Part

About LAMMPS and this manual

LAMMPS stands for **L**arge-scale **A**tomic/**M**olecular **M**assively **P**arallel **S**imulator.

LAMMPS is a classical molecular dynamics simulation code focusing on materials modeling. It was designed to run efficiently on parallel computers and to be easy to extend and modify. Originally developed at Sandia National Laboratories, a US Department of Energy facility, LAMMPS now includes contributions from many research groups and individuals from many institutions. Most of the funding for LAMMPS has come from the US Department of Energy (DOE). LAMMPS is open-source software distributed under the terms of the GNU Public License Version 2 (GPLv2).

The LAMMPS website has a variety of information about the code. It includes links to an online version of this manual, an online forum where users can post questions and discuss LAMMPS, and a GitHub site where all LAMMPS development is coordinated.

The content for this manual is part of the LAMMPS distribution in its doc directory.

- The version of the manual on the LAMMPS website corresponds to the latest LAMMPS feature release. It is available at: https://docs.lammps.org/.
- A version of the manual corresponding to the latest LAMMPS stable release (state of the *stable* branch on GitHub) is available online at: https://docs.lammps.org/stable/
- A version of the manual with the features most recently added to LAMMPS (state of the *develop* branch on GitHub) is available at: https://docs.lammps.org/latest/

If needed, you can build a copy on your local machine of the manual (HTML pages or PDF file) for the version of LAMMPS you have downloaded. Follow the steps on the *Build the LAMMPS documentation* page.

The manual is organized into three parts:

1. The *User Guide* with information about how to obtain, configure, compile, install, and use LAMMPS,
2. the *Programmer Guide* with information about how to use the LAMMPS library interface from different programming languages, how to modify and extend LAMMPS, the program design, internal programming interfaces, and code design conventions,
3. the *Command Reference* with detailed descriptions of all input script commands available in LAMMPS.

Part I

User Guide

CHAPTER

ONE

INTRODUCTION

These pages provide a brief introduction to LAMMPS.

1.1 Overview of LAMMPS

LAMMPS is a classical molecular dynamics (MD) code that models ensembles of particles in a liquid, solid, or gaseous state. It can model atomic, polymeric, biological, solid-state (metals, ceramics, oxides), granular, coarse-grained, or macroscopic systems using a variety of interatomic potentials (force fields) and boundary conditions. It can model 2d or 3d systems with sizes ranging from only a few particles up to billions.

LAMMPS can be built and run on single laptop or desktop machines, but is designed for parallel computers. It will run in serial and on any parallel machine that supports the MPI message-passing library. This includes shared-memory multicore, multi-CPU servers and distributed-memory clusters and supercomputers. Parts of LAMMPS also support OpenMP multi-threading, vectorization, and GPU acceleration.

LAMMPS is written in C++ and requires a compiler that is at least compatible with the C++-11 standard. Earlier versions were written in F77, F90, and C++-98. See the History page of the website for details. All versions can be downloaded as source code from the LAMMPS website.

LAMMPS is designed to be easy to modify or extend with new capabilities, such as new force fields, atom types, boundary conditions, or diagnostics. See the *Modifying & extending LAMMPS* section of for more details.

In the most general sense, LAMMPS integrates Newton's equations of motion for a collection of interacting particles. A single particle can be an atom or molecule or electron, a coarse-grained cluster of atoms, or a mesoscopic or macroscopic clump of material. The interaction models that LAMMPS includes are mostly short-ranged in nature; some long-range models are included as well.

LAMMPS uses neighbor lists to keep track of nearby particles. The lists are optimized for systems with particles that are repulsive at short distances, so that the local density of particles never becomes too large. This is in contrast to methods used for modeling plasma or gravitational bodies (like galaxy formation).

On parallel machines, LAMMPS uses spatial-decomposition techniques with MPI parallelization to partition the simulation domain into subdomains of equal computational cost, one of which is assigned to each processor. Processors communicate and store "ghost" atom information for atoms that border their subdomain. Multi-threading parallelization and GPU acceleration with particle-decomposition can be used in addition.

