Introduction to LAMMPS

Stan Moore

SC22 Student Cluster Competition
About Me

- Stan Moore
  - One of the LAMMPS code developers at Sandia National Laboratories in Albuquerque, New Mexico
  - Been at Sandia for 10 years
  - Main developer of the KOKKOS package in LAMMPS (runs on GPUs and multi-core CPUs)
  - Expertise in long-range electrostatics
  - PhD in Chemical Engineering, dissertation on molecular dynamics method development for predicting chemical potential
Molecular Dynamics

- Classical molecular dynamics (MD) models atom behavior using Newton’s laws of motions
- Normally use an empirical expression for forces (no explicit electrons)
- Atom positions $\rightarrow$ forces $\rightarrow$ velocities $\rightarrow$ new positions
- Spherical cutoff gives $O(N)$ linear scaling, can simulate billions of atoms on a supercomputer
LAMMPS Overview

- Large-scale Atomic/Molecular Massively Parallel Simulator
- https://lammps.org
  - Open source, C++ code
  - Bio, materials, mesoscale
  - Particle simulator at varying length and time scales
    - Electrons → atomistic → coarse-grained → continuum
  - Spatial-decomposition of simulation domain for parallelism
  - Energy minimization, dynamics, non-equilibrium MD
  - GPU and OpenMP enhanced
  - Can be coupled to other scales: QM, kMC, FE, CFD, …
Statistical Mechanics Basics

- **Statistical Mechanics:** relates macroscopic observations (such as temperature and pressure) to microscopic states (i.e. atoms)

- **Phase space:** a space in which all possible states of a system are represented. For $N$ particles: $6N$-dimensional phase space (3 position variables and 3 momentum variables for each particle)

- **Ensemble:** an idealization consisting of a large number of virtual copies of a system, considered all at once, each of which represents a possible state that the real system might be in, i.e. a probability distribution for the state of the system
Statistical Mechanics Basics

- **Statistical Mechanics**: relates macroscopic observations (such as temperature and pressure) to microscopic states (i.e. atoms)

- **Phase space**: a space in which all possible states of a system are represented. For $N$ particles: $6N$-dimensional phase space (3 position variables and 3 momentum variables for each particle)

- **Ensemble**: an idealization consisting of a large number of virtual copies of a system, considered all at once, each of which represents a possible state that the real system might be in, i.e. a probability distribution for the state of the system
Molecular Dynamics: What is it?

Mathematical Formulation
- Classical Mechanics
- Atoms are Point Masses: $r_1, r_2, \ldots, r_N$
- Positions, Velocities, Forces: $r_i, v_i, F_i$
- Potential Energy Function = $V(r^N)$
- $6N$ coupled ODEs

Newton’s Equations:

\[
\frac{dr_i}{dt} = v_i \\
\frac{dv_i}{dt} = \frac{F_i}{m_i} \\
F_i = -\frac{d}{dr_i} V(r^N)
\]
MD Versatility

Coupling to Solid Mechanics

Biophysics

Granular Flow

Materials Science

Chemistry
MD Time & Length Scales

- Quantum mechanical electronic structure calculations (QM) provide accurate description of mechanical and chemical changes on the atom-scale, but limited to ~1000 atoms
- Atom-scale phenomena drive a lot of interesting physics, chemistry, materials science, mechanics, biology...but it usually plays out on a much larger scale
- Mesoscale: much bigger than an atom, much smaller than a glass of soda
- QM and continuum/mesoscale models (CM) can not be directly compared—large scale MD can bridge gap

MD Basics

- Atoms can be modeled as points (most common), finite-size spheres, or other shapes (e.g. ellipsoids)
- Can model atomic-scale (all-atom model) or meso/continuum scale with MD-like models
- Typically use an orthogonal or triclinic (skewed) simulation cell
- Commonly use periodic boundary conditions: reduces finite size effects from boundaries and simulates bulk conditions

2D Triclinic
Simple Example: Crack
Interatomic Potentials

- Quantum chemistry: solves Schrödinger equation (electron interactions) to get forces on atoms. Accurate but very computationally expensive and only feasible for small systems: ~1000 atoms
- Molecular dynamics: uses empirical force fields, sometimes fit to quantum data. Not as accurate but much faster
- MD typically only considers pair-wise or three-body interactions, scales as $O(N)$ (billion atom simulations are considered huge)

Interatomic Potential

![Lennard-Jones Potential](image)

- Repulsive wall
- Attractive tail
Accuracy = Higher Cost

Moore’s Law for Interatomic Potentials
Plimpton and Thompson, MRS Bulletin (2012).
ReaxFF Potential

