More Advanced SPARTA Capabilities

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.

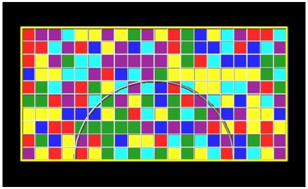
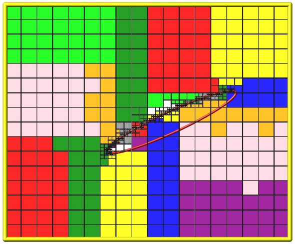
Outline - plus 15-minute break at ∼3 PM!

1. Parallelism
   - SPARTA algorithms and strong scaling
2. Input script options
   - variables
   - if/then/else and looping
   - running multiple simulations
3. How to do the following:

- boundary conditions
- particle species and mixtures
- particle creation
- gas-phase collision and chemistry models
- working with surfaces
  - create, read, move, delete
  - collision/chemistry models
- adaptive gridding
- load balancing
- ambipolar approximation
- diagnostic computes
- time-averaging of stats
- stats and dump output
- use SPARTA as a library

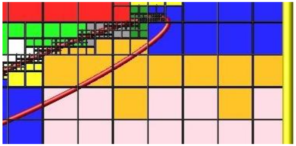
Parallelization in SPARTA

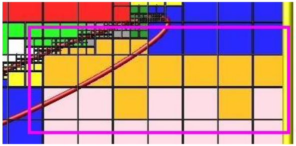
- Every processor owns copy of:
  - all parent grid cells (could be one or many)
  - geometry for all surface elements
- Each processor owns 1/P subset of:
  - child grid cells (those not further subdivided)
  - particles (inside its child cells)
  - grid stats (for its child cells)
  - surface element stats
- Assignment of cells to processors can be scattered or clumped

Ghost grid cells

- Each processor also owns ghost cells
  - ghost cell = copy of another proc's cell with geometry/surf info
  - enables particles to complete move without communication
  - for scattered decomp: each proc owns copy of all cells
  - for clumped decomposition: only nearby cells are ghosted

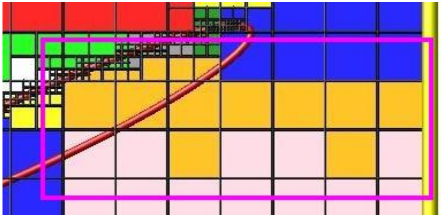




- You set cutoff distance for ghost cells
  - command: global gridcut 0.1
  - good setting = max particle move distance for 1 timestep

Parallel communication each timestep

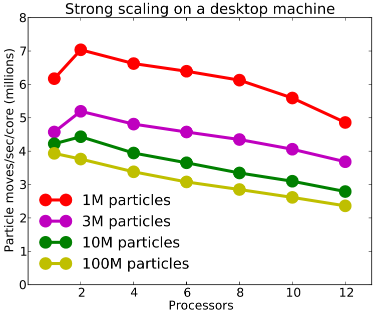
- Each timestep:
  1. each particle is ray-traced thru one or more grid cells
  2. can bounce off surface elements, perform surface chemistry
  3. send particles to new processors that own final grid cells



- If cutoff is sufficient: one communication after all moves
  else: multiple move/comm iterations within timestep

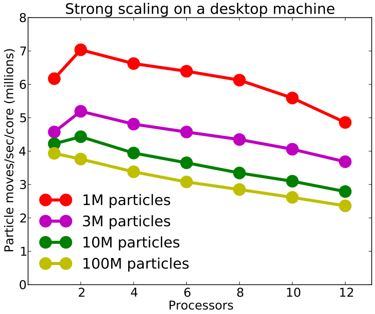
Parallel performance

- Should get parallel speed-up so long as enough particles/proc
- Parallel results ≃ serial results statistically, due to RNGs
- Bench directory has 3 test problems
  - free molecular flow, collisional flow, flow around sphere
  - spa_icc -v x 100 -v y 100 -v z 100 < in.collide



Parallel performance

- Should get parallel speed-up so long as enough particles/proc
- Parallel results $\simeq$ serial results statistically, due to RNGs
- Bench directory has 3 test problems
  - free molecular flow, collisional flow, flow around sphere
  - spa_icc -v x 100 -v y 100 -v z 100 < in.collide



- See http://sparta.sandia.gov/bench.html for more data
- Michael will discuss large problem scalability on large HPC

