Driving LAMMPS with Python

Dr. Richard Berger

LAMMPS Workshop, Temple University

August 18th, 2016

TEMPLE UNIVERSITY

Outline

Outline

Step 1: Building LAMMPS as a shared library

```
cd $LAMMPS_DIR/src

# generate custom Makefile
python2 Make.py -m mpi -png -a file

# add PACKAGES if necessary
make yes-MOLECULE

# compile shared library using Makefile
make mode=shlib auto
```

Step 2: Installing the LAMMPS Python package

```
cd $LAMMPS_DIR/python
python install.py
```

Warning!

Recompiling the shared library requires reinstaling the Python package.

Alternative: Installation in a Python virtualenv

```
# create virtualenv named 'testing'
virtualenv testing

# activate 'testing' environment
source testing/bin/activate

# install LAMMPS package in virtualenv
(testing) cd $LAMMPS_DIR/python
(testing) python install.py

# install other useful packages
(testing) pip install matplotlib jupyter mpi4py

...

# return to original shell
(testing) deactivate
```

TEMPLE UNIVERSITY

Benefits of using a virtualenv

- Isolate your system Python installation from your development installation
- Installation can happen in your user directory without root access (useful for HPC clusters)
- Installing packages through pip allows you to get newer versions of packages than e.g. through apt-get or yum package managers
- You can even install specific old versions of a package if necessary

Prerequisite (e.g., on Ubuntu):

```
apt-get install python-virtualenv
```

Python Interfaces

lammps.lammps

- uses C-Types
- direct memory access to native C++ data
- provides functions to send and receive data to LAMMPS
- requires knowledge of how LAMMPS works

lammps.PyLammps

- higher-level abstraction built on top of original C-Types interface
- manipulation of Python objects
- communication with LAMMPS is hidden from API user
- shorter, more concise Python code

Outline

PyLammps

Motivation

- Create a simpler, Python-like interface to LAMMPS
- API should be discoverable (no knowlege of the C++ code necessary)
- IPython notebook integration

Usage

```
from lammps import PyLammps
L = PyLammps()
```

Commands

LAMMPS Input Script

```
region box block 0 10 0 5 -0.5 0.5
```

Original Python Interface

```
L.command("region box block 0 10 0 5 -0.5 0.5")
```

PyLammps

```
L.region("box block", 0, 10, 0, 5, -0.5, 0.5)
```

Commands - Easier parametrization

Original Python Interface

```
L.command( \
    "region box block %f %f %f %f %f %f" % \
    (xlo, xh ylo, yhi, zlo, zhi) \
)
```

PyLammps

```
L.region("box block", xlo, xhi, ylo, yhi, zlo, zhi)
```

System state

`L.system`

Dictionary describing the system state such as bounding box or number of atoms

`L.system.xlo, L.system.xhi`

bounding box limits in x-direction

`L.system.natoms`

number of atoms in the system

System state

`L.communication`

communication configuration, such as number of threads or processors

`L.fixes`

list of fixes

`L.computes`

list of computes

`L.dumps`

list of dumps

`L.groups`

list of groups

Accessing variables

Defining a LAMMPS variable

```
L.variable("a index 2")
```

List of variables

```
L.variables
```

Accessing the variable value

```
a = L.variables['a']
a.value
```

Evaluating expressions

Access internal computed values

```
result = L.eval("ke") # kinetic energy
result = L.eval("pe") # potential energy
```

Evaluate arbitrary LAMMPS expression

```
result = L.eval("v_t/2.0")
```

Accessing atom data

List of atoms

```
L.atoms
```

Atom properties

```
L.atoms[0].id
L.atoms[0].type
L.atoms[0].position
L.atoms[0].velocity
L.atoms[0].force
```

Update atom position

```
L.atoms[0].position = (1.0, 0.0, 1.0)
```

Outline

Outline

Using LAMMPS with mpi4py

mpi4py allows you to use MPI to run Python in parallel

Installation

```
pip install mpi4py
```

Using mpi4py

hello.py

```
from mpi4py import MPI

me = MPI.COMM_WORLD.rank
nprocs = MPI.COMM_WORLD.size

print("Hello from process %d out of %d!" % (me,nprocs))

MPI.Finalize()
```

Running Python with MPI:

```
# run on 4 processors
mpirun -np 4 python hello.py
```

Using LAMMPS with mpi4py

melt.py

```
from mpi4py import MPI
from lammps import lammps

L = lammps()
L.file("in.melt")

MPI.Finalize()
```

Run LAMMPS in parallel using mpi4py

```
mpirun -np 4 python melt.py
```

PyLammps and mpi4py (Experimental)

Warning!

Any command must be executed by all MPI processes. However, evaluations and querying the system state is only available on rank 0.

```
from mpi4py import MPI
from lammps import PyLammps

L = PyLammps()
L.file("in.melt")

if MPI.COMM_WORLD.rank == 0:
    print("Potential energy: ", L.eval("pe"))

MPI.Finalize()
```