is paper describes objective technical results and analysis. Any subjective views or opinions that might be expres
the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Governm

Lessons Learned from Develo

SAND2019-3057C

Steve Plimpton
Sandia National Labs

ACS Spring 2019 Meeting
Sustainable Software for Computational Molecular Science
Orlando, FL - April 2019

Sandia National Laboratories

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. Presentation: SAND2017-5724C

National Nuclear Security Administration

What is a community code

- Has a community of users (large or small)
- Has a community of developers (full or part time or casual)

What is a community code

- Has a community of users (large or small)
- Has a community of developers (full or part time or casual)
- ASAP: Make your code as simple as possible to understand and extend
  - easy to do easy things
  - possible to do hard things
  - low barrier for users to become developers
- AMAP: Enable it to be used in as many as possible ways
  - comp materials science is increasingly multiple tools: multiphysics, multiscale, workflows, machine learning
  - allow creative people to do things you didn't envision

What is a community code

- Has a community of users (large or small)
- Has a community of developers (full or part time or casual)
- ASAP: Make your code as simple as possible to understand and extend
  - easy to do easy things
  - possible to do hard things
  - low barrier for users to become developers
- AMAP: Enable it to be used in as many as possible ways
  - comp materials science is increasingly multiple tools: multiphysics, multiscale, workflows, machine learning
  - allow creative people to do things you didn't envision
- More details, light read, bit of humor:
  S. Plimpton and J. Gale,
  Developing Community Codes for Materials Modeling,
  Current Opinion in Solid State and Materials Science (COSSMS), 17 (2013).

ASAP = as simple as possible

Make your code as simple as possible to understand and extend

Idea #1: Simple coding style

- C++, but really mostly C with classes
- Avoid C++ obfuscation:
  - overloaded operators
  - complex templating
  - STL
- Low-level kernels are C-style code
- Low-level data structs are C-style (e.g. Fortran-like arrays)
- Detailed comments for tricky things

ASAP idea #2: Modularity

Via styles = virtual parent + child classes

- Pair, fix, compute, dump, long-range solver, etc
- LAMMPS is ~1M lines of code, only ~5% is core
- ~95% is add-on styles from 100s of contributors

ASAP idea #2: Modularity

Via styles = virtual parent + child classes

- Pair, fix, compute, dump, long-range solver, etc
- LAMMPS is ~1M lines of code, only ~5% is core
- ~95% is add-on styles from 100s of contributors

Write/derive a new style to work with rest of LAMMPS:

- Often need to write just one or a few methods (functions)
- Add 2 files (compute_mymsd.cpp/h) to src dir, re-compile
- One ifdef line in header file:
  ComputeStyle(mymsd,ComputeMyMSD)
- Enables input script command like this:
  compute id upper mymsd com yes average yes
- Produces output that other commands can use as input

ASAP idea #3: Allow tailoring of timestep

In hindsight, this is most flexible feature of LAMMPS

User can control "what" happens "when" within each timestep

communicate ghost atoms

build neighbor list (once in a while)
compute forces — pair, bond, Kspace styles
communicate ghost forces

output to screen and files

ASAP idea #3: Allow tailoring of timestep

In hindsight, this is most flexible feature of LAMMPS

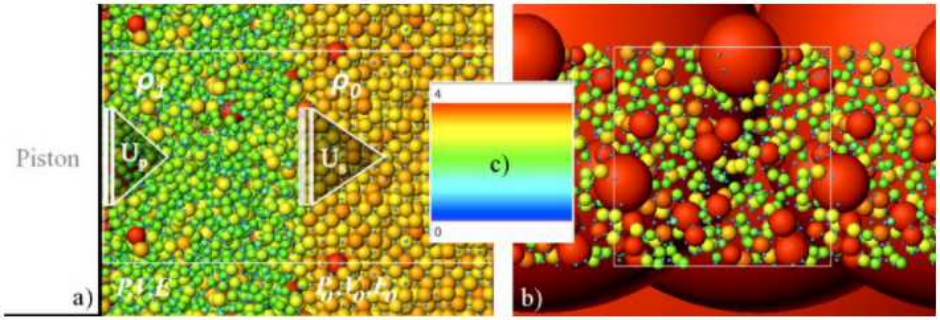
User can control "what" happens "when" within each timestep

