Modifying and Extending SPARTA

Steve Plimpton
Sandia National Labs
sjplimp@sandia.gov

DSMC15 Short Course
Sept 2015 - Kapaa, Hawaii

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.

SPARTA is designed to be extensible

- Enabled by object orientation and SPARTA styles
- See doc/Section_modify.html for overview
- But before you start writing code,
  you can send us (Steve, Michael) an email and ask ...
  1. can SPARTA already do this?
  2. how hard would it be to implement?
  3. is my plan a good way to implement this?
- We can give you some feedback on your idea

5 ways to modify SPARTA: easy to hard

1. Input script options
   - syntax is a simple programming language
   - if (then else), jump, next, label, include, print
   - use of variables
   - shell command to invoke other programs
2. Write a Python script that invokes SPARTA
   - instantiate one or more SPARTA instances
   - invoke SPARTA input script commands
   - invoke functions in SPARTA library interface (extensible)
   - grab SPARTA data, alter it, pass it back
   - see python directory of distro for examples
3. Couple to another code
4. Small changes to existing customizable files
5. Write code for a new style

#3 Couple to another code

- Other code calls SPARTA
  - Section howto 4.6: Library interface to SPARTA
  - C-style, so can be called from C++/C/Fortran/Python
  - easy to extend, just add functions to library.cpp/h
  - add wrapper method to python/sparta.py for Python
  - umbrella Python script can invoke SPARTA and other code, pass info between them

#3 Couple to another code

- Other code calls SPARTA
  - Section howto 4.6: Library interface to SPARTA
  - C-style, so can be called from C++/C/Fortran/Python
  - easy to extend, just add functions to library.cpp/h
  - add wrapper method to python/sparta.py for Python
  - umbrella Python script can invoke SPARTA and other code, pass info between them

- SPARTA calls other code (e.g. ParaView Catalyst)
  - Section howto 4.7: Coupling SPARTA to other codes
  - wrap the other code in a compute or fix (stay tuned)
  - pass appropriate SPARTA data (e.g. grid data)
  - other code returns new data (e.g. to alter BC)
  - when build SPARTA, link with the other code

#4 Make small changes to existing files

Look for customize comments in appropriate src file

#4 Make small changes to existing files

Look for customize comments in appropriate src file

1. Add a keyword to stats_style or dump particle/grid/surf
   - see src/stats.cpp or src/dump_*.cpp
   - complicated calculation better done as new Compute
2. Add a new function to variable formulas
   - see src/variable.cpp
   - math functions, special functions, math operators, etc
   - follow syntax rules for args of similar functions

#5 Write code for a new style

- A style is a child class derived from a parent class
- ~50% of SPARTA code base is add-on styles

#5 Write code for a new style

- A style is a child class derived from a parent class
- ~50% of SPARTA code base is add-on styles
- 9 kinds of styles (ls src/style*.h; cat style_compute.h)
  - collision model = collide style
  - gas reaction model = react style
  - surface collision model = surf_collide style
  - surface reaction model = surf_react style
  - diagnostics = compute style
  - operate within timestep = fix style
  - geometric region = region style
  - output = dump style
  - input command = command style
    - create_box, balance_grid, run, ...

Steps to write a new style

- Manual Chapter 8: Modifying & Extending SPARTA
- Examine the parent *.h file which defines style interface
  - class variables the child class sets and/or uses
  - methods a child class must define (pure virtual)
  - optional methods a child class can define (virtual)
- Find an existing *.cpp/h child file similar to what you want
  - write a new child similar to that one
  - or derive from it if only need limited changes
- Create fix_foo.cpp/h, drop in src dir, re-build
- Can now use fix ID foo ... in input script

```
#ifdef FIX_CLASS
FixStyle(balance,FixBalance)
#else
class FixBalance :  public Fix ...
#endif
```

Adding a new surface collision model

- Two current models: specular and diffuse
  - surf_collide_specular.cpp/h
- Parent class interface: surf_collide.h
  - virtual void init() = 0;
  - virtual Particle::OnePart *
    collide(Particle::OnePart *&, double *, int) = 0;
  - virtual void dynamic()
- Must provide two pure virtual methods, third is optional

Specular surface collision model

Input script: surf_collide upper specular
See surf_collide_specular.cpp/h, 81 lines (half comments)

