materials modeling workflows

SAND2021-2303C

Steve Plimpton, Aidan Thompson, and Mitch Wood
Sandia National Laboratories

TMS 2021 Virtual Annual Meeting

Symposium on Practical Tools for
Integration and Analysis in Materials Engineering

March 2021

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

LAMMPS molecular dynamics code

- https://lammps.sandia.gov
- https://github.com/lammps/lammps

- Classical molecular dynamics for materials modeling
- Sub-atomic to atomic to CG to meso to continuum scales
- Most people use it as a **stand-alone code**

LAMMPS molecular dynamics code

- https://lammps.sandia.gov
- https://github.com/lammps/lammps

- Classical molecular dynamics for materials modeling
- Sub-atomic to atomic to CG to meso to continuum scales
- Most people use it as a **stand-alone code**

- Features that enable **code coupling** and **workflows**
  1. Build and call LAMMPS as a **library**
  2. Wrap with **Python** or call Python from LAMMPS or both
  3. Support for **MDI** (MolSSI Driver Interface) for code coupling
  4. Inter-operability with **OpenKIM** database of potentials
  5. **Machine-learning** capabilities

(1) LAMMPS as a library

- Enables coupled simulations: multiphysics, multiscale
- Enables use in a complex workflow
- Often just requires bit of **re-design** at highest level

(1) LAMMPS as a library

- Enables coupled simulations: multiphysics, multiscale
- Enables use in a complex workflow
- Often just requires bit of **re-design** at highest level
- Allow **multiple instantiations** of your "simulator"
  - another code can call it (from any language)
  - can embed it multiple times within a higher-scale model
- **Code details**
  - no global variables
  - don't use MPI_COMM_WORLD (not using all processors)
- Provide a **library interface** (API) to basic functionality
  - C-style interface has maximum portability
  - callable from any language (C++,C,Fortran,etc)
  - enables scripting (from **Python**)
  - enables wrapping/connecting with a **GUI** or **visualizer**
  - API should be easily extensible, just add a new function

Main.cpp for LAMMPS

```
include "lammps.h"
int main(int argc, char **argv) {
  MPI_Init(&argc,&argv);
  LAMMPS *lammps =
    new LAMMPS(argc,argv,MPI_COMM_WORLD);
  lammps->input->file(); // read input script
  delete lammps;
  MPI_Finalize();
}
```

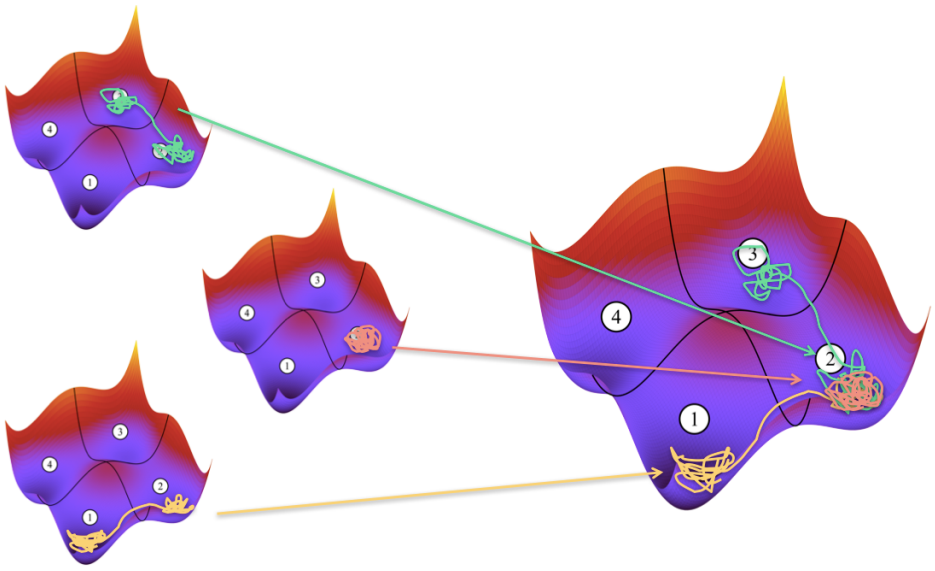
Main.cpp for LAMMPS

```
include "lammps.h"
int main(int argc, char **argv) {
  MPI_Init(&argc,&argv);
  LAMMPS *lammps =
    new LAMMPS(argc,argv,MPI_COMM_WORLD);
  lammps->input->file(); // read input script
  delete lammps;
  MPI_Finalize();
}
```

