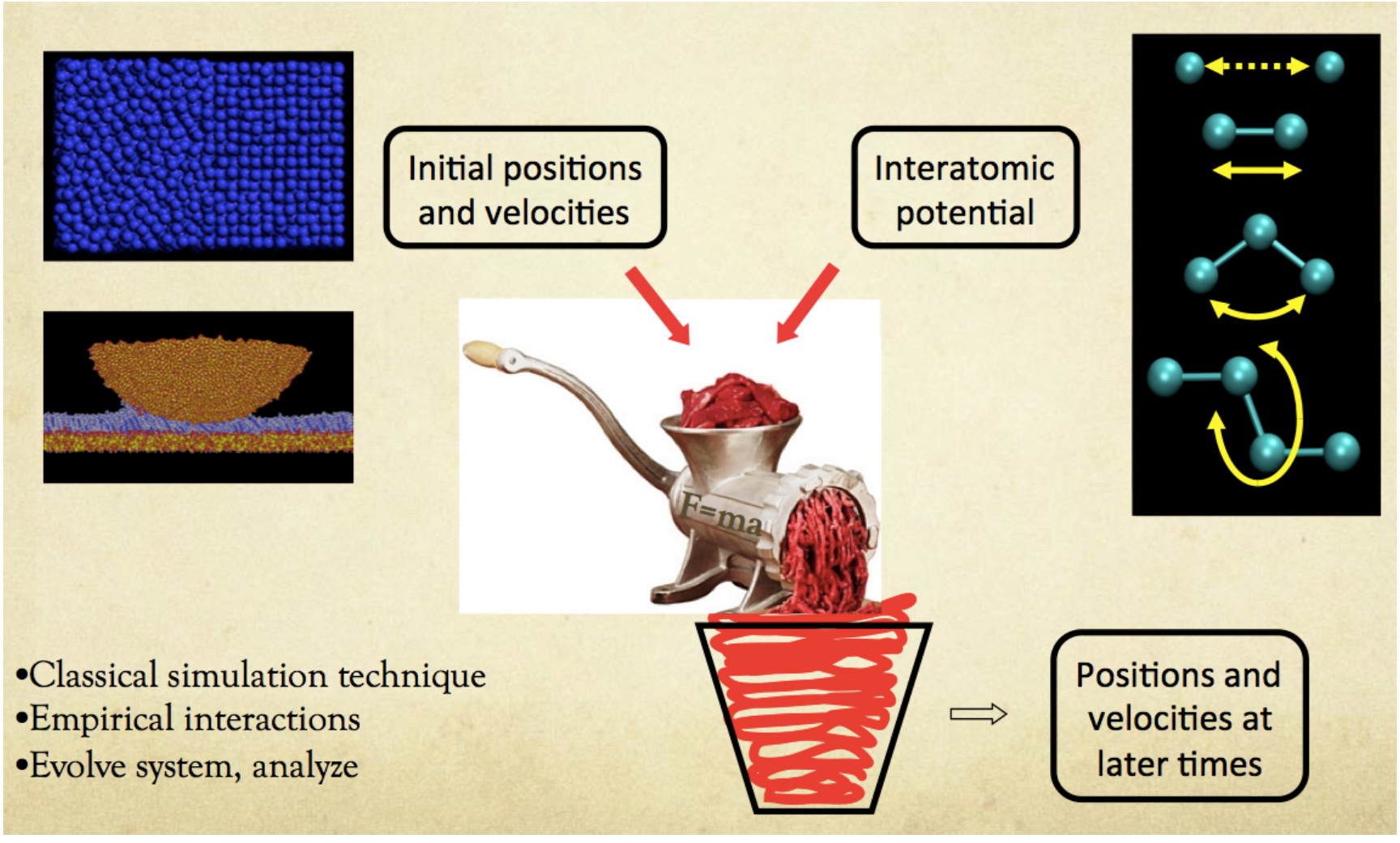
LAMMPS Features and Capabilities

Steve Plimpton
Sandia National Labs
sjplimp@sandia.gov

LAMMPS Users and Developers Workshop
International Centre for Theoretical Physics (ICTP)
March 2014 - Trieste, Italy

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
Presentation: SAND2014-2239C

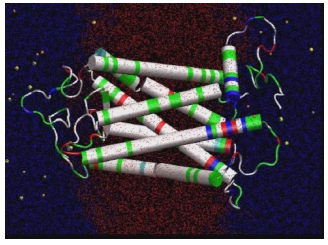
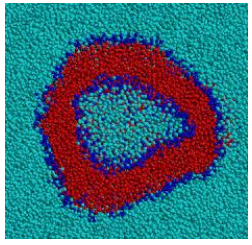
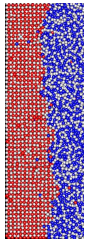
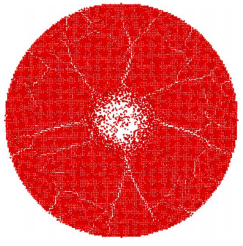
Classical MD in a nutshell



LAMMPS from 10,000 meters

Large-scale Atomic/Molecular Massively Parallel Simulator
http://lammps.sandia.gov

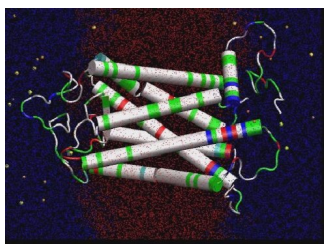
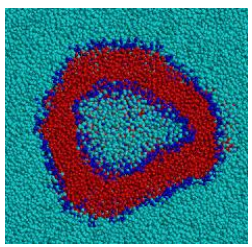
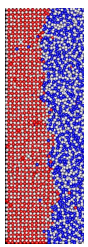
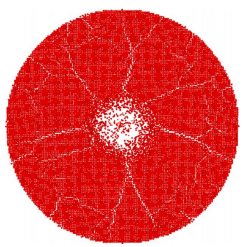
- Classical MD code
- Open source, portable C++
- 3-legged stool: soft matter, solids, mesoscale

LAMMPS from 10,000 meters

Large-scale Atomic/Molecular Massively Parallel Simulator
http://lammps.sandia.gov

- Classical MD code
- Open source, portable C++
- 3-legged stool: soft matter, solids, mesoscale

- Particle simulator at varying length and time scales
  electrons $\Rightarrow$ atomistic $\Rightarrow$ coarse-grained $\Rightarrow$ continuum
- Spatial-decomposition of simulation domain for parallelism
- MD, non-equilibrium MD, energy minimization
- GPU and OpenMP enhanced
- Can be coupled to other scales: QM, kMC, FE, CFD, ...

Reasons to use LAMMPS

1. Versatile
   - bio, materials, mesoscale
   - atomistic, coarse-grained, continuum
   - use with other codes, e.g. multiscale models
2. Good parallel performance
3. Easy to extend
   - Tuesday AM - Modifying & Extending LAMMPS
   - Wednesday PM - Hands-on: Writing new code for LAMMPS
4. Well documented
   - extensive web site
   - 1300 page manual
5. Active and supportive user community
   - 45K postings to mail list, 1500 subscribers
   - quick turn-around on Qs posted to mail list

Resources for learning LAMMPS

- Examples: about 35 sub-dirs under examples in distro
- Manual: doc/Manual.html
  - Intro, Commands, Packages, Accelerating
  - Howto, Modifying, Errors
- Alphabetized command list: one doc page per command
  - doc/Section_commands.html 3.5
- Web site: http://lammps.sandia.gov
  - Pictures, Movies - examples of others work
  - Papers - find a paper similar to what you want to model
  - Workshops - slides from LAMMPS simulation talks
- Mail list: search it, post to it
  - http://lammps.sandia.gov/mail.html
- These slides (more info than I can probably present!)

Structure of typical input scripts

1. Units and atom style
2. Create simulation box and atoms
   - region, create_box, create_atoms, region commands
     - lattice command vs box units
   - read_data command
     - data file is a text file
     - look at examples/micelle/data.micelle
     - see read_data doc page for full syntax
3. Define groups
4. Set attributes of atoms: mass, velocity
5. Pair style for atom interactions
6. Fixes for time integration and constraints
7. Computes for diagnostics
8. Output: thermo, dump, restart
9. Run or minimize
10. Rinse and repeat (script executed one command at a time)

Debugging an input script

LAMMPS tries hard to flag many kinds of errors and warnings

1. If an input command generates the error ...
   - % lmp_linux -echo screen < in.polymer
   - re-read the doc page for the command
2. For input, setup, run-time errors ...
   - search doc/Section_errors.html for text of error message
   - also for warnings, they are usually important
   - if specific input command causes problems, look for IMPORTANT NOTE info on doc page
   - look in the source code file at the line number
3. Search the mail list, others may have similar problem
   - google for: lammps-users fix npt, or error message

Debugging an input script

LAMMPS tries hard to flag many kinds of errors and warnings

1. If an input command generates the error ...
   - % lmp_linux -echo screen < in.polymer
   - re-read the doc page for the command
2. For input, setup, run-time errors ...
   - search doc/Section_errors.html for text of error message
   - also for warnings, they are usually important
   - if specific input command causes problems, look for IMPORTANT NOTE info on doc page
   - look in the source code file at the line number
3. Search the mail list, others may have similar problem
   - google for: lammps-users fix npt, or error message
4. Remember: an input script is like a program
   - start with small systems
   - start with one processor
   - turn-on complexity one command at a time
   - monitor thermo output, viz the results (use dump image)

Debug by examining screen output

```
LAMMPS (15 Aug 2013)
Lattice spacing in x,y,z = 1.28436 2.22457 1.28436
Created orthogonal box = (0 0 -0.321089)
            to (51.3743 22.2457 0.321089)
  4 by 1 by 1 MPI processor grid
Created 840 atoms
120 atoms in group lower
120 atoms in group upper
240 atoms in group boundary
600 atoms in group flow
Setting atom values ...
  120 settings made for type
Setting atom values ...
  120 settings made for type
Deleted 36 atoms, new total = 804
Deleted 35 atoms, new total = 769
```

Thermodynamic output

Look for blow-ups or NaNs, print every step if necessary

```
WARNING: Temperature for thermo pressure is not
         for group all (../thermo.cpp:436)
Setting up run ...
Memory usage per processor = 2.23494 Mbytes
Step Temp E_pair E_mol TotEng Press Volume
0 1.0004177 0 0 0.68689281 0.46210058 1143.0857
1000 1 -0.32494012 0 0.36166587 1.2240503 1282.5239
2000 1 -0.37815616 0 0.30844982 1.0642877 1312.5691
...
...
...
25000 1 -0.36649381 0 0.32011217 0.98366691 1451.5444
25000 1 -0.38890426 0 0.29770172 0.95284427 1455.9361
Loop time of 1.76555 on 4 procs for
         25000 steps with 769 atoms
```

Timing info

```
Loop time of 1.76555 on 4 procs for
        25000 steps with 769 atoms

Pair time (%) = 0.14617 (8.27903)
Neigh time (%) = 0.0467809 (2.64966)
Comm time (%) = 0.307951 (17.4422)
Outpt time (%) = 0.674575 (38.2078)
Other time (%) = 0.590069 (33.4213)
```

Run statistics

Per-processor values at end of run

```
Nlocal:  192.25 ave 242 max 159 min
Histogram:  2 0 0 0 0 1 0 0 0 1
Nghost:  43 ave 45 max 39 min
Histogram:  1 0 0 0 0 0 0 0 2 1
Neighs:  414 ave 588 max 284 min
Histogram:  2 0 0 0 0 0 1 0 0 1

Total # of neighbors = 1656
Ave neighs/atom = 2.15345
Neighbor list builds = 1641
Dangerous builds = 1
```

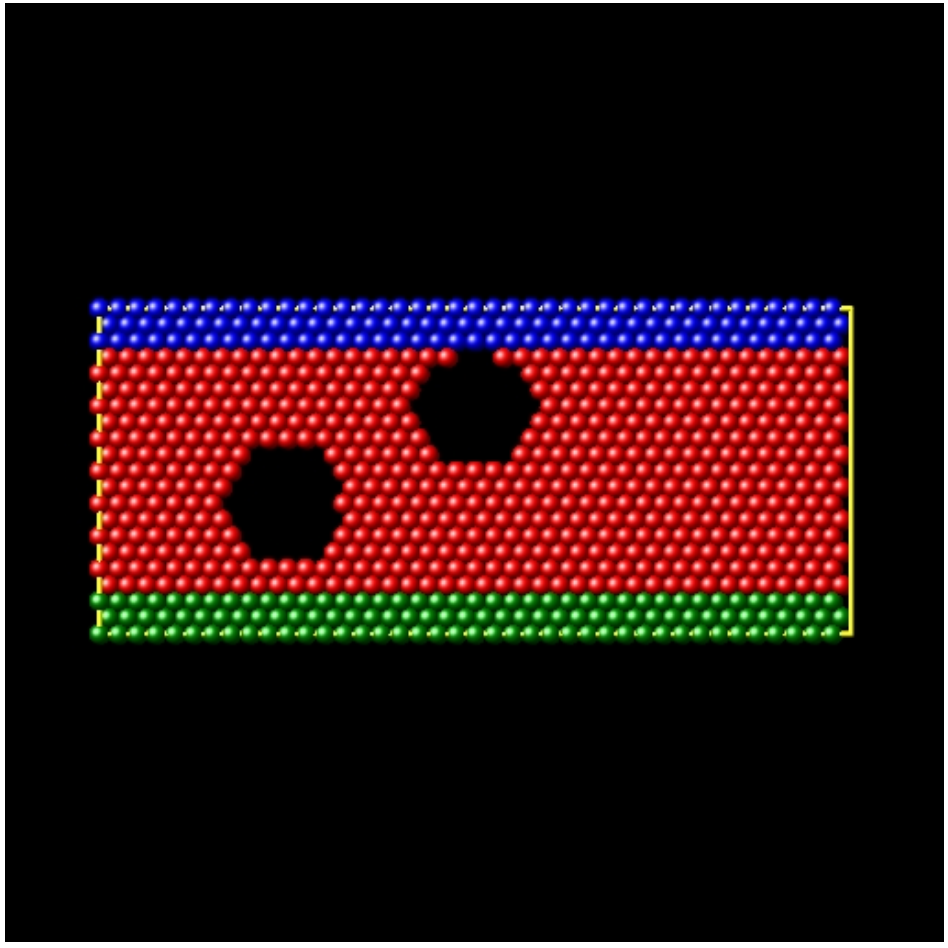
Debug by visualization - what does your system do?

Dump image for instant JPGs

- image.16500.jpg
- ImageMagick display
- Mac Preview

Make/view a movie

- ImageMagick
  convert *.jpg image.gif
- open in browser
  open -a Safari image.gif
- Mac QuickTime
  open image sequence
- Windows Media Player
- VMD, AtomEye, ...



