

HPC molecular simulations using LAMMPS

Paul S. Crozier

HPC Advisory Council Stanford Workshop

December 6 - 7, 2011



Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.





Acknowledgements

Ilmenau Univ. of Tech., Germany

Christian Trott & Lars Winterfeld

Sandia National Laboratories

Steve Plimpton & Aidan Thompson

Creative Consultants

Greg Scantlen & Mary Payne

Oak Ridge National Lab

Mike Brown, Scott Hampton & Pratul Agarwal

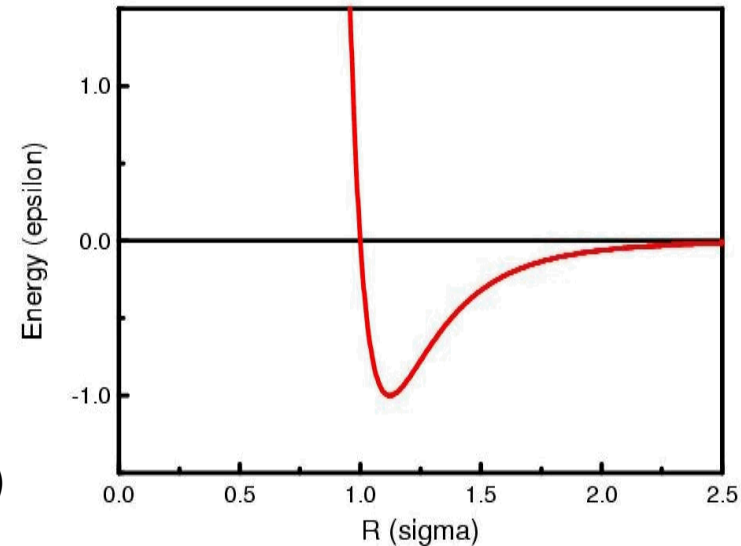


Presentation Outline

- 1. Molecular dynamics basics**
- 2. Why use LAMMPS?**
- 3. Why run LAMMPS on GPUs?**
- 4. What is the GPU-LAMMPS project?**
- 5. What can LAMMPS now do on GPUs?**

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 \rightarrow 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 $\frac{1}{2}$ step (using **F**)
 - update **X** (using **V**)
 - build neighbor lists (occasionally)
 - compute **F** (using **X**)
 - apply constraints & boundary conditions (on **F**)
 - update **V** by $\frac{1}{2}$ 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**

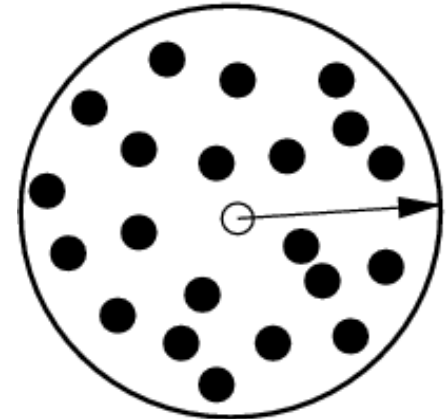
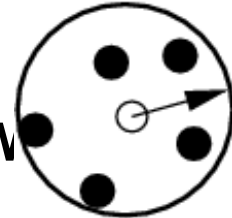


Classical MD in Parallel

- MD is inherently parallel
 - forces on each atom can be computed simultaneously
 - X and V can be updated simultaneously
- Most MD codes are parallel
 - via distributed-memory message-passing paradigm (MPI)
- Computation scales as N = number of atoms
 - ideally would scale as N/P in parallel
- Can distribute:
 - atoms communication = scales as N
 - forces communication = scales as N/\sqrt{P}
 - space communication = scales as N/P or $(N/P)^{2/3}$

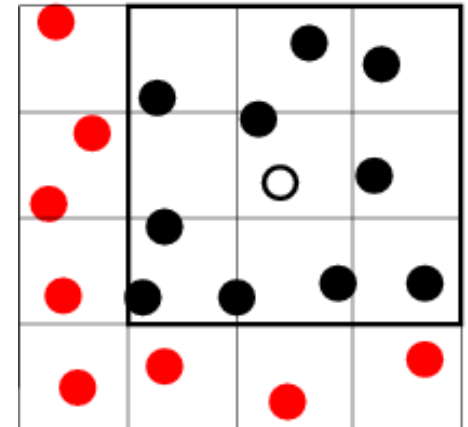
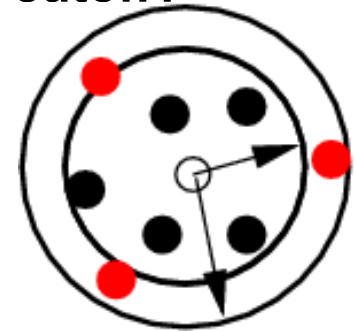
Cutoff in Force Field

- Forces = 80% of CPU cost
- Short-range forces $\rightarrow O(N)$ scaling for classical MD
 - constant density assumption
 - pre-factor is cutoff-dependent
- # of pairs/atom = cubic in cutoff
 - 2x the cutoff \rightarrow 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



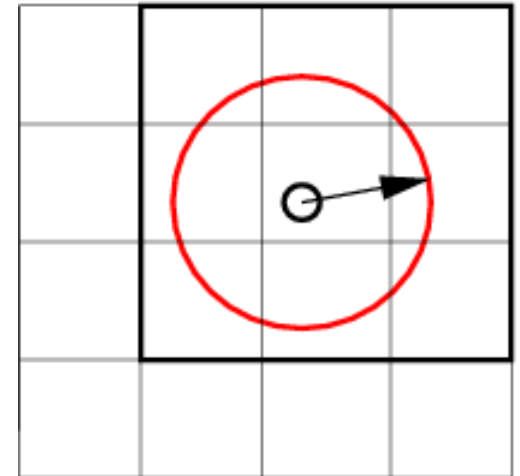
Neighbor Lists

- **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)
 - $R_{\text{neigh}} = R_{\text{force}} + \Delta_{\text{skin}}$
 - **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
 - $V_{\text{sphere}} = 4/3 \pi r^3$
 - $V_{\text{cube}} = 27 r^3$
- 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