- Top-level LAMMPS is just a single class
- Just **remove main.cpp** to compile as a library
- Library call returns ptr to an instance of LAMMPS
  - C++ caller: use that ptr
  - anything else: include opaque ptr in other lib calls

ParSplice using LAMMPS as an MD engine

- Developed by Danny Perez (LANL) and collaborators
  Long-time dynamics through parallel trajectory splicing,
  D Perez, et. al., JCTC, 12, 18-28 (2016).



- Enables μs-ms time-accurate runs with 10^2-10^6 MD replicas
  - for rare-event systems (solid-state)
  - assigns configs to explore PE landscape, run each for few ps
  - detects and catalogs events in a distributed database
  - splices event sequence into long trajectories with correct stats
- Extended LAMMPS library API to facilitate this usage mode

Provide callback functions

Callback = when library calls back to the calling program

Provide callback functions

Callback = when library calls back to the calling program

Example:

- QM code instantiates LAMMPS as a library
- QM code invokes MD run
  - uses **fix external**, sets a function pointer
  - triggers callback at appropriate point in timestep
- LAMMPS calls back to QM each step to get QM forces
- QM code can access LAMMPS data (coords, forces, etc)

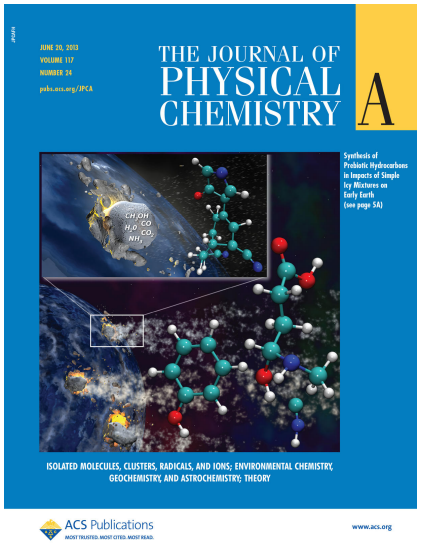
Provide callback functions

Callback = when library calls back to the calling program

Example:

- QM code instantiates LAMMPS as a library
- QM code invokes MD run
  - uses **fix external**, sets a function pointer
  - triggers callback at appropriate point in timestep
- LAMMPS calls back to QM each step to get QM forces
- QM code can access LAMMPS data (coords, forces, etc)

- Nir Goldman (LLNL) runs Fortran **DFTB+** with LAMMPS in this manner
- *Prebiotic Chemistry Within a Simple Impacting Icy Mixture*, N. Goldman and I. Tamblyn, J Phys Chem A, 117, 5124 (2013).



Other examples for using LAMMPS as a library

Tom Swinburne (CINaM Marseille, CNRS)

- TAMMBER: explore energy landscape to build kMC model
- 1000s of TAD runs, energy minimizations
- Phys Rev Materials 2018, DOI: 10.1103/PhysRevMaterials.2.053802

Noam Bernstein (NRL)

- Nested sampling for P-T-composition phase diagrams
- TE-HMC & GMC with LAMMPS lib as MD engine via ASE
- Phys Rev E 2017, DOI: 10.1103/PhysRevE.96.043311

ASE = Atomic Simulation Environment

- `https://wiki.fysik.dtu.dk/ase`
- simulation protocol (setup, run, analyze, viz) in Python
- supports dozens of QM and MD simulators as **calculators**
- can use LAMMPS either as a library or a stand-alone code

(2) LAMMPS with Python

- **Python** = most popular scripting language for scientific computing, workflows, machine learning

(2) LAMMPS with Python

- **Python** = most popular scripting language for scientific computing, workflows, machine learning

