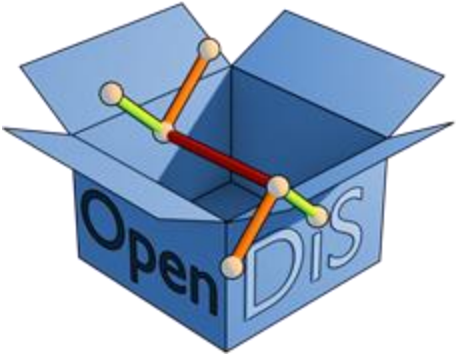
Wrap-up of Session 3

- **What we covered**
  - OpenDiS code structure
  - How to develop new module – start with PyDiS
  - How to develop new module – high performance implementation in ExaDiS
  - How to override the simulation driver – example: write vtk files

- **What we will cover next**
  - How to add user-defined enhancements, e.g. user-defined loading conditions
  - User-defined force contribution
  - Dislocation-precipitate interactions
  - Demo library for boundary value problem (BVP)

Feel free to send questions in zoom chat

Email us with questions during the break: bertin1@llnl.gov, caiwei@stanford.edu



https://opendis.github.io/OpenDiS/

OpenDiS Workshop

Open Dislocation **S**imulator

Session 4

Nicolas Bertin, Lawrence Livermore National Laboratory
Wei Cai, Stanford University

LLNL-PRES-2006898

Welcome! Here is the Zoom protocol:

1. Please mute your microphone
2. Feel free to turn on your camera (optional/encouraged)
3. Feel free to "raise hand" if you have a question/comment
4. Feel free to send your question/comment to chat

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 Lawrence Livermore National Security, LLC.

Wrap-up of Session 4

- **What we covered**
  - OpenDiS code structure
  - How to develop new module – start with PyDiS
  - How to develop new module – high performance implementation in ExaDiS
  - How to override the simulation driver – example: write vtk files

- **What we will cover in Session 4**
  - How to add user-defined enhancements
    - User-defined loading conditions
    - User-defined force contribution
    - Simple dislocation / precipitate interactions
    - A demo library (BVP)

Feel free to send questions in zoom chat
Email us with questions during the break: bertin1@llnl.gov, caiwei@stanford.edu

Outline

- User-defined loading conditions
- User-defined force contribution
- Simple dislocation-precipitate interactions
- A Demo Library for boundary value problem (BVP)

Example: user-defined loading conditions

```
from pyexadis_base import SimulateNetwork
class SimulationDriver(SimulateNetwork):
    """ User-defined simulation driver to impose a strain-rate tensor loading
    """
    def __init__(self, *args, **kwargs) -> None:
        super(SimulationDriver, self).__init__(*args, **kwargs)
        self.strain_rate_tensor = kwargs.get("strain_rate_tensor")
        state = kwargs.get("state")
        self.MU, self.NU = state["mu"], state["nu"]
        self.LA = 2*self.MU*self.NU/(1-2*self.NU)

    # Override step_update_response() function to apply strain-rate tensor loading
    def step_update_response(self, N: DisNetManager, state: dict):
        """step_update_response: update applied stress and rotation if needed
        """
        if self.loading_mode == 'strain_rate_tensor':

            # get values of plastic strain, plastic spin, and density computed internally in exadis
            dEp, dWp, state["density"] = N.get_disnet(ExaDisNet).net.get_plastic_strain()
            dEp = np.array(dEp).ravel()[[0,4,8,5,2,1]] # xx,yy,zz,yz,xz,xy
            dWp = np.array(dWp).ravel()[[5,2,1]] # yz,xz,xy
            state["dEp"], state["dWp"] = dEp, dWp

            # update strain and stress states based on strain rate tensor
            dE = self.strain_rate_tensor * state["dt"]
            dEe = dE - dEp # elastic strain
            dstress = self.LA*np.sum(dEe[0:3])*np.array([1,1,1,0,0,0]) + 2*self.MU*dEe

            # increment stress and strain tensors
            state["applied_stress"] += dstress
            state["Etot"] += dE

            def von_mises(T):
                S = np.array(T[[0,5,4,5,1,3,4,3,2]]).reshape(3,3)
                Sdev = S - np.trace(S)/3.0*np.eye(3)
                return np.sqrt(3.0/2.0*np.dot(Sdev.ravel(), Sdev.ravel()))

            # store strain and stress values used for the output (e.g. von Mises)
            state["strain"] = von_mises(state["Etot"])
            state["stress"] = von_mises(state["applied_stress"])

        else:
            # call base class function
            super().step_update_response(N, state)

        return state
```

Call user-defined driver to run the simulation

```
strain_rate_tensor = 1e3*np.array([1.,1.,1.,0.,0.,0.]) # xx,yy,zz,yz,xz,xy

sim = SimulationDriver(..., strain_rate_tensor=strain_rate_tensor)
sim.run(N, state)
```

