SimPhoNy-LAMMPS Documentation

Release 0.1.3.dev0

SimPhoNy, EU FP7 Project (Nr. 604005) www.simphony-project.eu

May 12, 2015

Contents

CHAPTER 1

simphony-lammps

The LAMMPS engine-wrapper for the SimPhoNy framework (www.simphony-project.eu).

1.1 Repository

simphony-lammps is hosted on github: https://github.com/simphony/simphony-lammps-md

1.2 Requirements

- enum34 >= 1.0.4
- pyyaml >= 3.11
- simphony-common >= 0.1.1

1.2.1 Optional requirements

To support the documentation built you need the following packages:

- sphinx >= 1.2.3

To run the unit tests you additionally need the following packages:

- numpy >= 1.4.1

1.3 Installation

The package requires python 2.7.x. Installation is based on setuptools:

```
# build and install
python setup.py install
```

or:

```
# build for in-place development
python setup.py develop
```

1.3.1 LAMMPS installation

This engine-wrapper uses LAMMPS Molecular Dynamics Simulator. A recent stable version (9 Dec 2014, tagged r12824) of LAMMPS is supported and has been tested. See `install_lammps.sh` for an example installation instructions. For general LAMMPS install information, see http://lammps.sandia.gov/index.html

LAMMPS installation varies depending on which interface is being used. See the manual for more details.

1.4 Usage

After installation, the user should be able to import the `lammps` engine plugin module by:

```
from simphony.engine import lammps
  engine = lammps.LammpsWrapper()
```

1.5 Testing

To run the full test-suite run:

```
python -m unittest discover
```

1.6 Documentation

To build the documentation in the doc/build directory run:

```
python setup.py build_sphinx
```

Note:

- One can use the –help option with a setup.py command to see all available options.
- The documentation will be saved in the `./build` directory.

1.7 Directory structure

- simlammps – hold the lammps-md wrapper implementation
- examples – holds different examples
- doc – Documentation related files
  - source – Sphinx rst source files
  - build – Documentation build directory, if documentation has been generated using the `make` script in the `doc` directory.

CHAPTER 2

User Manual

2.1 LAMMPS engine for SimPhoNy

The SimPhoNy engine for LAMMPS is available in the SimPhoNy through the engine plugin named `lammps`

After installation, the user should be able to import the `lammps` engine plugin module:

```
from simphony.engine import lammps
  engine = lammps.LammpsWrapper()
```

2.1.1 Interface to LAMMPS

The SimPhoNy LAMMPS engine (see `LammpsWrapper`) can be configured to interface with LAMMPS in two separate ways:

- FILE-IO - input and output files are used to configure and run LAMMPS engine
- INTERNAL - the LAMMPS library interface is used to run LAMMPS and access the internal state.

Despite performance differences, it should not matter whether the user is using the File-IO or the INTERNAL interface as the API is the same.

```
# wrapper defaults to FILE-IO
from simphony.engine import lammps
    engine = lammps.LammpsWrapper()

# or use INTERNAL wrapper
from simphony.engine import lammps
    engine = lammps.LammpsWrapper(use_internal_interface=true)
```

2.1.2 Installation of LAMMPS

This engine-wrapper uses LAMMPS Molecular Dynamics Simulator. A recent stable version (9 Dec 2014, tagged r12824) of LAMMPS is supported and has been tested. See `install_lammps.sh` for an example installation instructions. For general LAMMPS install information, see http://lammps.sandia.gov/index.html

The installation of LAMMPS differs depending on what interface is used:

- FILE-IO: There needs to be an executable called “lammps” that can be found in the PATH.

- INTERNAL: The LAMMPS-provided Python wrapper to LAMMPS needs to be installed.Instructions on building LAMMPS as a shared library and installing the Python wrapper can be found on LAMMPS website (http://lammps.sandia.gov/doc/Section_python.html#py_3)

2.1.3 Limitations of the INTERNAL interface

The following are known limitations when using the INTERNAL interface to LAMMPS:

- Deletion of particles (i.e. `delete_particle`) is not yet supported
- Adding particles to the LAMMPSParticleContainer can fail when coordinates of the added particles is outside of the simulation box configured by the user.
- Currently an upper limit of particle types (CUBA.MATERIAL_TYPE) is set due to the fact that LAMMPS only allows the number of types be configured at start (and not changed later)
- No notification is provided to the user when an internal error occurs in the LAMMPS shared library as the library calls *exit(1)* and the process immediately exists (without an exception or writing to standard output/error).

2.2 Molecular Dynamics Engine

The SimPhoNy engine for LAMMPS currently supports a simple bulk atomistic molecular dynamics simulations with NVE thermodynamic ensemble.

2.2.1 Configuration of the engine (CM, SP, BC)

The user can configure the engine by setting specific values using CUBA keywords. The CUBA keywords supported for each model component are provided below. Additionally, there are non-CUBA keywords used for configuration which are also currently being used for configuration.

