LAMMPS – An Object Oriented Scientific Application

Dr. Axel Kohlmeyer
(with a little help from several friends)
Senior Scientific Computing Expert

Information and Telecommunication Section
The Abdus Salam International Centre
for Theoretical Physics

http://sites.google.com/site/akohlmey/

akohlmey@ictp.it

The Abdus Salam
International Centre
for Theoretical Physics

Workshop on Computer Programming and
Advanced Tools for Scientific Research Work

LAMMPS is a Collaborative Project

A few lead developers and many significant contributors:

- Steve Plimpton, Paul Crozier, Aidan Thompson
  (Sandia National Laboratory, Albuquerque NM)

  - Roy Pollock (LLNL), Ewald and PPPM solvers
  - Mike Brown (ORNL), GPU package
  - Greg Wagner (Sandia), MEAM package for MEAM potential
  - Mike Parks (Sandia), PERI package for Peridynamics
  - Rudra Mukherjee (JPL), POEMS package for rigid body motion
  - Reese Jones (Sandia), USER-ATC package for coupling to continuum
  - Ilya Valuev (JIHT), USER-AWPMD package for wave-packet MD
  - Christian Trott (Sandia), USER-CUDA package
  - A. Jaramillo-Botero (Caltech), USER-EFF electron force field package
  - Christoph Kloss (JKU), LIGGGHTS package for DEM and fluid coupling
  - Metin Aktulga (LBL), USER-REAXC package for C version of ReaxFF
  - Georg Gunzenmuller (EMI), USER-SPH package
  - Axel Kohlmeyer (Temple U, ICTP), USER-OMP, USER-CG-CMM, USER-COLVARS, USER-MOLFILE packages, SMD and IMD support

LAMMPS is an Extensible Project

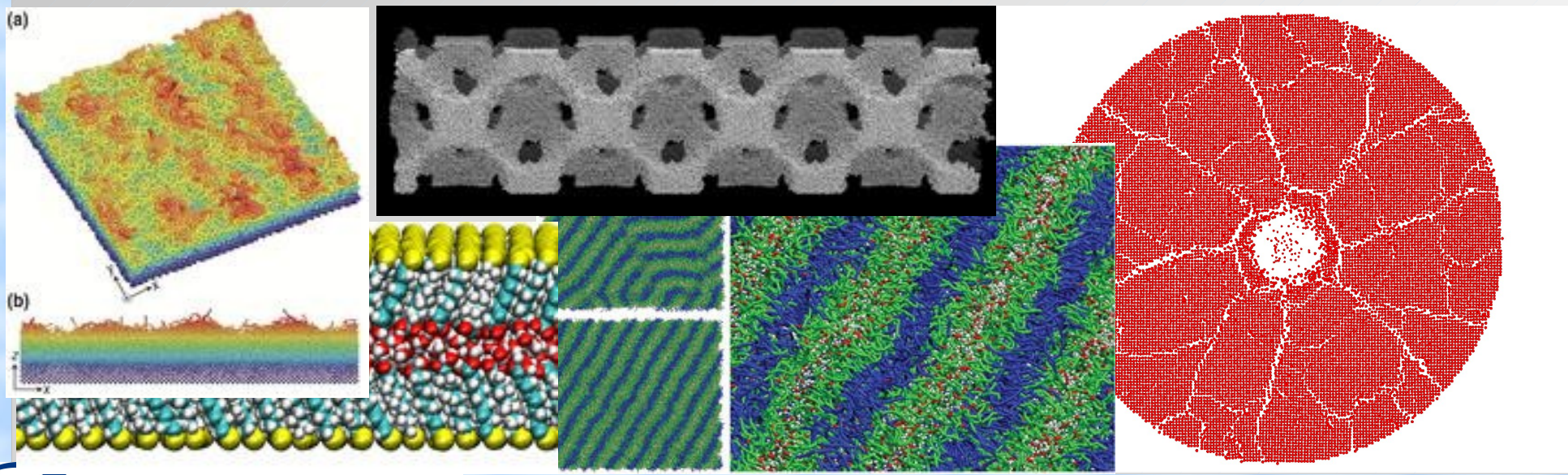
- ~2300 C/C++/CUDA files, 50 Fortran files, about 620,000 lines of code in core executable
- Only about 200 files are essential, about 530 files are compiled by default, 1820 are optional
- Optional files are included through derived C++ classes, extra functionality in bundled libraries
- Three levels of “package support”:
  - Core packages (officially supported)
  - USER-<NAME> packages (supported by individuals)
  - USER-MISC package (mixed bag of everything else)

A Short History of LAMMPS

- Started around 1995 as a DOE/Industry partnership under the lead of Steve Plimpton
- Development used Fortran 77 until 1999
- Converted to Fortran 90 for dynamical memory management. Final Fortran version in 2001
- Current version is a complete rewrite in C++ merging in features from several MD codes written at Sandia (ParaDyn, Warp, GranFlow, GRASP) and many community contributions

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 General Features

- Classical Molecular Dynamics (MD)

  - runs on a single processor or in parallel
  - distributed-memory message-passing parallelism (MPI)
  - GPU (CUDA and OpenCL) and OpenMP support for many code features
  - spatial-decomposition of simulation domain for parallelism
  - open-source distribution
  - highly portable C++
  - optional libraries used: MPI, serial FFT, JPEG
  - 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
  - Python wrapper included, combine with Pizza.py toolkit
  - couple with other codes: LAMMPS calls other code, other code calls LAMMPS, or umbrella code calls both

Particle and Model Types

- simple atoms, metals
- coarse-grained particles (e.g. bead-spring polymers)
- united-atom polymers or organic molecules
- all-atom polymers, organic molecules, proteins, DNA
- granular materials
- coarse-grained mesoscale models
- finite-size spherical and ellipsoidal particles
- finite-size line segment (2d) and triangle (3d) particles
- point dipolar particles
- rigid collections of particles
- hybrid combinations of these

Force Fields

- Simple pairwise additive potentials: Lennard-Jones, Buckingham, Morse, Born-Mayer-Huggins, Yukawa, Soft, Class 2 (COMPASS), Mie, hydrogen bond, tabulated, Coulombic, point-dipole
- Manybody potentials: EAM, Finnis/Sinclair EAM, modified EAM (MEAM), embedded ion method (EIM), EDIP, ADP, Stillinger-Weber, Tersoff, REBO, AIREBO, ReaxFF, COMB, BOP
- Electron force fields: eFF, AWPMD
- Coarse-grained: DPD, GayBerne, REsquared, colloidal, DLVO, SDK
- Mesoscopic potentials: Granular media, Peridynamics, SPH
- Potentials for bond/angles/dihedrals: harmonic, FENE, Morse, nonlinear, Class 2, quartic (breakable), CHARMM, OPLS, cvff, umbrella
- implicit solvent potentials: hydrodynamic lubrication, Debye
- long-range Coulombics and dispersion: Ewald, Wolf, PPPM (similar to particle-mesh Ewald), Ewald/N for long-range Lennard-Jones
- hybrid potentials: multiple pair, bond, angle, dihedral, improper potentials can be used in one simulation
- overlaid potentials: superposition of multiple pair potentials

Ensembles, Boundary Conditions

- 2d or 3d systems
- orthogonal or non-orthogonal (triclinic symmetry) simulation domains
- constant NVE, NVT, NPT, NPH, Parinello/Rahman integrators
- thermostatting options for groups and geometric regions of atoms
- pressure control via Nose/Hoover or Berendsen barostatting in 1 to 3 dimensions, coupled and uncoupled
- simulation box deformation (tensile and shear)
- harmonic constraint forces, collective variables (MTD, ABF, SMD)
- rigid body constraints
- SHAKE bond and angle constraints
- bond breaking, formation, swapping
- walls of various kinds
- non-equilibrium molecular dynamics (NEMD)
- Properties and manipulations can be controlled by custom functions