Outline

- User-defined loading conditions
- **User-defined force contribution**
- Simple dislocation-precipitate interactions
- A Demo Library for boundary value problem (BVP)

Example: user-defined force contribution

Add additional stress to base force model

```
class CalForceUserStress:
    """ User-defined CalForce function to add a
    user-defined stress contribution to a base CalForce model.
    """
    def __init__(self, state: dict, calforce_base: CalForce, **kwargs) -> None:
        self.calforce_base = calforce_base
        self.add_user_stress = 1

    def PreCompute(self, N: DisNetManager, state: dict) -> dict:
        self.calforce_base.PreCompute(N, state)
        return state

    def NodeForce(self, N: DisNetManager, state: dict, pre_compute=True) -> dict:

        # Compute base forces
        self.calforce_base.NodeForce(N, state, pre_compute)

        # Add user-defined stress contribution
        if self.add_user_stress:
            self.AddUserStress(N, state)

        return state

    def OneNodeForce(self, N: DisNetManager, state: dict, tag, update_state=True) -> np.array:
        ...
        return f
```

User-defined stress contribution

```
    def AddUserStress(self, N: DisNetManager, state: dict) -> dict:

        G = N.get_disnet(ExaDisNet)
        cell = G.cell
        rn = G.get_nodes_data()["positions"]
        segs = G.get_segs_data()
        segsnid = segs["nodeids"] # segment connectivity to nodes

        # segments end-nodes positions and Burgers vectors
        r1 = np.array(cell.closest_image(Rref=np.array(cell.center()), R=rn[segsnid[:,0]]))
        r2 = np.array(cell.closest_image(Rref=r1, R=rn[segsnid[:,1]]))
        r_ij = r2-r1
        b_ij = segs["burgers"]

        # compute user stress at segment mid-positions
        Rseg = 0.5*(r1+r2)
        user_stress = UserStress(Rseg, cell, state)

        # add PK force from user stress to current force
        sig_user = user_stress[:,[0,5,4,5,1,3,4,3,2]].reshape(-1,3,3)
        sigb = np.einsum('kij,kj->ki', sig_user, b_ij)
        fpkuser = 0.5 * np.cross(sigb, r_ij)
        f = G.get_forces() # current nodal forces
        np.add.at(f, segsnid[:,0], fpkuser)
        np.add.at(f, segsnid[:,1], fpkuser)

        # store new forces in state dictionnary
        state["nodeforces"] = f

        return state

def UserStress(R, cell, state):
    """ User-defined additional stress contribution
    """
    user_stress = np.zeros((R.shape[0],6)) # xx,yy,zz,yz,xz,xy in Pa
    ...
    return user_stress
```

Outline

- User-defined loading conditions
- User-defined force contribution
- **Simple dislocation-precipitate interactions**
- A Demo Library for boundary value problem (BVP)

Example: simple dislocation/precipitate interaction


```
from pyexadis_base import SimulateNetwork, NodeConstraints
class SimulationDriver(SimulateNetwork):
    """SimulationDriver
    User-defined simulation driver that adds the precipitate-dislocation interaction
    """
    def __init__(self, *args, **kwargs) -> None:
        super(SimulationDriver, self).__init__(*args, **kwargs)
        self.precips = kwargs.get("precips")

    def precipitates_intersection(self, N: DisNetManager, state: dict, xold):
        """precipitates_intersection:
        Adjust node positions of segments entering a precipitate
        """
        # Detect nodes that are inside precipitates and move
        # them to the precipitate surface
        data = N.export_data()
        pos = data["nodes"]["positions"]
        cons = data["nodes"]["constraints"]

        def sphere_intersect(ro, rd, ce, ra):
            oc = ro - ce
            b = np.einsum('ij,ij->i', oc, rd)
            qc = oc - b[:,None]*rd
            h = ra**2 - np.einsum('ij,ij->i', qc, qc)
            x, y = -1.0, -1.0
            ind = (h >= 0.0)
            h[ind] = np.sqrt(h[ind])
            x, y = -b-h, -b+h
            x[~ind], y[~ind] = -1, -1
            return x, y

        for p in self.precips:
            dpos = xold-pos
            dposn = np.linalg.norm(dpos, axis=1)
            ind = np.argwhere((np.sum((pos-p.pos)**2, axis=1) < p.radius**2) & (dposn > 1e-10)).ravel()
            d = dpos[ind]/dposn[ind,None]
            x, y = sphere_intersect(pos[ind], d, p.pos, p.radius)
            ind = ind[x<0]
            pos[ind] += y[x<0,None]*d
            #cons[ind] = NodeConstraints.PINNED_NODE

        # Commit changes
        N.get_disnet(ExaDisNet).import_data(data)

    # Override step_begin to store beginning-of-step node positions
    def step_begin(self, N: DisNetManager, state: dict):
        """step_begin: invoked at the begining of each time step
        """
        self.xold = N.export_data()["nodes"]["positions"].copy()

    # Override step_post_integrate to handle dislocation-precipitate intersections
    def step_post_integrate(self, N: DisNetManager, state: dict):
        """step_post_integrate: invoked after time-integration of each time step
        """
        self.precipitates_intersection(N, state, self.xold)
```

User-defined visualization module!