- Reactive model that captures bond breaking and formation via bond order parameter
- Many complex equations required to capture physics
- Includes charge equilibration model (QEq)

\[ BO_{ij} = \exp \left[ \rho_\sigma \left( \frac{r_{ij}}{r^0_\sigma} \right)^q \right] + \exp \left[ \rho_\pi \left( \frac{r_{ij}}{r^0_\pi} \right)^q \right] + \exp \left[ \rho_{\pi\pi} \left( \frac{r_{ij}}{r^0_{\pi\pi}} \right)^q \right] \]

\[ E_{\text{system}} = E_{\text{bond}} + E_{\text{lone pair}} + E_{\text{over coord}} + E_{\text{under coord}} + E_{\text{valence angle}} + E_{\text{torsion}} + E_{\text{conjugation}} + E_{\text{H Bonds}} + E_{\text{vdW}} + E_{\text{coulomb}} \]

SNAP Machine Learning Potential

- ML interatomic potential (IAP) have three critical parts:
  - Descriptors of the local environment
  - Energy and force functions expressed in the descriptors
  - Training (regression method) on large amount of ‘ground truth’ energies and forces
- Demonstrated *ab initio* accuracy in classical MD!

SNAP Performance on V100 GPU

~32x speedup in 3 years!
Neighbor Lists

- Neighbor lists are a list of neighboring atoms within the interaction cutoff + skin for each central atom
- Extra skin allows lists to be built less often
Newton Option

- Newton flag to *off* means that if two interacting atoms are on different processors, both processors compute their interaction and the resulting force information is not communicated.
- Setting the newton flag to *on* saves computation but increases communication.
- Performance depends on problem size, force cutoff lengths, a machine’s compute/communication ratio, and how many processors are being used.
- Newton off typically better for GPUs.

```plaintext
newton on
newton off
```
Half Neighbor List

- With newton flag on, each pair is stored only once (usually better for CPUs), requires atomic operations for thread-safety
Full Neighbor List

- Each pair stored twice which doubles computation but reduces communication and doesn’t require atomic operations for thread safety (can be faster on GPUs)
MPI Parallelization Approach

- Domain decomposition: each processor owns a portion of the simulation domain and atoms therein.
Ghost Atoms

- The processor domain is also extended to include needed ghost atoms (copies of atoms located on other processors).
- Communicated via MPI (message passing interface).
Load-balancing

- **Balance** (static) and **fix balance** (dynamic) commands
- “shift” style operates by adjusting planar cuts between processors
- Works well for 1D density variations
  - solid/gas or liquid/gas interfaces
- Less well for general 2D/3D variations
2D and 3D Load-balancing

- “rcb” style is a tiling method, works better for 2D and 3D variations
Molecular Topology

- Bonds: constrained length between two atoms
- Angles: constrained angle between three atoms
- Dihedrals: interactions between quadruplets of atoms
- Impropers: “improper” interactions between quadruplets of atoms

```plaintext
bond_style              harmonic
angle_style             charmm
dihedral_style           charmm
improper_style           harmonic
```
Long-Range Electrostatics

- Truncation doesn’t work well for charged systems due to long-ranged nature of Coulombic interactions
- Use Kspace style to add long-range electrostatics. PPPM method usually fastest, uses FFTs
- Specify a relative accuracy (i.e. 1e-4)
- Use `pair_style *coul/long` such as `lj/cut/coul/long` instead of `*coul/cut`
- Can vary Coulomb cutoff length and get the same answer

```
pair_style       lj/cut/coul/long 10.0
kspace_style    pppm 1e-4
```
Basic MD Timestep

- During each timestep (without neighborlist build):

  1. Initial integrate
  2. MPI communication
  3. Compute forces (pair, bonds, kspace, etc.)
  4. Additional MPI communication (if newton flag on)
  5. Final integrate
  6. Output (if requested on this timestep)

*Computation of diagnostics (fixes or computes) can be scattered throughout the timestep*
LAMMPS Files

- **Input file**: text file with LAMMPS commands used to run a simulation
- **Log file**: text file with thermodynamic output from simulation
- **Dump file**: snapshot of atom properties, i.e. forces
- **Restart file**: binary checkpoint file with data needed to restart simulation
- **Data file**: text file that can be used to start or restart simulation
Downloading LAMMPS

- Github ([https://github.com/lammps/lammps](https://github.com/lammps/lammps))
  - [https://github.com/lammps/lammps/releases](https://github.com/lammps/lammps/releases)
  - Clone or download button, then download zip file
  - `git clone ...` (beyond this tutorial)

- LAMMPS Website ([https://lammps.org](https://lammps.org))
  - Go to “download” link
  - Download gzipped tar file

- **Stable version**: rigorous testing
- **Development version**: still tested but not as rigorous, latest features, performance optimizations, and bug fixes (but could also have new bugs)
Compiling LAMMPS

