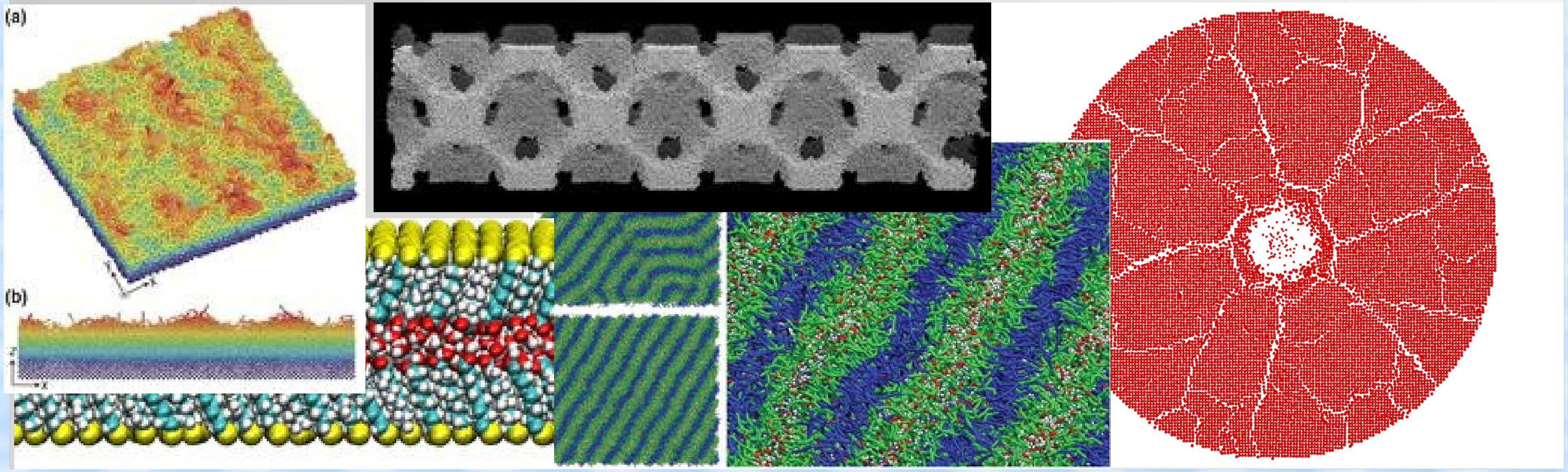
Collaborative Software Management: The LAMMPS Project

Dr. Axel Kohlmeyer

Research Professor, Department of Chemistry
Associate Director, ICMS
College of Science and Technology
Temple University, Philadelphia
axel.kohlmeyer@temple.edu

What LAMMPS Is

- Large-scale Atomic/Molecular Massively Parallel Simulator (each word is an attribute)
- Three-legged stool, supported by force fields and methods:
  one foot in biomolecules and polymers (soft materials)
  one foot in materials science (solids)
  one foot in mesoscale to continuum



LAMMPS is an Extensible Project

- ~3600 C/C++ files with about 1,100,000 lines of code in core executable, plus bundled libs
- Only about 300 files are essential, about 800 files are compiled by default, 2900 are optional
- Optional files are included through derived C++ classes, extra functionality in bundled libraries
- Three levels of “package support”:
  - Core packages (officially supported, included in source)
  - lammps-plugins packages (unsupported, compatible)
  - external packages (supported by individuals)

LAMMPS is a Collaborative Project

A few core developers and many contributors:

- Steve Plimpton (retired, formally Sandia National Lab (SNL))
- Axel Kohlmeyer (Temple University)
- Aidan Thompson, Stan Moore, Joel Clemmer (SNL)
- Trung Nguyen (Chicago University)
- Richard Berger (Los Alamos National Lab, formerly Temple U)

  - Roy Pollock (LLNL), Ewald and PPPM solvers
  - Mike Brown (ORNL/Intel), GPU package, INTEL package
  - Greg Wagner (Sandia), MEAM package for MEAM potential
  - Mike Parks (Sandia), PERI package for Peridynamics
  - Reese Jones (Sandia), ATC package for coupling to continuum
  - Christian Trott (Sandia), KOKKOS package
  - Metin Aktulga (Michigan State), REAXFF package
  - Georg Gunzenmuller (EMI), SPH, MACHDYN package
  - Ray Shan (Materials Research), COMB package, QEQ package

- In total over 300 people with significant contributions to LAMMPS

Why Use LAMMPS?