Defining variables in input scripts

- Styles: index, loop, equal, atom, ...
  - variable x index run1 run2 run3 run4
  - variable x loop 100
  - variable x equal trap(f_JJ[3])*${scale}
  - variable x atom -(c_p[1]+c_p[2]+c_p[3])/(3*vol)

Defining variables in input scripts

- Styles: index, loop, equal, atom, ...
  - variable x index run1 run2 run3 run4
  - variable x loop 100
  - variable x equal trap(f_JJ[3])*${scale}
  - variable x atom -(c_p[1]+c_p[2]+c_p[3])/(3*vol)
- Formulas can be complex
  - see doc/variable.html
  - thermo keywords (temp, press, ...)
  - math operators & functions (sqrt, log, cos, ...)
  - group and region functions (count, xcm, fcm, ...)
  - various special functions (min, ave, trap, stride, stagger, ...)
  - per-atom vectors (x, vx, fx, ...)
  - output from computes, fixes, other variables
- Formulas can be time- and/or spatially-dependent

Using variables in input scripts

- Substitute in any command via $x or ${myVar}
- Can define them as command-line arguments
  - % lmp_linux -v myTemp 350.0 < in.polymer
- Use in next command to increment a variable
  - with jump command to create loops
- Many commands allow them as arguments
  - fix addforce 0.0 v_fy 1.0
  - dump_modify every v_count
  - region sphere 0.0 0.0 0.0 v_radius
- Allows time- and spatially-dependent commands

Power tools for input scripts

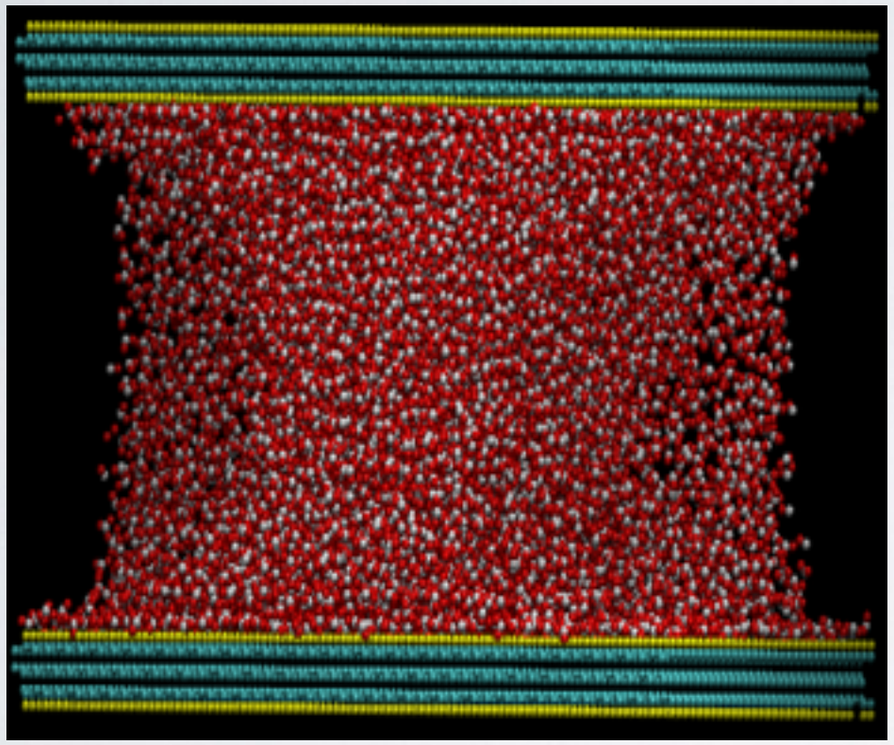
- Filename options:
  - dump.*.% for per-snapshot or per-processor output
  - read_data data.protein.gz
  - read_restart old.restart.*
- If/then/else via if command
- Insert another script via include command
  - useful for long list of parameters

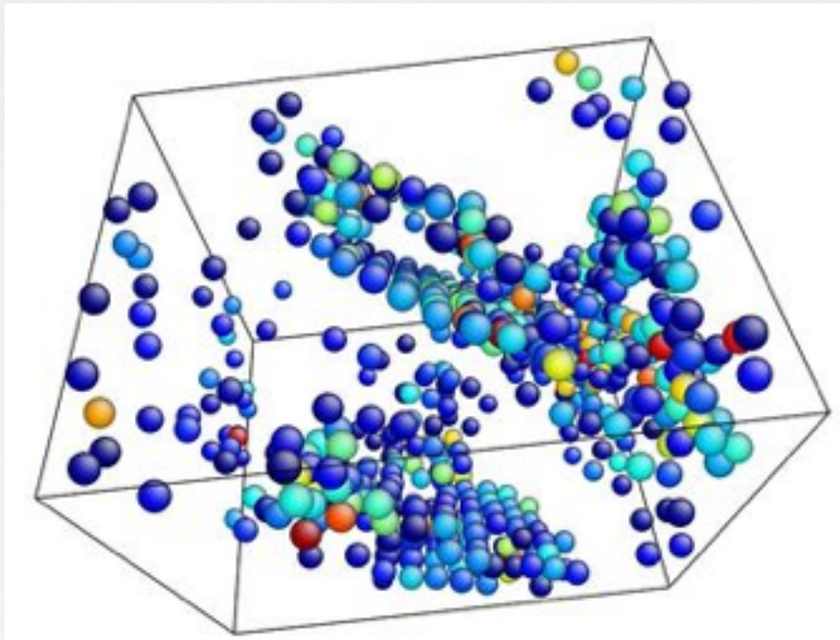
Power tools for input scripts

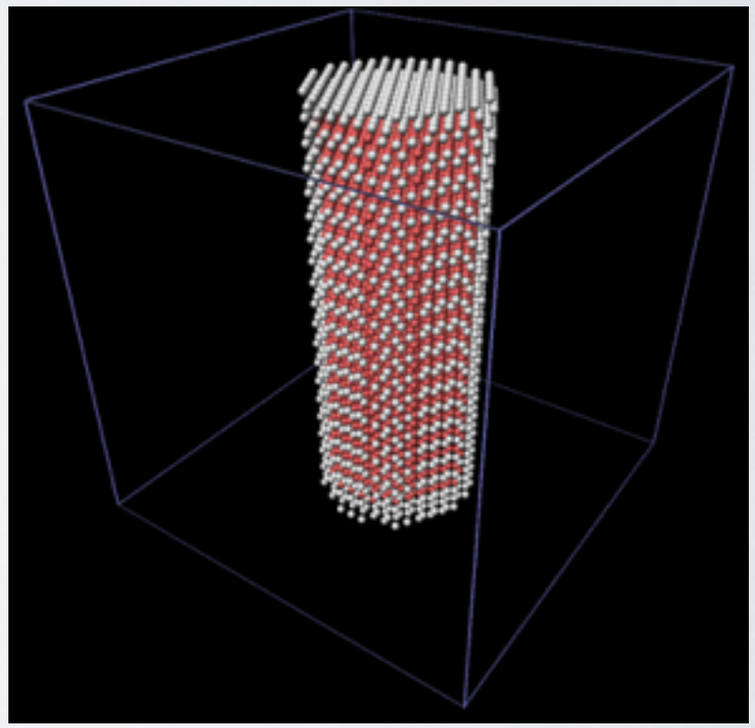
- Filename options:
  - dump.*.% for per-snapshot or per-processor output
  - read_data data.protein.gz
  - read_restart old.restart.*
- If/then/else via if command
- Insert another script via include command
  - useful for long list of parameters
- Looping via next and jump commands
  - easy to run incrementally and stop when condition met
  - see examples on jump command doc page
- Invoke a shell command or external program
  - shell cd subdir1
  - shell my_analyze out.file $n ${param}
- Various ways to run multiple simulations from one script
  - see Section_howto 6.4 of manual

Elements of ICME Research Workshop

Molecular Dynamics with LAMMPS







Elements of ICME Research Workshop
UIUC
July 23-25, 2014

Andrew L. Ferguson
Materials Science and Engineering
University of Illinois at Urbana-Champaign

LAMMPS

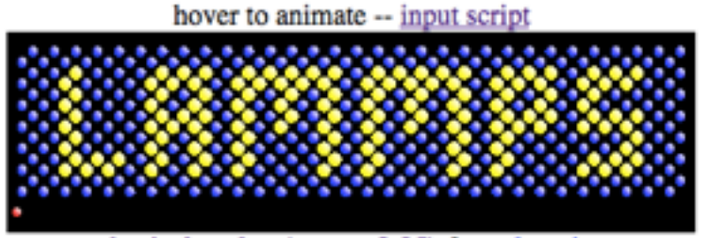
http://lammps.sandia.gov

Large-scale **A**tomic/**M**olecular **M**assively **P**arallel **S**imulator

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	Workshops
FAQ	Latest Features & Bug Fixes	Tutorials	Movies	Offsite LAMMPS packages & tools	Funding	User Scripts and HowTos
Wish list	Unfixed bugs	MD to LAMMPS glossary	Benchmarks	Visualization	Open source	Contribute to LAMMPS
.	.	Commands	Citing LAMMPS	Related Modeling codes	.	.

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

LAMMPS has potentials for solid-state materials (metals, semiconductors) and soft matter (biomolecules, polymers) 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 is distributed by Sandia 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 LAMMPS development has come primarily from DOE (OASCR, OBER, ASCI, LDRD, Genomes-to-Life) and is acknowledged here.

The LAMMPS WWW site is hosted by Sandia, which has this Privacy and Security statement.

History

- Born mid-90's in cooperation between Sandia, LLNL, Cray, Bristol Meyers Squibb, and Dupont — now developed at Sandia under DOE funding

- Current release in C++ w/ MPI

- **Open source and free under GPL**

- *Platforms:* Linux, Mac, Windows

- *Format:* exe, RPM, PPA, SVN, Git, Homebrew, tarball

Usability

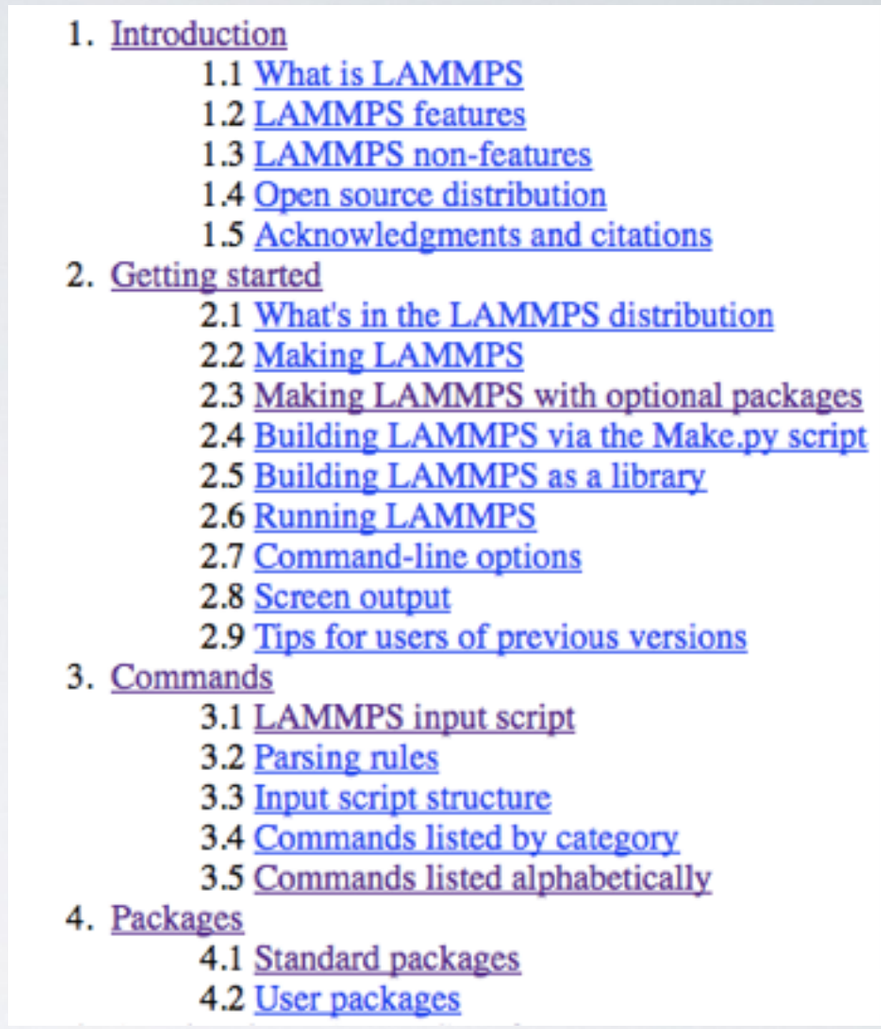
- Run initialization and control via **input script**
- Call from command line as `./lmp_linux < in.comp`
- **No GUI**, but some python tools available (http://lammps.sandia.gov/doc/Section_python.html)