CM

- `CUBA.NUMBER_OF_TIME_STEPS`
- `CUBA.TIME_STEP`
- **`CUBAExtension.THERMODYNAMIC_ENSEMBLE`**
  - **supported values:**
    - "NVE" - use constant NVE integration

SP

- `CUBAExtension.PAIR_POTENTIALS`

BC

- `CUBAExtension.BOX_FACES`

Example

Here is an example configuration:

```
# CM
engine.CM[CUBA.NUMBER_OF_TIME_STEPS] = 10000
engine.CM[CUBA.TIME_STEP] = 0.003
engine.CM_extension[CUBAExtension.THERMODYNAMIC_ENSEMBLE] = "NVE"

# SP
pair_potential = ("lj:\n"
                  "  global_cutoff: 1.12246\n"
                  "  parameters:\n"
                  "  - pair: [1, 1]\n"
                  "    epsilon: 1.0\n"
                  "    sigma: 1.0\n"
                  "    cutoff: 1.2246\n"
                  "  - pair: [1, 2]\n"
                  "    epsilon: 1.0\n"
                  "    sigma: 1.0\n"
                  "    cutoff: 1.2246\n"
                  "  - pair: [1, 3]\n"
                  "    epsilon: 1.0\n"
                  "    sigma: 1.0\n"
                  "    cutoff: 1.2246\n"
                  "  - pair: [2, 2]\n"
                  "    epsilon: 1.0\n"
                  "    sigma: 1.0\n"
                  "    cutoff: 1.2246\n"
                  "  - pair: [2, 3]\n"
                  "    epsilon: 1.0\n"
                  "    sigma: 1.0\n"
                  "    cutoff: 1.2246\n"
                  "  - pair: [3, 3]\n"
                  "    epsilon: 1.0\n"
                  "    sigma: 1.0\n"
                  "    cutoff: 1.0001\n")
engine.SP_extension[CUBAExtension.PAIR_POTENTIALS] = pair_potential

# BC
engine.BC_extension[CUBAExtension.BOX_FACES] = (
    "periodic", "periodic", "periodic")
```

2.2.2 Configuring the engine's state

The MD engine only supports Particles.

Particles

The Particles container's `data` should have `CUBA.MASS` and `CUBA.MATERIAL_TYPE` defined.

Individual particles

The individual particles should have `CUBA.VELOCITY` defined.

Example

Here is an example configuration:

```
pc = ParticleContainer
data = DataContainer()
data[CUBA.MASS] = mass
data[CUBA.MATERIAL_TYPE] = material_type

pc.data = data

pc_w = engine.add_particles(pc)

vectors = [(25.0, 0.0, 0.0),
           (0.0, 22.0, 0.0),
           (0.0, 0.0, 1.0)]

pc_w.data_extension = {CUBAExtension.BOX_VECTORS: vectors,
                       CUBAExtension.BOX_ORIGIN: (0.0, 0.0, 0.0)}

random.seed(42)
for _ in range(10):
    coord = (random.uniform(0.0, 25.0),
             random.uniform(0.0, 22.0),
             0.0)
    p = Particle(coordinates=coord)
    p.data[CUBA.VELOCITY] = (0.0, 0.0, 0.0)
    pc_w.add_particle(p)
```

2.2.3 Additional notes

- Units : all quantities are unitless.

CHAPTER 3

API Reference

3.1 Engine

The LAMMPS engine for SimPhoNy (`LammpsWrapper` in module `simlammps.lammps_wrapper`) implements the SimPhoNy's abstract interface definition for engines (`ABCModelingEngine`). This LAMMPS engine is available through the engine plug-in to the SimPhoNy library.

Engine

`LammpsWrapper`([use_internal_interface])	Wrapper to LAMMPS-md

Implementation

LAMMPS SimPhoNy Wrapper

This module provides a wrapper for LAMMPS-md

class `simlammps.lammps_wrapper.`**`LammpsWrapper`**(*use_internal_interface=False*)

Bases: `simphony.cuds.abc_modeling_engine.ABCModelingEngine`

Wrapper to LAMMPS-md

`add_lattice`(*lattice*)

`add_mesh`(*mesh*)

`add_particles`(*particles*)

Add particles.

Parameters

- **particles** (*ABCParticles*) – particles to be added.
- **Returns** –
- **———-** –
- **ABCParticles** – The particles newly added to Lammps. See get_particles for more information.

`delete_lattice`(*name*)

`delete_mesh`(*name*)

delete_particles(*name*)
Delete particles

Parameters **name** (*str*) – name of particles to delete

get_lattice(*name*)

get_mesh(*name*)

get_particles(*name*)
Get particles

The returned particle container can be used to query and change the related data inside LAMMPS.

Parameters **name** (*str*) – name of particle container to return

iter_lattices(*names=None*)

iter_meshes(*names=None*)

iter_particles(*names=None*)
Returns an iterator over a subset or all of the particle containers. The iterator iterator yields (name, particlecontainer) tuples for each particle container.

Parameters **names** (*list of str*) – names of specific particle containers to be iterated over. If names is not given, then all particle containers will be iterated over.

run()
Run lammps-engine based on configuration and data

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

S

A

D

G

I

L

R

S