Methods

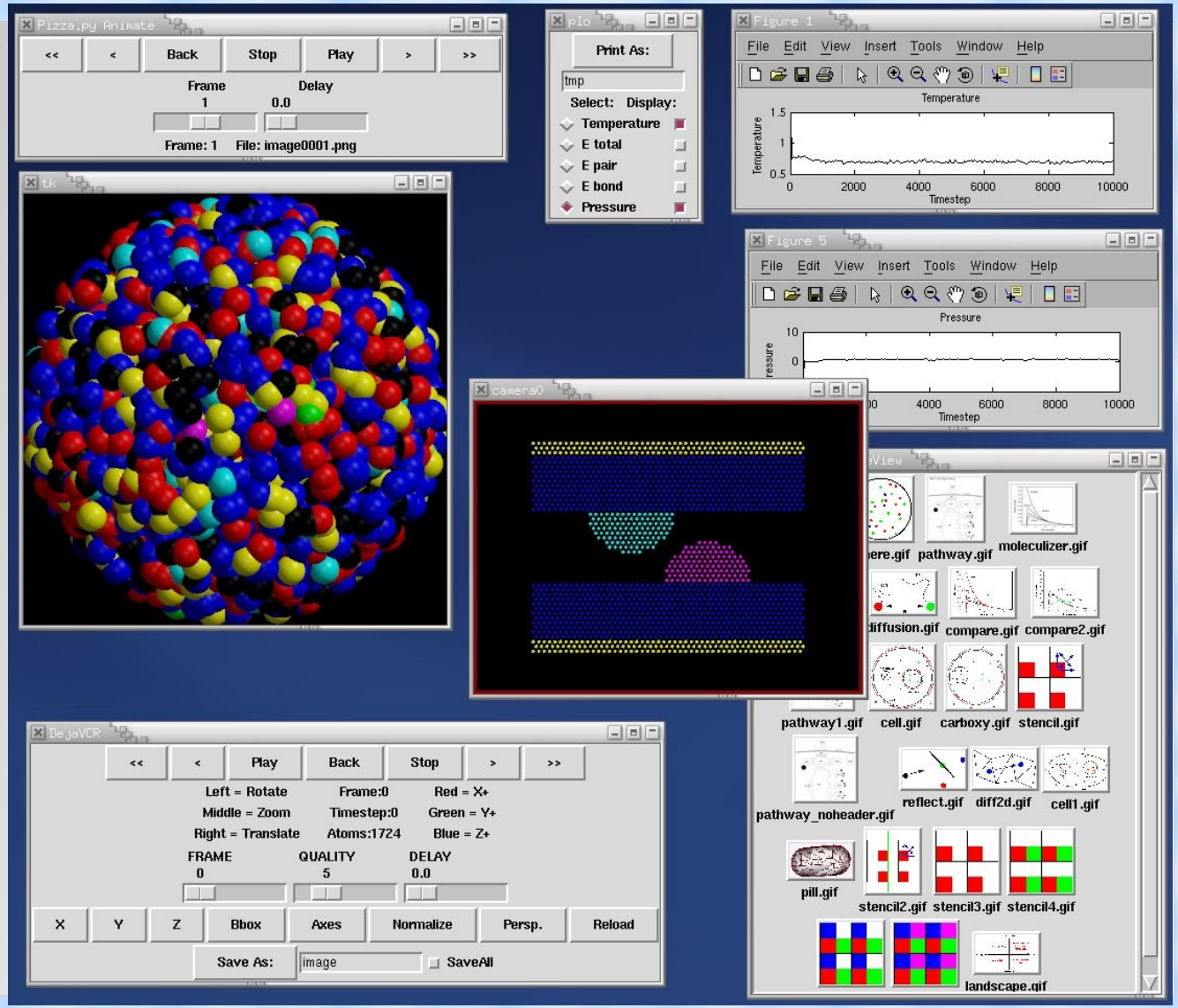
- Integrators:
  - Velocity Verlet, r-RESPA multi-timestepping, Brownian dynamics, rigid bodies
  - Energy minimization with various algorithms
- Multi-replica methods:
  - Nudged-elastic band
  - Parallel replica dynamics
  - Temperature accelerated MD
  - Parallel tempering MD
  - Split short-range / long-range force computation

Not so Common Features

- generalized aspherical particles
- stochastic rotation dynamics (SRD)
- real-time visualization and interactive MD
- atom-to-continuum coupling with finite elements
- coupled rigid body integration via the POEMS library
- grand canonical Monte Carlo insertions/deletions
- Direct Simulation Monte Carlo for low-density fluids
- Peridynamics mesoscale modeling
- targeted and steered molecular dynamics
- two-temperature electron model
- On-the-fly parallel processing of data (direct and via rerun)

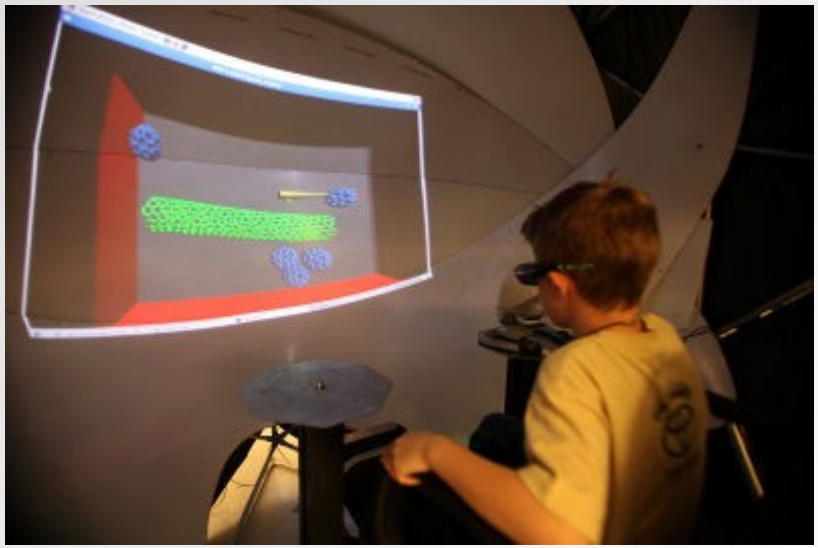
Pizza.py Companion Toolkit

- Each tool is a Python class
- Use multiple tools simultaneously from command-line, scripts, or GUIs
- Tools for building LAMMPS input, reading LAMMPS output, conversion, analysis, plotting, viz, etc
- GUI-based tool to run a LAMMPS simulation in real-time ...