Al_fcc.in

```
# ---------- Initialize Simulation --------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms ---------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential ---------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings ---------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options ---------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization ---------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

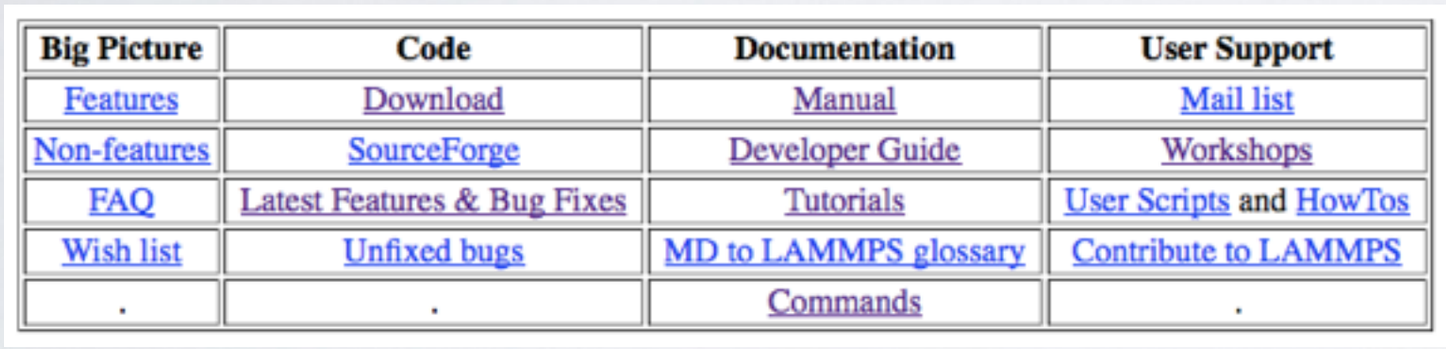
Documentation

- Excellent manual (http://lammps.sandia.gov/doc/Manual.html)



1. Introduction
   - 1.1 What is LAMMPS
   - 1.2 LAMMPS features
   - 1.3 LAMMPS non-features
   - 1.4 Open source distribution
   - 1.5 Acknowledgments and citations
2. Getting started
   - 2.1 What's in the LAMMPS distribution
   - 2.2 Making LAMMPS
   - 2.3 Making LAMMPS with optional packages
   - 2.4 Building LAMMPS via the Make.py script
   - 2.5 Building LAMMPS as a library
   - 2.6 Running LAMMPS
   - 2.7 Command-line options
   - 2.8 Screen output
   - 2.9 Tips for users of previous versions
3. Commands
   - 3.1 LAMMPS input script
   - 3.2 Parsing rules
   - 3.3 Input script structure
   - 3.4 Commands listed by category
   - 3.5 Commands listed alphabetically
4. Packages
   - 4.1 Standard packages
   - 4.2 User packages

- Introductory Tutorials and HowTos (http://lammps.sandia.gov/howto.html)

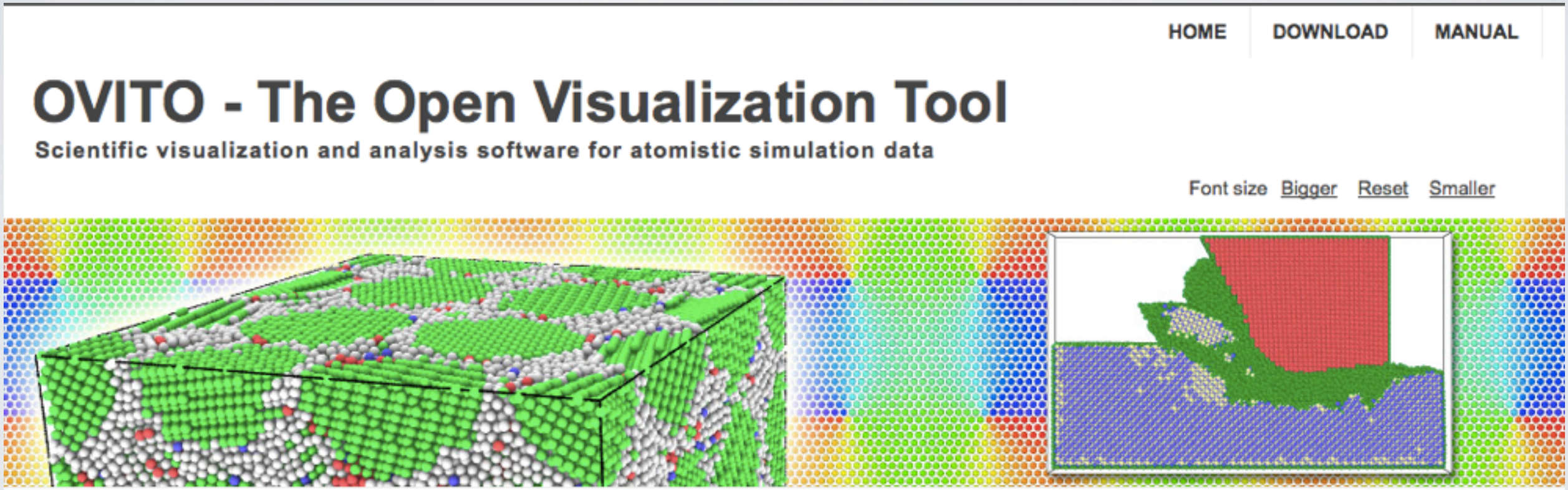


Big Picture	Code	Documentation	User Support
Features	Download	Manual	Mail list
Non-features	SourceForge	Developer Guide	Workshops
FAQ	Latest Features & Bug Fixes	Tutorials	User Scripts and HowTos
Wish list	Unfixed bugs	MD to LAMMPS glossary	Contribute to LAMMPS
.	.	Commands	.

- Friendly user base and mailing list (http://lammps.sandia.gov/mail.html)
- Excellent third-party tutorials hosted by CAVS @ MSU (https://icme.hpc.msstate.edu/mediawiki/index.php/LAMMPS_tutorials)

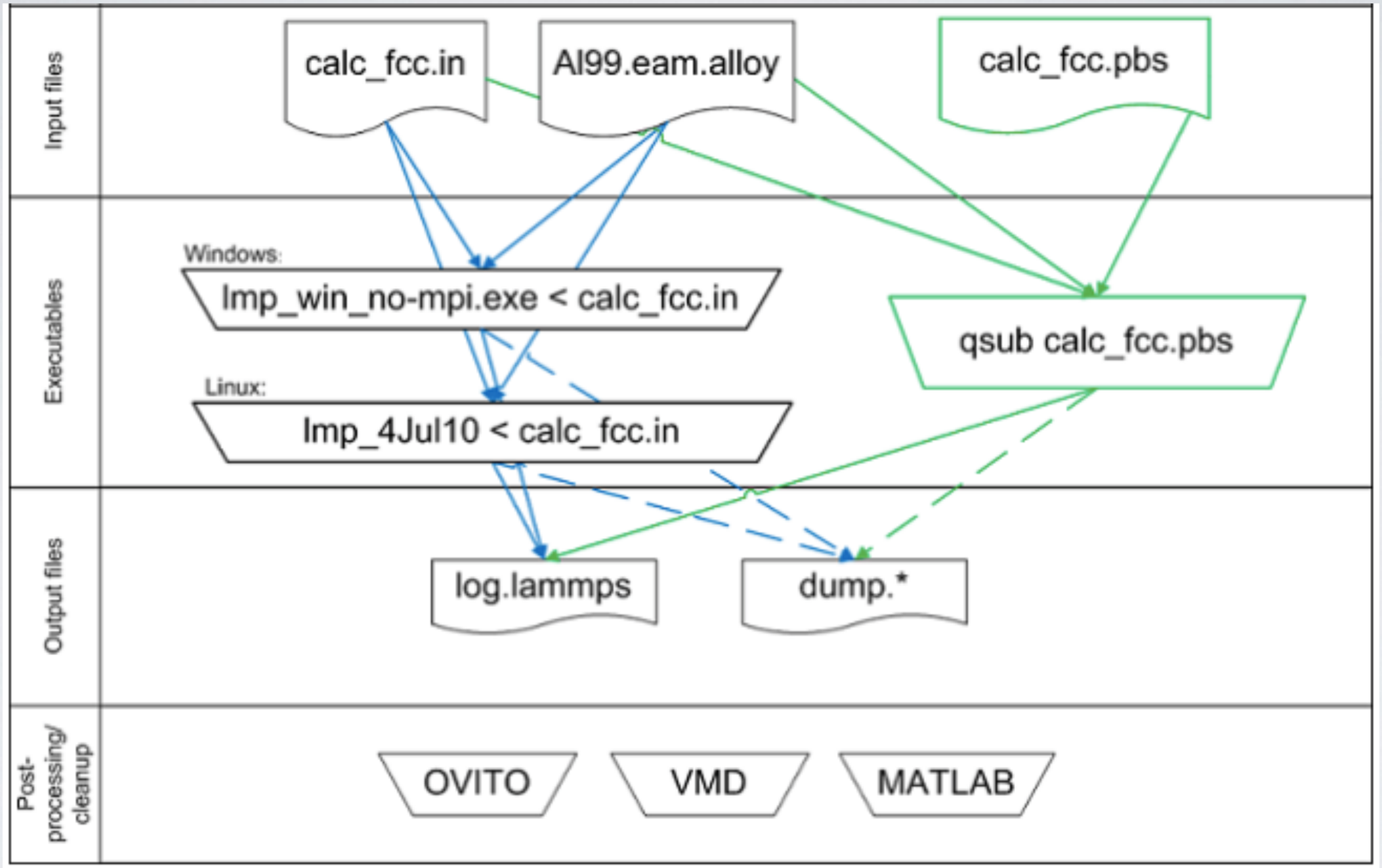
Visualization

- LAMMPS has no built-in visualization capability
- **OVITO** is a free, user-friendly and powerful visualization engine available for Linux, Mac and Windows



http://www.ovito.org

Running a simulation



https://icme.hpc.msstate.edu/mediawiki/index.php/LAMMPS_Help

VIII. Hands-on with LAMMPS

Adapted from materials developed by Mark A. Tschopp (US ARL) and hosted at https://icme.hpc.msstate.edu

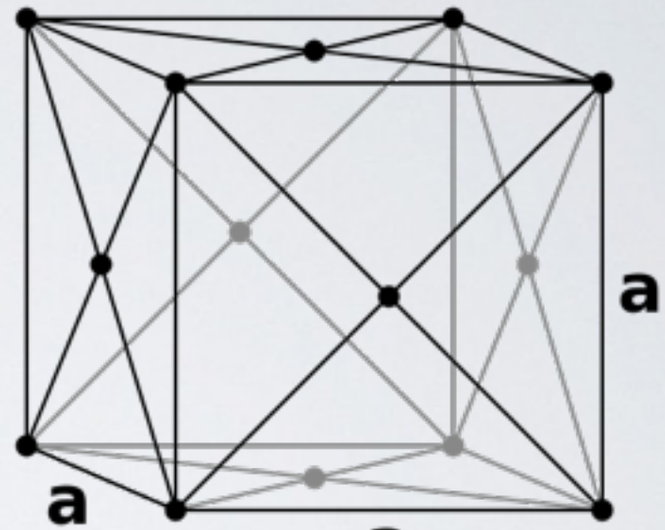
Tutorial 1: Al cohesive energy

Tutorial I: Al cohesive energy

- We will use LAMMPS to estimate the Al fcc cohesive energy, $\mathbf{E_{cohe}}$, and lattice parameter, **a**

$$E_{cohe} = E_{solid} - \sum_{atoms} E_{isolated}$$

0



- Experimentally, $\mathbf{E_{cohe}}$ = -3.39 eV/atom* and **a** = 4.0495 Å*

- **Strategy:** We shall use a modern EAM potential for Al and optimize $\mathbf{E_{cohe}}$ as a function of **a**

*Charles Kittel. Introduction to Solid State Physics, 8th edition. Hoboken, NJ: John Wiley & Sons, Inc, 2005. *http://periodictable.com/Elements/013/data.html

Tutorial 1: Al cohesive energy

1. Download **Al99.eam.alloy** EAM potential from NIST Interatomic Potentials Repository Project (http://www.ctcms.nist.gov/potentials)

Elements

	1												13	14	15	16	17	18
1	1 H																	2 He
2	3 **Li**	4 Be											5 B	6 **C**	7 N	8 **O**	9 F	10 Ne
3	11 **Na**	12 **Mg**	3	4	5	6	7	8	9	10	11	12	13 **Al**	14 **Si**	15 P	16 **S**	17 Cl	18 Ar
4	19 **K**	20 Ca	21 Sc	22 **Ti**	23 **V**	24 **Cr**	25 Mn	26 **Fe**	27 **Co**	28 **Ni**	29 **Cu**	30 **Zn**	31 Ga	32 Ge	33 As	34 **Se**	35 Br	36 Kr
5	37 **Rb**	38 Sr	39 Y	40 **Zr**	41 **Nb**	42 **Mo**	43 Tc	44 **Ru**	45 Rh	46 **Pd**	47 **Ag**	48 **Cd**	49 In	50 Sn	51 Sb	52 **Te**	53 I	54 Xe
6	55 **Cs**	56 Ba	*	72 Hf	73 **Ta**	74 **W**	75 Re	76 Os	77 Ir	78 **Pt**	79 **Au**	80 **Hg**	81 Tl	82 **Pb**	83 Bi	84 Po	85 At	86 Rn
7	87 Fr	88 Ra	**	104 Rf	105 Db	106 Sg	107 Bh	108 Hs	109 Mt	110 Ds	111 Rg	112 Cn	113 Uut	114 Fl	115 Uup	116 Lv	117 Uus	118 Uuo

*	57 La	58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu
**	89 Ac	90 Th	91 Pa	92 **U**	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr

Tutorial 1: Al cohesive energy

2. Obtain LAMMPS input file **Al_fcc.in** from http://ferguson.matse.illinois.edu/download/Al.zip

Al_fcc.in

Al99.eam.alloy

lmp_mac

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation ---------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms ---------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential ---------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings ---------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options ---------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization ---------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

For style *metal*, these are the units:

- mass = grams/mole
- distance = Angstroms
- time = picoseconds
- energy = eV
- velocity = Angstroms/picosecond
- force = eV/Angstrom
- torque = eV
- temperature = Kelvin
- pressure = bars
- dynamic viscosity = Poise
- charge = multiple of electron charge (1.0 is a proton)
- dipole = charge*Angstroms
- electric field = volts/Angstrom
- density = gram/cm^dim

- **#** specifies a comment
- x,y,z **periodic boundaries**

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation --------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms --------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential --------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings --------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options --------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization --------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

- Specify **fcc lattice** with **a**=4 Å
- Define **cuboidal block** labeled **box** holding **one lattice cell**
- Create **box** with **1** atom type

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation ---------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms ---------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential ---------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings ---------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options ---------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization ---------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

- Specify fcc lattice **orientation**
- Create atoms of type **1** on lattice sites within **box**
- **Replicate domain** by **2x2x2** in x,y,z

[`replicate 1 1 1` would be more parsimonious for this trivially periodic system]

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation ---------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms ---------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential ---------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings ---------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options ---------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization ---------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

- Define form of pairwise interaction potential as **eam/alloy**
  [misnomer, EAM is n-body]

- Use **Al** block of **Al99.eam.alloy** - specifies cutoff, F, ρ, and Φ - for all pairs
  [for one atom type, **1 1** fine]

- **2 Å skin thickness** for **neighbor list binning**

- Build neighbor list every **10 steps**, but **check** atom moved more than half skin thickness

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation ---------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms ---------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential ---------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings ---------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options ---------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization ---------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

- Define **computes** - quantities recalculated every time step [cf. **variables**, which evaluate a formula when called]

- Reference computes as **c_<name>**

- **c_eng** defined over **all** atoms to compute **potential energy per atom**

- **c_eatoms** performs **sum reduce** of c_eng vector over **all** atoms [alternatively: `compute eatoms all pe`]

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation ---------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms ---------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential ---------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings ---------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options ---------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization ---------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

- A **dump** specifies how to write output data
- Tag dump with id **1** to write to **dump.relax** every **1** steps the coords of **all** of the **atoms**
- Dump format:

```
ITEM: TIMESTEP
0
ITEM: NUMBER OF ATOMS
32
ITEM: BOX BOUNDS pp pp pp
0 8
0 8
0 8
ITEM: ATOMS id type xs ys zs
1 1 0 0 0
2 1 0.25 0.25 0
3 1 0.25 0 0.25
4 1 0 0.25 0.25
...
```

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation --------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms --------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential --------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings --------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options --------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization --------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

- Reset time steps to **0**
- A **fix** is an operation applied at every time step
- Define fix **1** operating on **all** atoms **relaxes box** to an external **isotropic pressure** of **0.0 bar** with a **0.1% maximum fractional volume change per step**

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation --------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms ---------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential ---------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings ---------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options ---------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization ---------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

- Output **thermodynamic info** to screen every **10** steps [use **fix** / **dump** for file write]
- Customize thermo output
- Perform energy minimization by **conjugate gradient**
- **Minimize** $E = E_{FF} + E_{fix}$ with $\Delta E = 10^{-25}$ (i.e., 1 part in 10^{25}) and $\Delta f = 10^{-25}$, and a maximum of 5000 iterations and 10000 energy evaluations

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation ---------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms ---------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential ---------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings ---------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options ---------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization ---------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

- Define **variables** as formulas evaluated when called [cf. **computes**, simulation values recomputed each step]

- Reference variables as **v_<name>**

- natoms = # atoms
  teng = total PE (c_eatoms)
  a = lattice parameter
  (box side in x divided by # x replicas = 2)
  ecoh = cohesive energy /atom

Tutorial 1: Al cohesive energy

Al_fcc.in

```
# ---------- Initialize Simulation ---------------------
units metal
dimension 3
boundary p p p
atom_style atomic

# ---------- Create Atoms ---------------------
lattice         fcc 4
region  box block 0 1 0 1 0 1 units lattice
create_box      1 box

lattice fcc 4 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms 1 box
replicate 2 2 2

# ---------- Define Interatomic Potential ---------------------
pair_style eam/alloy
pair_coeff * * Al99.eam.alloy Al
neighbor 2.0 bin
neigh_modify delay 10 check yes

# ---------- Define Settings ---------------------
compute eng all pe/atom
compute eatoms all reduce sum c_eng

# ---------- Dump Options ---------------------
dump            1 all atom 1 dump.relax

# ---------- Run Minimization ---------------------
reset_timestep 0
fix 1 all box/relax iso 0.0 vmax 0.001
thermo 10
thermo_style custom step pe lx ly lz press pxx pyy pzz c_eatoms
min_style cg
minimize 1e-25 1e-25 5000 10000

variable natoms equal "count(all)"
variable teng equal "c_eatoms"
variable a equal "lx/2"
variable ecoh equal "v_teng/v_natoms"

print "Total energy (eV) = ${teng};"
print "Number of atoms = ${natoms};"
print "Lattice constant (Angstoms) = ${a};"
print "Cohesive energy (eV/atom) = ${ecoh};"

print "All done!"
```

- Print terminal output to screen

Tutorial 1: Al cohesive energy

3. Let's run! `./lmp_mac < Al_fcc.in`

```
tuckernuck:1_Al_cohesive_energy alf$ ./lmp_mac < Al_fcc.in
LAMMPS (1 Feb 2014)
Lattice spacing in x,y,z = 4 4 4
Created orthogonal box = (0 0 0) to (4 4 4)
  1 by 1 by 1 MPI processor grid
Lattice spacing in x,y,z = 4 4 4
Created 4 atoms
Replicating atoms ...
  orthogonal box = (0 0 0) to (8 8 8)
  1 by 1 by 1 MPI processor grid
  32 atoms
WARNING: Resetting reneighboring criteria during minimization (../min.cpp:173)
Setting up minimization ...
Memory usage per processor = 3.39898 Mbytes
Step PotEng Lx Ly Lz Press Pxx Pyy Pzz eatoms
       0    -107.3423           8           8           8    29590.11    29590.11    29590.11    29590.11    -107.3423
      10   -107.51283        8.08        8.08        8.08    5853.9553    5853.9553    5853.9553    5853.9553   -107.51283
      14      -107.52         8.1         8.1         8.1    2.726913    2.726913    2.726913    2.726913      -107.52
Loop time of 0.00931406 on 1 procs for 14 steps with 32 atoms

Minimization stats:
  Stopping criterion = linesearch alpha is zero
  Energy initial, next-to-last, final =
        -107.342298373      -107.51999962      -107.51999962
  Force two-norm initial, final = 28.3679 0.00268005
  Force max component initial, final = 28.3679 0.00268005
  Final line search alpha, max atom move = 0.00145753 3.90625e-06
  Iterations, force evaluations = 14 23

Pair  time (%) = 0.00601649 (64.5958)
Neigh time (%) = 0 (0)
Comm  time (%) = 0.00095582 (10.2621)
Outpt time (%) = 0.000850677 (9.13326)
Other time (%) = 0.00149107 (16.0088)

Nlocal:    32 ave 32 max 32 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Nghost:    1067 ave 1067 max 1067 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Neighs:    2240 ave 2240 max 2240 min
Histogram: 1 0 0 0 0 0 0 0 0 0