1.2 What does a LAMMPS version mean

The LAMMPS "version" is the date when it was released, such as 1 May 2014. LAMMPS is updated continuously, and we aim to keep it working correctly and reliably at all times. Also, several variants of static code analysis are run regularly to maintain or improve the overall code quality, consistency, and compliance with programming standards, best practices and style conventions. You can follow its development in a public git repository on GitHub.

Each version of LAMMPS contains all the documented *features* up to and including its version date. For recently added features, we add markers to the documentation at which specific LAMMPS version a feature or keyword was added or significantly changed.

1.2.1 Identifying the Version

The version date is printed to the screen and log file every time you run LAMMPS. There also is an indication, if a LAMMPS binary was compiled from version with modifications **after** a release. It is also visible in the file src/version.h and in the LAMMPS directory name created when you unpack a downloaded tarball. And it is on the first page of the *manual*.

- If you browse the HTML pages of the online version of the LAMMPS manual, they will by default describe the most current feature release version of LAMMPS. In the navigation bar on the bottom left, there is the option to view instead the documentation for the most recent *stable* version or the documentation corresponding to the state of the development branch.
- If you browse the HTML pages included in your downloaded tarball, they describe the version you have, which may be older than the online version.

1.2.2 LAMMPS releases, branches, and tags

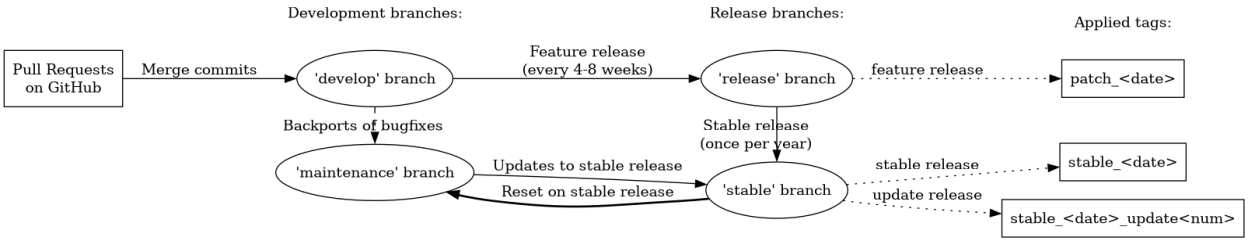

Fig. 1: Relations between releases, main branches, and tags in the LAMMPS git repository

Development

Modifications of the LAMMPS source code (like bug fixes, code refactoring, updates to existing features, or addition of new features) are organized into pull requests. Pull requests will be merged into the *develop* branch of the git repository after they pass automated testing and code review by the LAMMPS developers.

Feature Releases

When a sufficient number of new features and updates have accumulated *and* the LAMMPS version on the *develop* branch passes an extended set of automated tests, we release it as a *feature release*, which are currently made every 4 to 8 weeks. The *release* branch of the git repository is updated with every such *feature release* and a tag in the format patch_1May2014 is added. A summary of the most important changes of these releases for the current year are posted on this website page. More detailed release notes are available on GitHub.

Stable Releases

About once a year, we release a *stable release* version of LAMMPS. This is done after a "stabilization period" where we apply only bug fixes and small, non-intrusive changes to the *develop* branch but no new features. At the same time, the code is subjected to more detailed and thorough manual testing than the default automated testing. After such a *stable release*, both the *release* and the *stable* branches are updated and two tags are applied, a patch_1May2014 format and a stable_1May2014 format tag.

Stable Release Updates

Between *stable releases*, we collect bug fixes and updates back-ported from the *develop* branch in a branch called *maintenance*. From the *maintenance* branch we make occasional *stable update releases* and update the *stable* branch accordingly. The first update to the stable_1May2014 release would be tagged as stable_1May2014_update1. These updates contain no new features.

