Building a reusable LAMMPS script library

Craig Tenney[1] Edward Maginn[2]

[1]Sandia National Laboratories

[2]Department of Chemical and Biomolecular Engineering
University of Notre Dame

August 2011

UNIVERSITY OF NOTRE DAME

Outline

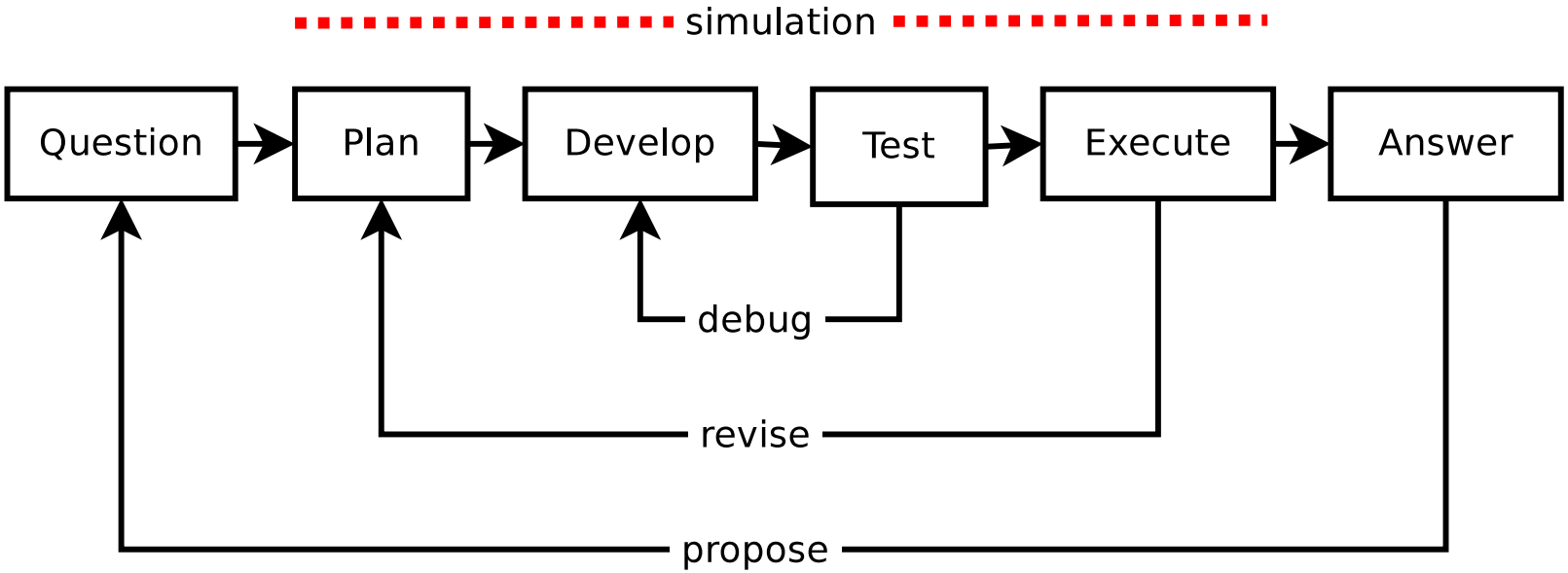
motivation

Something done once will need doing again. . .
after I've forgotten how to do it.

motivation

Something done once will need doing again. . .
after I've forgotten how to do it.

Research workflow:



Goal: loop more efficiently

- code re-use
- friendly documentation

simulation strategy v0.1

Model: one script per job

- create new script when needed
- copy and tweak existing scripts when possible

Pros:

- easiest implemention
- clear connection between job and script

Cons:

- scattered script collection
- no clear evolutionary record
- frequent reinvention and regression

simulation strategy v0.2

Model: one script for many jobs

- simulation parameters are script variables
- variables are passed from command line (e.g.“lammps -var name value -in myscript”)

Pros:

- improved code re-use
- fairly easy implementation

Cons:

- weaker connection between job and script
- interactive and batch jobs handled differently

current simulation strategy v0.3

Model: one script *template* per job type

- create custom script from general-purpose template (similar to building web pages dynamically)
- run LAMMPS with custom script

Pros:

- clear connection between job and script

Cons:

- requires specialized front-end

implementation details

two primary components

- library of LAMMPS template scripts
  (simulation setup, modification, and production tasks)
- front-end to create custom script from template
  (shell script)

implementation details

template = script with parameters set via "index" variables

- variables should have safe-ish default values
- runtime script is template copy with *new* variable values
  (e.g. "variable myvar index 0" in template
  becomes "variable myvar index 100" in new script)

implementation details

template = script with parameters set via "index" variables

- variables should have safe-ish default values
- runtime script is template copy with *new* variable values
  (e.g. "variable myvar index 0" in template
  becomes "variable myvar index 100" in new script)

template library

- for generality, most templates expect restart file as input
- forcefield-specific setup templates use data files
- template can include special "help" comment section

implementation details

front-end

- provide optional “help” framework
- collect parameter variables from command line
- create custom script from specified template

help example: available templates

```
$ lmp_mkscript
```

help example: available templates

```
$ lmp_mkscript
Available templates:

alter.lammpsin
any.lammpsin
data2restart.lammpsin
minimize.lammpsin
rnemd.lammpsin
sllod.lammpsin
therm_cond.lammpsin

Usage:
lmp_mkscript template [-outname=NAME] [-variable=VALUE]...
```