Total # of neighbors = 2240
Ave neighs/atom = 70
Neighbor list builds = 0
Dangerous builds = 0
Total energy (eV) = -107.51999962032;
Number of atoms = 32;
Lattice constant (Angstoms) = 4.05;
Cohesive energy (eV/atom) = -3.359999988135;
All done!
```

building system

serial run

thermo

minimization stopping criteria

CPU accounting

atom accounting

neighbor accounting (dangerous builds)

terminal print

Tutorial 1: Al cohesive energy

4. Analysis

	LAMMPS	Expt.	
Lattice constant / Å	4.05	4.0495	*
Cohesive energy / eV/atom	-3.36	-3.39	*

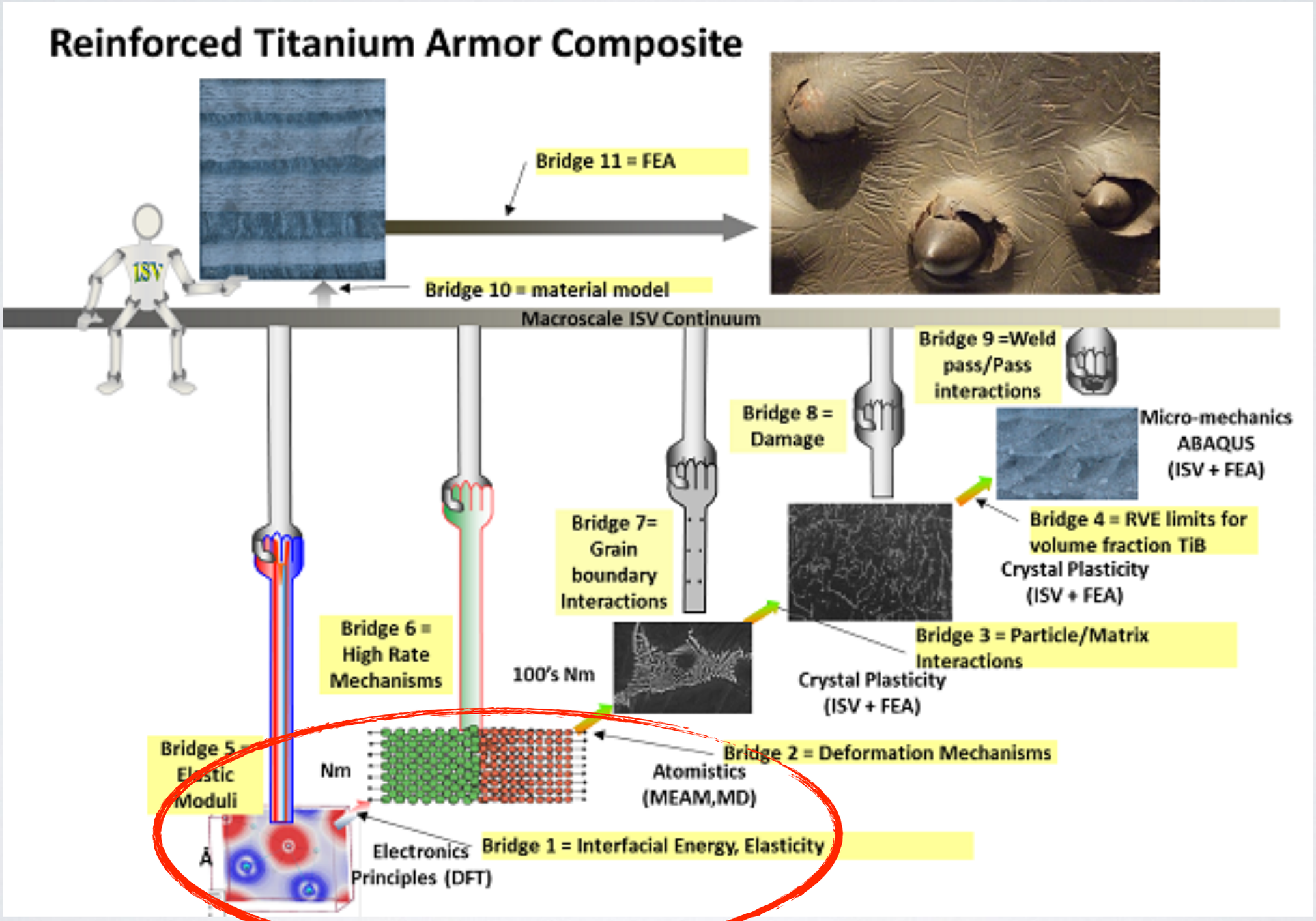
- We should be shocked if these quantities did **not** agree — EAM FF parametrized wrt experimental data

- **Q.** What about if we were studying a new material with experimentally unknown $\mathbf{E_{cohe}}$ and **a**?

*Charles Kittel. Introduction to Solid State Physics, 8th edition. Hoboken, NJ: John Wiley & Sons, Inc, 2005. *http://periodictable.com/Elements/013/data.html

Tutorial 1: Al cohesive energy

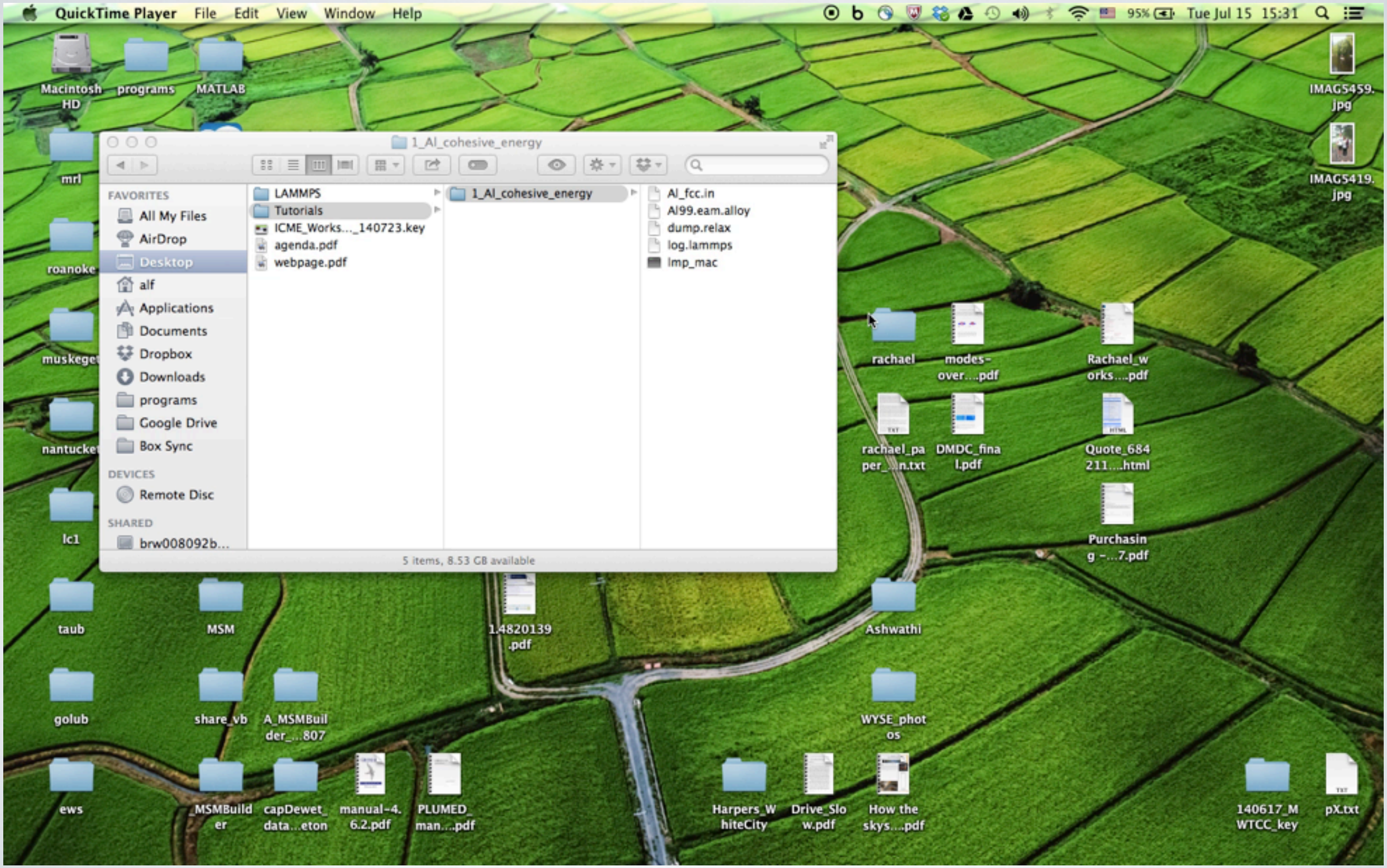
- **A.** ICME!



https://icme.hpc.msstate.edu/mediawiki/index.php?title=File:Titanium_armor_length_scale_Bridging_plot.png&limit=20

Tutorial 1: Al cohesive energy

5. Visualization in OVITO

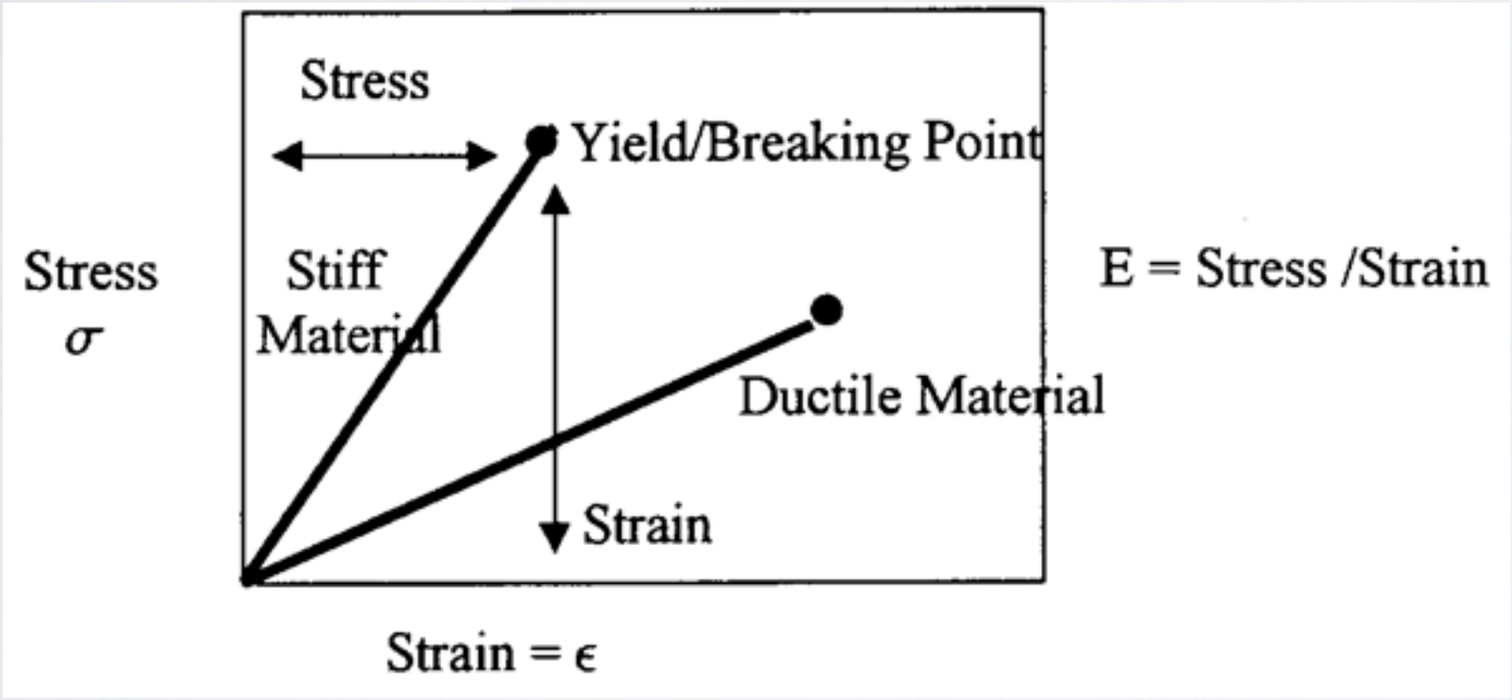


Tutorial 11: Young's modulus of Al

Tutorial 11: Young's modulus of Al

- OK, but weren't we meant to do MD?

- Right! Now that we can generate an equilibrated Al fcc lattice, let's use LAMMPS to estimate Young's modulus, **E**



$$E_{Al}^{exptl} = 69 \text{ GPa}^{*}$$

- **Strategy:** Apply an artificial extensional force to a fcc Al xtal and measure stress/strain relationship

*http://www.engineeringtoolbox.com/young-modulus-d_417.html http://www.mbari.org/staff/conn/botany/methods/methods/mmaterial.htm

Tutorial 11: Young's modulus of Al

1. Download **Al99.eam.alloy** EAM potential from NIST Interatomic Potentials Repository Project (http://www.ctcms.nist.gov/potentials)

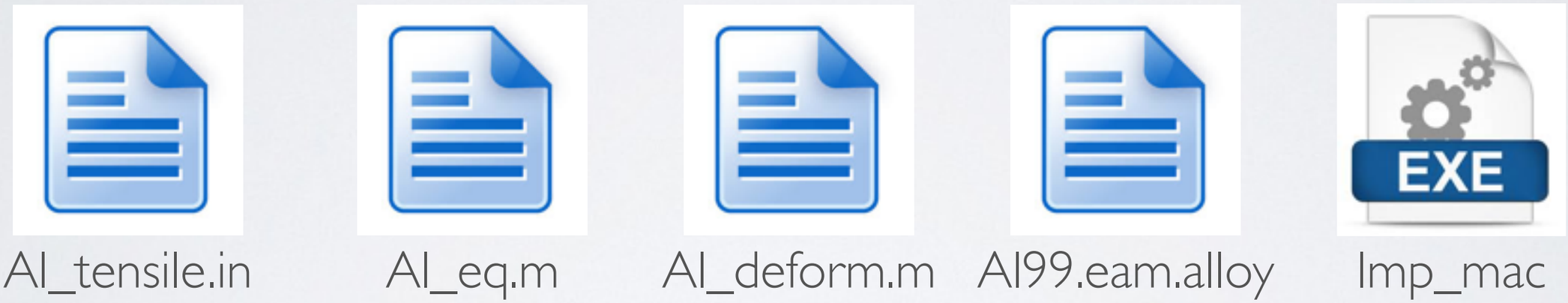
Elements

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1 H																	2 He
2	3 **Li**	4 Be											5 B	6 **C**	7 N	8 **O**	9 F	10 Ne
3	11 **Na**	12 **Mg**											13 **Al**	14 **Si**	15 P	16 **S**	17 Cl	18 Ar
4	19 **K**	20 Ca	21 Sc	22 **Ti**	23 **V**	24 **Cr**	25 Mn	26 **Fe**	27 **Co**	28 **Ni**	29 **Cu**	30 **Zn**	31 Ga	32 Ge	33 As	34 **Se**	35 Br	36 Kr
5	37 **Rb**	38 Sr	39 Y	40 **Zr**	41 **Nb**	42 **Mo**	43 Tc	44 **Ru**	45 Rh	46 **Pd**	47 **Ag**	48 **Cd**	49 In	50 Sn	51 Sb	52 **Te**	53 I	54 Xe
6	55 **Cs**	56 Ba	*	72 Hf	73 **Ta**	74 **W**	75 Re	76 Os	77 Ir	78 **Pt**	79 **Au**	80 **Hg**	81 Tl	82 **Pb**	83 Bi	84 Po	85 At	86 Rn
7	87 Fr	88 Ra	**	104 Rf	105 Db	106 Sg	107 Bh	108 Hs	109 Mt	110 Ds	111 Rg	112 Cn	113 Uut	114 Fl	115 Uup	116 Lv	117 Uus	118 Uuo

*	57 La	58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu
**	89 Ac	90 Th	91 Pa	92 **U**	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr

Tutorial 11: Young's modulus of Al

2. Obtain LAMMPS input files **Al_tensile.in**, **Al_eq.m**, and **Al_deform.m** from http://ferguson.matse.illinois.edu/download/Al.zip



Tutorial 11: Young's modulus of Al

Al_tensile.in

```
# ------------------------ INITIALIZATION -----------------------------
units           metal
dimension       3
boundary        p       p       p
atom_style      atomic
variable latparam equal 4.05

# ------------------------ ATOM DEFINITION ----------------------------
lattice         fcc ${latparam}
region          whole block 0 10 0 10 0 10
create_box      1 whole

lattice         fcc ${latparam} orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
create_atoms    1 region whole
replicate       1 1 1

# ------------------------ FORCE FIELDS -------------------------------
pair_style      eam/alloy
pair_coeff      * * Al99.eam.alloy Al

neighbor        2.0 bin
neigh_modify    delay 0 every 10 check yes

# ------------------------ SETTINGS -----------------------------------
compute         csym all centro/atom fcc
compute         eng all pe/atom
```

- Set lattice parameter **variable** to **a** = $\mathbf{a_{eq}}$ = 4.05 Å
- Specify two **computes** to calculate **pe/atom** and **centrosymmetry parameter**

$$CS = \sum_{i=1}^{N/2} |\vec{R}_i + \vec{R}_{i+N/2}|^2$$