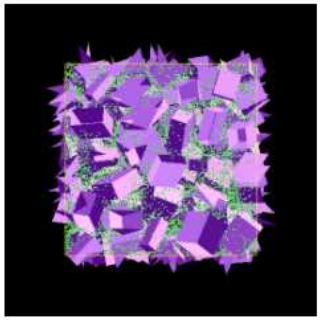
Loop over timesteps:

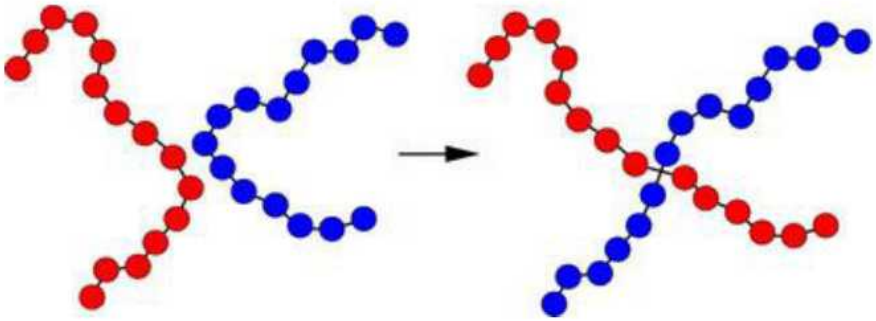
fix initial	NVE ⇒ NPT, rigid-body integration
communicate ghost atoms	
fix neighbor	insert particles, MC moves
build neighbor list (once in a while)	
compute forces	pair, bond, Kspace styles
communicate ghost forces	
fix force	SHAKE, Langevin, walls, springs, gravity
fix final	NVE ⇒ NPT, rigid-body integration
fix end	box deformation, alchemy, diagnostics
output to screen and files	

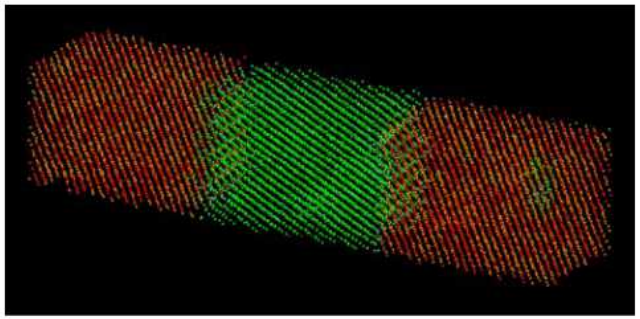
Examples of what "styles" have enabled

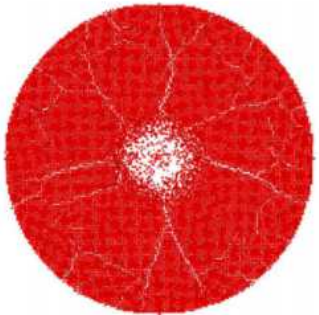
Many were models we never imagined MD could be used for

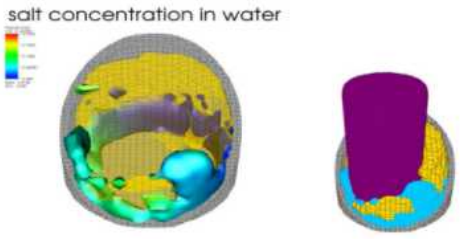


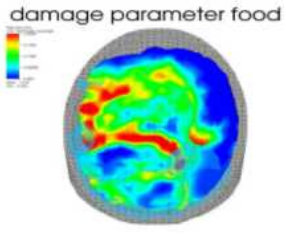


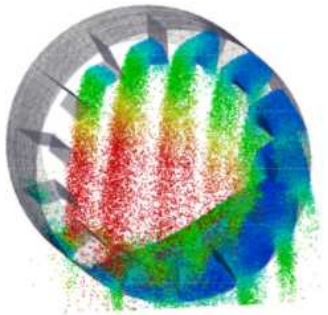












AMAP = as many as possible

Enable your code to be used in as many as possible ways

- Make it a library
- Provide callbacks
- Via Python (or other scripting language)
- Client/server coupling