1.3 LAMMPS features

LAMMPS is a classical molecular dynamics (MD) code with these general classes of functionality:

1. *General features*
2. *Particle and model types*
3. *Interatomic potentials (force fields)*
4. *Atom creation*
5. *Ensembles, constraints, and boundary conditions*
6. *Integrators*
7. *Diagnostics*
8. *Output*
9. *Multi-replica models*
10. *Pre- and post-processing*
11. *Specialized features (beyond MD itself)*

1.3.1 General features

- runs on a single processor or in parallel
- distributed memory message-passing parallelism (MPI)
- shared memory multi-threading parallelism (OpenMP)
- spatial decomposition of simulation domain for MPI parallelism
- particle decomposition inside spatial decomposition for OpenMP and GPU parallelism
- GPLv2 licensed open-source distribution
- highly portable C++-11 (optional packages may require C++17)
- modular code with most functionality in optional packages
- only depends on MPI library for basic parallel functionality, MPI stub for serial compilation
- other libraries are optional and only required for specific packages
- GPU (CUDA, OpenCL, HIP, SYCL), Intel Xeon Phi, and OpenMP support for many code features
- easy to extend with new features and functionality
- runs from an input script
- syntax for defining and using variables and formulas
- syntax for looping over runs and breaking out of loops
- run one or multiple simulations simultaneously (in parallel) from one script
- build as library, invoke LAMMPS through library interface (from C, C++, Fortran) or provided Python wrapper or SWIG based wrappers
- couple with other codes: LAMMPS calls other code, other code calls LAMMPS, umbrella code calls both, MDI coupling interface
- call out to Python for computing forces, time integration, or other tasks
- plugin interface for loading external features at runtime
- large integrated collection of tests

1.3.2 Particle and model types

(See *atom style* command)

- atoms
- coarse-grained particles (e.g. bead-spring polymers)
- united-atom polymers or organic molecules
- all-atom polymers, organic molecules, proteins, DNA
- metals
- metal oxides
- granular materials
- coarse-grained mesoscale models
- finite-size spherical and ellipsoidal particles

- finite-size line segment (2d) and triangle (3d) particles
- finite-size rounded polygons (2d) and polyhedra (3d) particles
- point dipole particles
- particles with magnetic spin
- rigid collections of n particles
- hybrid combinations of these

1.3.3 Interatomic potentials (force fields)

(See *pair style*, *bond style*, *angle style*, *dihedral style*, *improper style*, *kspace style* commands)

- pairwise potentials: Lennard-Jones, Buckingham, Morse, Born-Mayer-Huggins, Yukawa, soft, Class II (COMPASS), hydrogen bond, harmonic, gaussian, tabulated, scripted
- charged pairwise potentials: Coulombic, point-dipole
- many-body potentials: EAM, Finnis/Sinclair, MEAM, MEAM+SW, EIM, EDIP, ADP, Stillinger-Weber, Tersoff, REBO, AIREBO, ReaxFF, COMB, Streitz-Mintmire, 3-body polymorphic, BOP, Vashishta
- machine learning potentials: ACE, AGNI, GAP, Behler-Parrinello (N2P2), POD, RANN, SNAP
- interfaces to ML potentials distributed by external groups: ANI, ChIMES, DeepPot, HIPNN, MTP
- long-range interactions for charge, point-dipoles, and LJ dispersion: Ewald, Wolf, PPPM (similar to particle-mesh Ewald), MSM, ScaFaCoS
- polarization models: *QEq*, *core/shell model*, *Drude dipole model*
- charge equilibration (QEq via dynamic, point, shielded, Slater methods)
- coarse-grained potentials: DPD, GayBerne, REsquared, colloidal, DLVO, oxDNA / oxRNA, SPICA
- mesoscopic potentials: granular, Peridynamics, SPH, mesoscopic tubular potential (MESONT)
- semi-empirical potentials: multi-ion generalized pseudopotential theory (MGPT), second moment tight binding + QEq (SMTB-Q)
- electron force field (eFF, AWPMD)
- bond potentials: harmonic, FENE, Morse, nonlinear, Class II (COMPASS), quartic (breakable), tabulated, scripted
- angle potentials: harmonic, CHARMM, cosine, cosine/squared, cosine/periodic, Class II (COMPASS), tabulated, scripted
- dihedral potentials: harmonic, CHARMM, multi-harmonic, helix, Class II (COMPASS), OPLS, tabulated, scripted
- improper potentials: harmonic, cvff, umbrella, Class II (COMPASS), tabulated
- polymer potentials: all-atom, united-atom, bead-spring, breakable
- water potentials: TIP3P, TIP4P, SPC, SPC/E and variants
- interlayer potentials for graphene and analogues, hetero-junctions
- metal-organic framework potentials (QuickFF, MO-FF)
- implicit solvent potentials: hydrodynamic lubrication, Debye
- force-field compatibility with CHARMM, AMBER, DREIDING, OPLS, GROMACS, Class II (COMPASS), UFF, ClayFF, DREIDING, AMOEBA, INTERFACE