- constructor - invoked when input script command is read
  - if (narg != 2)
    error->all(FLERR,"Illegal surf_collide specular command");
- collide() - invoked when particle hits a surface element
  - 22 lines of code (w/out comments)
  - 3 inputs: particle, surface norm, index of reaction model

```
collide(OnePart *&ip, double *norm, int isr)
  if (isr >= 0) do reaction (may create new jp)
  if (ip) MathExtra::reflect3(ip->v,norm);
  if (jp) MathExtra::reflect3(jp->v,norm);
  // call ambipolar fix (if exists) so can bookkeep
  return jp;
```

Diffuse surface collision model

Input script: surf_collide heatwall diffuse 300.0 0.8
See surf_collide_diffuse.cpp/h, 281 lines

- constructor
  - args for temperature & accommodation coeff
  - temperature can be time-dependent variable
  - optional args for translation/rotation of surf element
- collide() - invoked when particle hits a surface element
  - Bird formula for velocity after diffuse reflection
  - split velocity into normal/tangential components
  - randomize new tangential component
  - account for surface translation/rotation
  - allow for possible surface reaction
- dynamic() - if variable Temp, update once per timestep

Adding a new compute

Recall that computes calculate some property of system
Always for the current timestep

Adding a new compute

Recall that computes calculate some property of system
Always for the current timestep

If you want to calculate a ...

- Global result:
  - write a compute_scalar(), compute_vector(), and/or compute_array() methods
  - store result in scalar, vector[i], array[i][j] (vars in compute.h)
  - example: compute_temp.cpp
    - loop over particles
    - MPI_Allreduce of KE $\Rightarrow$ scalar temperature
- Per particle result:
  - write a compute_per_particle() method
  - store result in vector_particle[i], array_particle[i][j]
  - example: compute_ke_particle.cpp
    - loop over particles
    - ke[i] = 0.5*mass * (v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);

Adding a new compute for grid or surface properties

- Per grid cell result:
  - write a compute_per_grid() method
  - store result in vector_grid[i], array_grid[i][j]
  - example: compute_grid.cpp
    - loop over particles
    - u[i] += mass*v[0];
    - norm_mass[igroup][icell] += mass;
    - used to calculate $U = \sum_i (m_i V x_i) / \sum_i (m_i)$
- Per surface element result:
  - write a surf_tally() method
    - called when particle hits surface element
    - args = element, before/after particle properties
  - store result in vector_surf[i], array_surf[i][j]
  - example: compute_surf.cpp
    - use mass, Vpre, Vpost to accumulate pressure P
    - $\Delta p = mass * (V_{post} - V_{pre})$
    - $P \mathrel{+}= (\Delta p \bullet \hat{n}) / (\mathrm{Area}\, \Delta t / \mathrm{Fnum})$

Adding a new fix

- Fixes can insert operations into the timestep loop
- Via start_of_step() and end_of_step() methods
- Provide setmask() method:

  mask |= START_OF_STEP;

Loop over timesteps:

move particles
communicate particles
collisions and reactions

output to screen and files

Adding a new fix

- Fixes can insert operations into the timestep loop
- Via start_of_step() and end_of_step() methods
- Provide setmask() method:

  mask |= START_OF_STEP;

Loop over timesteps:

fix start-of-step	emit/face, emit/face/file, emit/surf, ...
move particles	
communicate particles	
collisions and reactions	
fix end-of-step	ave/time, balance, adapt, move/surf, ...
output to screen and files	

Other operations fixes can perform

See fix.h for details

- Invoke & access output from computes or variables
  - fix ave/time, fix ave/grid, fix ave/surf
- Create output, similar to computes
  - fix ave/time, fix ave/grid, fix ave/surf
  - global, per-particle, per-grid-cell, per-surf vectors/arrays
- Define new per-particle attributes
  - example: fix ambipolar
  - ionambi = integer flag for ion or not
  - velambi[3] = velocity of electron associated with ion

Contribute your new code to the SPARTA distro

- Why release it as part of SPARTA?
  - open source philosophy
  - fame and fortune, name on author page and in source code
  - acquire users of your feature
    - find and fix bugs
    - extend its functionality
    - become collaborators
- Key points for a speedy release:
  - doc pages for new commands, in SPARTA format (doc/*.txt)
  - avoid changes (if possible) to core SPARTA files
  - ask ahead if you think changes are necessary
- Then just email us the files