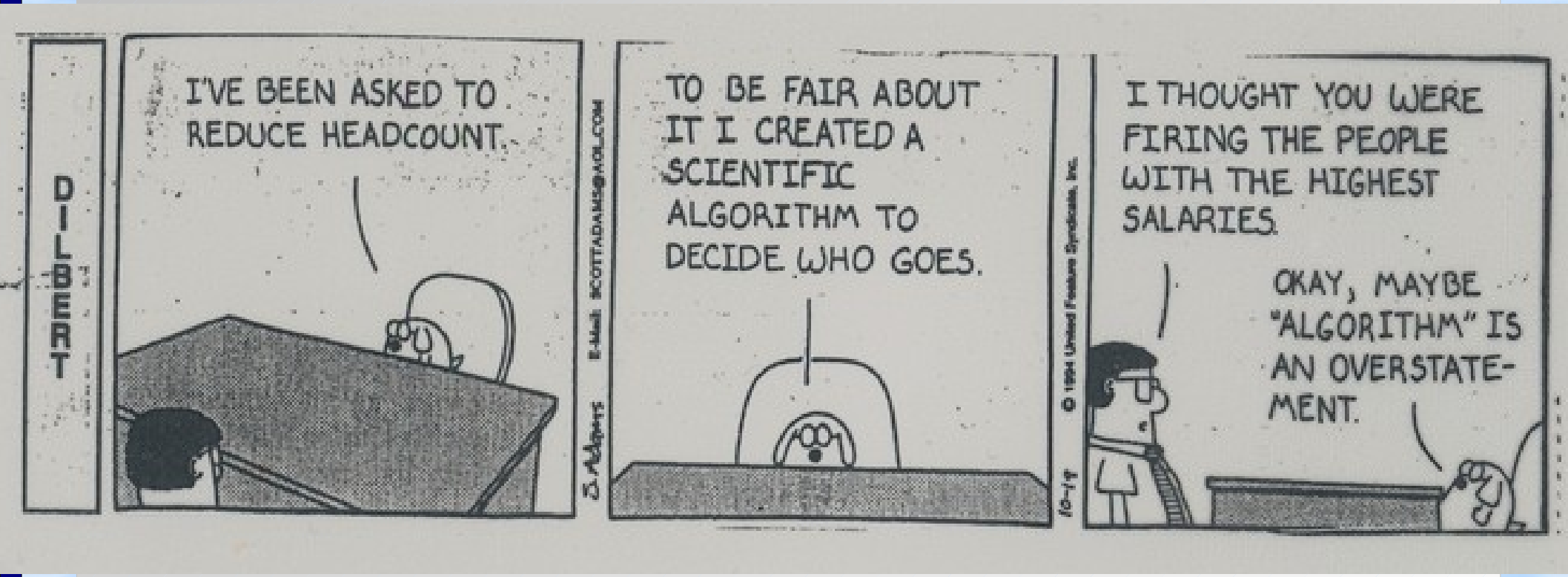


LAMMPS for Outreach
The Nano Dome

- Single person immersive, stereo-3d, haptic, and interactive simulation/visualization environment
- Combines HPC, visualization, molecular simulation, virtual reality, and STEM outreach



What's an Algorithm?



Timescale in Classical MD

- Timescale of simulation is most serious bottleneck in MD
- Timestep size limited by atomic oscillations:
  - C-H bond = 10 fmsec → ½ to 1 fmsec timestep
  - Debye frequency = 10^{13} → 2 fmsec timestep
- Reality is often on a much longer timescale:
  - protein folding (msec to seconds)
  - polymer entanglement (msec and up)
  - glass relaxation (seconds to decades)
  - nanoparticle rheology (milliseconds to seconds)
- Even smaller timestep in tight-binding or quantum-MD

Particle-Time Metric

- Atom * steps = size of your simulation
- 10^{12} is supercomputer scale → 10^6 atoms for 10^6 timesteps
  2 months on a 1.7 GHz Pentium (simple LJ system)
  few hours on 100s of processors
- 1 cubic micron (10^{10} atoms) for a nanosecond (10^6 steps)
  1000 flops per atom per step → 10^{19} flops
  MD is 10% of peak → 1 day on a Petaflop machine

Serial Performance

- Low-level data structures
  - C-like, Fortran-like
  - x[N][3] = coordinates = 3N contiguous memory locations
  - one simulation allocates many atom-based arrays
- Neighbor lists
  - O(N) binning
  - Verlet list with skin, stored in large “pages” of integers
  - keep for 10-20 steps
  - biggest memory requirement in code
- Performance is same as C and same as Fortran
  - we don’t do things that slow down pair and neighbor routines
  - people do care how fast your code is

Classical MD in Parallel

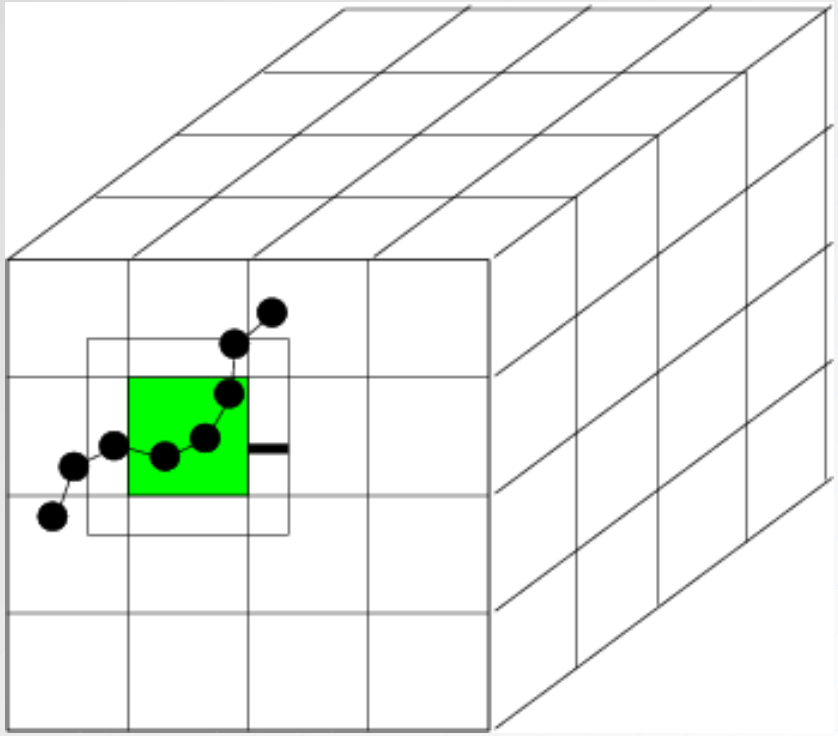
- MD is inherently parallel
  - forces on each atom can be computed simultaneously
  - X and V can be updated simultaneously

- Most widely used MD codes are parallelized via distributed-memory message-passing style parallelism

- MPI → www-unix.mcs.anl.gov/mpi
  - assembly-language of parallel computing
  - lowest-common denominator
  - most portable
  - runs on all parallel machines:
    - SMP shared-memory
    - hybrids = multi-node with multiple procs or cores / node

Parallelism via Spatial-Decomposition

- Physical domain divided into 3d boxes, one per processor
- Communication of “ghost” atoms via
  nearest-neighbor 6-way stencil
- Each processor computes forces on atoms in its box
- Atoms "carry along" molecular topology
  as they migrate to new procs