- Flexible choice of per particle attributes
- Large choice of potential functions
- Flexible handling of boundary conditions
- Large choice of ensembles and “manipulators”
- Efficient parallelization (MPI + OpenMP/GPU)
- On-the-fly analysis and powerful scripting
- Easy to add new features or modify code
- Library interface for coupling to other codes

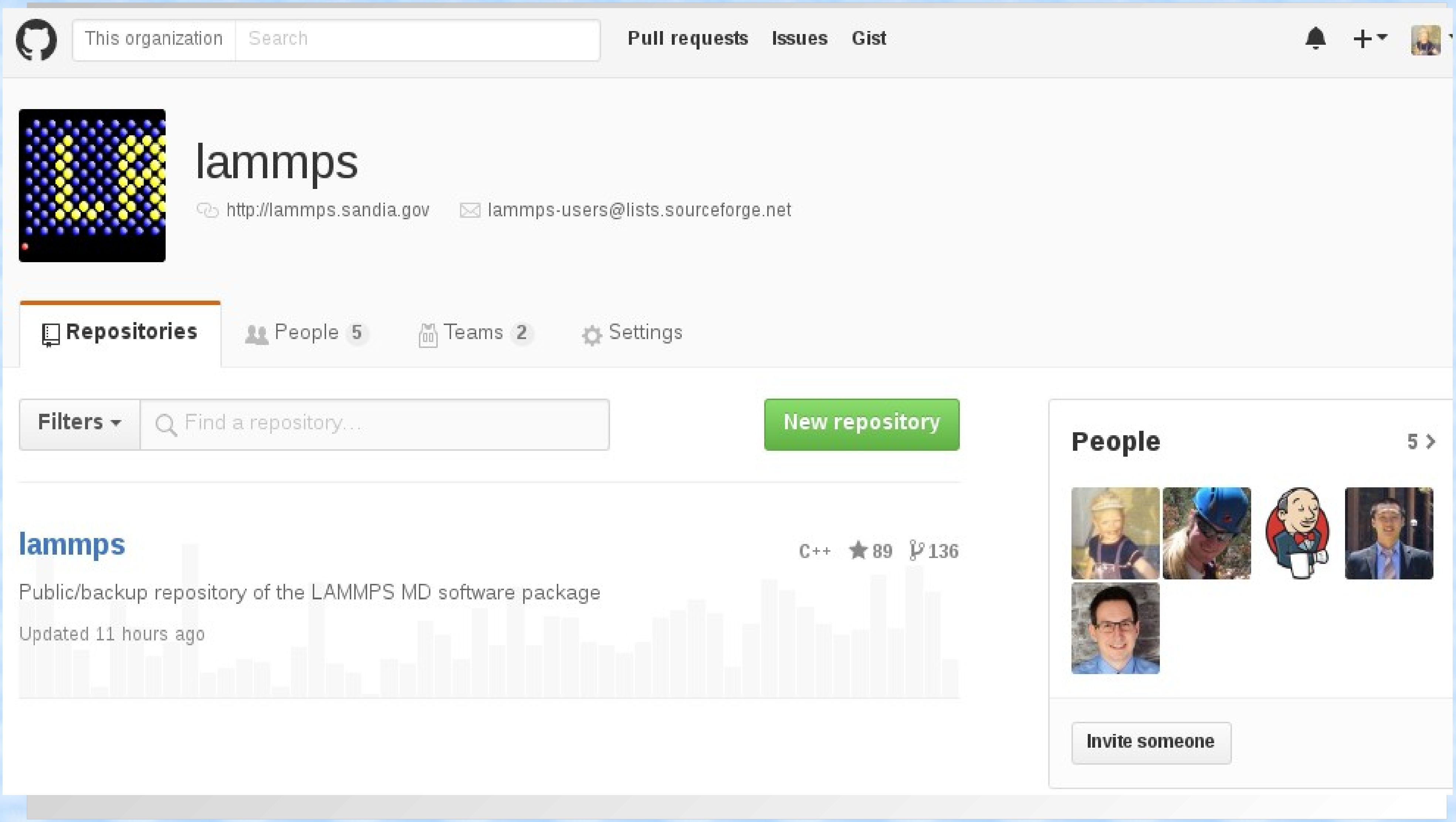
Development Infrastructure

- Public Git repository on GitHub
- 4 Branches: *develop* (development), *release* (feature releases), *stable* (stable releases), *maintenance* (updates/bugfixes for stable)
- All changes to LAMMPS *must* be submitted as pull request (even from maintainers!), pass automated testing, and developer review
- Forum on MatSci.org for discussing LAMMPS
- Communication on development also as comments to GitHub issues and pull requests