Presentation Outline

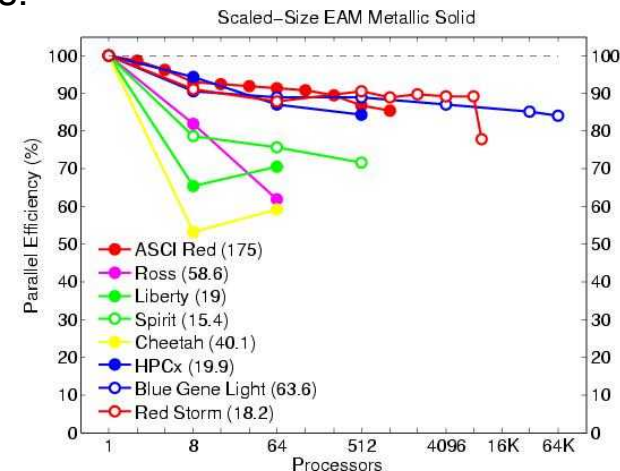
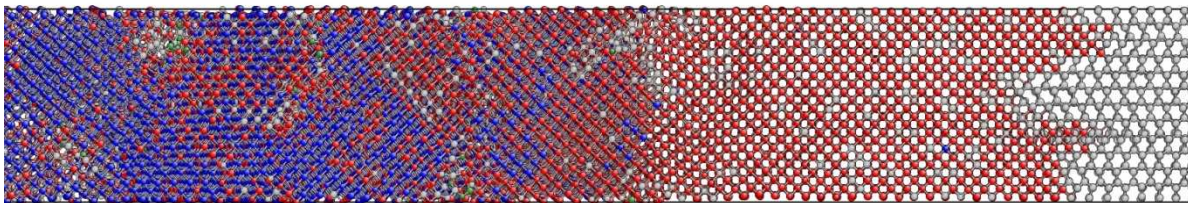
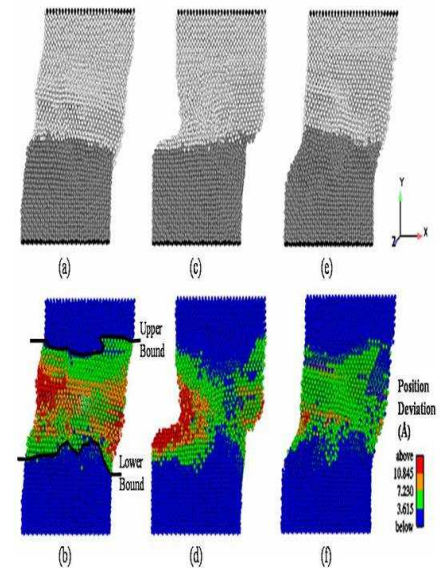
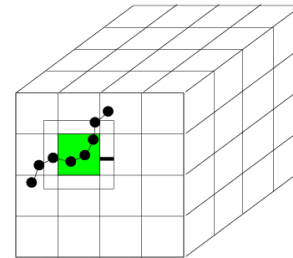
1. Molecular dynamics basics
2. Why use LAMMPS?
3. Why run LAMMPS on GPUs?
4. What is the GPU-LAMMPS project?
5. What can LAMMPS now do on GPUs?

Why Use LAMMPS?

(Large-scale Atomic/Molecular Massively Parallel Simulator)

<http://lammps.sandia.gov>

- Classical MD code.
- Open source, highly portable C++.
- Freely available for download under GPL.
- Easy to download, install, and run.
- Well documented.
- Easy to modify or extend with new features and functionality.
- Active user's e-mail list with over **650** subscribers.
- Upcoming users' workshop: Aug 9 – 11, 2011.
- Since Sept. 2004: over 50k downloads, grown from 53 to 175 kloc.
- Spatial-decomposition of simulation domain for parallelism.
- Energy minimization via conjugate-gradient relaxation.
- Radiation damage and two temperature model (TTM) simulations.
- Atomistic, mesoscale, and coarse-grain simulations.
- Variety of potentials (including many-body and coarse-grain).
- Variety of boundary conditions, constraints, etc.



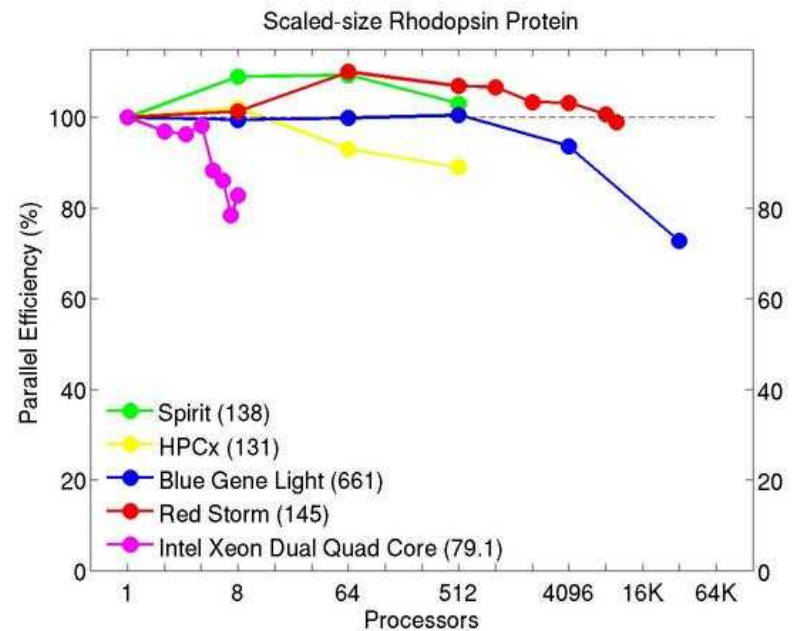
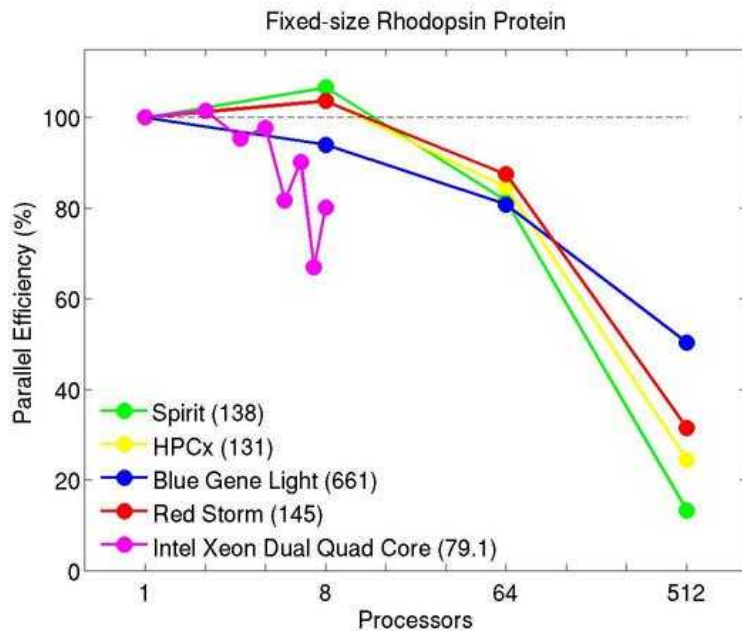
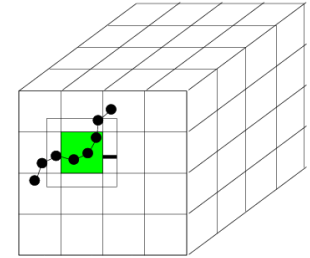
Why Use LAMMPS?