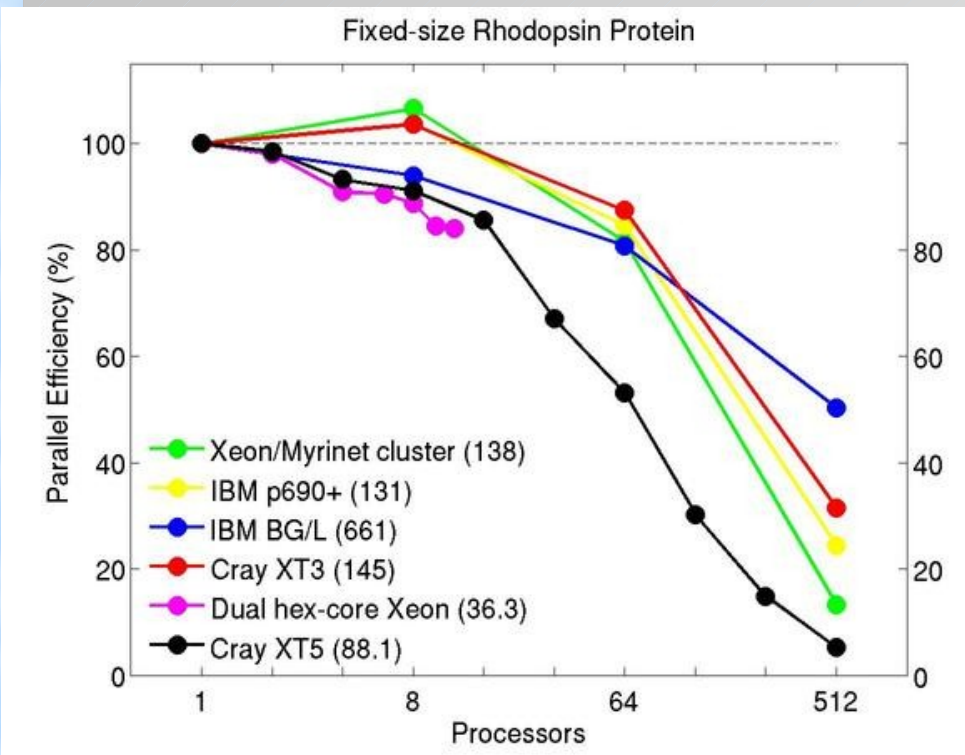


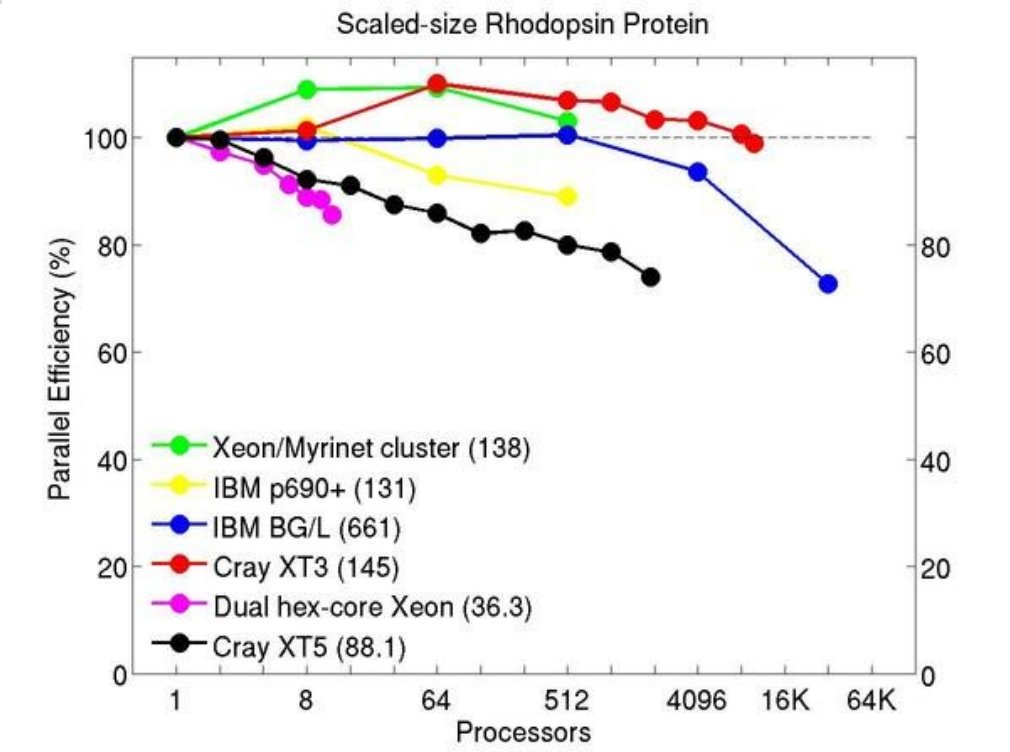
- Work hard for optimal scaling:
  N/P so long as load-balanced

- Computation scales as N/P
- Communication scales
  sub-linear as $(N/P)^{2/3}$
  (for large problems)
- Memory scales as N/P

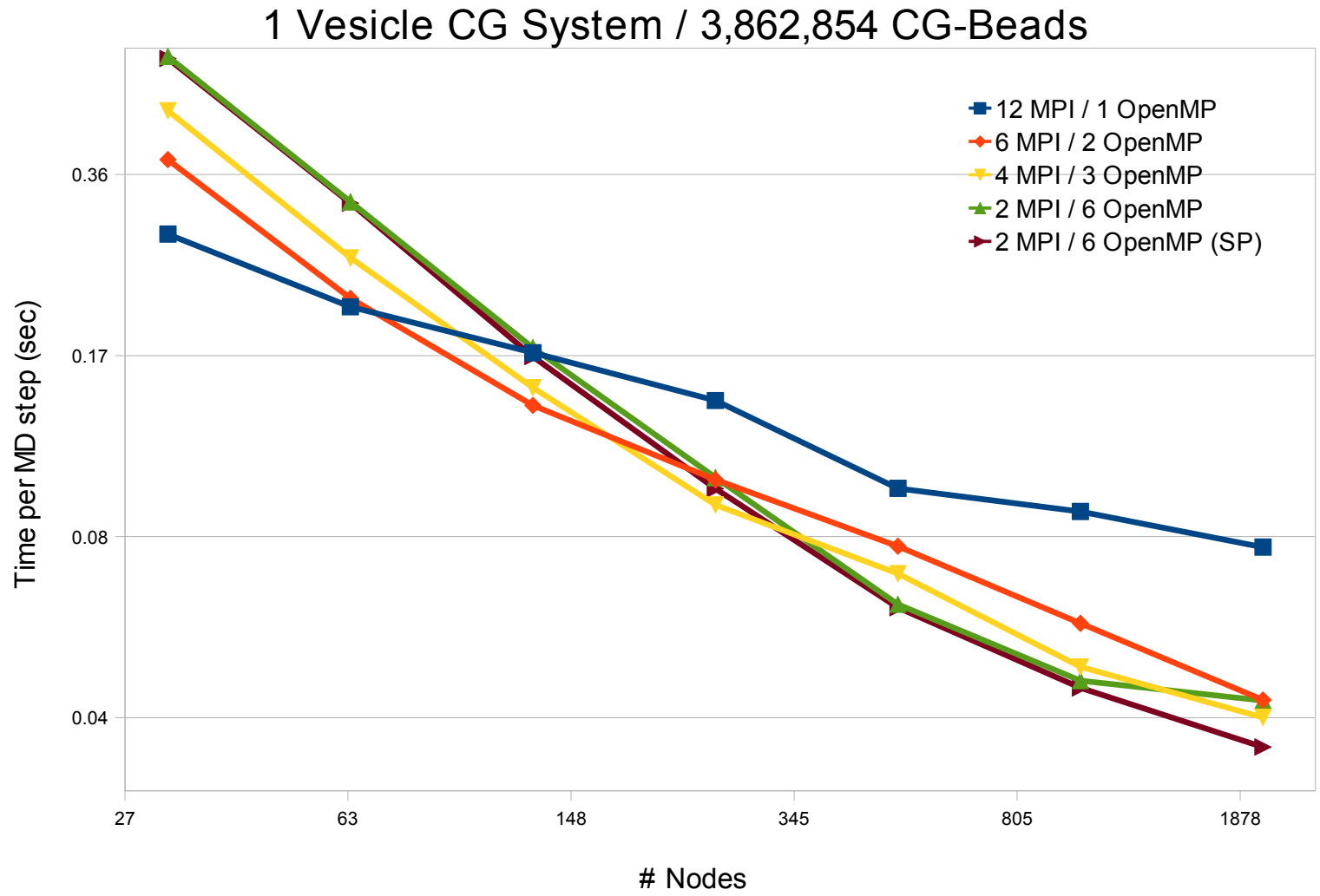
LAMMPS Performance

- Fixed-size (32K atoms) & scaled-size (32K/proc) parallel efficiencies
- Protein (rhodopsin) in solvated lipid bilayer