1. **Ctypes** module makes it easy to wrap any C-style API
   - every LAMMPS library function is exposed to Python
   - also allows callbacks to a Python function
2. **Multiprocessing** module to loop over 1000s of tasks
   $\Rightarrow$ each of which can instantiate LAMMPS as a lib
3. **Mpi4py** module enables parallel Python and split of MPI
   $\Rightarrow$ run one or more LAMMPS instances in parallel

(2) LAMMPS with Python

- **Python** = most popular scripting language for scientific computing, workflows, machine learning

1. **Ctypes** module makes it easy to wrap any C-style API
   - every LAMMPS library function is exposed to Python
   - also allows callbacks to a Python function
2. **Multiprocessing** module to loop over 1000s of tasks
   $\Rightarrow$ each of which can instantiate LAMMPS as a lib
3. **Mpi4py** module enables parallel Python and split of MPI
   $\Rightarrow$ run one or more LAMMPS instances in parallel

- LAMMPS data structs accessible by **Numpy** (no data copy)

```
nlocal = lammps.extract_global("nlocal")
cptr = lammps.extract_atom("x")
coords = np.ctypeslib._array(cptr,shape=(nlocal,3))
```

- Python versions of LAMMPS **styles**: pair, fix, variable

(3) MolSSI Driver Interface (MDI) for code coupling

Goal: Enable scientific codes to easily exchange data (for materials modeling)

- **MDI** = library created by Taylor Barnes at MolSSI
- MolSSI = Molecular Sciences Software Institute (NSF funded)
  - https://molssi.org
  - https://molssi-mdi.github.io/MDI_Library
- Nice paper: TA Barnes et al, *The MolSSI Driver Interface Project: A framework for standardized, on-the-fly interoperability between computational molecular sciences codes*, Comp Phys Comm, 261, 107688 (2021).
- Standardized, high-level way to couple codes via **client/server** model (driver/engine in MDI lingo)

MDI in a nutshell for materials modeling

- Example #1: AIMD or QM/MM
  - **MD** = driver: sends coords to QM
  - **QM** = engine: computes and returns forces
  - QM/MM Python script is driver, MD and QM are both engines
- Example #2: Monte Carlo using MD as an engine
  - **MC** = driver: makes a complex move (configurational bias)
  - **MD** = engine: evaluates energy, (optionally) runs MD

MDI in a nutshell for materials modeling

- Example #1: AIMD or QM/MM
  - **MD** = driver: sends coords to QM
  - **QM** = engine: computes and returns forces
  - QM/MM Python script is driver, MD and QM are both engines
- Example #2: Monte Carlo using MD as an engine
  - **MC** = driver: makes a complex move (configurational bias)
  - **MD** = engine: evaluates energy, (optionally) runs MD
- More examples: ML, NEB, sampling, PIMD, etc
- Driver can use **multiple instances** of multiple engines
- Umbrella drivers can be Python scripts

MDI in a nutshell for materials modeling

- Example #1: AIMD or QM/MM
  - **MD** = driver: sends coords to QM
  - **QM** = engine: computes and returns forces
  - QM/MM Python script is driver, MD and QM are both engines
- Example #2: Monte Carlo using MD as an engine
  - **MC** = driver: makes a complex move (configurational bias)
  - **MD** = engine: evaluates energy, (optionally) runs MD
- More examples: ML, NEB, sampling, PIMD, etc
- Driver can use **multiple instances** of multiple engines
- Umbrella drivers can be Python scripts
- Codes **exchange strings & data** via MDI calls (MPI-like)
  - ">COORDS" = here are my atom coords
  - "<COORDS" = send me your atom coords
  - under the hood: MPI or TCP/IP messaging (user choice)
  - MD can make MDI calls in middle of timestep

Advantages of client/server mode of code coupling

1. Two codes can be **stand alone** executables
2. Codes in **different languages**: C/C++, Fortran, Python
3. Easy to run codes on **different #s of processors**:
   - mpirun -np 4 driver -mdi ... -np 60 engine -mdi ...
   - sockets means 2 codes can run in different places
4. Use MDI-defined or custom **data exchange protocols**:
   - any MD code can work with any QM code