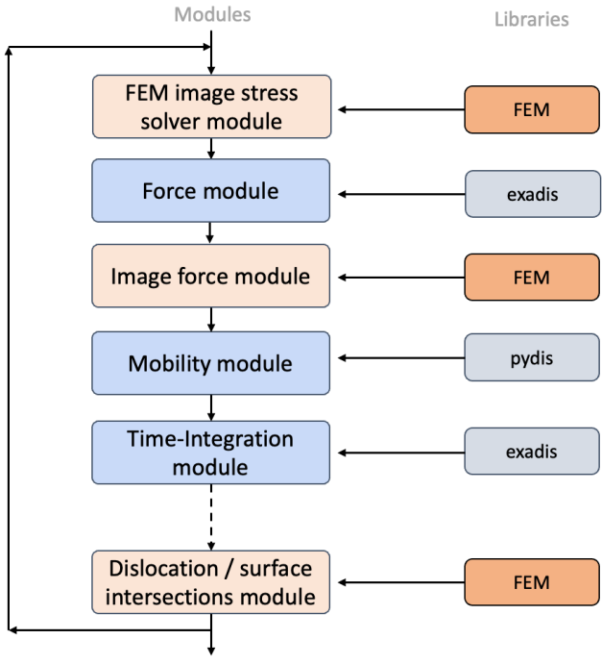
```
import matplotlib.pyplot as plt
from pyexadis_base import VisualizeNetwork
class VisualizeNetworkPrecips(VisualizeNetwork):
    """VisualizeNetworkPrecips
    User-defined visualizer to plot dislocations and precipitates
    """
    def __init__(self, *args, **kwargs) -> None:
        super(VisualizeNetworkPrecips, self).__init__(*args, **kwargs)
        self.precips = kwargs.get("precips")

    def plot_sphere(self, ax, pos, radius):
        u = np.linspace(0, 2 * np.pi, 20)
        v = np.linspace(0, np.pi, 20)
        x = pos[0] + radius * np.outer(np.cos(u), np.sin(v))
        y = pos[1] + radius * np.outer(np.sin(u), np.sin(v))
        z = pos[2] + radius * np.outer(np.ones(np.size(u)), np.cos(v))
        ax.plot_surface(x, y, z, zorder=3.5, alpha=1.0)

    def plot_disnet(self, N: DisNetManager, state: dict={},
                    plot_nodes=True, plot_segs=True, plot_cell=True, trim=False,
                    fig=None, ax=None, block=False, pause_seconds=0.01):
        # plot dislocations using base class
        fig, ax = super().plot_disnet(N, state, plot_nodes, plot_segs, plot_cell,
                                      trim, fig, ax, block, pause_seconds=-1)
        # plot precipitates
        ax.computed_zorder = False
        for p in self.precips:
            self.plot_sphere(ax, p.pos, p.radius)
        # display plot
        plt.draw()
        plt.show(block=block)
        plt.pause(pause_seconds)
```

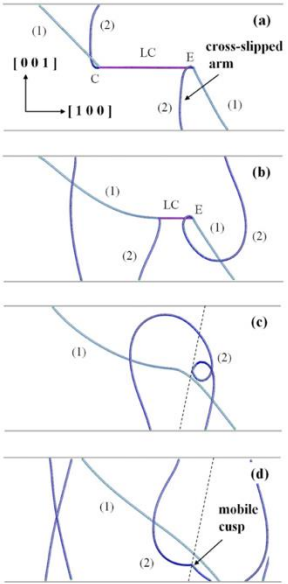
Outline

- User-defined loading conditions
- User-defined force contribution
- Simple dislocation-precipitate interactions
- **A Demo Library for boundary value problem (BVP)**