Development Cycle

- Continuous release procedure
- Version indicated by date of release
- Feature releases about every 2-3 months
- One stable release per year with additional testing and bugfix-only stabilization period, bugfixes backported to stable release for one year
- Continuous integration with Jenkins and GitHub Actions to run integration and unit test on all pull requests with multiple configurations
- Regression tests after merge to develop branch

https://github.com/lammps



lammps

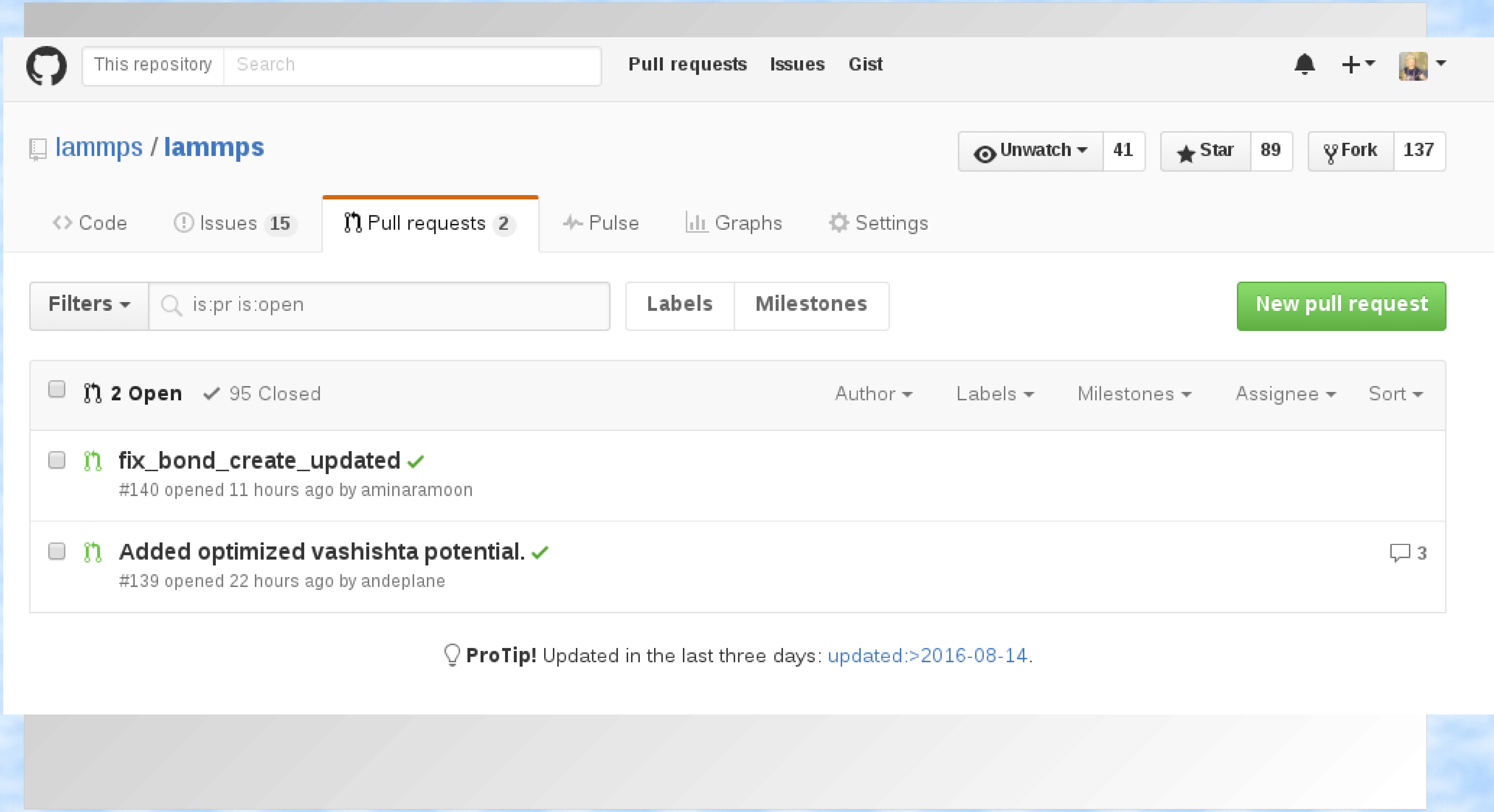
http://lammps.sandia.gov   lammps-users@lists.sourceforge.net