AMAP idea #1: Make your code a library

- Enables coupled simulations: multi-physics, multi-scale
- Enables use in a complex workflow
- Often just requires bit of re-design at highest level

AMAP idea #1: Make your code a library

- Enables coupled simulations: multi-physics, multi-scale
- Enables use in a complex workflow
- Often just requires bit of re-design at highest level
- Code details
  - no global variables
  - don't use MPI_COMM_WORLD (not using all processors)
- Allow multiple instantiations of your "simulator"
  - another code can call it (from any language)
  - enables multiple embeddings within a higher-scale model
- Provide a library interface (API) to basic functionality
  - C-style interface has maximum portability
  - callable from any language (C++,C,Fortran)
  - API is easily extensible, just add a new function
  - enables scripting (from Python)
  - enables wrapping with a GUI or visualizer

Main.cpp for LAMMPS

```
include "lammps.h"
int main(int argc, char **argv) {
  MPI_Init(&argc,&argv);
  LAMMPS *lammps =
    new LAMMPS(argc,argv,MPI_COMM_WORLD);
  lammps->input->file(); // read input script
  delete lammps;
  MPI_Finalize();
}
```

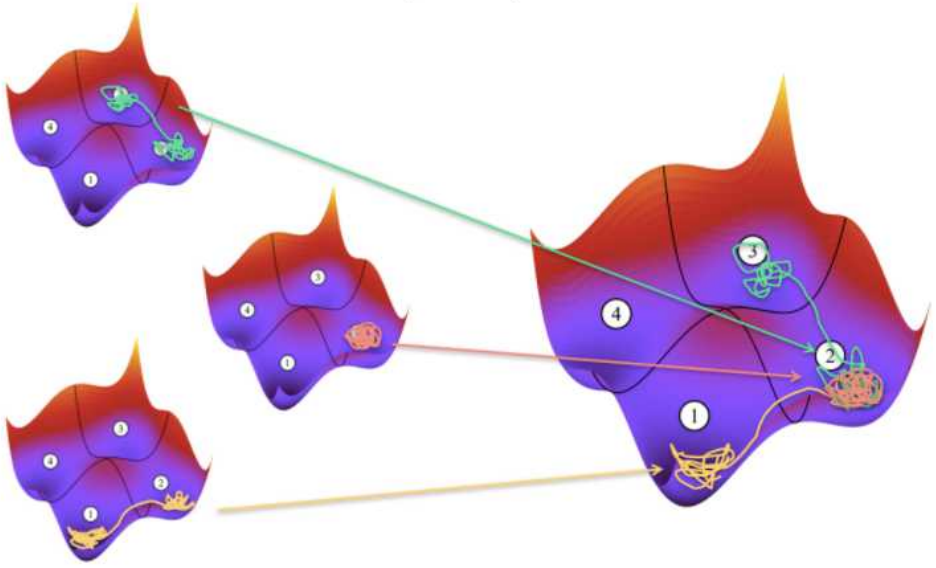
Main.cpp for LAMMPS

```
include "lammps.h"
int main(int argc, char **argv) {
  MPI_Init(&argc,&argv);
  LAMMPS *lammps =
    new LAMMPS(argc,argv,MPI_COMM_WORLD);
  lammps->input->file(); // read input script
  delete lammps;
  MPI_Finalize();
}
```

- LAMMPS itself is a single class
- Just remove main.cpp to compile as a library
- Library call returns ptr to an instance
  - C++ caller: use that ptr
  - anything else: pass ptr back in other lib calls

ParSplice using LAMMPS as an MD library

- Developed by Danny Perez (LANL) and collaborators
  Long-time dynamics through parallel trajectory splicing,
  D Perez, et. al., JCTC, 12, 18-28 (2016).