Answer 1: (Parallel) Performance

Protein (rhodopsin) in solvated lipid bilayer

Fixed-size (32K atoms) & scaled-size (32K/proc) parallel efficiencies

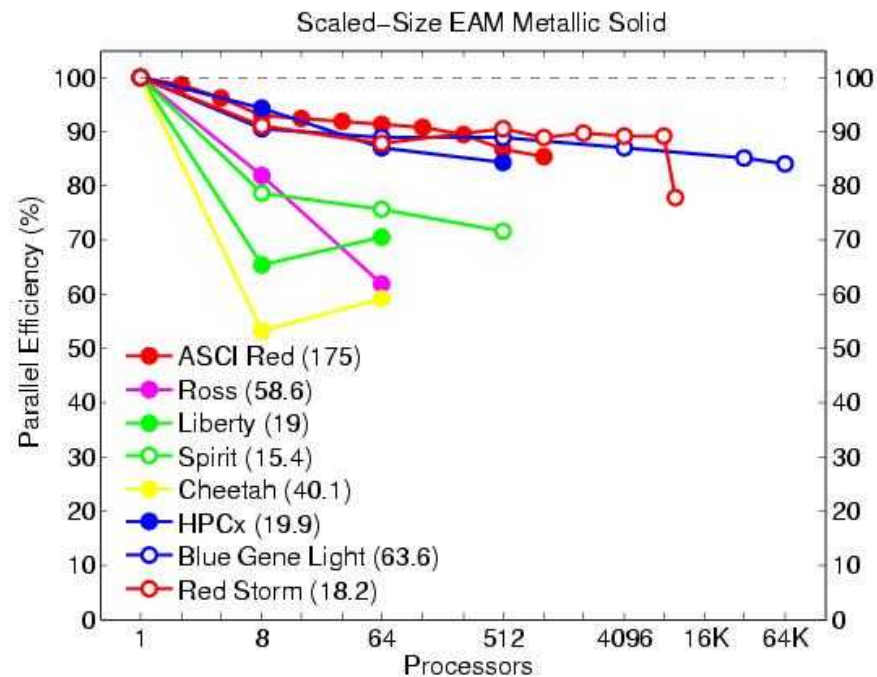
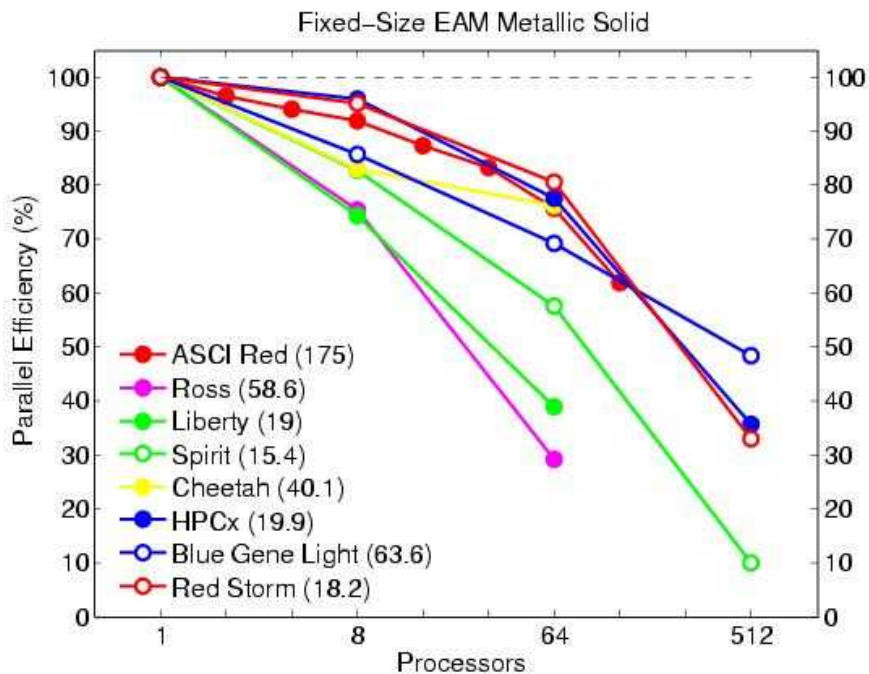
Billions of atoms on 64K procs of Blue Gene or Red Storm



Typical speed: $5E-5$ core-sec/atom-step (LJ $1E-6$, ReaxFF $1E-3$)

Parallel performance, EAM

- Fixed-size (32K atoms) and scaled-size (32K atoms/proc) parallel efficiencies
- Metallic solid with EAM potential

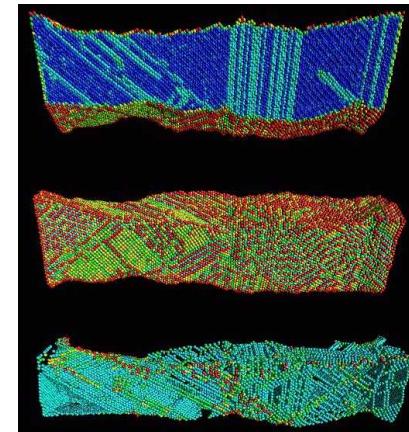
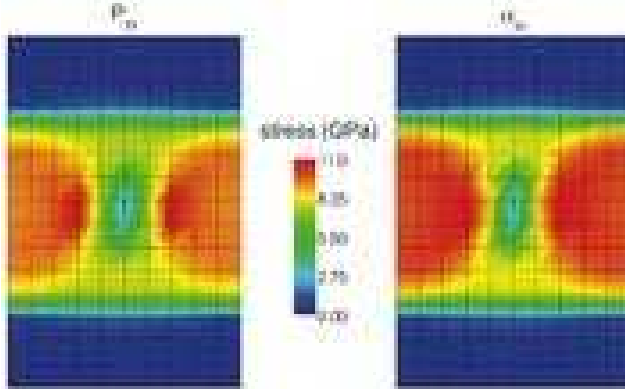


- Billions of atoms on 64K procs of Blue Gene or Red Storm
- Opteron processor speed: $5.7E-6$ sec/atom/step (0.5x for LJ, 12x for protein)

Why Use LAMMPS?

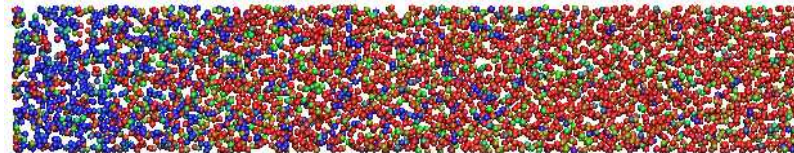
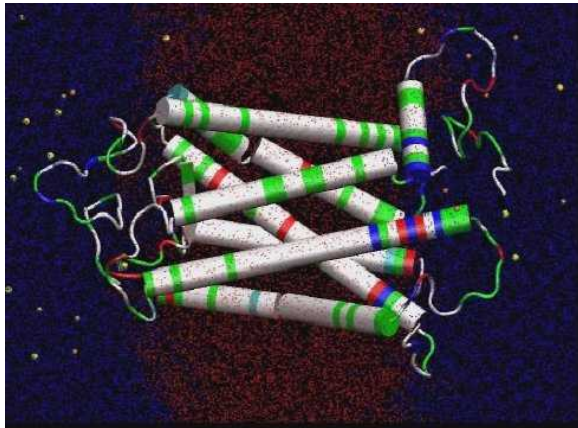
Answer 2: Versatility