- [https://docs.lammps.org/Build.html](https://docs.lammps.org/Build.html)
- Need C++11 compiler (GNU, Intel, Clang)
- Need MPI library, or can use the “STUBS” library
- Many Makefiles in src/MAKE
- LAMMPS also has CMake interface
Running LAMMPS

- [executable] -in [input_script]

In serial:

  ./lmp_serial -in in.lj

In parallel:

  mpirun -np 2 lmp_mpi -in in.lj

Many other command line options, see [https://docs.lammps.org/Run_options.html](https://docs.lammps.org/Run_options.html)
Optional Packages

- [https://docs.lammps.org/Packages_list.html](https://docs.lammps.org/Packages_list.html)
- LAMMPS is very modular and has several optional packages
- For example, SNAP potential needs ML-SNAP package installed

Traditional Make:

```
make yes-ml-snap
make no-ml-snap
```

CMAKE:

```
-D PKG_ML-SNAP=yes
```
Accelerator Packages

- [https://docs.lammps.org/Speed_packages.html](https://docs.lammps.org/Speed_packages.html)
- Some hardware components like GPUs, and multithreaded CPUs require special code (i.e. OpenMP, CUDA) to fully take advantage of the hardware
- LAMMPS has 5 accelerator packages:
  - OPENMP
  - INTEL
  - OPT
  - GPU
  - KOKKOS
Running OPT Package

- Compile LAMMPS with OPT package
- Run with 8 MPI: `mpiexec -np 8 ./lmp_exe -in in.lj -sf opt`
- `-sf opt` is the `suffix` command: automatically appends `/opt` onto anything it can
- For example, `pair_style lj/cut` automatically becomes `pair_style lj/cut/opt` (no changes to input file needed)

https://docs.lammps.org/suffix.html
OPENMP Package

- https://docs.lammps.org/Speed_omp.html
- Uses OpenMP to enable multithreading on CPUs
- MPI parallelization in LAMMPS is almost always more effective than OpenMP on CPUs
- When running with MPI across multi-core nodes, MPI often suffers from communication bottlenecks, so using MPI+OpenMP per node could be faster
- The more nodes per job and the more cores per node, the more pronounced the bottleneck and the larger the benefit from MPI+OpenMP
- OPENMP package may vectorize (SIMD) better than vanilla code
Running OPENMP Package

- Compile LAMMPS with OPENMP package
- Run with 2 MPI and 2 OpenMP threads:

```bash
export OMP_NUM_THREADS=2
mpiexec -np 2 ./lmp_exe -in in.lj -sf omp
```
INTEL Package

- [https://docs.lammps.org/Speed_intel.html](https://docs.lammps.org/Speed_intel.html)
- Allows code to vectorize and run well on Intel CPUs (with or without OpenMP threading)
- Can also be used in conjunction with the OPENMP package
- Normally best performance out of all accelerator packages for CPUs
- Supports reduced precision: mixed FP64+FP32 or pure single FP32
Running INTEL Package

- Compile LAMMPS with INTEL package
- To run using 2 MPI and 2 threads on a Intel CPU:

  mpiexec -np 2 ./lmp_exe -in in.lj -pk intel 0 omp 2 mode double -sf intel

- \texttt{-pk} is the package command that sets package options, see [https://docs.lammps.org/package.html](https://docs.lammps.org/package.html)
GPU Package

- [https://docs.lammps.org/Speed_gpu.html](https://docs.lammps.org/Speed_gpu.html)
- Designed for one or more GPUs coupled to many CPU cores
- Only pair runs on GPU, fixes/bonds/computes run on CPU
- Atom-based data (e.g. coordinates, forces) move back and forth between the CPU(s) and GPU every timestep
- Asynchronous force computations can be performed simultaneously on the CPU(s) and GPU if using Kspace
- Provides NVIDIA and more general OpenCL support
- Supports reduced precision: mixed FP64+FP32 or pure single FP32
Running GPU Package

- Compile GPU library found in lib/gpu
- Compile LAMMPS with GPU package
- Run with 16 MPI and 4 GPUs: mpiexec -np 16 ./lmp_exe -in in.lj -sf gpu -pk gpu 4
- Best to use CUDA MPS (Multi-Process Service) if using multiple MPI ranks per GPU
- Automatically overlaps pair-style on GPU with Kspace on CPU
Kokkos

- Abstraction layer between programmer and next-generation platforms
- Allows the same C++ code to run on multiple hardware (GPU, CPU)
- Kokkos consists of two main parts:
  1. Parallel dispatch—threaded kernels are launched and mapped onto backend languages such as CUDA or OpenMP
  2. Kokkos views—polymorphic memory layouts that can be optimized for a specific hardware
- Used on top of existing MPI parallelization (MPI + X)
- See https://kokkos.github.io/kokkos-core-wiki for more info
LAMMPS KOKKOS Package

- [https://docs.lammps.org/Speed_kokkos.html](https://docs.lammps.org/Speed_kokkos.html)
- **Need C++14 compiler**
- Supports OpenMP and GPUs
- Designed so that everything (pair, fixes, computes, etc.) runs on the GPU, minimal data transfer from GPU to CPU
- GPU performance penalty if kernel isn’t ported to Kokkos
- Only double precision FP64 support
- Package options can toggle full and half neighbor list, newton on/off, etc.
  