- access to the OpenKIM Repository of potentials via the *kim command*
- hybrid potentials: multiple pair, bond, angle, dihedral, improper potentials can be used in one simulation
- overlaid potentials: superposition of multiple pair potentials (including many-body) with optional scale factor

1.3.4 Atom creation

(See *read_data*, *lattice*, *create_atoms*, *delete_atoms*, *displace_atoms*, *replicate* commands)

- read in atom coordinates from files
- create atoms on one or more lattices (e.g. grain boundaries)
- delete geometric or logical groups of atoms (e.g. voids)
- replicate existing atoms multiple times
- displace atoms

1.3.5 Ensembles, constraints, and boundary conditions

(See *fix* command)

- 2d or 3d systems
- orthogonal or non-orthogonal (triclinic symmetry) simulation domains
- constant NVE, NVT, NPT, NPH, Parrinello/Rahman integrators
- thermostatting options for groups and geometric regions of atoms
- pressure control via Nose/Hoover or Berendsen barostatting in 1 to 3 dimensions
- simulation box deformation (tensile and shear)
- harmonic (umbrella) constraint forces
- rigid body constraints
- SHAKE / RATTLE bond and angle constraints
- motion constraints to manifold surfaces
- Monte Carlo bond breaking, formation, swapping, template based reaction modeling
- atom/molecule insertion and deletion
- walls of various kinds, static and moving
- non-equilibrium molecular dynamics (NEMD)
- variety of additional boundary conditions and constraints

1.3.6 Integrators

(See *run*, *run_style*, *minimize* commands)

- velocity-Verlet integrator
- Brownian dynamics
- rigid body integration
- energy minimization via conjugate gradient, steepest descent relaxation, or damped dynamics (FIRE, Quickmin)
- rRESPA hierarchical timestepping
- fixed or adaptive time step
- rerun command for post-processing of dump files

1.3.7 Diagnostics

- see various flavors of the *fix* and *compute* commands
- introspection command for system, simulation, and compile time settings and configurations

1.3.8 Output

(*dump*, *restart* commands)

- log file of thermodynamic info
- text dump files of atom coordinates, velocities, other per-atom quantities
- dump output on fixed and variable intervals, based timestep or simulated time
- binary restart files
- parallel I/O of dump and restart files
- per-atom quantities (energy, stress, centro-symmetry parameter, CNA, etc.)
- user-defined system-wide (log file) or per-atom (dump file) calculations
- custom partitioning (chunks) for binning, and static or dynamic grouping of atoms for analysis
- spatial, time, and per-chunk averaging of per-atom quantities
- time averaging and histogramming of system-wide quantities
- atom snapshots in native, XYZ, XTC, DCD, CFG, NetCDF, HDF5, ADIOS2, YAML formats
- on-the-fly compression of output and decompression of read in files