```
Bulk lattice = 0
Dislocation core ~ 1.0
Stacking faults ~ 5.0
Free surface ~ 23.0
```

http://lammps.sandia.gov/doc/compute_centro_atom.html#Kelchner

Tutorial 11: Young's modulus of Al

```
########################################
# EQUILIBRATION

# reset timer
reset_timestep  0

# 2 fs time step
timestep        0.002

# initial velocities
velocity        all create 300 12345 mom yes rot no

# thermostat + barostat
fix             1 all npt temp 300 300 1 iso 0 0 1 drag 1.0

# instrumentation and output
variable s1 equal "time"
variable s2 equal "lx"
variable s3 equal "ly"
variable s4 equal "lz"
variable s5 equal "vol"
variable s6 equal "press"
variable s7 equal "pe"
variable s8 equal "ke"
variable s9 equal "etotal"
variable s10 equal "temp"
fix writer all print 250 "${s1} ${s2} ${s3} ${s4} ${s5} ${s6} ${s7} ${s8} ${s9} ${s10}" file Al_eq.txt screen no

# thermo
thermo          500
thermo_style    custom step time cpu cpuremain lx ly lz press pe temp

# dumping trajectory
dump            1 all atom 250 dump.eq.lammpstrj

# 24 ps MD simulation (assuming 2 fs time step)
run             12000

# clearing fixes and dumps
unfix           1
undump          1

# saving equilibrium length for strain calculation
variable tmp equal "lx"
variable L0 equal ${tmp}
print "Initial Length, L0: ${L0}"
```

- Instrumentation, perform MD integration with Verlet (default) algorithm, and record terminal relaxed box size

Tutorial 11: Young's modulus of Al

```
########################################
# DEFORMATION

# reset timer
reset_timestep  0

# 2 fs time step
timestep 0.002

# thermostat + barostat
fix             1 all npt temp 300 300 1 y 0 0 1 z 0 0 1 drag 1.0

# nonequilibrium straining in x-direction at strain rate = 1x10^10 / s = 1x10^-2 / ps in units metal
variable srate equal 1.0e10
variable srate1 equal "v_srate / 1.0e12"
fix             2 all deform 1 x erate ${srate1} units box remap x

# instrumentation and output
# for units metal, pressure is in [bars] = 100 [kPa] = 1/10000 [GPa] => p2, p3, p4 are in GPa
variable strain equal "(lx - v_L0)/v_L0"
variable p1 equal "v_strain"
variable p2 equal "-pxx/10000"
variable p3 equal "-pyy/10000"
variable p4 equal "-pzz/10000"
fix writer all print 125 "${p1} ${p2} ${p3} ${p4}" file Al_deform.txt screen no

# thermo
thermo          500
thermo_style    custom step cpuremain v_strain v_p2 v_p3 v_p4 press pe temp

# dumping standard atom trajectories
dump            1 all atom 125 dump.deform.lammpstrj

# dumping custom cfg files containing coords + ancillary variables
dump            2 all cfg 125 dump.deform_*.cfg mass type xs ys zs c_csym c_eng fx fy fz
dump_modify     2 element Al


# 20 ps MD simulation (assuming 2 fs time step)
run             10000

# clearing fixes and dumps
unfix           1
unfix           2
unfix           writer
undump          1
undump          2