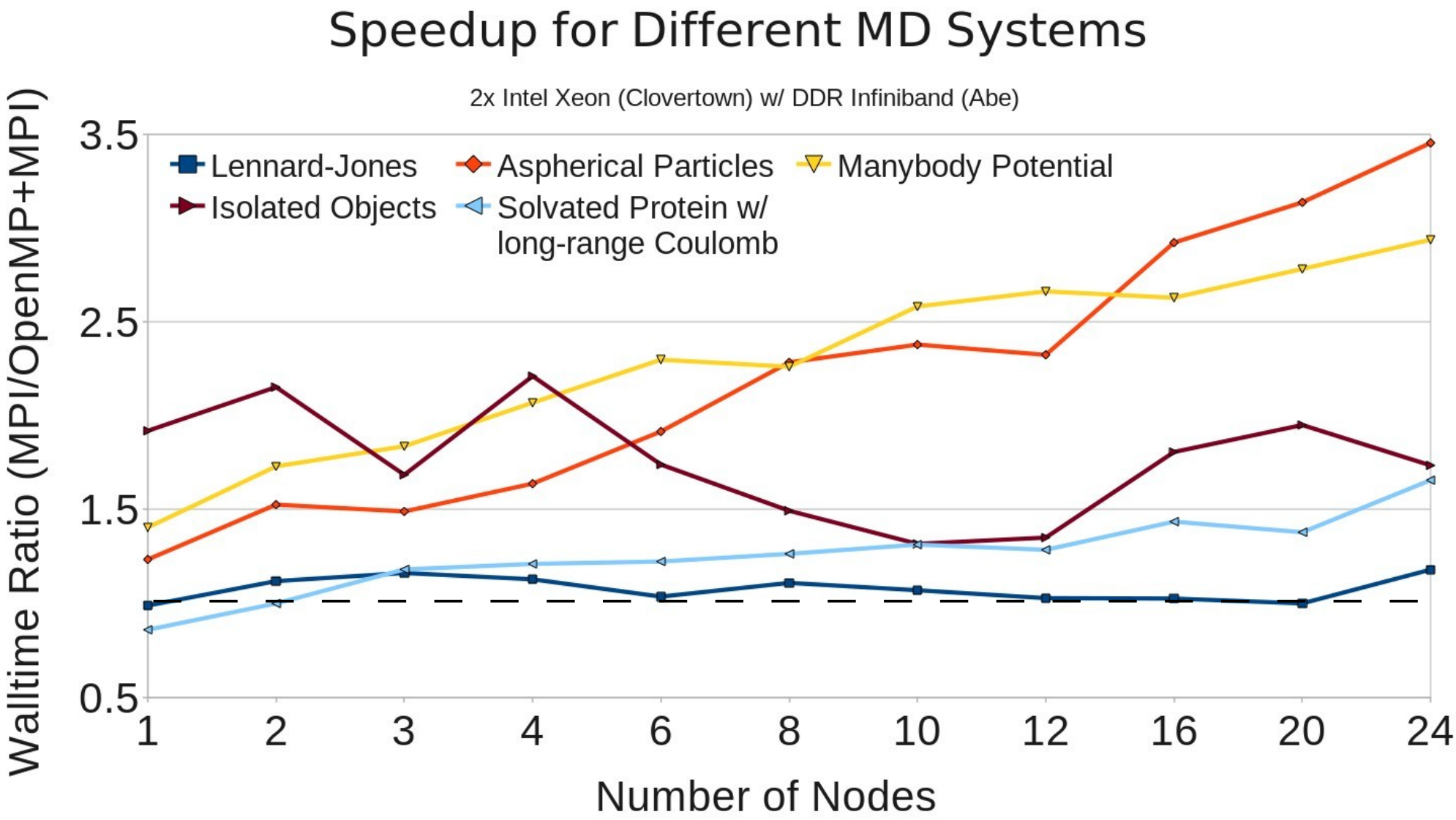




OpenMP/MPI Scaling on Cray XT5



OpenMP+MPI Best Effort vs. MPI



Extending LAMMPS

- In hindsight, this is best feature of LAMMPS
  > 80% of code is “extensions”
- Easy for us and others to add new features (“style”)
  new particle types
  new force fields
  new computations (T, per-atom stress, ...)
  new fix (BC, constraint, integrator, diagnostic, ...)
  new input command (read_data, velocity, run, ...)
- Adding a feature only requires 2 lines in a header file and recompiling

```
# include "pair_airebo.h"
PairStyle ( airebo, PairAIRebo )
```

- Enabled by C++
  virtual parent class for all styles, e.g. pair potentials
  defines interface the feature must provide
  compute(), init(), coeff(), restart(), etc

"Fixes" are Flexible

- Define particle attributes — mass, x, v, f, charge, bonds, angles, orientation, torque, dipole, shear history, ...
- Loop over timesteps:
  - fix_initial — NVE, NVT, NPT, rigid-body integration
  - communicate ghost atoms
  - fix_neighbor — insert particles
  - build neighbor list (once in a while)
  - compute forces
  - communicate ghost forces
  - fix_force — SHAKE, langevin drag, wall, spring, gravity
  - fix_final — NVE, NVT, NPT, rigid-body integration
  - fix_end — volume & T rescaling, diagnostics
  - output to screen and files
- Fixes operate on sub-groups of atoms, add per-atom storage, communicate, write status to restart file, ...

Hybrid Models

- Water/proteins on metal/silica surface
- Metal-semiconductor interface
- Metal islands on amorphous (LJ) substrate
- Specify 2 (or more) pair potentials:

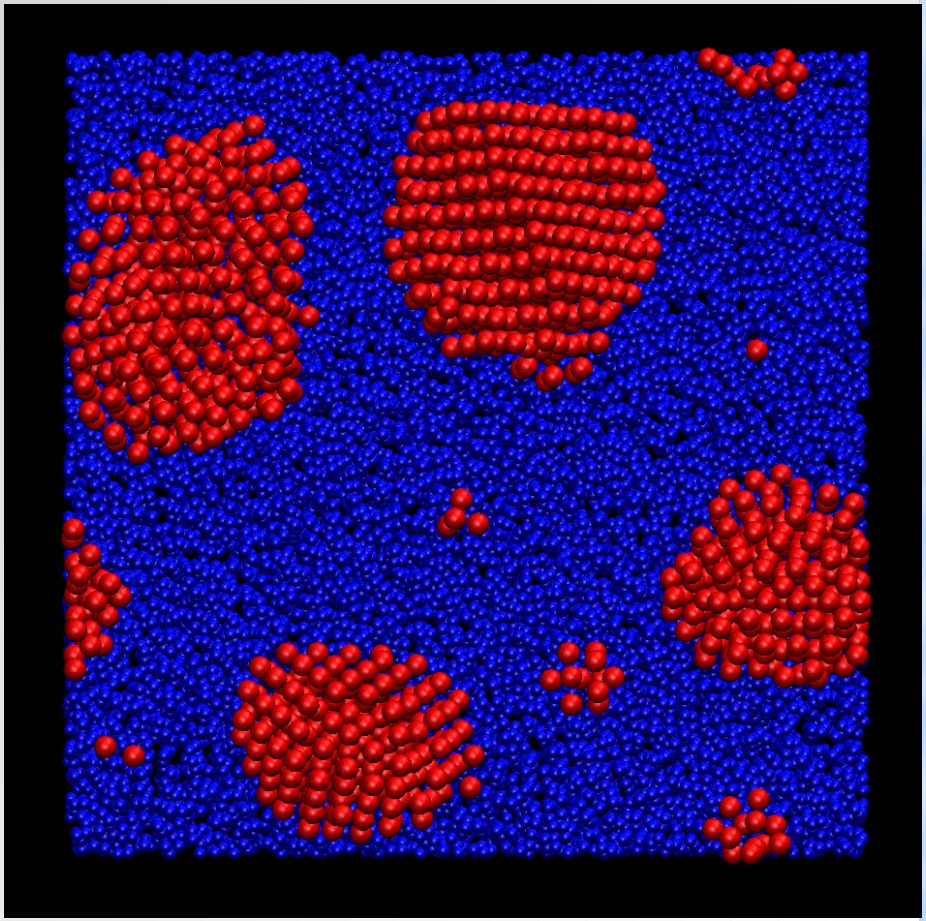
  A-A, B-B, A-B, etc
- Overlay potentials:

  add explicit h-bonds

  add coulomb
- Hybrid in two ways:

  potentials (pair, bond, etc)

  atom style (bio, metal, etc)



Multiple Processor Partitions

- Command-line switch:
  mpirun -np 32 lmp_ibm -partition 8x4 -in in.temper
  partition your 32 procs into 8 4-processor partitions

- "Variable", "loop", "jump" commands in input script
  variable loop t 250.0 300.0 350.0 400.0
  fix 1 all nvt $t $t 0.01

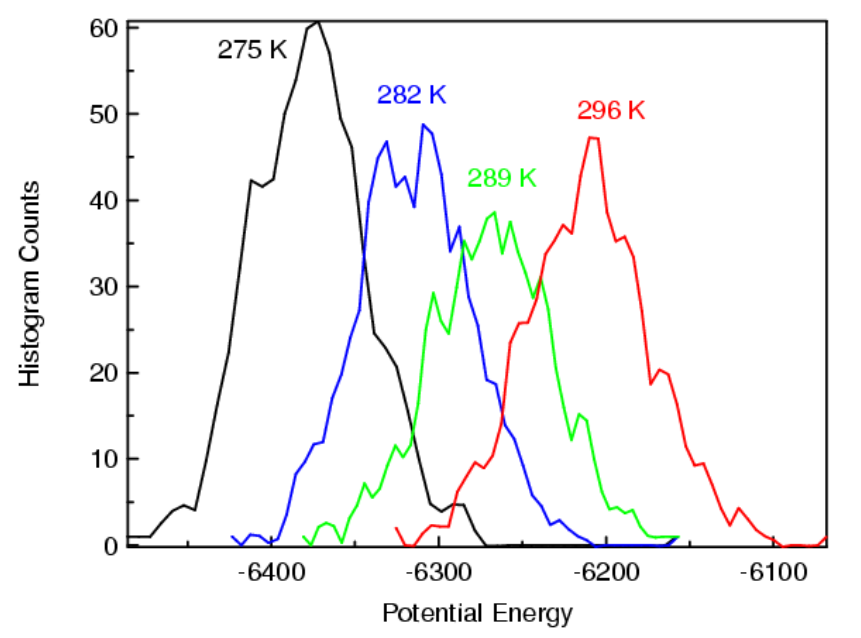
- Run 8 different simulations simultaneously
  at different temperatures
  from different input scripts

- Run 100 simulations one after the other on 8 partitions

- Run 8 simulations with loose coupling → parallel tempering

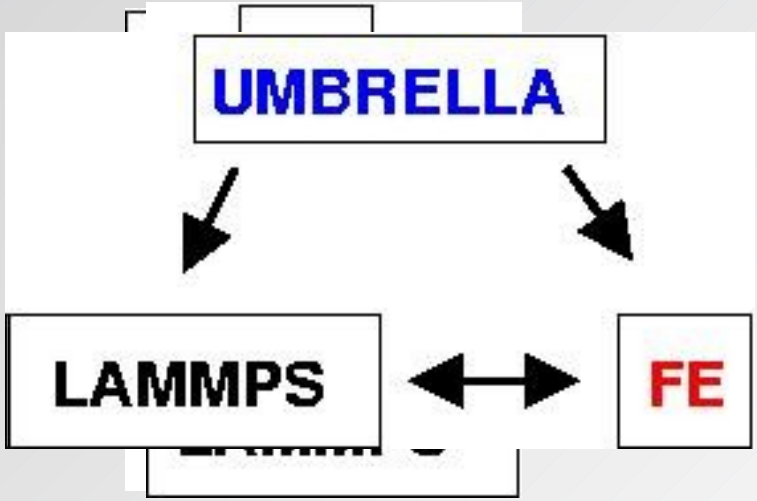
Parallel Tempering

- More efficient sampling of polymer/protein conformations
- Every 100 steps:
  pair up ensembles
  attempt a temperature swap
  Monte Carlo accept/reject
- Need overlapping energy histograms
- Unstuck from energy minima
- Each ensemble cycles up and down thru temperatures



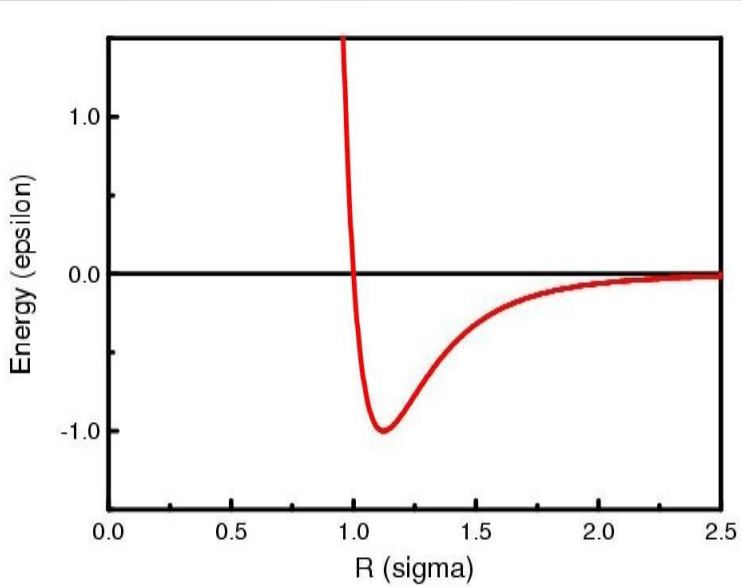
Coupling LAMMPS to Other Codes

- Method 1: MD is the driver
  - MD → FE
  - enabled by fixes, link to external library
  - coupled rigid body solver from RPI
- Method 2: Other code is the driver
  - FE → MD
  - build LAMMPS as a library
  - call from C++, C, Fortran
  - low-overhead to run MD in spurts
  - invoke low-level ops (get/put coords)
- Method 3: Umbrella code is the driver
  - Umbrella code calls MD and FE
  - RPI group linking LAMMPS to their FE codes for deformation problems
  - could run LAMMPS on P procs, FE on Q procs, talk to each other
- Challenge: balance the computation so both codes run efficiently