1.3.9 Multi-replica models

- *nudged elastic band*
- *hyperdynamics*
- *parallel replica dynamics*
- *temperature accelerated dynamics*
- *parallel tempering*
- path-integral MD: *first variant*, *second variant*
- multi-walker collective variables with *Colvars* and *Plumed*

1.3.10 Pre- and post-processing

- A handful of pre- and post-processing tools are packaged with LAMMPS, some of which can convert input and output files to/from formats used by other codes; see the *Tools* page.
- Our group has also written and released a separate toolkit called Pizza.py which provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in Python and is available for download from the Pizza.py WWW site.

1.3.11 Specialized features

LAMMPS can be built with optional packages which implement a variety of additional capabilities. See the *Optional Packages* page for details.

These are LAMMPS capabilities which you may not think of as typical classical MD options:

- *static* and *dynamic load-balancing*, optional with recursive bisectioning decomposition
- *generalized aspherical particles*
- *stochastic rotation dynamics (SRD)*
- *real-time visualization and interactive MD*, *built-in renderer for images and movies*
- calculate *virtual diffraction patterns*
- calculate *finite temperature phonon dispersion* and the *dynamical matrix of minimized structures*
- *atom-to-continuum coupling* with finite elements
- coupled rigid body integration via the *POEMS* library
- *QM/MM coupling*
- Monte Carlo via *GCMC* and *tfMC* and *atom swapping*
- *path-integral molecular dynamics (PIMD)* and *this as well*
- *Direct Simulation Monte Carlo* for low-density fluids
- *Peridynamics modeling*
- *Lattice Boltzmann fluid*
- *targeted* and *steered* molecular dynamics
- *two-temperature electron model*

1.4 LAMMPS non-features

LAMMPS is designed to be a fast, parallel engine for molecular dynamics (MD) simulations. It provides only a modest amount of functionality for setting up simulations and analyzing their output.

Originally, LAMMPS was not conceived and designed for:

- being run through a GUI
- building molecular systems, or building molecular topologies
- assign force-field coefficients automagically
- perform sophisticated analysis of your MD simulation
- visualize your MD simulation interactively
- plot your output data

Over the years many of these limitations have been reduced or removed. In part through features added to LAMMPS and in part through external tools that either closely interface with LAMMPS or extend LAMMPS.

Here are suggestions on how to perform these tasks:

- **GUI:** LAMMPS can be built as a library and a Python module that wraps the library interface is provided. Thus, GUI interfaces can be written in Python or C/C++ that run LAMMPS and visualize or plot its output. Examples of this are provided in the python directory and described on the *Python* doc page.

  Since version 2 August 2023 *a LAMMPS-GUI tool* is included in LAMMPS. Also, there are several external wrappers or GUI front ends that are mentioned on the Pre-/post-processing tools page of the LAMMPS homepage.

- **Builder:** Several pre-processing tools are packaged with LAMMPS. Some of them convert input files in formats produced by other MD codes such as CHARMM, AMBER, or Insight into LAMMPS input formats. Some of them are simple programs that will build simple molecular systems, such as linear bead-spring polymer chains. The moltemplate program is a true molecular builder that will generate complex molecular models. See the *Tools* page for details on tools packaged with LAMMPS. The Pre-/post-processing tools page of the LAMMPS homepage describes a variety of third party tools for this task. Furthermore, some internal LAMMPS commands allow reconstructing, or selectively adding topology information, as well as provide the option to insert molecule templates instead of atoms for building bulk molecular systems.

- **Force-field assignment:** The conversion tools described in the previous bullet for CHARMM, AMBER, and Insight will also assign force field coefficients in the LAMMPS format, assuming you provide CHARMM, AMBER, or BIOVIA (formerly Accelrys) force field files. The tools ParmEd and InterMol are particularly powerful and flexible in converting force field and topology data between various MD simulation programs.