########################################
# SIMULATION DONE
print "All done!"
```

- Nonequilibrium straining, instrumentation, and cfg trajectory dump

Tutorial 11: Young's modulus of Al

3. Let's run! `./lmp_mac < Al_tensile.in`

2_Al_Youngs_modulus — bash — 133×27

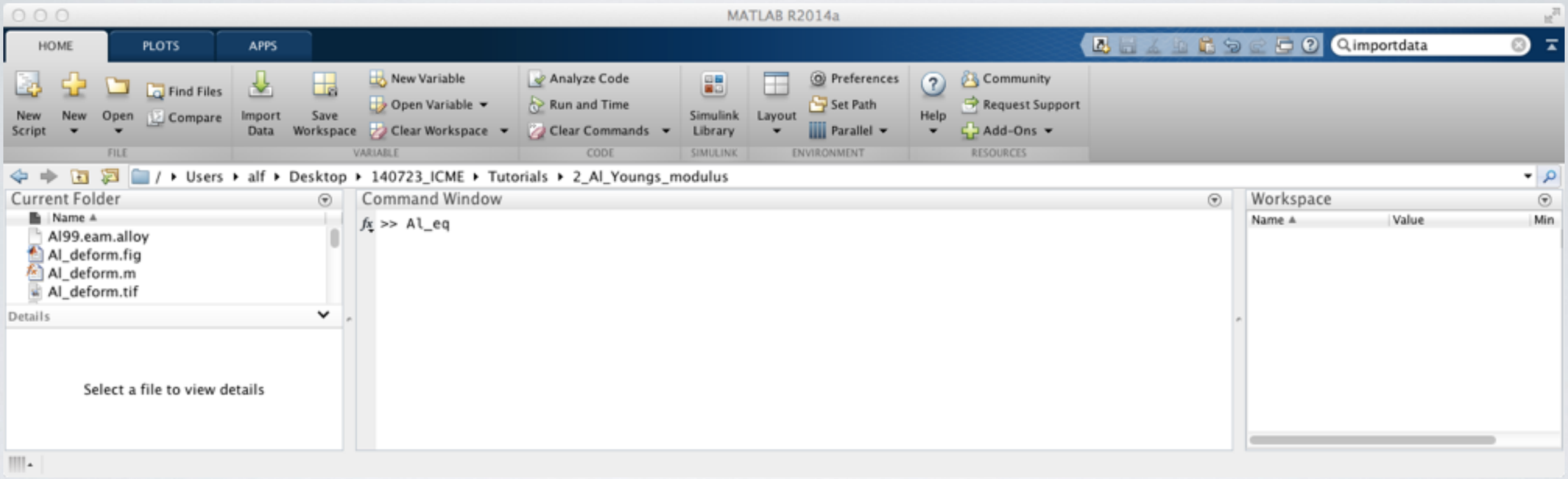
```
tuckernuck:2_Al_Youngs_modulus alf$ ./lmp_mac < Al_tensile.in
LAMMPS (1 Feb 2014)
Lattice spacing in x,y,z = 4.05 4.05 4.05
Created orthogonal box = (0 0 0) to (40.5 40.5 40.5)
  1 by 1 by 1 MPI processor grid
Lattice spacing in x,y,z = 4.05 4.05 4.05
Created 4000 atoms
Replicating atoms ...
  orthogonal box = (0 0 0) to (40.5 40.5 40.5)
  1 by 1 by 1 MPI processor grid
  4000 atoms
Setting up run ...
Memory usage per processor = 4.96236 Mbytes
Step Time CPU CPULeft Lx Ly Lz Press PotEng Temp
       0            0           0            0         40.5         40.5         40.5    2496.1233       -13440          300
     500            1    12.365961    284.41713    40.557806    40.557806    40.557806    781.69582   -13362.995    169.08671
    1000            2    24.741789    272.15969    40.573622    40.573622    40.573622    85.733564   -13355.919     178.0143
    1500            3    39.549843    276.84891     40.58055     40.58055     40.58055    222.05046   -13346.458    182.72414
    2000            4    54.536895    272.68448    40.588269    40.588269    40.588269    28.687955   -13340.533    194.24556
    2500            5    70.367178    267.39528    40.591944    40.591944    40.591944    191.32817   -13335.453    207.18274
    3000            6    83.555789    250.66737    40.595807    40.595807    40.595807    324.09009   -13329.002    216.94285
    3500            7    96.427479    234.18102    40.603551    40.603551    40.603551    330.98508   -13320.563    222.10308
    4000            8    110.11764    220.23529    40.611179    40.611179    40.611179    106.05206   -13316.256    234.27863
    4500            9    122.99169    204.98615     40.61865     40.61865     40.61865    21.917251    -13313.98    249.16077
    5000           10    135.84727    190.18618    40.625915    40.625915    40.625915    14.611906    -13307.48    254.46631
    5500           11    148.63189    175.65587    40.629588    40.629588    40.629588     30.56594   -13302.925    261.97643
    6000           12    161.44717    161.44717    40.631803    40.631803    40.631803    2.7573596   -13301.413    273.69236
```

N.B. This could take 8-10 minutes if your machine is old and slow (like mine)
Speed things up by reducing system size by factor of 2^3 in **Al_tensile.in**:

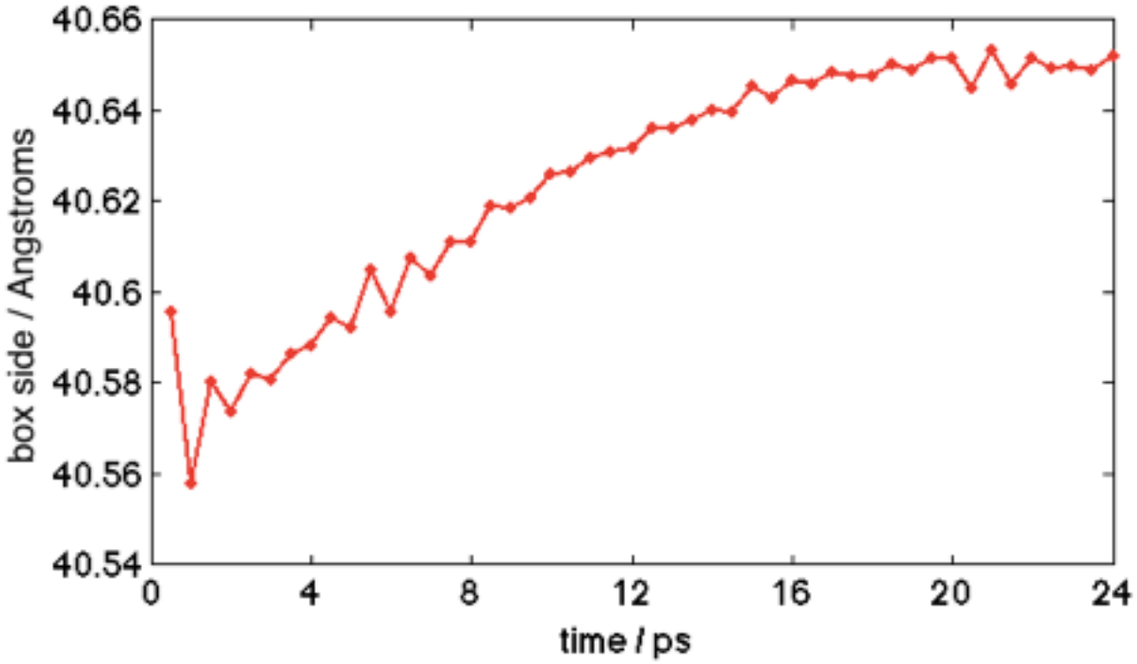
`region  whole block 0 5 0 5 0 5`

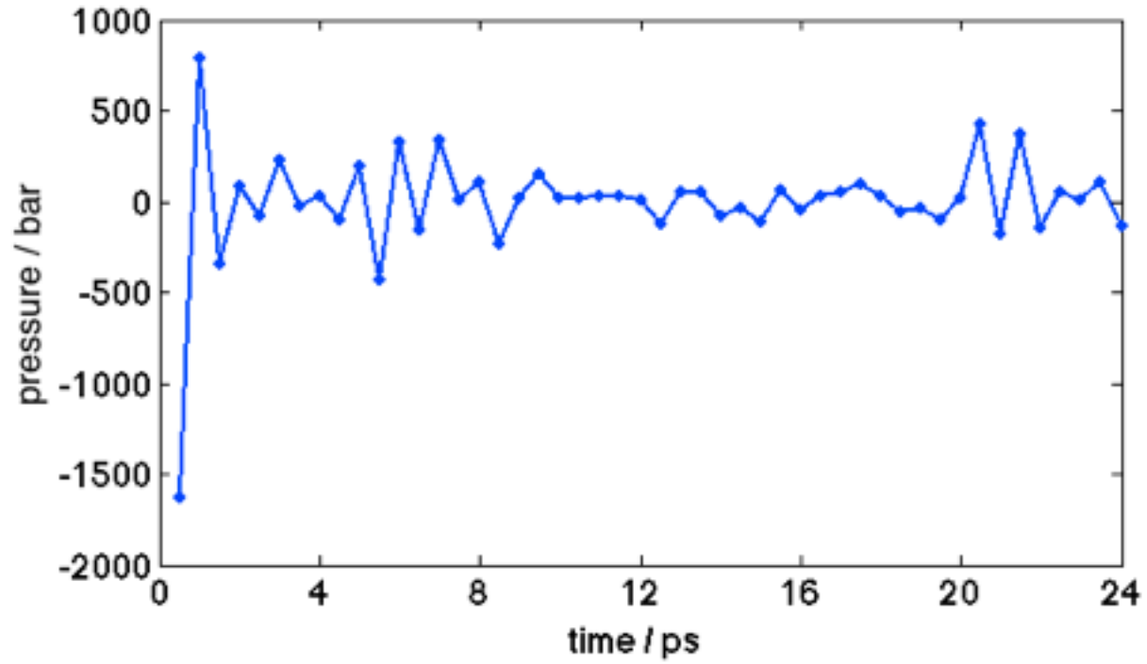
Tutorial 11: Young's modulus of Al

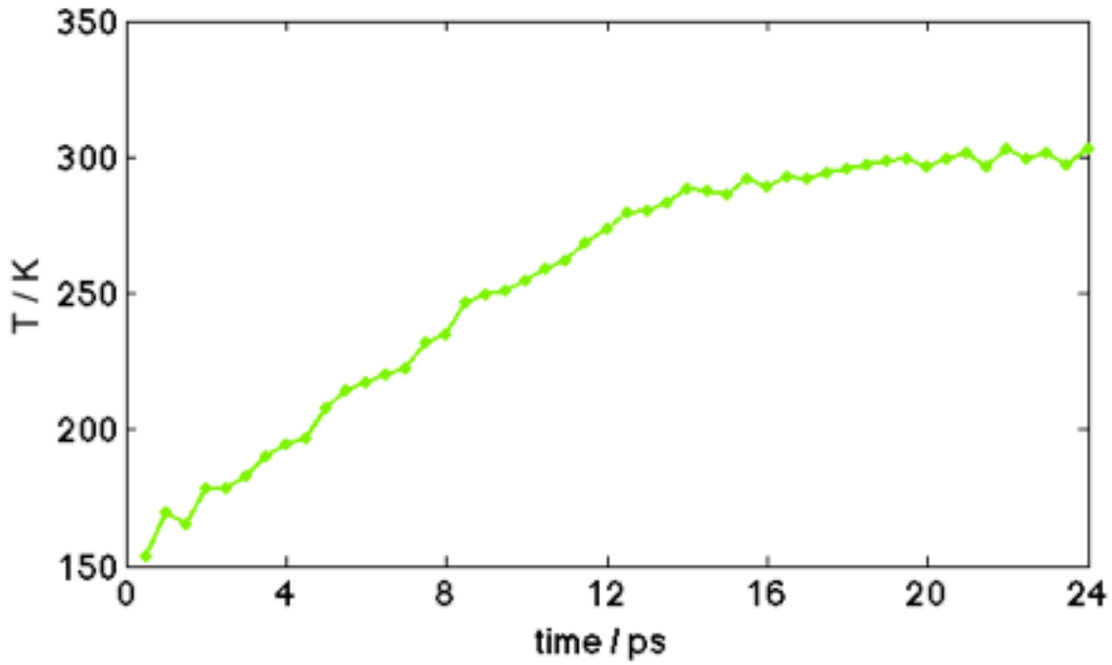
4. Analyze approach to equilibration using **Al_eq.m**

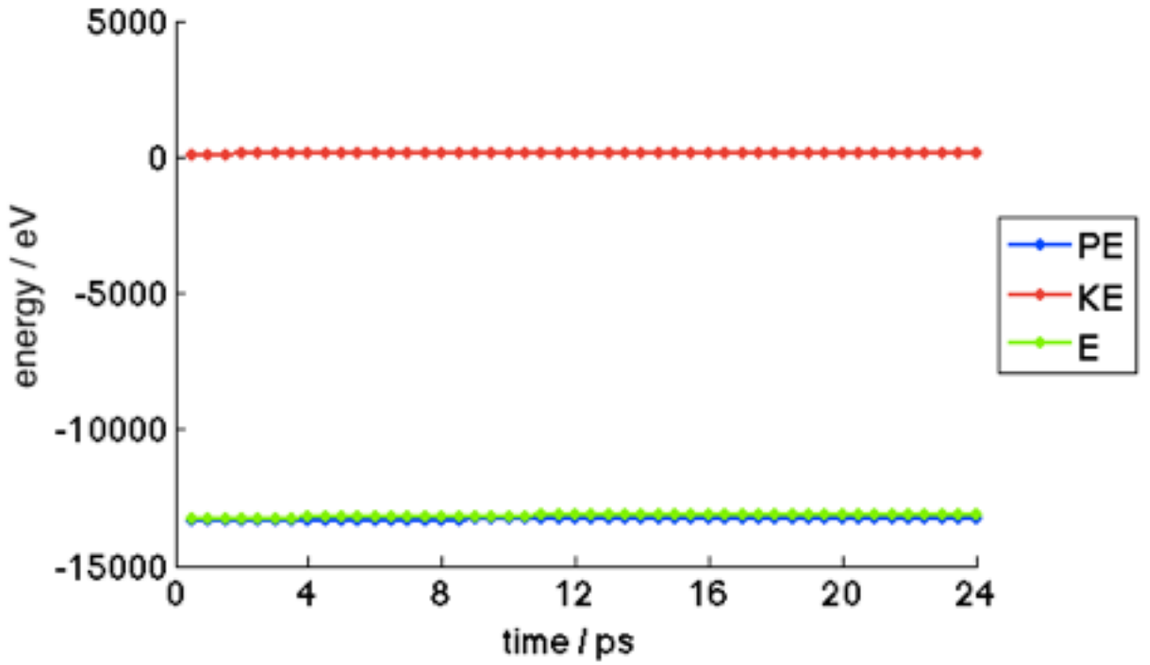


Tutorial 11: Young's modulus of Al





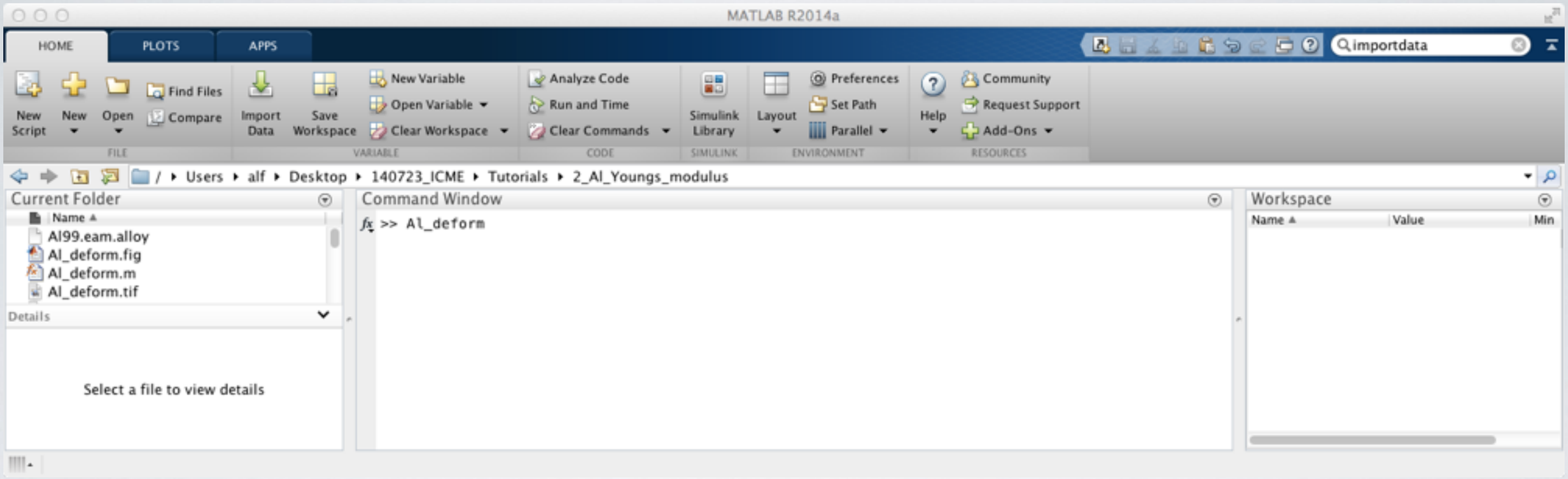




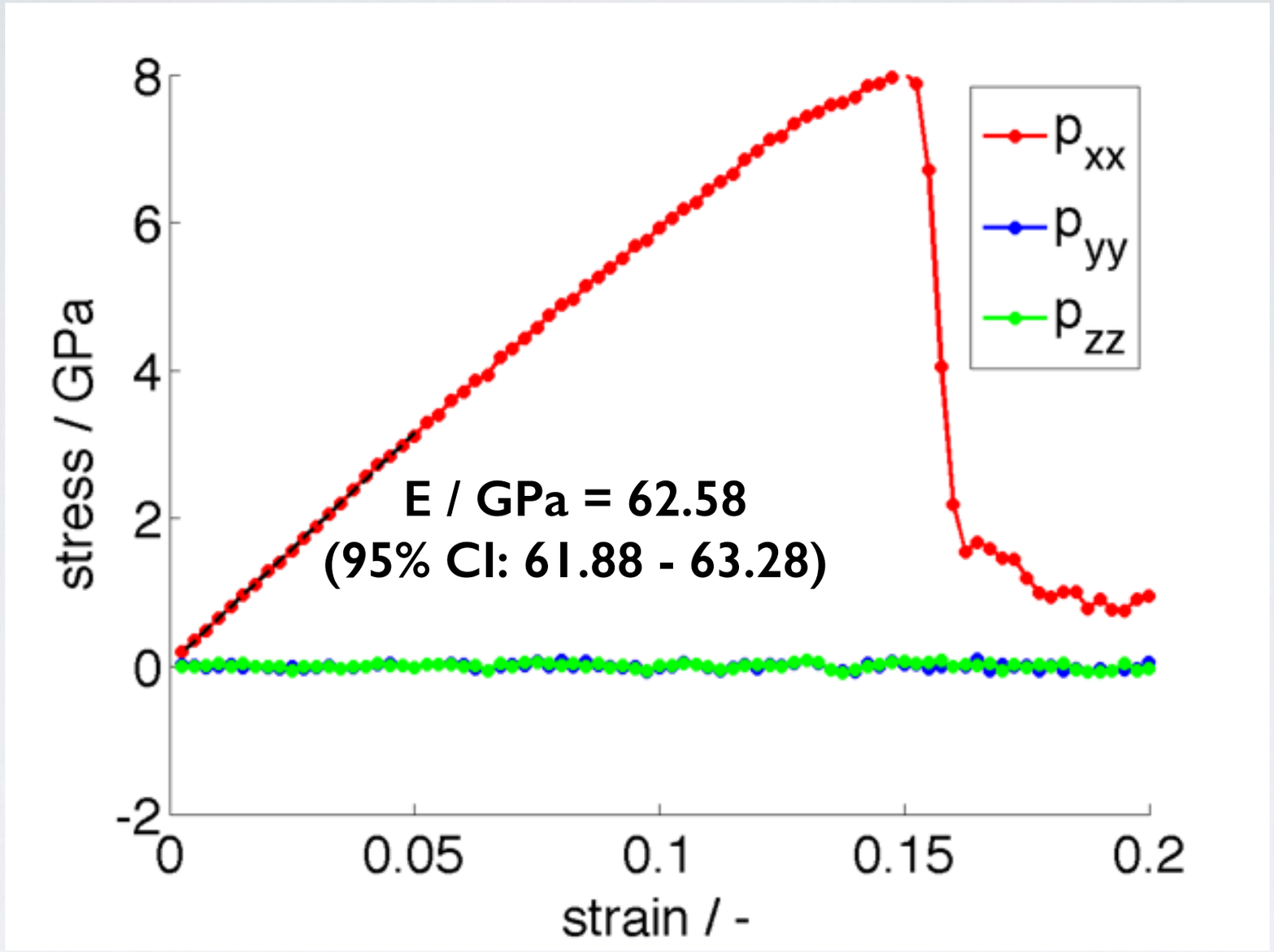
- Equilibrium attained in ~24 ps

Tutorial 11: Young's modulus of Al

5. Analyze deformation and estimate **E** using **Al_deform.m**



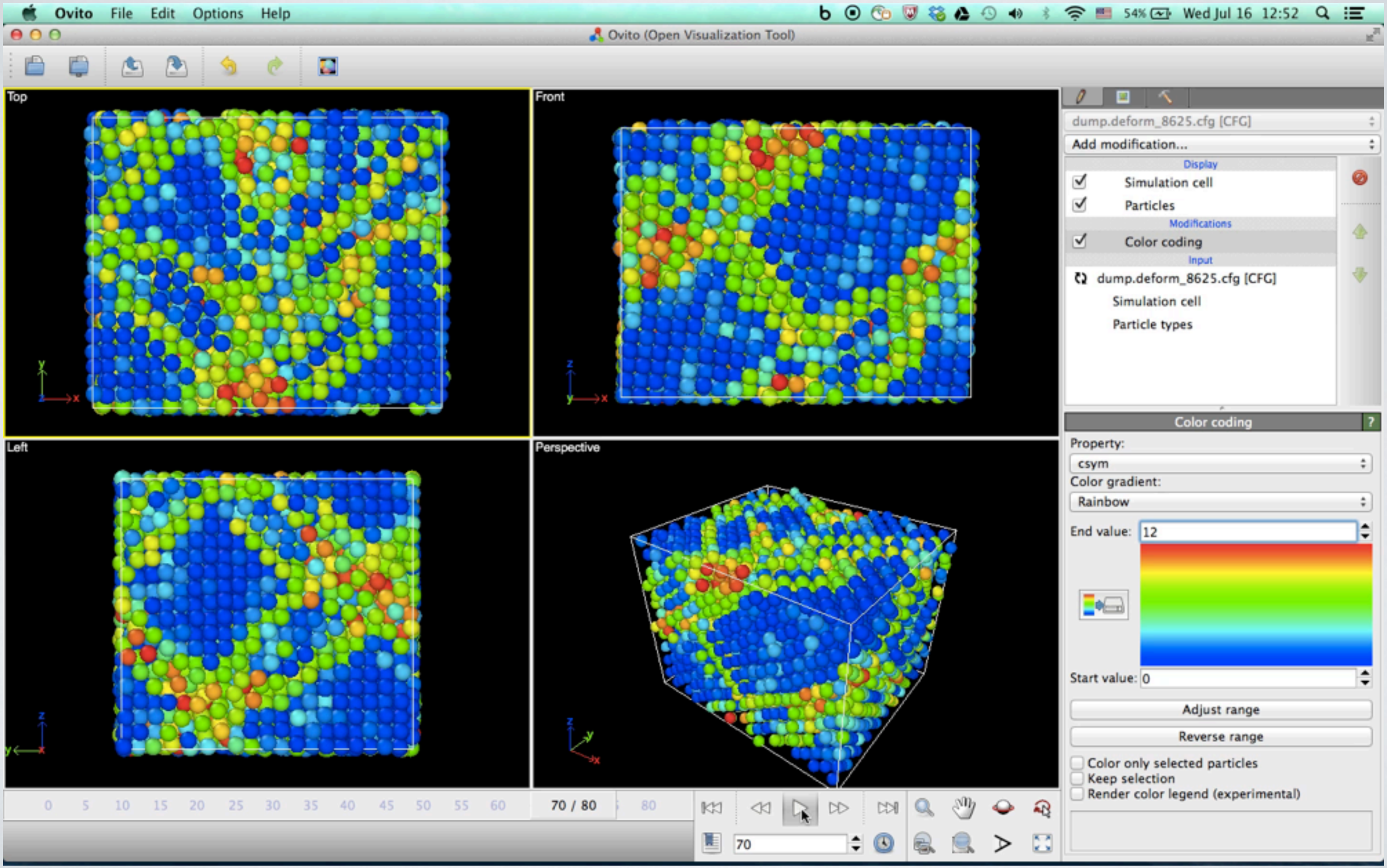
Tutorial 11: Young's modulus of Al



- Onset of homogeneous dislocation nucleation and end of elastic deformation at ~8 GPa
- **E** estimated by slope of linear fit over strain range [0-0.05]

Tutorial 11: Young's modulus of Al

6. Visualization of deformation in OVITO



Tutorial 11: Young's modulus of Al

7. Comparison to experiment

	LAMMPS	Expt.		Δ / %
Young's Modulus / GPa	62	69	*	10.1
Yield Stress / MPa	8000	10	*	79900

Young's Modulus

- We did pretty well, **E** within ±10%
- A rigorous study would check **E** convergence as a function of **system size**

Yield Stress

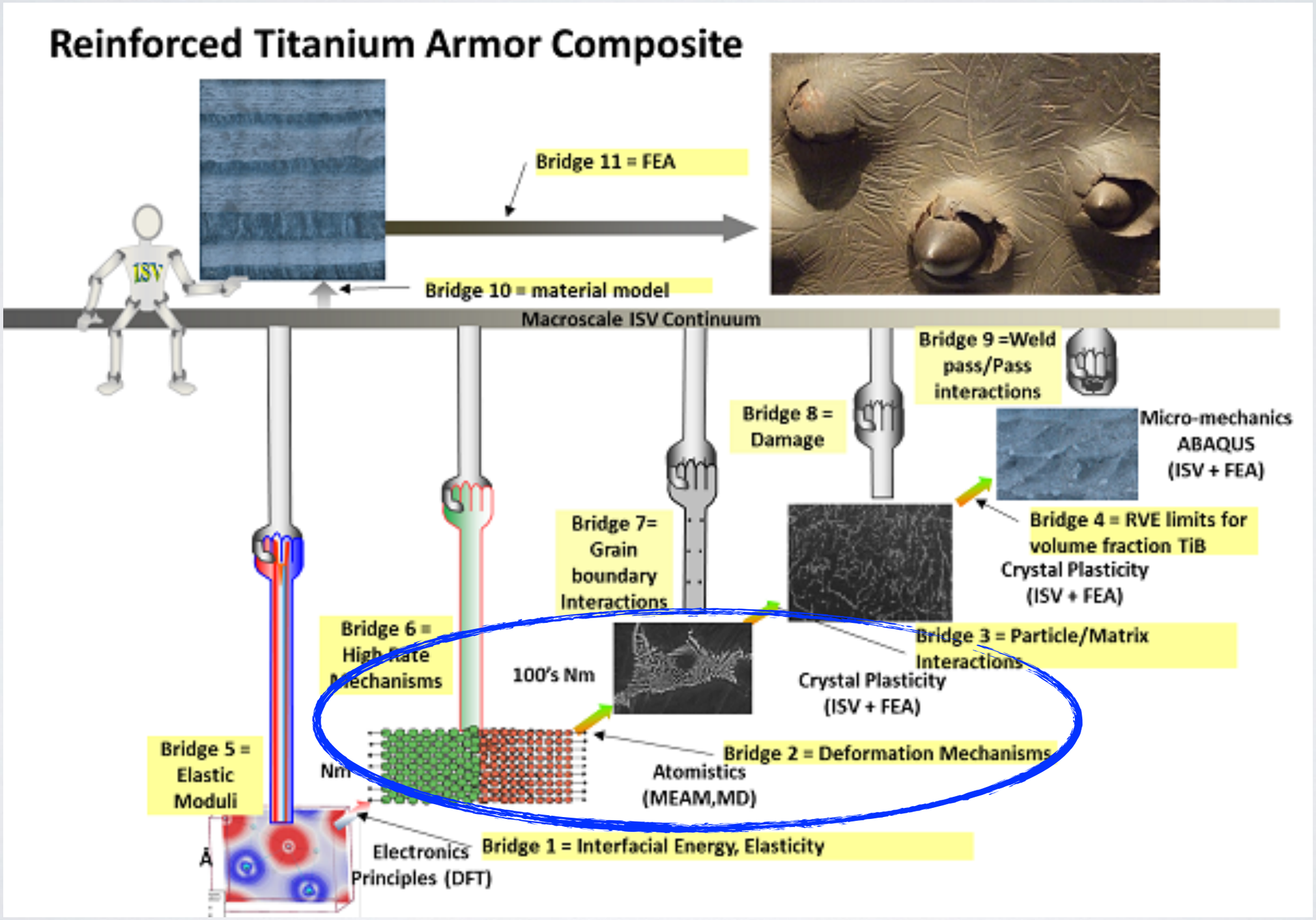
- Our estimate for yield stress is horrible! Off by ~3 orders of magnitude!
- Why did we do so badly? — We have a **perfect crystal**, *homogeneous* vs. *heterogeneous* nucleation.

*http://www.engineeringtoolbox.com/young-modulus-d_417.html

*Polmear, I. J. (1995). Light Alloys: Metallurgy of the Light Metals (3rd ed.). Butterworth-Heinemann

Tutorial 11: Young's modulus of Al

- **ICME** — for an experimentally uncharacterized material can "bridge up" MD **E** estimate to FEA model



https://icme.hpc.msstate.edu/mediawiki/index.php?title=File:Titanium_armor_length_scale_Bridging_plot.png&limit=20

Questions?

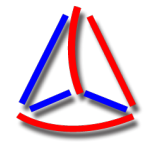
Molecular Dynamics Primer: LAMMPS Examples

Stefan Bringuier

Department of Materials Science and Engineering, University of Arizona

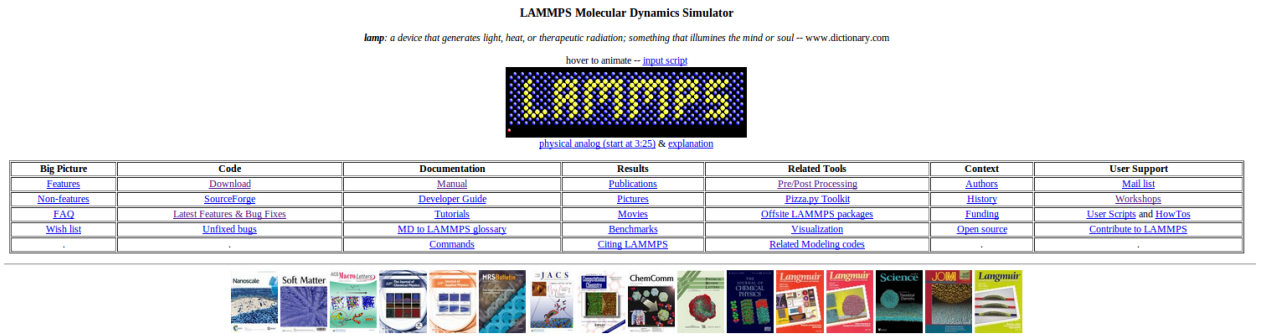
stefanb@email.arizona.edu
`http://u.arizona.edu/~stefanb`

April 2, 2014



LAMMPS

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



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	Workshops
FAQ	Latest Features & Bug Fixes	Tutorials	Movies	Offsite LAMMPS packages	Funding	User Scripts and HowTos
Wish list	Unfixed bugs	MD to LAMMPS glossary	Benchmarks	Visualization	Open source	Contribute to LAMMPS
.	.	Commands	Citing LAMMPS	Related Modeling codes	.	.

Adapted from lammps.sandia.gov

Design of LAMMPS

License

- LAMMPS is provided through GNU Public License
- Free to use,modify, and distribute

Code Layout

- Object-Oriented approach
- Parallelization via Message Passage Interface (MPI)
- Is invoked by commands through input scripts

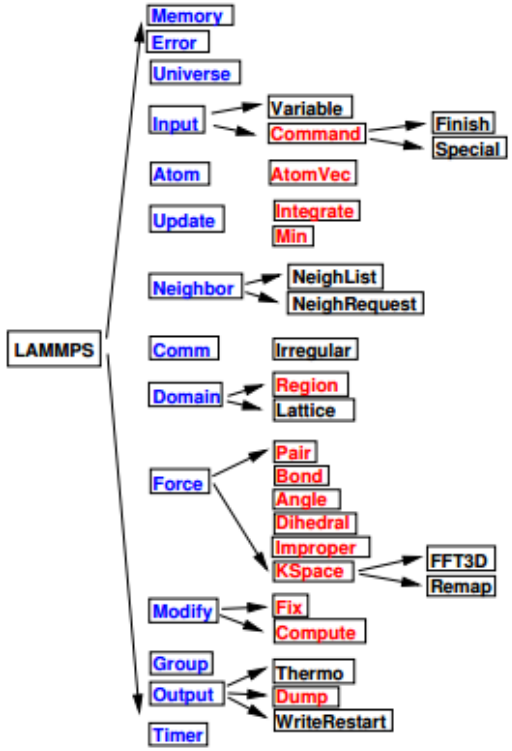


Figure 1: Class hierarchy within LAMMPS source code.

Adapted from lammps.sandia.gov

Obtaining LAMMPS

Download Source

Point your browser to `lammps.sandia.gov/download` and download the tarball. This allows you to compile LAMMPS to your liking!

Preferred Method For Beginners

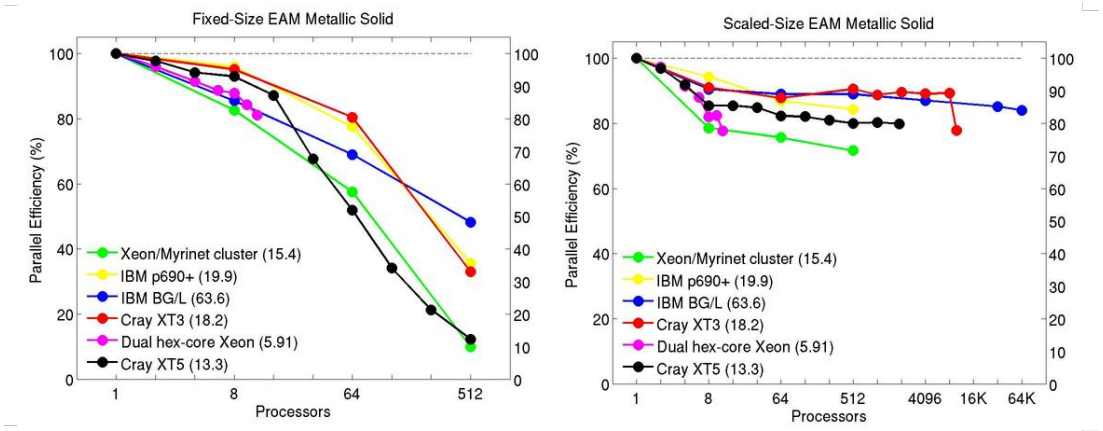
Many hard working people have set up repositories for linux/unix binaries as well as windows executables. Go to `lammps.sandia.gov/download` and find the appropriate link for *Pre-built* executables.

LAMMPS performance

Resource Usage

- Typically script prototyping and trouble shooting is done on user desktops and laptops.
- Actual MD simulations usually require significant resources
- High Performance Computing services
- Group computing server (12-36 cores)

Scaling



Adapted from lammps.sandia.gov

Resources

Use the Manual!

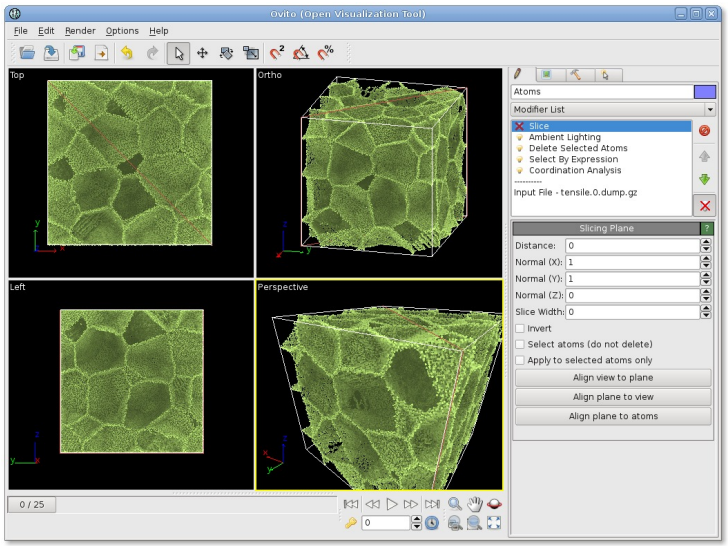
- The LAMMPS manual has almost all your questions answered.
- Manual: `http://lammps.sandia.gov/doc/Manual.html`
- Search the mailing list: `http://lammps.sandia.gov/mail.html`
- Subscribe and Email the mailing list (`https://lists.sourceforge.net/lists/listinfo/lammps-users`)

Adapted from www.martybucella.com

Continued...

Visualization Tools and Builders

- Checkout
  `http://lammps.sandia.gov/prepost.html`
- OVITO `http://www.ovito.org`
- VMD, `http://www.ks.uiuc.edu/Research/vmd`



Adapted from `http://www.ovito.org`

Outline of LAMMPS Input script

Command Line Driven

LAMMPS has no standard GUI. Every simulation is executed by supplying an input text script to the LAMMPS executable.

Parts of An Input Script

- Initialize: units, dimensions, etc.
- Atomic positions and velocities
- Settings:
  - Interatomic potential
  - Run time simulation parameters (e.g. timestep)
  - Fixes - operations during dynamics (e.g. thermostat)
  - Computes - calculation of properties during dynamics
- Run!

Example 1: Energy vs. Volume

Example 1: BCC Iron

This is more of a static calculation however we can get some useful material properties.

Listing 1: Simulation Settings