Classical MD Basics

- Each of N particles is a point mass
  - atom
  - group of atoms (united atom)
  - macro- or meso- particle
- Particles interact via empirical force laws
  - all physics in energy potential → force
  - pair-wise forces (LJ, Coulombic)
  - many-body forces (EAM, Tersoff, REBO)
  - molecular forces (springs, torsions)
  - long-range forces (Ewald)
- Integrate Newton's equations of motion
  - F = ma
  - set of N, coupled ODEs
  - advance as far in time as possible
- Properties via time-averaging ensemble snapshots (vs MC sampling)



MD Timestep

- Velocity-Verlet formulation:
  - update V by ½ step (using F)
  - update X (using V)
  - build neighbor lists (occasionally)
  - compute F (using X)
  - apply constraints & boundary conditions (on F)
  - update V by ½ step (using new F)
  - output and diagnostics

- CPU time break-down:
  - forces = 80%
  - neighbor lists = 15%
  - everything else = 5%

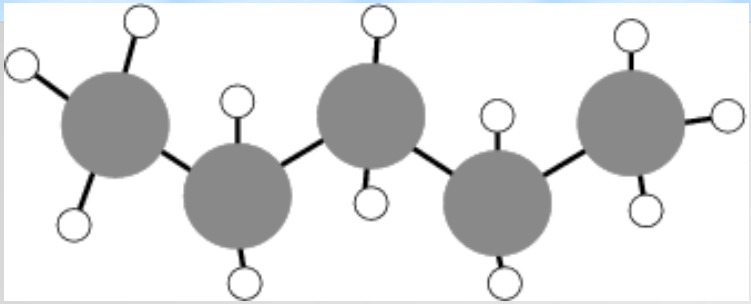
Computational Issues

- These have a large impact on CPU cost of a simulation:

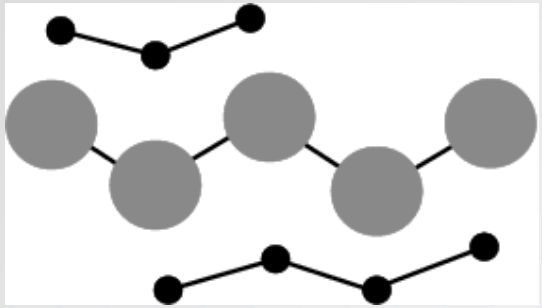
  Level of detail in model
  Cutoff in force field
  Long-range Coulombics
  Neighbor lists

  Newton's 3rd law (compute on ghost atoms, but more communication)
  Timestep size (vanilla, SHAKE, rRESPA)
  Parallelism (already discussed)

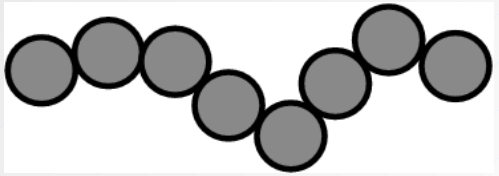
Level of Detail in Polymer Models



- All-atom:
  - Δt = 0.5-1.0 fmsec for C-H
  - C-C distance = 1.5 Angs
  - cutoff = 10 Angs
- United-atom:
  - # of interactions is 9x less
  - Δt = 1.0-2.0 fmsec for C-C
  - cutoff = 10 Angs
  - 20-30x savings over all-atom



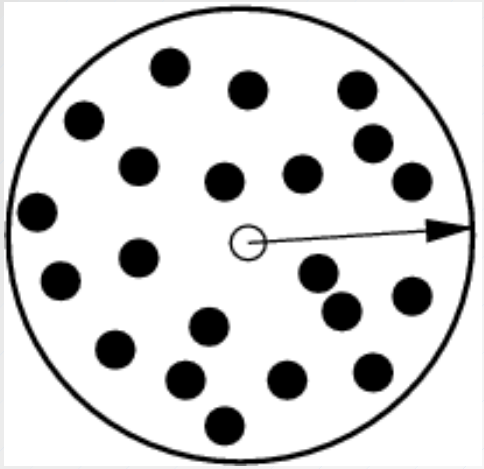
- Bead-Spring:
  - 2-3 C per bead
  - Δt ←→ fmsec mapping is T-dependent
  - $2^{1/6}\sigma$ cutoff → 8x in interactions
  - can be considerable savings over united-atom



- "Eternity" vs "Near-eternity" vs "Not quite possible"

Cutoff in Force Field

- Forces = 80% of CPU cost
- Short-range forces → O(N) scaling for classical MD
  constant density assumption
  pre-factor is cutoff-dependent
- # of pairs/atom = cubic in cutoff
  2x the cutoff → 8x the work
- Use as short a cutoff as can justify:
  LJ = 2.5σ (standard)
  all-atom and UA = 8-12 Angstroms
  bead-spring = $2^{1/6}\sigma$ (repulsive only)
  Coulombics = 12-20 Angstroms
  solid-state (metals) = few neighbor shells (due to screening)
- Test sensitivity of your results to cutoff



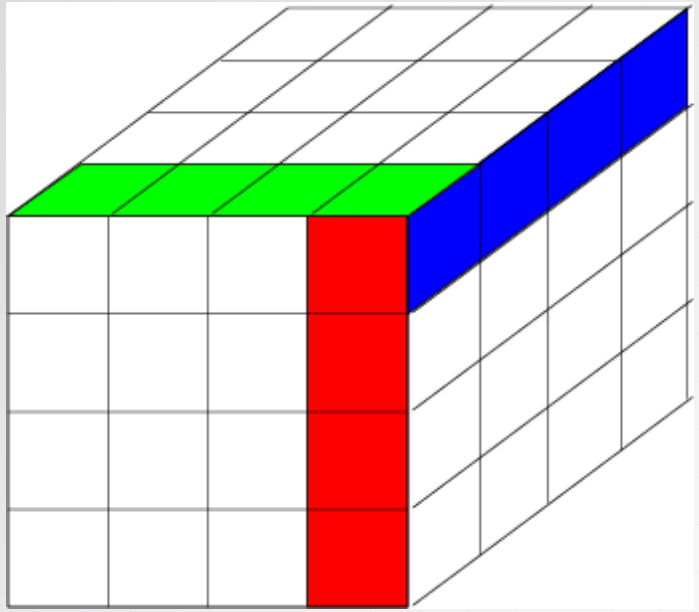
Long-range Coulombics

- Systems that need it:
  - Charged polymers (polyelectrolytes)
  - Organic & biological molecules
  - Ionic solids
  - Not metals (screening)
- Computational issue:
  - Coulomb energy only falls off as 1/r
- Options:

cutoff	scales as N (scales N^3 with cutoff), but large contribution at 10 Angs
Ewald	scales as $N^{3/2}$
particle-mesh Ewald	scales as $N\log_2 N$
multipole	scales as N (but large prefactor)
multigrid	scales as N (but large prefactor)

Parallel FFTs in LAMMPS

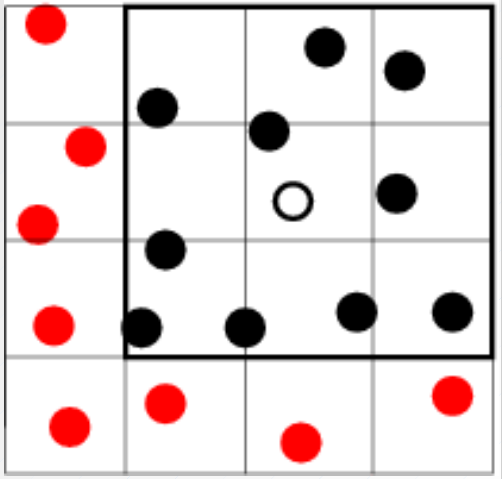
- 3d FFT is 3 sets of 1d FFTs
  in parallel, 3d grid is distributed across procs
  1d FFTs on-processor
  native library or FFTW (www.fftw.org)
  multiple "transposes" of 3d grid
  data transfer can be costly