**Solid
Mechanics**



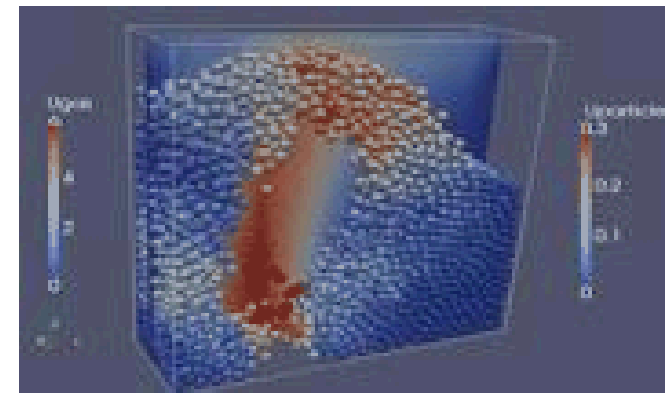
**Materials
Science**

Biophysics



Chemistry

**Granular
Flow**





Why Use LAMMPS?

Answer 3: Modularity

pair_reax.cpp

fix_nve.cpp

pair_reax.h

fix_nve.h

LAMMPS Objects

atom styles: atom, charge, colloid, ellipsoid, point dipole

pair styles: LJ, Coulomb, Tersoff, ReaxFF, AI-REBO, COMB, MEAM, EAM, Stillinger-Weber,

fix_styles: NVE dynamics, Nose-Hoover, Berendsen, Langevin, SLLOD, Indentation,...

compute styles: temperatures, pressures, per-atom energy, pair correlation function, mean square displacements, spatial and time averages

Goal: All computes works with all fixes work with all pair styles work with all atom styles



Why Use LAMMPS?

Answer 4: Potential Coverage

LAMMPS Potentials

pairwise potentials: Lennard-Jones, Buckingham, ...

charged pairwise potentials: Coulombic, point-dipole

manybody potentials: EAM, Finnis/Sinclair, modified EAM

(MEAM), embedded ion method (EIM), Stillinger-Weber, Tersoff, AI-REBO, ReaxFF, COMB

electron force field (eFF)

coarse-grained potentials: DPD, GayBerne, ...

mesoscopic potentials: granular, peridynamics

long-range Coulombics and dispersion: Ewald, PPPM (similar to particle-mesh Ewald)



Why Use LAMMPS?

Answer 4: Potentials

LAMMPS Potentials (contd.)

bond potentials: harmonic, FENE,...

angle potentials: harmonic, CHARMM, ...

dihedral potentials: harmonic, CHARMM,...

improper potentials: harmonic, cvff, class 2 (COMPASS)

polymer potentials: all-atom, united-atom, bead-spring, breakable

water potentials: TIP3P, TIP4P, SPC

implicit solvent potentials: hydrodynamic lubrication, Debye

force-field compatibility with common CHARMM, AMBER, OPLS,
GROMACS options



Why Use LAMMPS?

Answer 4: Range of Potentials

LAMMPS Potentials

Biomolecules: CHARMM, AMBER, OPLS, COMPASS (class 2),
long-range Coulombics via PPPM, point dipoles, ...

Polymers: all-atom, united-atom, coarse-grain (bead-spring FENE),
bond-breaking, ...

Materials: EAM and MEAM for metals, Buckingham, Morse, Yukawa,
Stillinger-Weber, Tersoff, AI-REBO, ReaxFF, COMB, eFF...

Mesoscale: granular, DPD, Gay-Berne, colloidal, peri-dynamics, DSMC...

Hybrid: can use combinations of potentials for hybrid systems:
water on metal, polymers/semiconductor interface,
colloids in solution, ...



Why Use LAMMPS?

Answer 5: Easily extensible

- **One of the best features of LAMMPS**
 - 80% of code is “extensions” via styles
 - only 35K of 175K lines is core of LAMMPS
- **Easy to add new features via 14 “styles”**
 - new particle types = atom style
 - new force fields = pair style, bond style, angle style, dihedral style, improper style
 - new long range = kspace style
 - new minimizer = min style
 - new geometric region = region style
 - new output = dump style
 - new integrator = integrate style
 - new computations = compute style (global, per-atom, local)
 - new fix = fix style = BC, constraint, time integration, ...
 - new input command = command style = read_data, velocity, run, ...
- **Enabled by C++**
 - virtual parent class for all styles, e.g. pair potentials
 - defines interface the feature must provide
 - compute(), init(), coeff(), restart(), etc



Presentation Outline

1. Molecular dynamics basics
2. Why use LAMMPS?
3. Why run LAMMPS on GPUs?
4. What is the GPU-LAMMPS project?
5. What can LAMMPS now do on GPUs?



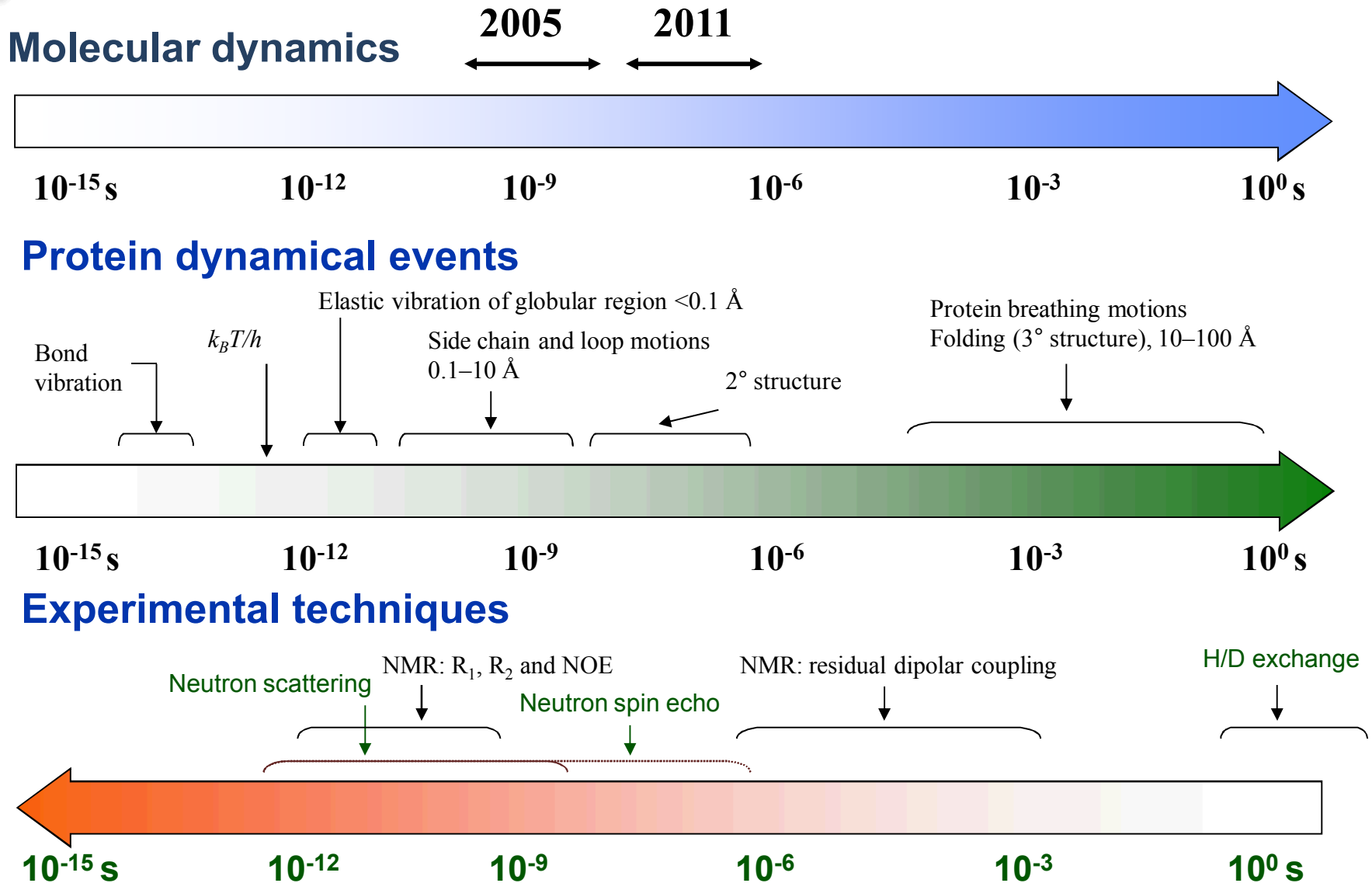
Why run LAMMPS on GPUs?