lammps   C++ ★ 89 136

Public/backup repository of the LAMMPS MD software package

Updated 11 hours ago

Contributing Code via Pull Requests



lammps / lammps

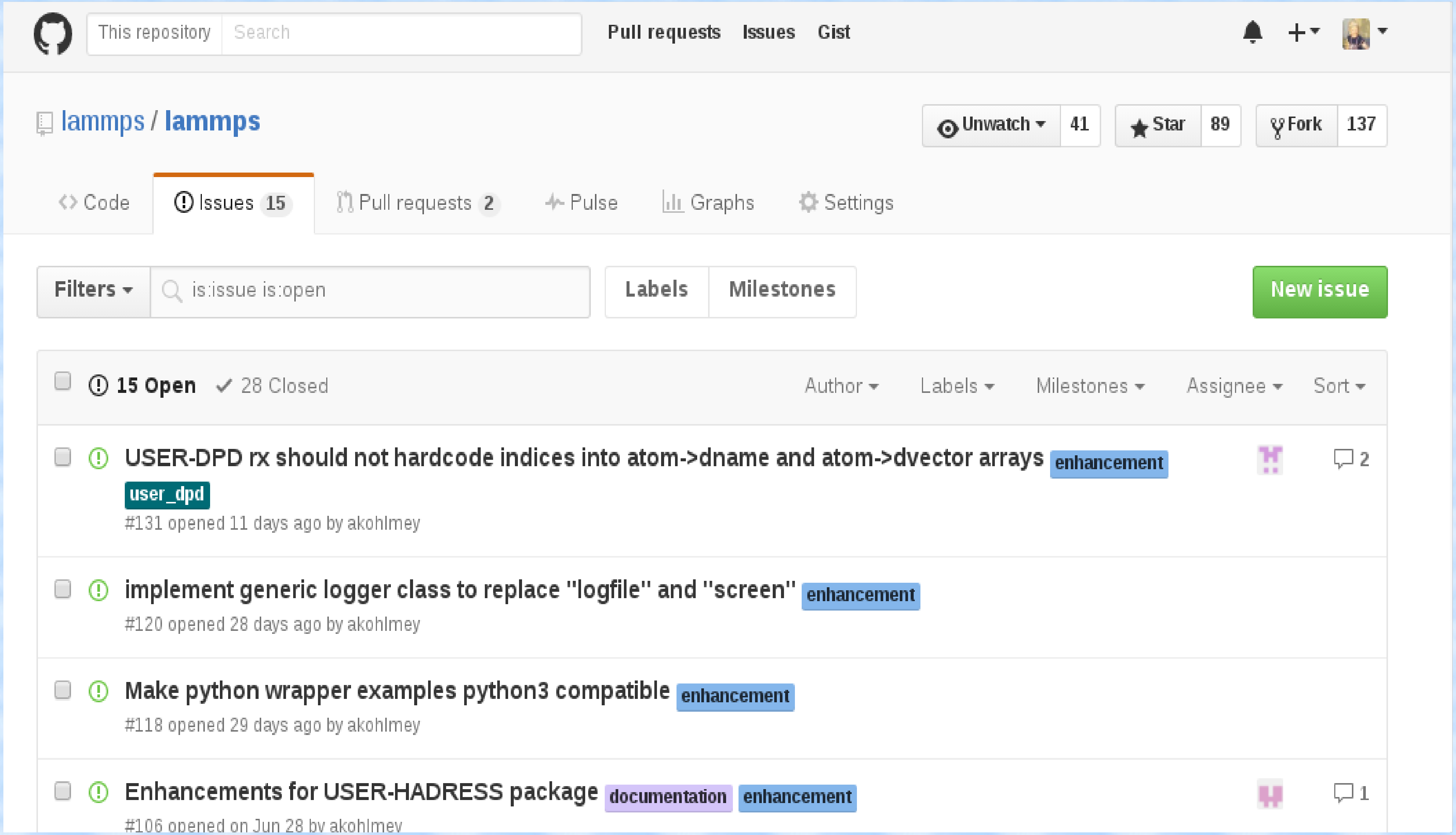
2 Open ✓ 95 Closed

fix_bond_create_updated ✓
#140 opened 11 hours ago by aminaramoon

Added optimized vashishta potential. ✓ 3
#139 opened 22 hours ago by andeplane

ProTip! Updated in the last three days: updated:>2016-08-14.