- Enables μs-ms time-accurate runs with 10^2-10^6 MD replicas
  - for rare-event systems (solid-state)
  - chooses configs to explore PE landscape, run each for few ps
  - detects and catalogs events in a distributed database
  - splices event sequence into long trajectories with correct stats
- Extended LAMMPS library API to facilitate this usage mode

AMAP idea #2: Provide callback functions

Callback = library code calls back to the calling program

AMAP idea #2: Provide callback functions

Callback = library code calls back to the calling program

Example:

- QM code instantiates LAMMPS
- QM code invokes MD run
  - uses fix external, sets a function pointer
- LAMMPS calls back each step to get QM forces
- Trigger at appropriate point in timestep
- QM caller can access LAMMPS data (coords, forces, etc)

AMAP idea #2: Provide callback functions

Callback = library code calls back to the calling program

Example:

- QM code instantiates LAMMPS
- QM code invokes MD run
  - uses fix external, sets a function pointer
- LAMMPS calls back each step to get QM forces
- Trigger at appropriate point in timestep
- QM caller can access LAMMPS data (coords, forces, etc)

- Nir Goldman (LLNL) runs Fortran DFTB+ with LAMMPS in this manner
- *Prebiotic Chemistry Within a Simple Impacting Icy Mixture*, N. Goldman and I. Tamblyn, J Phys Chem A, 117, 5124 (2013).

AMAP idea #3: Wrap/extend your code with Python

- Python = most popular scripting language for scientific computing, workflows, machine learning
- Ctypes module makes it easy to wrap a C-style API
  - every library function can be exposed to Python
  - including callbacks to a Python function
- Multiprocessing module to loop over 1000s of jobs
  $\Rightarrow$ each of which can instantiate a lib
- Mpi4py module enables parallel Python and split of MPI
  $\Rightarrow$ run one or more lib instances in parallel
- Expose data structs in your code to Numpy (no data copy)

```
nlocal = lammps.extract_global("nlocal")
cptr = lammps.extract_atom("x")
coords = np.ctypeslib._array(cptr,shape=(nlocal,3))
```

- Allow Python versions of styles: pair, fix, variable

Example of Python + LAMMPS for neural net potentials

Nick Lubbers (LANL) and Mitch Wood, Aidan Thompson (Sandia)

- Machine learned and neural net (NN) potentials:
  - Goal: quantum accuracy in an empirical potential
  - Analytic energy based only on geometry of atomic configs
  - Fit coefficients to large database of QM configs

Example of Python + LAMMPS for neural net potentials

Nick Lubbers (LANL) and Mitch Wood, Aidan Thompson (Sandia)

- Machine learned and neural net (NN) potentials:
  - Goal: quantum accuracy in an empirical potential
  - Analytic energy based only on geometry of atomic configs
  - Fit coefficients to large database of QM configs
- Fitting via a Python workflow:
  - Manages QM database
  - Passes LAMMPS individual configs:
    - computes geometric descriptors (bispectrum, SOAP via Quip)
    - computes energy, forces, virial for current best-fit potential
  - Optimizes to QM truth, over hyper-parameters

Example of Python + LAMMPS for neural net potentials

Nick Lubbers (LANL) and Mitch Wood, Aidan Thompson (Sandia)

- Machine learned and neural net (NN) potentials:
  - Goal: quantum accuracy in an empirical potential
  - Analytic energy based only on geometry of atomic configs
  - Fit coefficients to large database of QM configs
- Fitting via a Python workflow:
  - Manages QM database
  - Passes LAMMPS individual configs:
    - computes geometric descriptors (bispectrum, SOAP via Quip)
    - computes energy, forces, virial for current best-fit potential
  - Optimizes to QM truth, over hyper-parameters
- Run MD in LAMMPS via Python driver and NN potential
  - LAMMPS calls back to Python when forces needed
  - Python accesses neighbor list, builds list of configs
  - Python invokes NN on each config's descriptors $\Rightarrow$ force
  - Can use PyTorch for GPU-accelerated NN computation