(Some motivations for adding GPU capabilities to LAMMPS)

- **User community interest**
- **The future is many-core, and GPU computing is leading the way**
- **GPUs and Nvidia's CUDA are the industry leaders**
 - Already impressive performance, each generation better
 - More flops per watt
 - CUDA is way ahead of the competition
- **Other MD codes are working in this area and have shown impressive speedups**
 - HOOMD
 - NAMD
 - Folding@home
 - Others

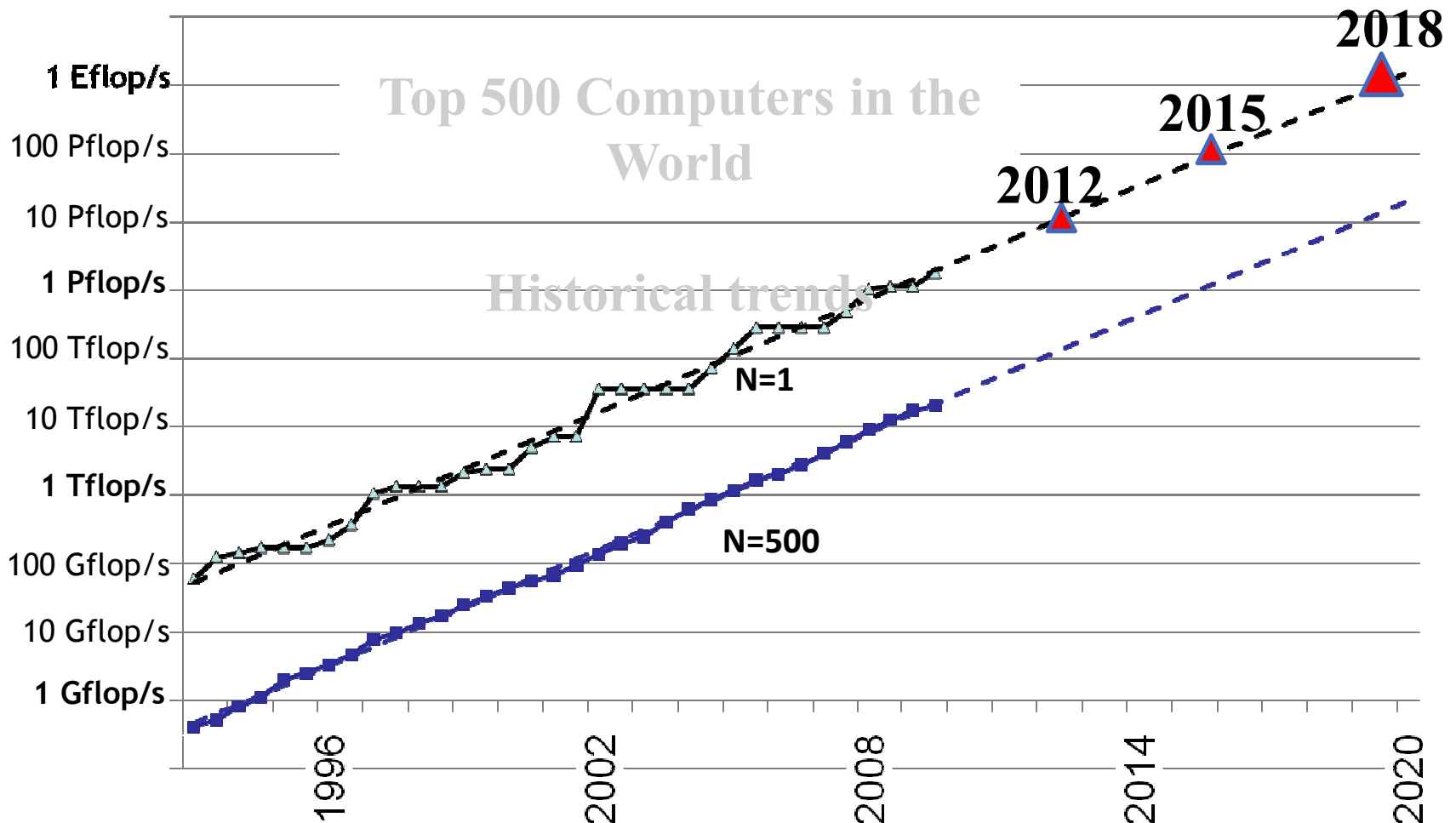
Why run LAMMPS on GPUs?

(Motivation: Access longer time-scales)



Why run LAMMPS on GPUs?