Advantages of client/server mode of code coupling

1. Two codes can be **stand alone** executables
2. Codes in **different languages**: C/C++, Fortran, Python
3. Easy to run codes on **different #s of processors**:
   - mpirun -np 4 driver -mdi ... -np 60 engine -mdi ...
   - sockets means 2 codes can run in different places
4. Use MDI-defined or custom **data exchange protocols**:
   - any MD code can work with any QM code

- **LAMMPS** about to release MDI support (driver & engine)
- Other codes with some level of MDI support:
  - **MD**: Tinker, OpenMM
  - **QM**: Q Espresso, Molpro, MOPAC, GAMESS, NWChem, Q-Chem, Turbomole, CP2K, Siesta, ...
  - All except QE support MDI via **QCEngine** wrapper (files)
- **Key point**: any code can use MDI, not just MD & QM

(4) Interoperability with OpenKIM project

Open Knowledgebase of Interatomic Models (NSF funded)
https://openkim.org

- Goal: enable reliable, reproducible, portable atomistic sims
- Repository of interatomic potentials (tested, versioned)
- Material properties calculated for all potentials
- Any materials code can use potentials via KIM API

(4) Interoperability with OpenKIM project

Open Knowledgebase of Interatomic Models (NSF funded)
https://openkim.org

- Goal: enable reliable, reproducible, portable atomistic sims
- Repository of interatomic potentials (tested, versioned)
- Material properties calculated for all potentials
- Any materials code can use potentials via KIM API
- LAMMPS has a **pair_style kim** command:
  - any KIM model can be used in a LAMMPS simulation
  - **model** = analytic potential plus material-specific parameters
  - KIM has ∼500 such models
  - also has ∼20 analytic potentials not natively in LAMMPS
- LAMMPS input scripts can **query KIM website** in real time
  - list of models that match criteria
  - model parameters: lattice type, lattice constant, α coeff, etc

Example workflow using LAMMPS + KIM repository

```
variable species index Ni        # grab list of Ni potentials
kim query model index get_available_models species=[${species}]

label loop
    log log.${model}
    kim init ${model} metal      # initialize for metal units
    kim query alat ...           # this potential's lattice constant
    lattice fcc ${alat}
    create_box ...
    create_atoms ...
    kim interactions ${species}  # invokes pair_style kim with model
    dump 1 all custom 1000 dump.${model}
    run 100000
    clear
    next model

jump SELF loop
```

Example workflow using LAMMPS + KIM repository

```
variable species index Ni        # grab list of Ni potentials
kim query model index get_available_models species=[${species}]

label loop
    log log.${model}
    kim init ${model} metal      # initialize for metal units
    kim query alat ...           # this potential's lattice constant
    lattice fcc ${alat}
    create_box ...
    create_atoms ...
    kim interactions ${species}  # invokes pair_style kim with model
    dump 1 all custom 1000 dump.${model}
    run 100000
    clear
    next model

jump SELF loop
```

- Could loop on species, query alloy models, run N sims at once, each in parallel

(5) Machine learning: workflows for fitting ML potentials

- Machine learning interatomic potentials for classical MD:
  - trained on **large volumes** of QM data
  - forces via **geometry-based** eqs or NN, not physics-based eqs
  - **goal**: QM accuracy in an MD simulation for cheap O(N) cost

(5) Machine learning: workflows for fitting ML potentials

- Machine learning interatomic potentials for classical MD:
  - trained on **large volumes** of QM data
  - forces via **geometry-based** eqs or NN, not physics-based eqs
  - **goal**: QM accuracy in an MD simulation for cheap O(N) cost
- Typical training procedure is a workflow:
  - DFT data for small configs: energy, forces, virial = **QM truth**
  - Each atom + neighborhood $\Rightarrow$ geometric **descriptors**
  - Descriptors used in **big matrix** eq or as input to **NN**
  - Solve big matrix (e.g. least squares) or iteratively train NN
  - Optimize over additional **hyper parameters** for best fit

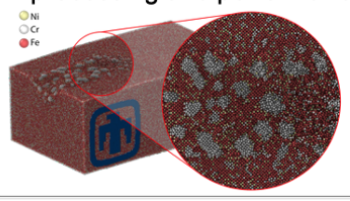
(5) Machine learning: workflows for fitting ML potentials