AMAP idea #4: Client/server coupling of 2 scientific codes

- Example #1: ab initio MD, or QM/MM
  - MD is client: sends coords to QM
  - QM is server: computes and returns forces
- Example #2: Monte Carlo using MD as an engine
  - MC is client: makes a complex move (e.g. configurational bias)
  - MD is server: evaluates energy, (optionally) runs MD

AMAP idea #4: Client/server coupling of 2 scientific codes

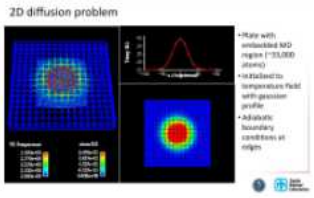
- Example #1: ab initio MD, or QM/MM
  - MD is client: sends coords to QM
  - QM is server: computes and returns forces
- Example #2: Monte Carlo using MD as an engine
  - MC is client: makes a complex move (e.g. configurational bias)
  - MD is server: evaluates energy, (optionally) runs MD
- Four advantages of client/server style coupling:
  1. 2 codes are independent, can be different languages
  2. 2 codes can easily run on different # of procs
  3. Data exchange can be via files, sockets, or MPI
     - sockets means 2 codes can run in different places
  4. Easy to design a data exchange protocol:
     - any MD code can work with any QM code
     - so long as each code implements the protocol

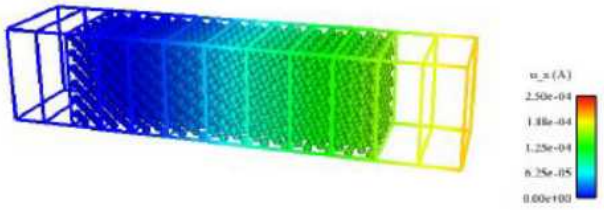
AMAP idea #4: Client/server coupling of 2 scientific codes

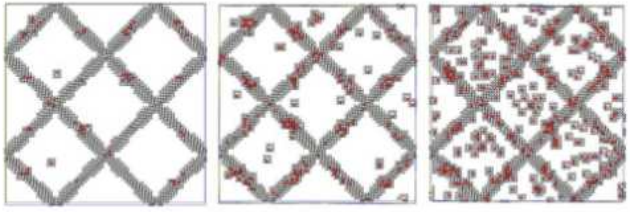
- Example #1: ab initio MD, or QM/MM
  - MD is client: sends coords to QM
  - QM is server: computes and returns forces
- Example #2: Monte Carlo using MD as an engine
  - MC is client: makes a complex move (e.g. configurational bias)
  - MD is server: evaluates energy, (optionally) runs MD
- Four advantages of client/server style coupling:
  1. 2 codes are independent, can be different languages
  2. 2 codes can easily run on different # of procs
  3. Data exchange can be via files, sockets, or MPI
     - sockets means 2 codes can run in different places
  4. Easy to design a data exchange protocol:
     - any MD code can work with any QM code
     - so long as each code implements the protocol
- Recently added client or server options to LAMMPS
- Based on CSlib: http://cslib.sandia.gov (BSD license)
- Using it for ab initio MD with NWChem and VASP

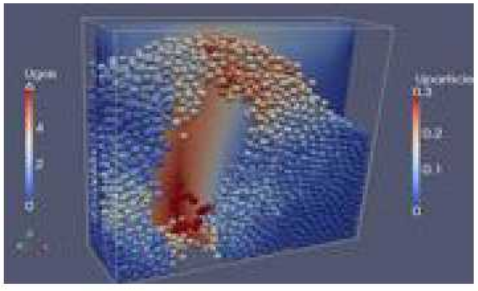
Examples of code-coupling with MD

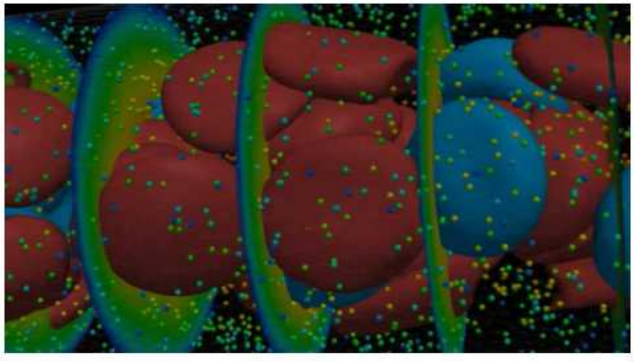
Again, many are models we never imagined MD could be part of