(Exaflop machines are coming, and they'll likely have GPUs or something similar)

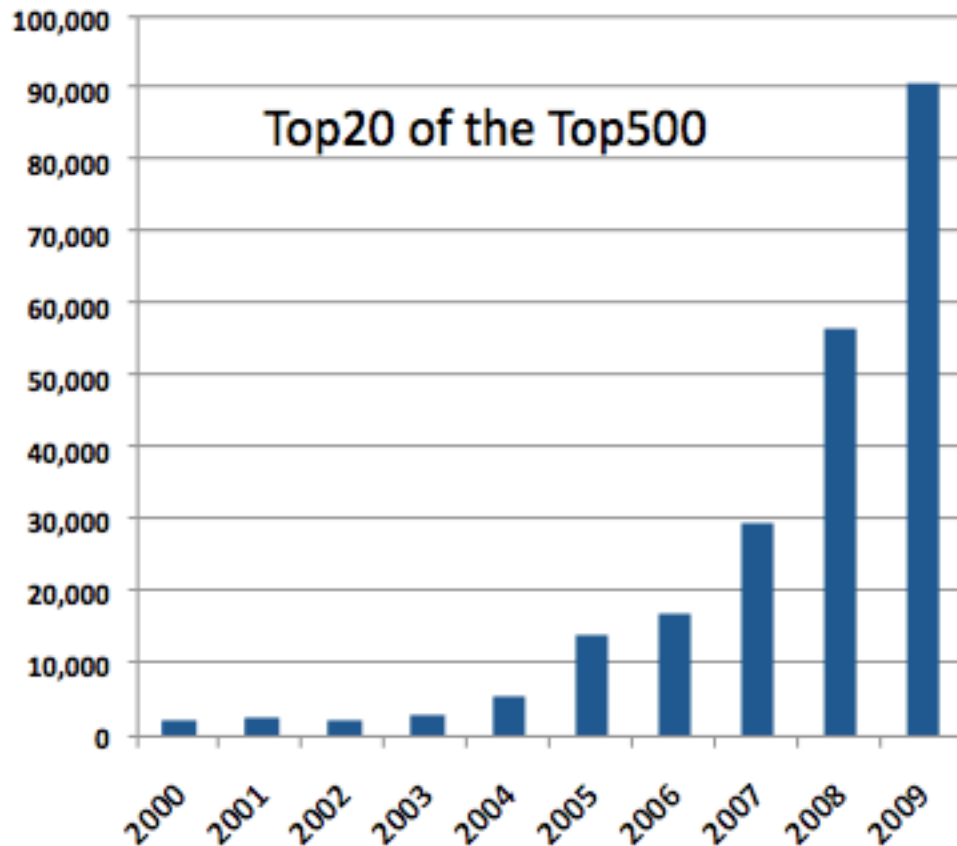


Courtesy: Al Geist (ORNL), Jack Dongarra (UTK)

Why run LAMMPS on GPUs?

(Supercomputers of the future will have much higher concurrency, like GPUs.)

Average Number of Processors Per Supercomputer



Fundamental assumptions of system software architecture and application design did not anticipate exponential growth in parallelism

Courtesy: Al Geist (ORNL)



Why run LAMMPS on GPUs?

(Future architectures very likely to have GPU-like accelerators.)

- **Fundamentally different architecture**
 - Different from the traditional (homogeneous) machines
- **Very high concurrency: Billion way in 2020**
 - Increased concurrency on a single node
- **Increased Floating Point (FP) capacity from accelerators**
 - Accelerators (GPUs/FPGAs/Cell/? etc.) will add heterogeneity

Significant Challenges:

Concurrency, Power and Resiliency

2012 → 2015 → 2018 ...



Presentation Outline

1. Molecular dynamics basics
2. Why use LAMMPS?
3. Why run LAMMPS on GPUs?
4. What is the GPU-LAMMPS project?
5. What can LAMMPS now do on GPUs?

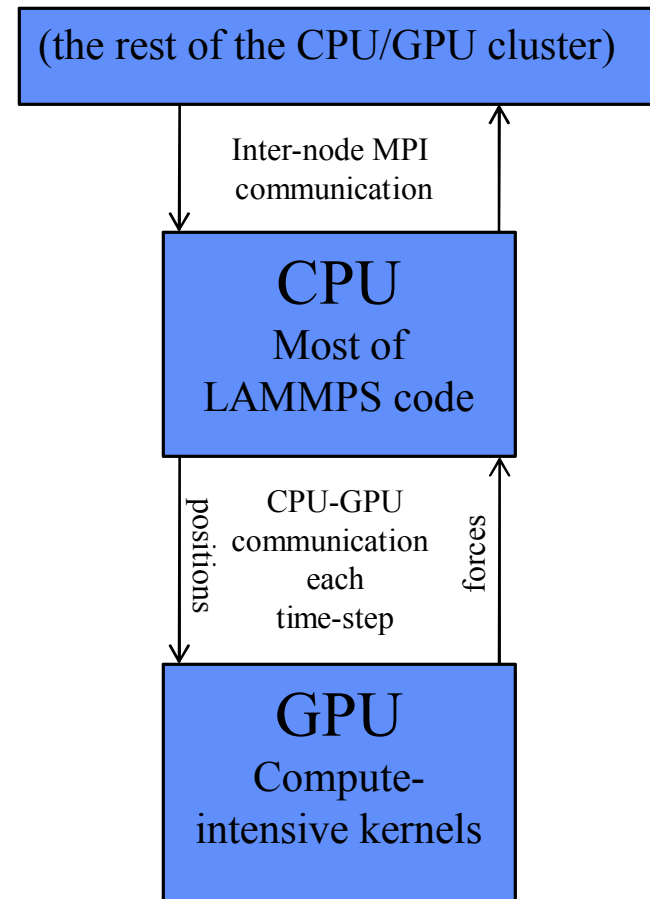
GPU-LAMMPS project goals & strategy

Goals

- Better time-to-solution on particle simulation problems we care about.
- Get LAMMPS running on GPUs (other MD codes don't have all of the cool features and capabilities that LAMMPS has).
- Maintain all of LAMMPS functionality while improving performance (roughly 2x – 100x speedup).
- Harness next-generation hardware, specifically CPU+GPU clusters.

Strategy

- Enable LAMMPS to run efficiently on CPU+GPU clusters. Not aiming to optimize for running on a single GPU.
- Dual parallelism
 - Spatial decomposition, with MPI between CPU cores
 - Force decomposition, with CUDA on individual GPUs
- Leverage the work of internal and external developers.





Presentation Outline

1. Molecular dynamics basics
2. Why use LAMMPS?
3. Why run LAMMPS on GPUs?
4. What is the GPU-LAMMPS project?
5. What can LAMMPS now do on GPUs?



A tale of two packages ...

LAMMPS now has two optional “packages” that allow it to harness GPUs: the “GPU package” (developed by developed by Mike Brown at ORNL) and the “USER-CUDA package,” (developed by Christian Trott at U Technology Ilmenau in Germany):

GPU package:

- Designed to exploit hardware configurations where one or more GPUs are coupled with many cores of multi-core CPUs.
- Atom-based data (e.g. coordinates, forces) moves back-and-forth between the CPU(s) and GPU every timestep.
- Neighbor lists can be constructed on the CPU or on the GPU.
- Asynchronous force computations can be performed simultaneously on the CPU(s) and GPU.
- OpenCL support.

USER-CUDA package:

- Designed to allow simulation to run entirely on the GPU (except for inter-processor MPI communication), so that atom-based data (e.g. coordinates, forces) do not have to move back-and-forth between the CPU and GPU.
- Speed-up advantage is better when the number of atoms per GPU is large .
- Data will stay on the GPU until a timestep where a non-GPU-ized fix or compute is invoked.
- Neighbor lists constructed on the GPU.
- Can use only one CPU (core) with each GPU.