- **Simulation analysis:** If you want to perform analysis on-the-fly as your simulation runs, see the *compute* and *fix* doc pages, which list commands that can be used in a LAMMPS input script. Also see the *Modify* page for info on how to add your own analysis code or algorithms to LAMMPS. For post-processing, LAMMPS output such as *dump file snapshots* can be converted into formats used by other MD or post-processing codes. To some degree, that conversion can be done directly inside LAMMPS by interfacing to the VMD molfile plugins. The *rerun* command also allows post-processing of existing trajectories, and through being able to read a variety of file formats, this can also be used for analyzing trajectories from other MD codes. Some post-processing tools packaged with LAMMPS will do these conversions. Scripts provided in the tools/python directory can extract and massage data in dump files to make it easier to import into other programs. See the *Tools* page for details on these various options.

  The Pre-/post-processing page on the LAMMPS homepage lists some external packages for analysis of MD simulation data, including data produced by LAMMPS.

- **Visualization:** LAMMPS can produce NETPBM, JPG, or PNG format snapshot images on-the-fly via its *dump image* command and pass them to an external program, FFmpeg, to generate movies from them. The *LAMMPS-GUI tool* has an *Snapshot Image Viewer* which uses *dump image* and allows to modify the visualization settings interactively. It also has a *Slide Show* feature where images created by *dump image* are collected during a simulation and can be animated interactively or exported to a movie with FFmpeg.

  For high-quality, interactive visualization, there are many excellent and free tools available. See the Visualization Tools page of the LAMMPS website for visualization packages that can process LAMMPS output data.

- **Plotting:** See the next bullet about Pizza.py as well as the *Python* page for examples of plotting LAMMPS output. Scripts provided with the *python* tool in the tools directory will extract and process data in log and dump files to make it easier to analyze and plot. See the *Tools* doc page for more discussion of the various tools.

  The *LAMMPS-GUI tool* has an *Chart Viewer* where *thermodynamic data* computed by LAMMPS is collected during the simulation and plotted immediately.

- **Pizza.py:** Our group has also written a separate toolkit called Pizza.py which can do certain kinds of setup, analysis, plotting, and visualization (via OpenGL) for LAMMPS simulations. It thus provides some functionality for several of the above bullets. Pizza.py is written in Python and is available for download from this page.

1.5 LAMMPS portability and compatibility

The primary form of distributing LAMMPS is through highly portable source code. But also several ways of obtaining LAMMPS as *precompiled packages or through automated build mechanisms* exist. Most of LAMMPS is written in C++, some support tools are written in Fortran or Python or MATLAB.

1.5.1 Programming language standards

Most of the C++ code currently requires a compiler compatible with the C++11 standard, the KOKKOS package currently requires C++17. Most of the Python code is written to be compatible with Python 3.5 or later or Python 2.7. Some Python scripts *require* Python 3 and a few others still need to be ported from Python 2 to Python 3.

1.5.2 Build systems

LAMMPS can be compiled from source code using a (traditional) build system based on shell scripts, a few shell utilities (grep, sed, cat, tr) and the GNU make program. This requires running within a Bourne shell (/bin/sh). Alternatively, a build system with different back ends can be created using CMake. CMake must be at least version 3.16.

1.5.3 Operating systems

The primary development platform for LAMMPS is Linux. Thus, the chances for LAMMPS to compile without problems on Linux machines are the best. Also, compilation and correct execution on macOS and Windows (using Microsoft Visual C++) is checked automatically for largest part of the source code. Some (optional) features are not compatible with all operating systems, either through limitations of the corresponding LAMMPS source code or through source code or build system incompatibilities of required libraries.

Executables for Windows may be created natively using either Cygwin or Visual Studio or with a Linux to Windows MinGW cross-compiler.

Additionally, FreeBSD and Solaris have been tested successfully.