  `-pk kokkos newton on neigh half`

- [https://docs.lammps.org/package.html](https://docs.lammps.org/package.html)
Running Kokkos Package

- Compile LAMMPS with the KOKKOS package
- Run with 4 MPI and 4 GPUs: `mpiexec -np 4 ./lmp_exe -in in.lj -k on g 4 -sf kk`
- Run with 4 OpenMP threads: `./lmp_exe -in in.lj -k on t 4 -sf kk`
Processor and Thread Affinity

- Use `mpirun` command-line arguments (e.g. `--bind-to core`) to control how MPI tasks and threads are assigned to nodes and cores.
- Also use OpenMP variables such as `OMP_PROC_BIND` and `OMP_PLACES`.
- One must also pay attention to NUMA bindings between tasks, cores, and GPUs. For example, for a dual-socket system, MPI tasks driving GPUs should be on the same socket as the GPU.
Lennard-Jones Benchmark

- lammps/bench/in.lj
- Simple pair-wise model
- Similar to argon liquid/gas
For KOKKOS package on GPUs, timing breakdown won’t be accurate without CUDA_LAUNCH_BLOCKING=1 (but will prevent kernel overlap and could slow down simulation)

Measuring performance

Loop time of 0.0174524 on 640 procs for 100 steps with 32000 atoms

Performance: 2475308.243 tau/day, 5729.880 timesteps/s
94.1% CPU use with 640 MPI tasks x no OpenMP threads

MPI task timing breakdown:

<table>
<thead>
<tr>
<th>Section</th>
<th>min time</th>
<th>avg time</th>
<th>max time</th>
<th>%varavg</th>
<th>%total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair</td>
<td>0.0010798</td>
<td>0.0013214</td>
<td>0.0016188</td>
<td>0.2</td>
<td>7.57</td>
</tr>
<tr>
<td>Neigh</td>
<td>0.00021591</td>
<td>0.00024434</td>
<td>0.0003079</td>
<td>0.0</td>
<td>1.40</td>
</tr>
<tr>
<td>Comm</td>
<td>0.015171</td>
<td>0.015479</td>
<td>0.01573</td>
<td>0.1</td>
<td>88.69</td>
</tr>
<tr>
<td>Output</td>
<td>9.0258e-05</td>
<td>0.00011218</td>
<td>0.00014501</td>
<td>0.0</td>
<td>0.64</td>
</tr>
<tr>
<td>Modify</td>
<td>0.00017915</td>
<td>0.00018453</td>
<td>0.00020567</td>
<td>0.0</td>
<td>1.06</td>
</tr>
<tr>
<td>Other</td>
<td>0.0001111</td>
<td></td>
<td></td>
<td></td>
<td>0.64</td>
</tr>
</tbody>
</table>
Performance of Different Potentials

LJ: single node

EAM: single node

ReaxFF HNS: single node

SNAP: single node
Parallel MD Performance

- MD parallelizes well: major parts of timestep (forces, neighbor list build, time integration) can be done in parallel through domain decomposition
- High communication overhead when strong scaling to a few 100 atoms/proc (depends on cost of the force-field)
- **Strong scaling**: hold system size fixed while increasing processor count (# of atoms/processor decreases)
- **Weak scaling**: increase system size in proportion to increasing processor count (# of atoms/processor remains constant)
- For perfect strong scaling, doubling the processor count cuts the simulation time in half
- For perfect weak scaling, the simulation time stays exactly the same when doubling the processor count
- Harder to maintain parallel efficiency with strong scaling because the compute time decreases relative to the communication time
Visualization Resources

- LAMMPS “dump image” command: https://docs.lammps.org/dump_image.html
- VMD: https://www.ks.uiuc.edu/Research/vmd/
- OVITO: https://www.ovito.org/about/ovito-pro/
Getting Help

- Look at LAMMPS documentation, latest version here: https://docs.lammps.org/Manual.html
- Search the MatSci LAMMPS forum archives https://matsci.org/lammps or join and post new questions
- LAMMPS reference paper: gives an overview of the code including its parallel algorithms, design features, performance, and brief highlights of many of its materials modeling capabilities https://doi.org/10.1016/j.cpc.2021.108171
Thank You

Questions?