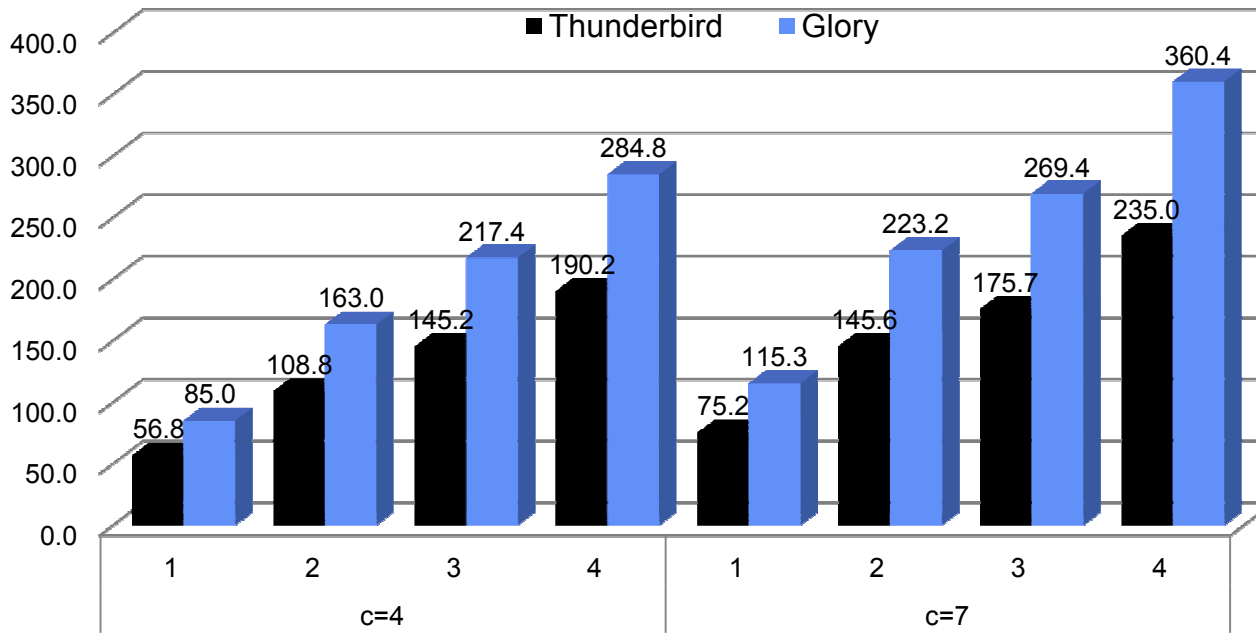
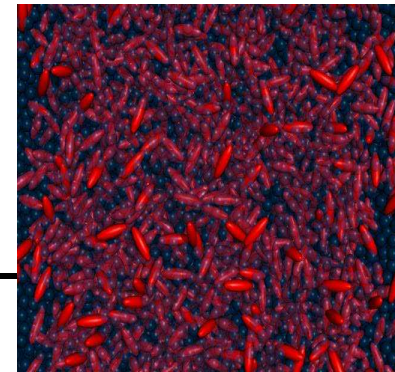


Force field coverage

- pairwise potentials: **Lennard-Jones, Buckingham, Morse**, Yukawa, soft, **class 2 (COMPASS)**, tabulated
- charged pairwise potentials: **Coulombic**, point-dipole
- manybody potentials: **EAM, Finnis/Sinclair EAM, modified EAM (MEAM), Stillinger-Weber, Tersoff**, AI-REBO, ReaxFF
- coarse-grained potentials: DPD, **GayBerne, REsquared**, colloidal, DLVO, **cg/cmm**
- mesoscopic potentials: **granular**, Peridynamics
- polymer potentials: all-atom, united-atom, bead-spring, breakable
- water potentials: **TIP3P**, TIP4P, **SPC**
- implicit solvent potentials: hydrodynamic lubrication, **Debye**
- long-range Coulombics and dispersion: Ewald, **PPPM (similar to particle-mesh Ewald)**, Ewald/N for long-range Lennard-Jones
- force-field compatibility with common **CHARMM**, AMBER, OPLS, **GROMACS** options

Gay-Berne potential

GPU Times Speedup vs 1 Core (c=cutoff, 32768 particles)



GPUs: 1, 2, 3, or 4 NVIDIA, 240 core, 1.3 GHz Tesla C1060 GPU(s)

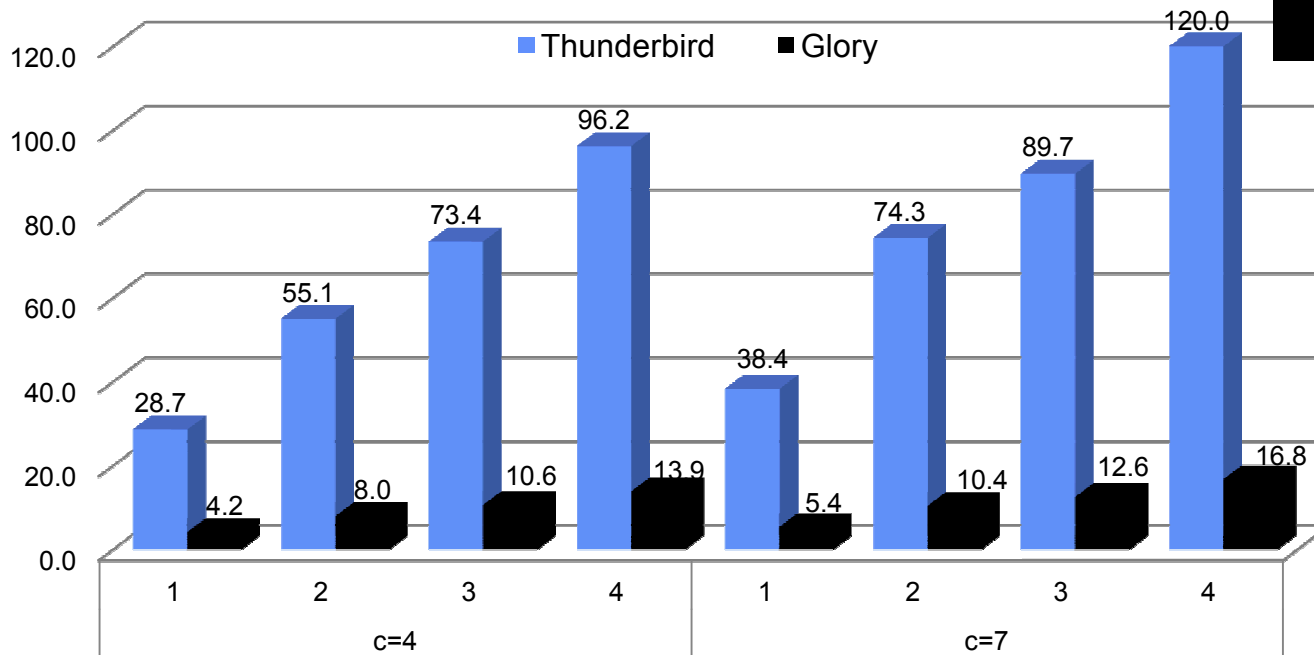
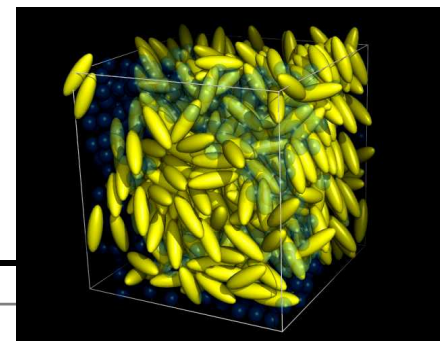
Thunderbird: 1 core of Dual 3.6 GHz Intel EM64T processors