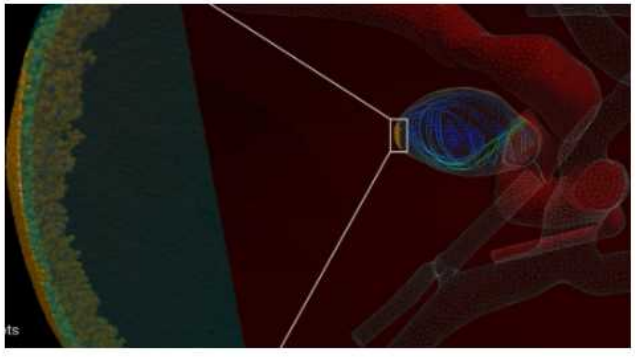












One more lesson learned

- Open-source licensing: how others distribute your code
  - GPL = anything added must be distributed openly via GPL
  - LGPL = proprietary code can be added & not distributed; changes to your LGPL code remain openly distributed
  - BSD = do whatever you want with my software: change it, make it closed source, sell it
- LAMMPS has been GPL for 15 years, now moving to LGPL

One more lesson learned

- Open-source licensing: how others distribute your code
  - GPL = anything added must be distributed openly via GPL
  - LGPL = proprietary code can be added & not distributed; changes to your LGPL code remain openly distributed
  - BSD = do whatever you want with my software: change it, make it closed source, sell it
- LAMMPS has been GPL for 15 years, now moving to LGPL

Paul Saxe (while at Materials Design, Inc):

"If you want your field (and its software) to be driven by economics and business interest, rather than government subsidy (DOE, NSF, NIH, ...), then companies have to be able to add value and protect their investment."

One more lesson learned

- Open-source licensing: how others distribute your code
  - GPL = anything added must be distributed openly via GPL
  - LGPL = proprietary code can be added & not distributed; changes to your LGPL code remain openly distributed
  - BSD = do whatever you want with my software: change it, make it closed source, sell it
- LAMMPS has been GPL for 15 years, now moving to LGPL

Paul Saxe (while at Materials Design, Inc):

"If you want your field (and its software) to be driven by economics and business interest, rather than government subsidy (DOE, NSF, NIH, ...), then companies have to be able to add value and protect their investment."

- GPL is a dis-incentive for this, LGPL is OK,
  BSD would be ideal (for companies, not for us)

One more lesson learned

- Open-source licensing: how others distribute your code
  - GPL = anything added must be distributed openly via GPL
  - LGPL = proprietary code can be added & not distributed; changes to your LGPL code remain openly distributed
  - BSD = do whatever you want with my software: change it, make it closed source, sell it
- LAMMPS has been GPL for 15 years, now moving to LGPL

Paul Saxe (while at Materials Design, Inc):

"If you want your field (and its software) to be driven by economics and business interest, rather than government subsidy (DOE, NSF, NIH, ...), then companies have to be able to add value and protect their investment."

- GPL is a dis-incentive for this, LGPL is OK,
  BSD would be ideal (for companies, not for us)
- Lesson: Make licensing decisions early in your software's life
- Can re-license later, but may be painful

Thanks and links

- Thanks to LAMMPS co-developers:
  Aidan Thompson, Stan Moore (Sandia)
  Axel Kohlmeyer (Temple U), many others

- LAMMPS: http://lammps.sandia.gov
- CSlib: http://cslib.sandia.gov

- S. Plimpton and J. Gale,
  Developing Community Codes for Materials Modeling,
  Current Opinion in Solid State and Materials Science
  (COSSMS), 17, 271-276 (2013).