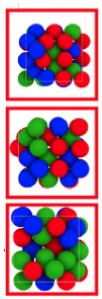
- Machine learning interatomic potentials for classical MD:
  - trained on **large volumes** of QM data
  - forces via **geometry-based** eqs or NN, not physics-based eqs
  - **goal**: QM accuracy in an MD simulation for cheap O(N) cost
- Typical training procedure is a workflow:
  - DFT data for small configs: energy, forces, virial = **QM truth**
  - Each atom + neighborhood $\Rightarrow$ geometric **descriptors**
  - Descriptors used in **big matrix** eq or as input to **NN**
  - Solve big matrix (e.g. least squares) or iteratively train NN
  - Optimize over additional **hyper parameters** for best fit
- **fitSNAP.py** $\Rightarrow$ `https://github.com/FitSNAP/FitSNAP`
  - used to train **SNAP** (spectral neighbor analysis potential)
  - calls LAMMPS as library to compute bispectrum descriptors
  - parallelism: **multiple LAMMPS instances**, large LSQ solve
  - output = vector of dozens or a few 100 bispectrum coeffs

Applications of SNAP via fitSNAP.py

High Entropy Alloys

- Complex chemical composition space limits accuracy of MD predictions
- Massive training spaces needed to generate a ML-IAP
- Predictive modeling of AM processing and performance

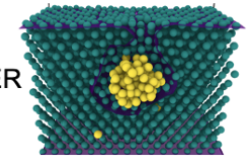


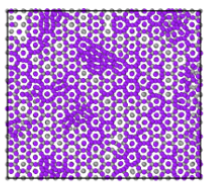


Plasma-Facing Materials

Office of Science

- Multiscale effort to predict lifetime and failure modes of ITER
- SNAP potentials generated for W/Be/N/H/He
- Bubble bursting at tungsten surface
- Beryllium deposition on tungsten leading to bimetallic ordering