OpenDiS: adding a user-defined library

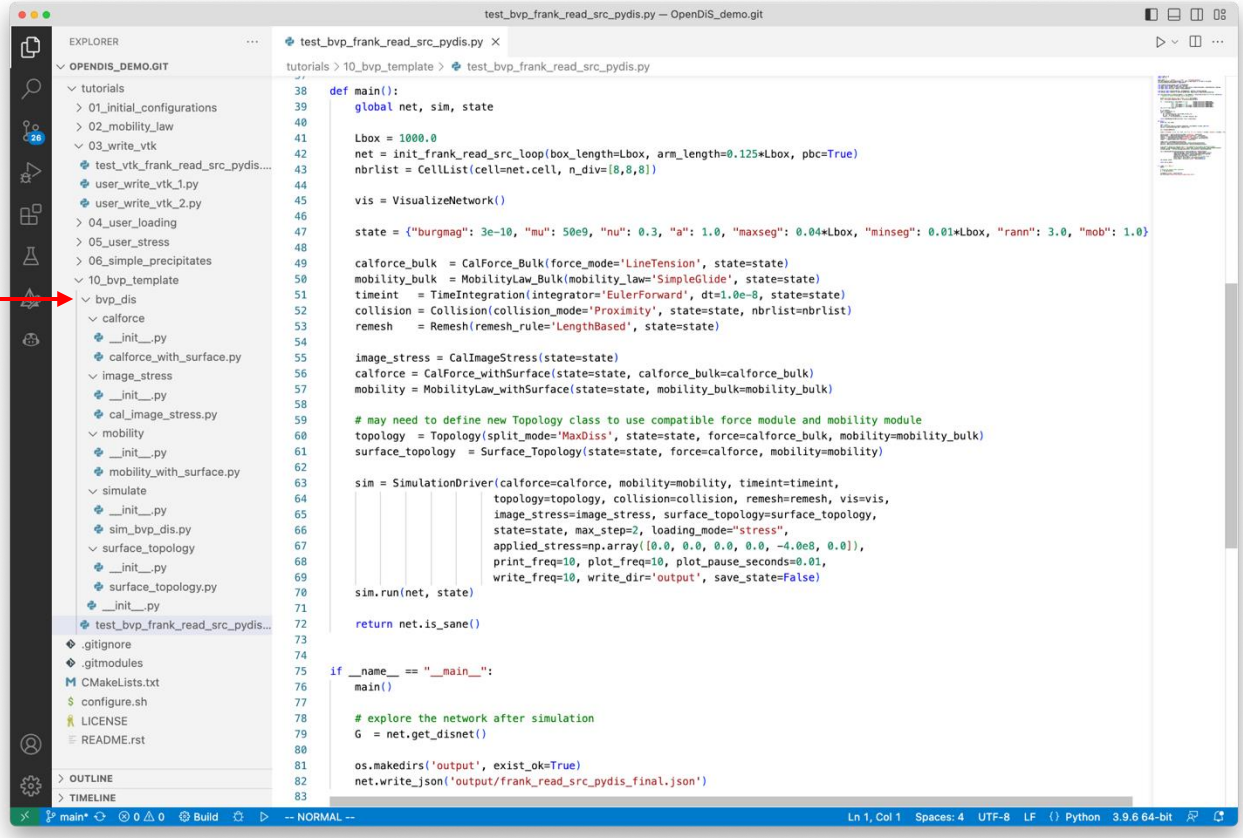
- Library: collection of modules that implement specific functionalities (e.g. FEM coupling, analysis tools, etc.)





Dislocation Junctions and Jogs in Free Standing Thin Films", *Modelling and Simulation in Materials Science and Engineering*, 19, 025002 (2011)

OpenDiS: adding a user-defined library



Example:

tutorials/10_bvp_template

Library (Python module):

bvp_dis

follows the same structure as core libraries:

pydis, exadis

Wrap-up

Wrap-up of the workshop

OpenDiS is a work in progress

- This is not a final product!

Contribution Opportunities

- Contributions are welcomed/needed in multiple areas
  - New modules, optimization, documentation
  - FEM coupling, multiphysics simulations
  - More complex physics (e.g. HEAs, solutes, precipitates)
  - Interface to ML
  - Continuous integration / testing
  - Container / Docker package (easy to use, reproducibility)
  - Infrastructure: platform to share script, data, etc.
- Benefits of contributing (e.g., advancing the field, networking , recognition, etc.)

- How to get started: Github fork -> pull/merge requests (+ contribution guidelines)

We invite you to join the project!

Wrap-up of the workshop

Contribution Opportunities

- Contributions are welcomed/needed in multiple areas
- Benefits of contributing (e.g., advancing the field, networking , recognition, etc.)
- How to get started: Github repository, contribution guidelines

We invite you to join the project!

Community and Collaboration

- Building a community around OpenDiS (e.g., forums, regular workshops, hackathons).
- Follow successful examples of open-source projects (LAMMPS, OpenKIM, etc.)
- Welcoming any feedback, ideas for improvements (project, code, etc.)

Resources

- Documentation
- GitHub repo / issues / discussions
- Contact: bertin1@llnl.gov, caiwei@stanford.edu

Thank you for your participation!

https://opendis.github.io/OpenDiS