help example: template options

```
$ lmp_mkscript any.lammpsin
```

help example: template options

```
$ lmp_mkscript any.lammpsin
# NVE, NVT, NPT, or NPH ensemble

 # these variables must be set from the command line:
variable        outname index basename_of_output_files
variable        infile index full_name_of_restart_file
 # ensemble/integrator: 1 = NVE, 2 = NVT, 3 = NPT, 4 = NPH
variable        ensemble index -1
variable        numsteps index 0

 # initial and final thermostat setpoints (if applicable)
variable        beg_temp index 300
variable        end_temp index $beg_temp
 # initial and final barostat setpoints (if applicable)
variable        beg_press index 1
variable        end_press index $beg_press
```

help example: template options (cont'd)

```
 # output options:
 # thermo_freq: output thermo snapshots at specified interv
variable        thermo_freq index 1000
 # dump_freq: dump trajectory snapshots at specified interv
variable        dump_freq index 100000
 # dump_detail: 1 = positions, 2 = +velocities, 3 = +forces
variable        dump_detail index 1
 # press_freq > 0: output pressure tensor data at specified
variable        press_freq index 0
 # mol_freq > 0: output molecule data at specified interval
variable        mol_freq index 0
```

help example: template options (cont'd)

```
 # detail options:
 # kspace: 1 = pppm, 2 = ewald, 3 = ewald/n, other = off
variable        kspace index 1
 # timestep: aka dt, e.g. 1 fs
variable        timestep index 1
 # restart_freq: save restart file every N steps (0 = only
variable        restart_freq index 1000000
 # new_step >= 0: reset current step to new_step
variable        new_step index -1
 # density > 0: rescale volume to result in specified dens
 # note for 'real' units: g/cm^3 * 0.6022 = g/mol/A^3
variable        density index 0

# ___end command line variable section___
Usage:
lmp_mkscript any.lammpsin [-outname=NAME] [-variable=VALUE]
```

script creation example: NVT @350K to density setpoint

```
$ lmp_mkscript any.lammpsin -outname=workshop \
-infile=myrestart.0 -ensemble=2 -numsteps=100000 \
-beg_temp=350 -density=0.6022 -dummy=nothing
```

script creation example: NVT @350K to density setpoint

```
$ lmp_mkscript any.lammpsin -outname=workshop \
-infile=myrestart.0 -ensemble=2 -numsteps=100000 \
-beg_temp=350 -density=0.6022 -dummy=nothing

template: /home/cmtenne/apps/dev/lammps.scripts/scripts/any
  variable substitution: outname=workshop
  variable substitution: infile=myrestart.0
  variable substitution: ensemble=2
  variable substitution: numsteps=100000
  variable substitution: beg_temp=350
  variable substitution: density=0.6022
WARNING: unknown variable 'dummy'
outscript: workshop.lammpsin
```

script snippets: setup

```
# set up simulation
read_restart    $infile
if "$kspace == 1" then "kspace_style pppm 0.0001"
if "$kspace == 2" then "kspace_style ewald 0.0001"
if "$kspace == 3" then "kspace_style ewald/n 0.0001"
neighbor        2.0 bin
neigh_modify    delay 5 check yes
timestep        $timestep

# rescale simulation box during simulation
if "$density <= 0" then "jump SELF no_density"
  variable vscale equal "(mass(all)/v_density/vol)^(1/3)"
  fix RESCALE all deform 10 &
    x scale $vscale y scale $vscale z scale $vscale
label no_density
```

script snippets: integrator

```
# pick desired ensemble/integrator
variable tdamp equal 100*$timestep
variable pdamp equal 1000*$timestep
if "$ensemble == 1" then &
  "fix NVE all nve" &
elif "$ensemble == 2" &
  "fix TFIX all nvt temp $beg_temp $end_temp $tdamp" &
elif "$ensemble == 3" &
  "fix TFIX all npt temp $beg_temp $end_temp $tdamp
    iso $beg_press $end_press $pdamp" &
elif "$ensemble == 4" &
  "fix TFIX all nph iso $beg_press $end_press $pdamp" &
else &
  "print 'Invalid ensemble/integrator!!!'" "exit"
```

script snippets: molecular data

```
# molecule properties
if "$mol_freq <= 0" then "jump SELF no_mol_props"
  compute MOLcom all com/molecule  # x y z
  variable mass atom mass
  variable px atom mass*vx
  variable py atom mass*vy
  variable pz atom mass*vz
  compute ke all ke/atom
  compute MOLkin all atom/molecule &
    v_mass v_px v_py v_pz c_ke
  fix MOL all ave/time 1 1 $mol_freq c_MOLcom c_MOLkin &
    mode vector file $outname.mol ave one &
    title3 "# mol x y z mass px py pz ke_total"
label no_mol_props
```

script snippets: run

```
# output useful info
variable Mass equal mass(all)
print "total_mass $Mass"
variable Charge equal charge(all)
print "total_charge $Charge"

# run simulation
if "$restart_freq > 0" then &
  "restart $restart_freq $outname.restart"
run             $numsteps
if "$restart_freq >= 0" then &
  "write_restart $outname.restart.*"
```

closing

LAMMPS script templates for repetitive tasks

- small initial development overhead
- long-term benefits
  - efficient code re-use
  - pseudo “man” pages