- FFTs for PPPM can scale poorly
  on large # of procs and on clusters
- Good news: Cost of PPPM is only ~2x more than 8-10 Ang cutoff
- Analytic differentiation (1/3rd the FFTs), hybrid OpenMP/MPI, split-Verlet method to counter scaling issues

Neighbor Lists

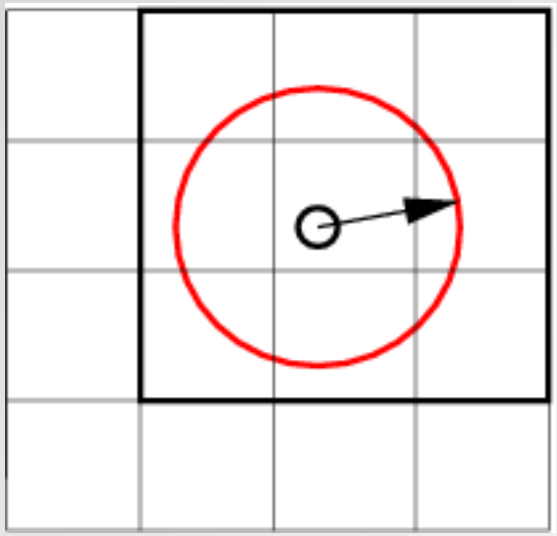
- Problem: how to efficiently find neighbors within cutoff?
- Simple solution:
  - for each atom, test against all others
  - $O(N^2)$ algorithm
- Verlet lists:
  - *Verlet, Phys Rev, 159, p 98 (1967)*
  - $R_{neigh} = R_{force} + \Delta_{skin}$
  - build list: once every few timesteps
  - other timesteps: scan thru larger list
    - for neighbors within force cutoff
  - rebuild list: any atom moves 1/2 of skin
- Link-cells (bins):
  - *Hockney, et al, J Comp Phys, 14, p 148 (1974)*
  - grid simulation box into bins of size R_{force}
  - each timestep: search 27 bins for neighbors



Neighbor Lists (continued)

- Verlet list is ~6x savings over bins
  $V_{sphere} = 4/3\ \pi\ r^3$
  $V_{cube} = 27\ r^3$

- Fastest methods do both:
  link-cell to build Verlet list
  Verlet list on non-build timesteps
  O(N) in CPU and memory
  constant-density assumption
  this is what LAMMPS implements



LAMMPS Input

- Reads an input script (ASCII text) via re-direction:
  lmp_mac -echo screen -in in.colloid

- One command per line, acted on immediately

- Command name + arguments

atom_style	molecular
read_data	water.data
fix	1 all nve
run	10000

- Have doc pages for individual commands handy!

- Examples and bench sub-directories have sample input scripts

Sample Input Script

```
# 3d Lennard-Jones melt

variable        x index 20                      # concept of variables
variable        y index 20
variable        z index 20

units           lj
atom_style      atomic

lattice         fcc 0.8442
region          box block 0 $x 0 $y 0 $z
create_box      1 box
create_atoms    1 box
mass            1 1.0

velocity        all create 1.44 87287 loop geom
pair_style      lj/cut 2.5
pair_coeff      1 1 1.0 1.0 2.5

neighbor        0.3 bin
neigh_modify    delay 0 every 20 check no

fix             1 all nve                       # concept of groups

run             100
```

LAMMPS Output

- log.lammps contains what is printed to screen
  thermodynamic info
  Pizza.py log tool, gnu tool, matlab tool

- "dump" command outputs snapshots of atom properties
  default format is simple: id, type, x, y, z
  other supported formats: XYZ, DCD, XTC
  conversion tools: PDB, Ensight, XYZ, VTK
  Rasmol, Raster3d, SVG, etc
  Pizza.py dump tool, pdbfile tool, xyz tool, etc

Bundled Example Problems

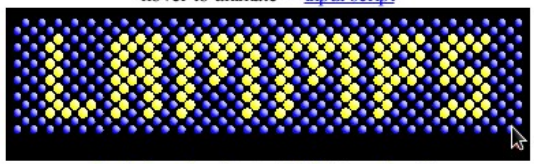
- colloid: colloid system with explicit solvent (2d)
- crack: crack growth in a LJ crystal (2d)
- dipole: dipolar particles (2d)
- ellipse: ellipsoidal GayBerne particles (2d)
- flow: Couette/Poisseuille flow between walls (2d)
- friction: rubbing of 2 irregular surfaces (2d)
- indent: crystal response to spherical indenter (2d)
- meam: MEAM potential (3d)
- melt: LJ lattice (3d)
- micelle: self-assembly of tiny lipid molecules (2d)
- min: energy minimization of LJ melt (2d)
- nemd: non-equilibrium MD run with triclinic box (2d)
- obstacle: flow around obstacles (2d)
- peptide: small peptide chain in water (3d)
- pour: granular particle pour and flow (2d/3d)
- rigid: rigid bodies (3d)
- shear: shear of a metal slab with void (quasi-3d)

lammps.sandia.gov

LAMMPS Molecular Dynamics Simulator

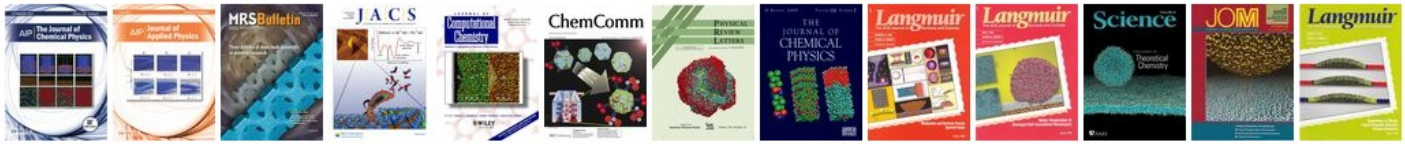
***lamp**: a device that generates light, heat, or therapeutic radiation; something that illumines the mind or soul* -- www.dictionary.com

hover to animate -- input script



physical analog (start at 3:25) & explanation

Big Picture	Code	Documentation	Results	Related Tools	Context	User Support
Features	Download	Manual	Publications	Pre/Post Processing	Authors	Mail list
Non-features	SourceForge	Developer Guide	Pictures	Pizza.py Toolkit	History	MD to LAMMPS glossary
FAQ	Latest Features & Bug Fixes	Tutorials	Movies	Offsite LAMMPS packages	Funding	User Scripts and HowTos
Wish list	Unfixed bugs	Commands	Benchmarks	Visualization	Open source	Workshops
.	.	.	Citing LAMMPS	Other MD codes	.	Contribute to LAMMPS



LAMMPS is a classical molecular dynamics code, and an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator.

LAMMPS has potentials for soft materials (biomolecules, polymers) and solid-state materials (metals, semiconductors) and coarse-grained or mesoscopic systems. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale.

LAMMPS runs on single processors or in parallel using message-passing techniques and a spatial-decomposition of the simulation domain. The code is designed to be easy to modify or extend with new functionality.

LAMMPS is distributed as an open source code under the terms of the GPL. The current version can be downloaded here. Links are also included to older F90/F77 versions. Periodic releases are also available on SourceForge.

lammps.sandia.gov/#nogo ...andia National Laboratories, a US Department of Energy laboratory. The main authors of LAMMPS are listed on this page along with contact info and other contributors. Funding for