```
# Energy:eV Distance:Angstrom Mass:Kg Time:picosec
units       metal

#Classical particles
atom_style   atomic

#p=Periodic, in x y z
boundary     p p p
```

Continued...

Listing 2: Specify The Structure

```
#lattice constant i.e. Fe-BCC 2.8665
variable a equal 2.8665

# BCC structure
lattice bcc $a

# Specify simulation box
region box block 0 5 0 5 0 5

# Initiate box with 1 atom type
create_box 1 box

# Create 1 atom type on lattice in simulation box
create_atoms 1 box

mass 1 55.85
```

Continued...

Listing 3: Interatomic Potential

```
# Interatomic potential - Embedded Atom Method
pair_style eam/fs

# interaction pairs, filename, Element parameters
pair_coeff * * Fe_mm.eam.fs Fe

#Specify build neighborlist, use cutoff+0.3 A
neighbor  0.3 bin

#Frequency to rebuild neighborlist
neigh_modify every 20 delay 0 check no
```

Continued...

Listing 4: Output Configuration

```
#Compute the energy per atom
compute eperat all pe/atom

#Output x,y,z of atoms LAMMPS standard format
dump config all atom 10 dump.FeBCC_${a}

#Custom output of atom properties
dump config all custom 10 dump.FeBCC_${a} &
 id type x y z c_eperat
```

Listing 5: Dynamics

```
timestep 0.001

#Ensemble
fix 1 all nve

#Frequency of Ensemble data output to screen
thermo 500

#Data that is output to screen
thermo_style custom step pe ke temp vol press
run  10000
```

Results of Example 1

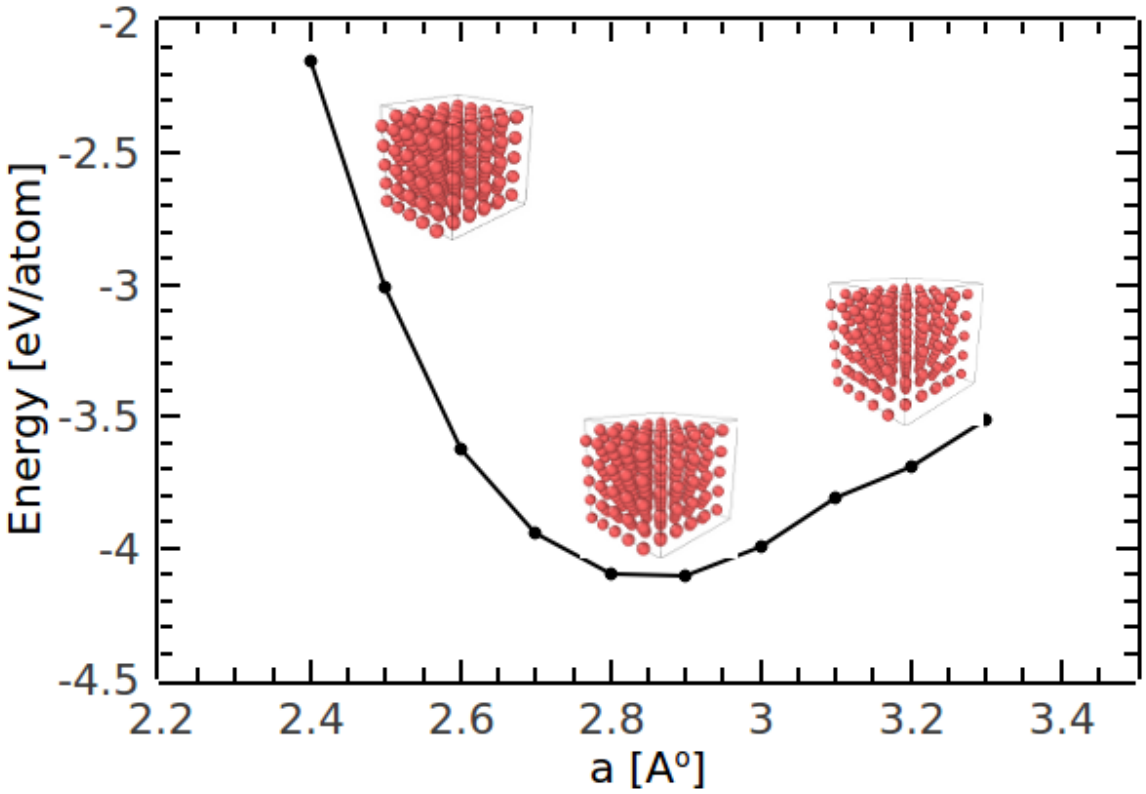


Figure: Fe-BCC Energy vs. lattice parameter. Experimental lattice parameter, a=2.865

Continued...

Elastic Constants

Not show in the input script of example 1 but we can obtain the static elastic constants with fair ease.

C_{11}	244 GPa
C_{12}	145 GPa
C_{44}	116 GPa

Table: Elastic constants for BCC Fe

Example 2: Melting Nanoparticle

Example 2: Attempt to observe melting of gold nanoparticle
Many of the commands for this example are similiar to the example 1. The nanoparticle was created by using the "region" command.
Potential - EAM for gold

Listing 6: RDF & minimize

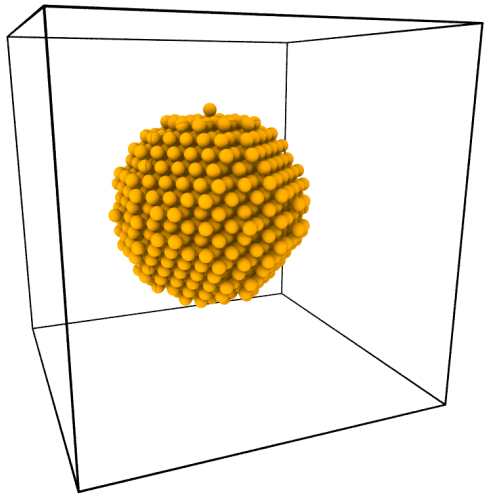
```
# Get the radial distribution function
compute rdf all rdf 50
fix rdf1 all ave/time 100 10 1000 c_rdf &
 ave running file nanogold.rdf mode vector

#Energy & Force tolerence ,max iterations.
minimize 1.0e-8 1.0e-8 1000 100000

#Minimize using conjugate gradient method
min_style cg
```

Listing 7: NVT-Melt

```
timestep 0.001

#Frequency of Ensemble data output to screen
thermo 5000
thermo_style custom step pe ke etotal temp vol press

#Thermalize to 298 K
#Create velocity distribution
velocity all create 298 39849 &
 mom yes rot yes dist gaussian