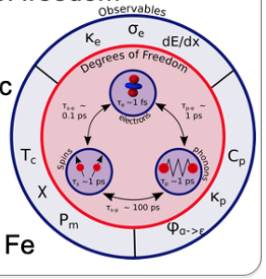




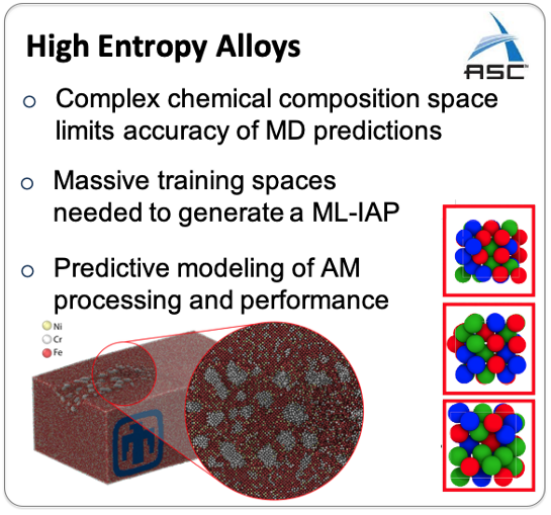
OAK RIDGE National Laboratory

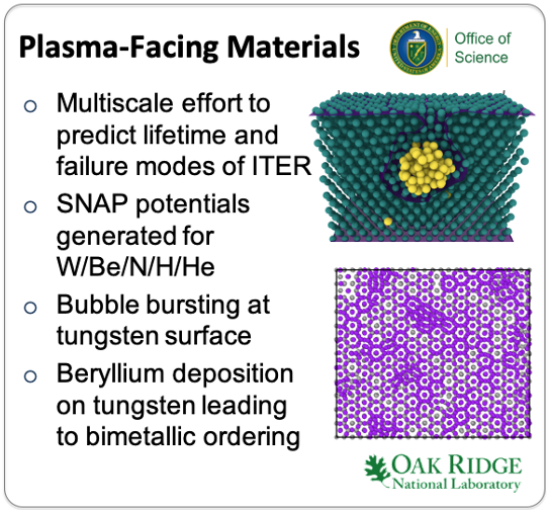
Coupled-Physics Modeling

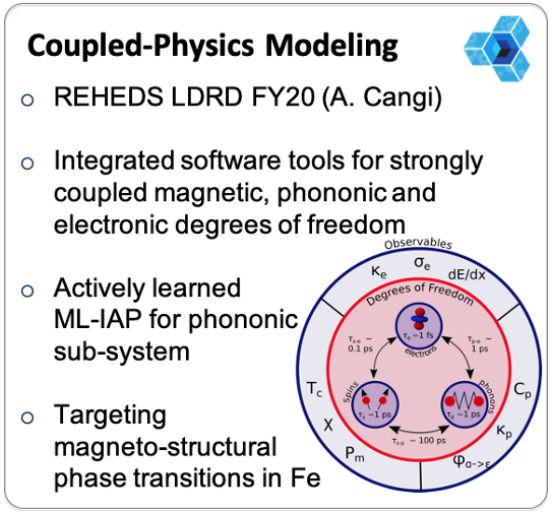
- REHEDS LDRD FY20 (A. Cangi)
- Integrated software tools for strongly coupled magnetic, phononic and electronic degrees of freedom
- Actively learned ML-IAP for phononic sub-system
- Targeting magneto-structural phase transitions in Fe



Applications of SNAP via fitSNAP.py

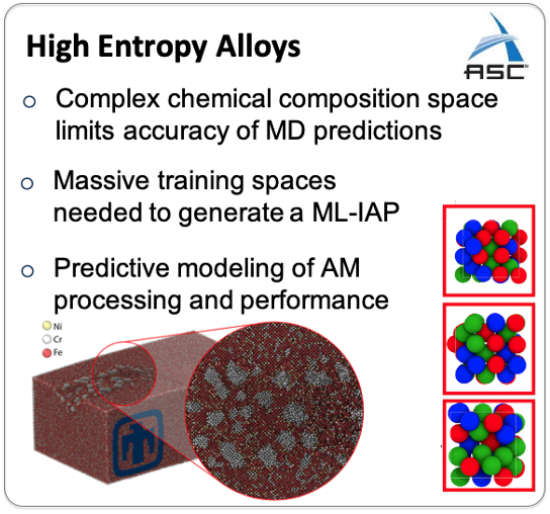


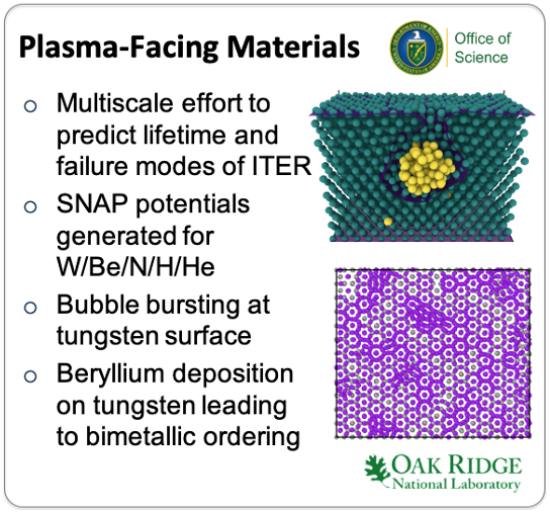


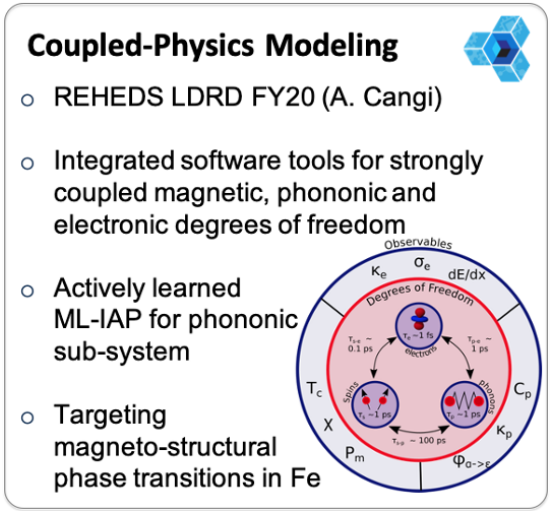


- Dislocation motion (Ta)
- Radiation damage (InP)
- Shock phase transitions (actinides)
- Super-ionic condutors (LiN)
- Vitrification (GeSe)
- Lithium-ion batteries (LiMoS)
- Thermoelectric materials (SiGeSnPb)