Reporting Bugs and Suggesting New Features



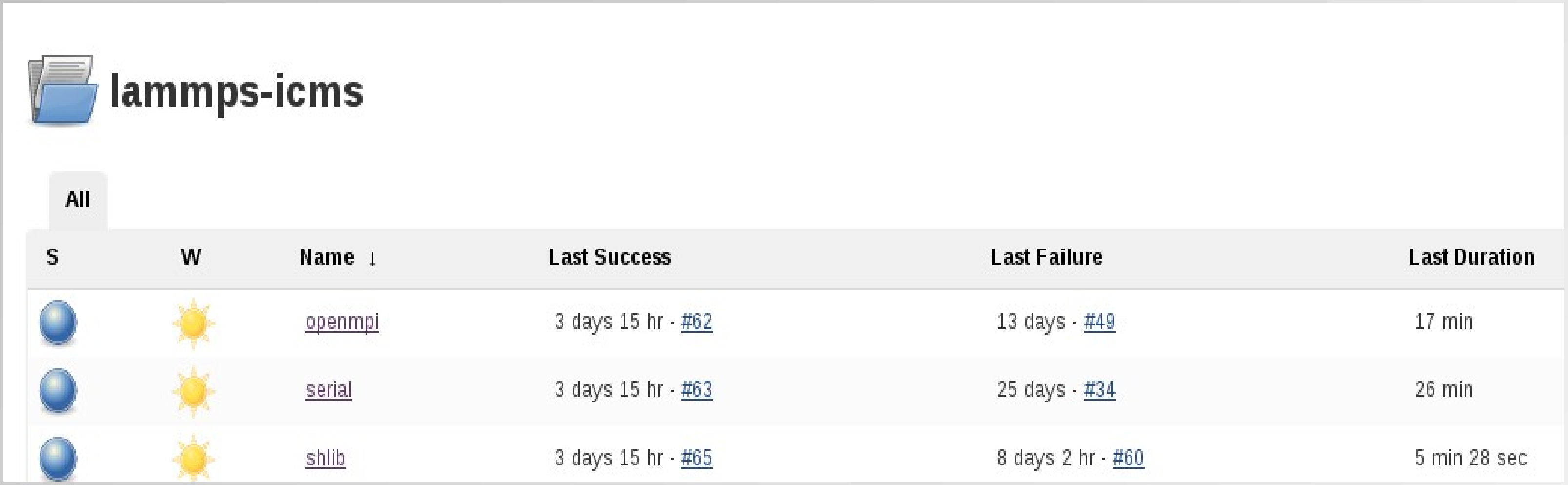
USER-DPD rx should not hardcode indices into atom->dname and atom->dvector arrays enhancement user_dpd
#131 opened 11 days ago by akohlmey — 2

implement generic logger class to replace "logfile" and "screen" enhancement
#120 opened 28 days ago by akohlmey

Make python wrapper examples python3 compatible enhancement
#118 opened 29 days ago by akohlmey

Enhancements for USER-HADRESS package documentation enhancement
#106 opened on Jun 28 by akohlmey — 1

Public Continuous Integration and Regression Testing at ci.lammps.org



lammps-icms

All

S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	openmpi	3 days 15 hr - #62	13 days - #49	17 min
●	☀	serial	3 days 15 hr - #63	25 days - #34	26 min
●	☀	shlib	3 days 15 hr - #65	8 days 2 hr - #60	5 min 28 sec

- Commits to GitHub repository are automatically checked against many inputs for errors
- Pull request contribution tested for compilability
- Advanced checks and pre-compiled packages

Development Procedure

- Clone original or forked git repository
- Check out the ‘develop’, ‘release’, or ‘stable’ branch depending on preference
- Create a “feature branch” for development
- Modify sources, test, commit
- Create a separate “feature branch” for each new feature or for bugfixes or modifications
- update with upstream before submitting PR

Code Submission Process

- Get GitHub account, create fork of LAMMPS
- Push (local) feature branch into forked repo
- Go to LAMMPS GitHub page and create a pull request after comparing branch to ‘develop’
- Fill out, modify the pull request template text
- Submit either as draft (=more changes coming) or regular pull request (=ready)
- Wait for automated integration tests to clear
- Fix issues with failed CI tests, if any

Code Review Process

- Only designated core developer may merge
- Code must pass all automatic CI tests
- Developers may request additional tests
- Review can be done by multiple developers
- Discretion of person doing the merge, if a code is sufficiently well reviewed and approved
- Minimum is one approval from a core developer
- Review requests, manual and automatic

Required Code Properties

- Code should follow documented coding style and conventions (not a strict requirement)
- No tabs, no trailing whitespace, no CR-LF
- All new/changed features must be documented
- Manual must build and pass spell-check tests
- Code has to build with legacy make and Cmake
- Added feature must provide some innovation
- No undesired side effects, no performance hit
- Higher scrutiny if changes to core code

More Required Code Properties

- Contributed code should be “valgrind clean”
- Code must work in parallel and serial
- Header files should not include library headers use forward declaration and PIMPL instead
- Limited use of C++ (STL) headers
- Base code must remain C++11 compatible
- Dependency on libraries only for “packages”
- Use of C++14 or later only in “packages”