# NVT ensemble ramp temperature to melt
fix              2 all nvt temp 298.00 2400.00 1.0
run  100000
```

Results for Example 2

Watch it Melt!



Further Confirmation

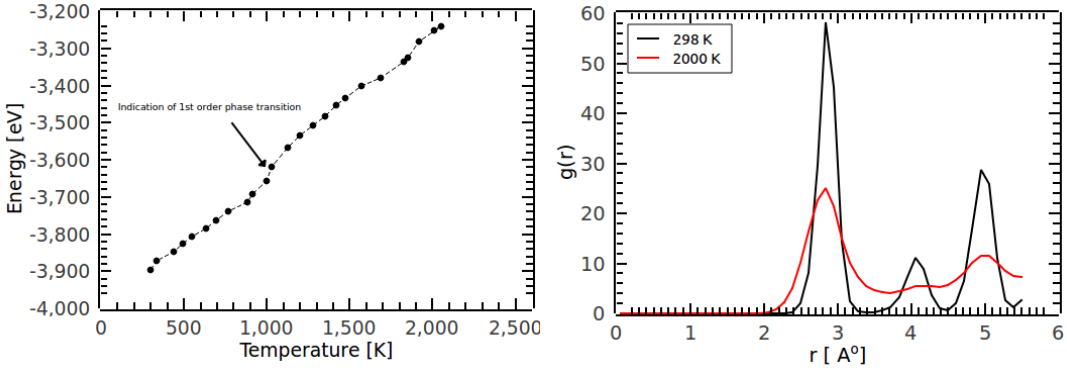


Figure: (a)Au nanoparticle melting (b) Radial distribution function

Things to Try!

- Change the diameter of the nanoparticle, what happens?
- Does the melting temperature change?
- What about a nanoparticle with different facets?
- Could we crystallize nanoparticles from a melt?

Example 3: Sputtering

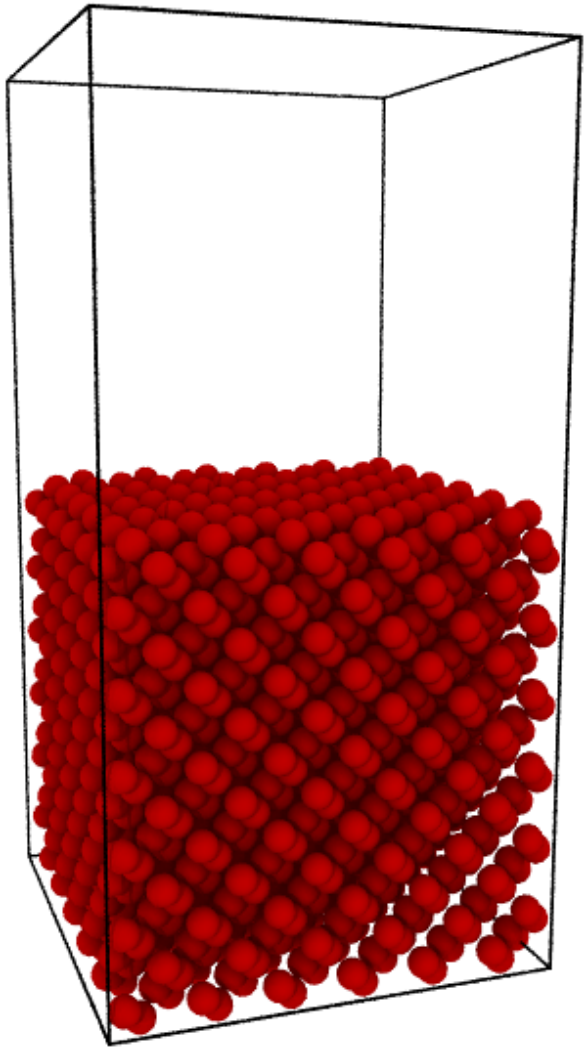
Example 3: This is a more advanced simulation (NEMD) which looks at the sputtering of carbon onto (100) Si.
Like examples 1 and 2 many of the basic commands are the same only a few other commands are added
Potential - Tersoff 1989 version

Listing 8: Regions & Sputter

```
# p = periodic, f= fixed
boundary        p p f
...
...
# Set the force on atoms in bottom to Zero
fix s1 fbot setforce 0.0 0.0 0.0
...
...
#Basic setup sputtering of carbon on Silicon.
fix 4 all deposit 500 2 500 95485 region sput near 1.2 &
vz -0.5 -1.5 target 12.0 12.0 28.0 units box
```

Results

Let it Rain!



Things To Try!

- What happens when you do this at higher temperatures?
- What is the effect of sputtering rate?
- How can we quantitatively measure thickness?
- Is exposed surface important?

Questions?

Please feel free to contact me to setup a 1-on-1 session if you have specific questions or need help running LAMMPS on your system(s).

If you would like to obtain a copy of this presentation you can email me at *stefanb@email.arizona.edu*

The Hidden Gem of LAMMPS

Dr. Axel Kohlmeyer

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

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

akohlmey@gmail.com

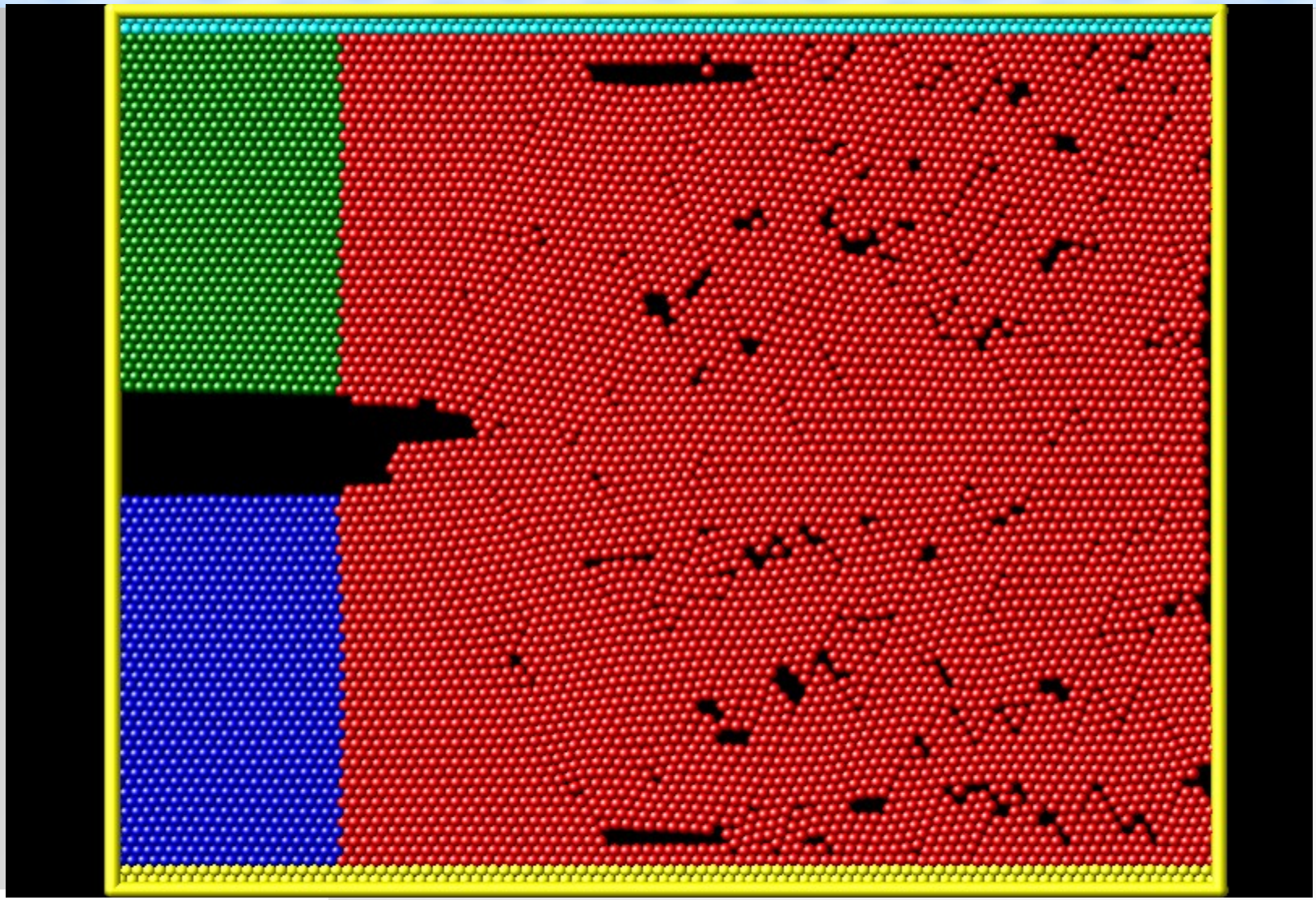
The **`dump image`** command

- Included in LAMMPS since 06/2011
  Author: Nathan Fabian (Sandia)
- Outputs 24-bit RGB images to “netpbm” format
- JPEG format available as compile time option
- Format selection based on filename extension:
  `.jpeg` or **`.jpg`** -> JPEG, any other -> PNM
- Dump filename format: **`image.*.pnm`**
  => one image per snapshot
  => '*' replaced by current timestep number

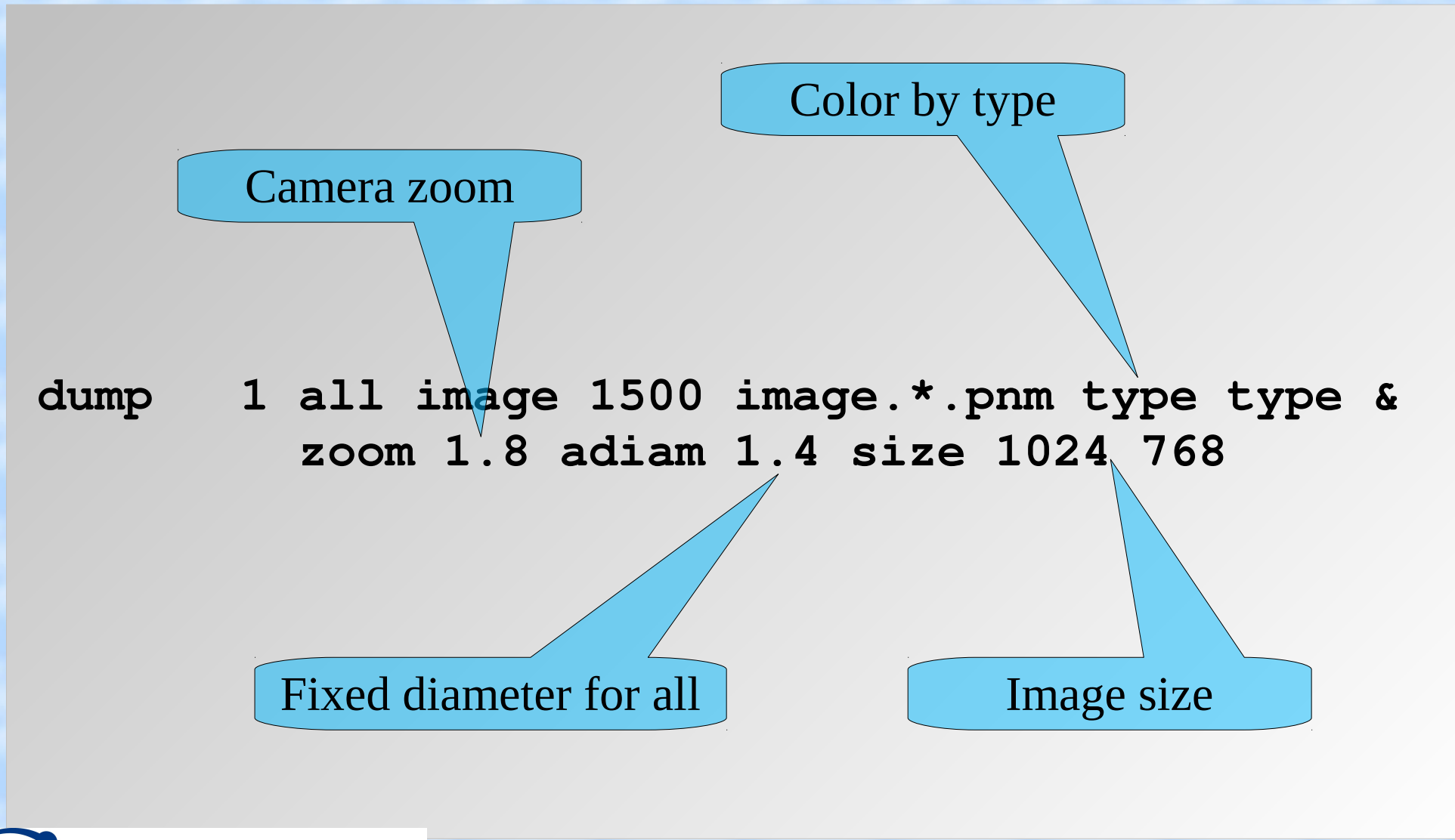
Advantages & Disadvantages

- Advantages:
  - No external visualizer (useful for quick checks)
  - Parallel rendering (with room for improvement)
  - No large data transfers; no (slow) remote display
  - Compressed output with JPEG images
  - Access to per atom properties, computes
- Disadvantages:
  - Cannot change anything after the fact
  - No periodic display, no complex visualizations

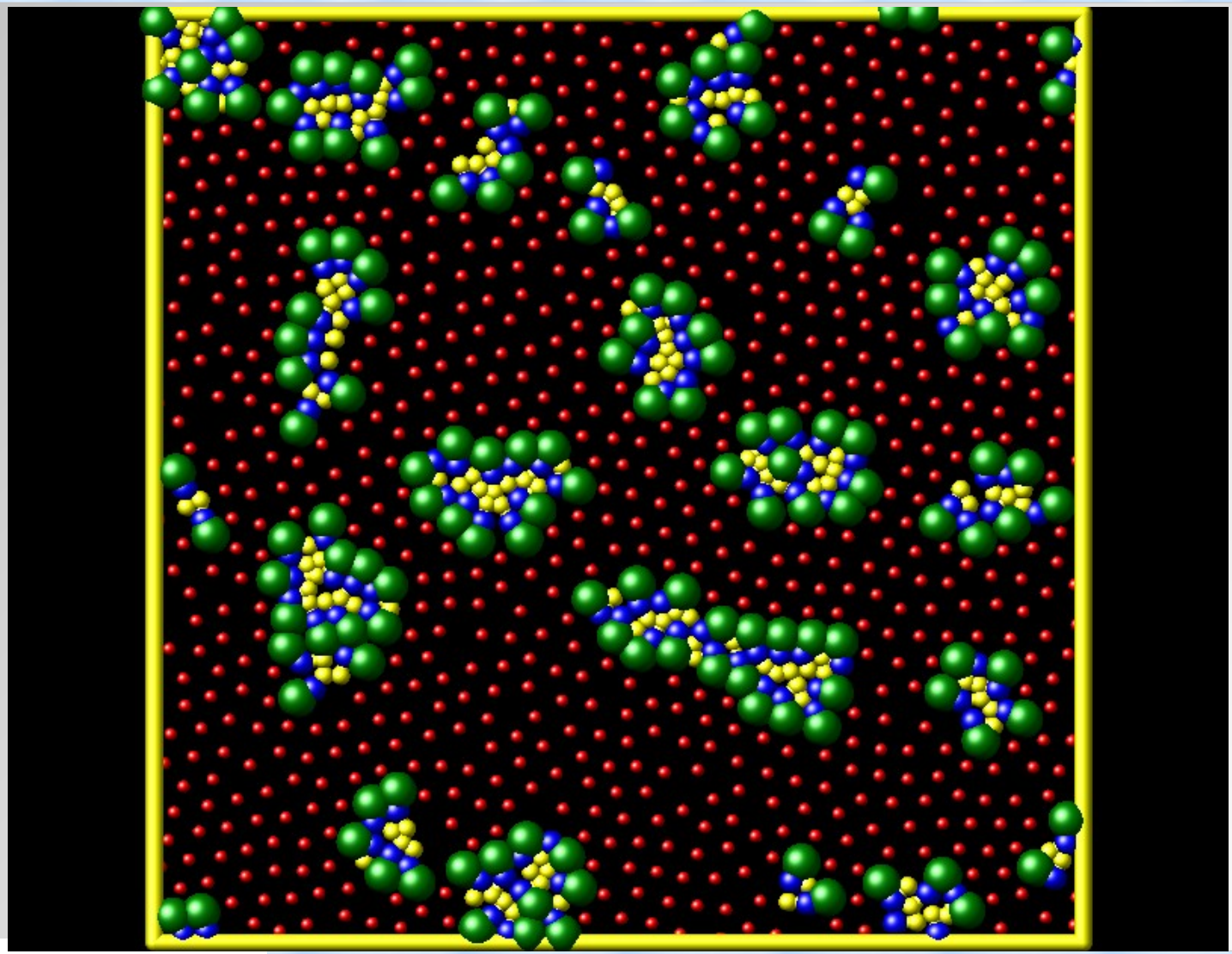
Simple Example: Crack



Settings used



Simple Example: Micelle



Settings used

```
dump    1 all image 10000 image.*.pnm type type &
          zoom 1.6 size 1024 768
dump_modify  1    adiam 1 0.5 adiam 2 1.5 &
                  adiam 3 1.0 adiam 4 0.75
```

Diameter set individually
for each type

Making Movies

- Convert stream of images into animation
- Call converter tools from LAMMPS via **`shell`**
  - Simplest option: **`convert`** tool of ImageMagick => animated GIF, MPEG1 => 8-bit color, low quality
  - HD quality possible via FFmpeg or Mencoder using DivX/MPEG-4 or H.264 compression:

```
shell ffmpeg -y -an -i:v snap-movie.%d.ppm &
  -r 24 -b:v 2400k -c:v libx264 crack.mp4
```

  - FFmpeg requires consecutive image numbering:

```
shell sh -c 't=0 ; for s in image.*.ppm; do
mv $s image.$t.ppm ; t=`expr $t + 1`; done'
```