Glory: 1 core of Quad Socket/Quad Core 2.2 GHz AMD

- Enables aspherical particle simulations
- ~30 times more expensive than LJ
- Particles in nature and manufacturing often have highly irregular shapes
- Used for liquid crystal simulations

Gay-Berne potential

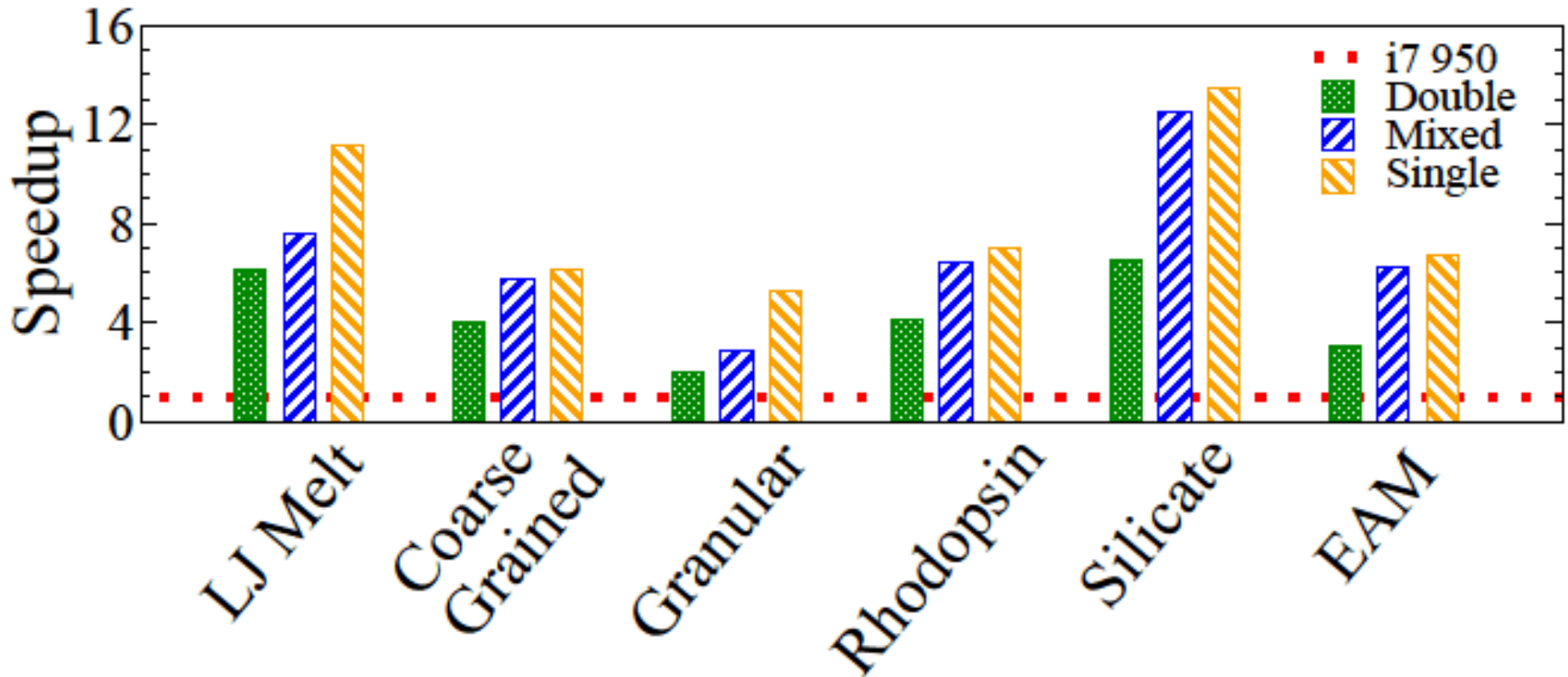
GPU Times Speedup vs 1 Node (c=cutoff, 32768 particles)



- Enables aspherical particle simulations
- ~30 times more expensive than LJ
- Particles in nature and manufacturing often have highly irregular shapes
- Used for liquid crystal simulations

GPU: 1, 2, 3, or NVIDIA, 240 core, 1.3 GHz Tesla C1060 GPU(s)
Thunderbird: 2 procs, Dual 3.6 GHz Intel EM64T processors
Glory: 16 procs, Quad Socket/Quad Core 2.2 GHz AMD

GPU speedups for several classes of materials



Typical speed-ups when using a single GTX 470 GPU versus a Quad-Core Intel i7 950 for various system classes.



Some lessons learned

1. Legacy codes can not be automatically “translated” to run efficiently on GPUs.
2. GPU memory management is tricky to optimize and strongly affects performance.
3. Host-device communication is a bottleneck.
4. Moving more of the calculation to the GPU improves performance, but requires more code conversion, verification, and maintenance.
5. Optimal algorithm on CPU is not necessarily the optimal algorithm on GPU or on a hybrid cluster (i.e. neighbor lists, long-range electrostatics).
6. Mixed- or single-precision is OK in some cases, and considerably faster. Preferable to give users the choice of precision.
7. Optimal performance requires simultaneous use of available CPUs, GPUs, and communication resources.



Remaining challenges

1. **Funding limitations**
2. **Merging of disparate efforts and “visions”**
3. **Better coverage: port more code (FFs, other) to GPU**
4. **Better performance (better algorithms & code)**
5. **Quality control (code correctness and readability)**
6. **Code maintenance**
 - a. **Synching with the rest of LAMMPS**
 - b. **User support**
 - c. **Bug fixes**



Future areas of LAMMPS development

1. Alleviations of time-scale and spatial-scale limitations
 2. Improved force fields for better molecular physics
 3. New features for the convenience of users
- **LAMMPS now has a rapidly-growing OpenMP capability (efficient harnessing of multi-core CPUs) in addition to the GPU capabilities discussed here.**
 - **We are currently developing a new long-range capability for LAMMPS based on the multi-summation method (MSM). This is an example of development of new algorithms that better suit next-generation HPC.**