Applications of SNAP via fitSNAP.py







- Dislocation motion (Ta)
- Radiation damage (InP)
- Shock phase transitions (actinides)
- Super-ionic condutors (LiN)
- Vitrification (GeSe)
- Lithium-ion batteries (LiMoS)
- Thermoelectric materials (SiGeSnPb)

Can train a SNAP potential for a **specific material or application**

New ML potential framework in LAMMPS

Work with Nick Lubbers (LANL)

- All MLIAPs perform 3 steps in MD context:
  1. compute descriptors from atoms
  2. compute model gradients with respect to descriptor gradients
  3. compute forces from model gradients
- Step 2 is same for all ML potentials (pair style provides it)
- Users can add new Descriptors and Models (steps 1 & 3)

New ML potential framework in LAMMPS

Work with Nick Lubbers (LANL)

- All MLIAPs perform 3 steps in MD context:
  1. compute descriptors from atoms
  2. compute model gradients with respect to descriptor gradients
  3. compute forces from model gradients
- Step 2 is same for all ML potentials (pair style provides it)
- Users can add new Descriptors and Models (steps 1 & 3)

Descriptors:

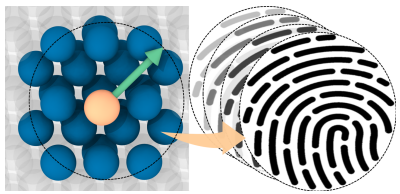
2-body, 3-body, graph methods

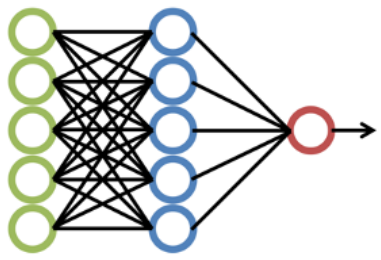
moment tensors, SOAP, bispectrum, ACE

Energy Models:

linear & kernel-ridge regression, Gaussian process

non-linear opt, NNs & hierarchical NNs





New ML potential framework in LAMMPS

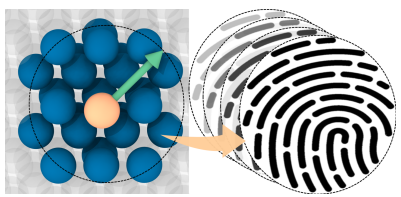
Work with Nick Lubbers (LANL)

- All MLIAPs perform 3 steps in MD context:
  1. compute descriptors from atoms
  2. compute model gradients with respect to descriptor gradients
  3. compute forces from model gradients
- Step 2 is same for all ML potentials (pair style provides it)
- Users can add new Descriptors and Models (steps 1 & 3)

Descriptors:

2-body, 3-body, graph methods

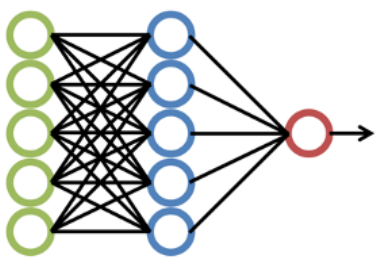
moment tensors, SOAP, bispectrum, ACE



Energy Models:

linear & kernel-ridge regression, Gaussian process

non-linear opt, NNs & hierarchical NNs



New pair style MLIAP allows:

- **mix-and-match** any Descriptor with any Model
- call back to Python, link with **PyTorch** for GPUs

Thanks for your attention

Thanks to Tom Swinburne (CINaM) & Noam Bernstein (NRL)
for lib examples, and to Ryan Elliott (U Minn) for OpenKIM work

LAMMPS: https://lammps.sandia.gov

MolSSI MDI: https://molssi-mdi.github.io/MDI_Library

OpenKIM: https://openkim.org

Email questions to:

athomp at sandia.gov (machine learning)
mitwood at sandia.gov (fitSNAP or SNAP applications)
sjplimp at sandia.gov (anything else)