Define variables in input scripts

- Styles: index, loop, equal, particle, world, ...
  - variable name index run1 run2 run3 run4
  - variable i loop 100
  - variable frac equal 100.0*c_count[2]/np
  - variable vmag particle sqrt(vx*vx+vy*vy+vz*vz)
  - variable temp world 200.0 250.0 300.0 350.0
  - index style vars can be set from command line

Define variables in input scripts

- Styles: index, loop, equal, particle, world, ...
  - variable name index run1 run2 run3 run4
  - variable i loop 100
  - variable frac equal 100.0*c_count[2]/np
  - variable vmag particle sqrt(vx*vx+vy*vy+vz*vz)
  - variable temp world 200.0 250.0 300.0 350.0
  - index style vars can be set from command line
- Formulas can be complex
  - see doc/variable.html
  - stats keywords (step, np, vol, ...)
  - math operators & functions (sqrt, log, cos, ...)
  - various special functions (min, ave, trap, stride, stagger, ...)
  - particle vectors (x, vx, mass, ...)
  - output from computes, fixes, other variables
- Formulas can be time-dependent

4 ways to use variables in input scripts

1. Substitute in any command via $x or ${myVar}
   - global fnum ${Fnum} nrho ${Nrho} vstream $v 0 0
   - read_surf sdata.${fname}
2. Immediate formula evaluation via $() syntax:
   - avoids need to define separate one-time variable
   - variable xmid equal (xlo+xhi)/2
   - region 1 block ${xmid} EDGE INF INF EDGE EDGE
   - region 1 block $((xlo+xhi)/2) EDGE INF INF EDGE EDGE
3. Next command increments a variable to next value
4. Some commands allow variables as arguments
   - surf_collide diffuse v_temp 1.0
   - dump_modify every v_every
   - dump image ... view v_theta v_phi ...

More options for input scripts

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

More options for input scripts

- Filename options:
  - dump.*.% for per-snapshot or per-processor output
  - read_surf sdata.huge.gz
  - read_restart old.restart.*
- If/then/else via if command
- Insert another script via include command
  - useful for long list of parameters
- Invoke a shell command or external program
  - shell cd subdir1
  - shell my_analyze out.file $n ${param}
- Looping via next and jump commands
  - increment a parameter, run some more, repeat
  - loop over multiple runs (next slide)
- Various ways to run multiple simulations from one script
  - see doc/Section_howto 4.3

Example script for multiple runs

Run 8 successive simulations on any number of processors:

```
variable  a loop 8
variable  rho index 1.0e18 4.0e18 1.0e19 4.0e19 ...
log       log.$a
global    nrho ${rho}
...
compute   myGrid grid all n temp
dump      1 all grid 1000 dump.$a id c_myGrid
run       100000
clear
next      rho
next      a
jump      in.flow
```

Example script for multiple runs

Run 8 successive simulations on any number of processors:

```
variable   a loop 8
variable   rho index 1.0e18 4.0e18 1.0e19 4.0e19 ...
log        log.$a
global     nrho ${rho}
...
compute    myGrid grid all n temp
dump       1 all grid 1000 dump.$a id c_myGrid
run        100000
clear
next       rho
next       a
jump       in.flow
```

Run 8 simulations on 3 partitions until finished:

- change a & rho to universe-style variables
- mpirun -np 12 spa_mpi -p 3x4 -in in.flow

Units and boundary conditions

- Units command
  - two choices: cgs or si (maybe more at some point)
  - all input in units (up to you), all output in units
- Dimension command = 2 or 3
- Boundary command for each of 6 box faces (4 in 2d)
  - outflow, periodic, reflective, axisymmetric, surface
- Each box face can also be inlet for particles (details later)
  - only really makes sense for outflow & surface boundaries

Units and boundary conditions

- Units command
  - two choices: cgs or si (maybe more at some point)
  - all input in units (up to you), all output in units
- Dimension command = 2 or 3
- Boundary command for each of 6 box faces (4 in 2d)
  - outflow, periodic, reflective, axisymmetric, surface
- Each box face can also be inlet for particles (details later)
  - only really makes sense for outflow & surface boundaries
- Surface boundary ⇒ assign collision, reaction models
  - surf_collide command: specular or diffuse
  - surf_react command: clone/delete or file of reactions
- Axisymmetric only allowed for ylo face
  - requires ylo = 0.0 in create_box command
  - particles move in 3d, projected back into 2d plane
  - line segments become 3d arcs for collision purposes
  - see global weight command for radial cell weighting
  - see doc/Section_howto 4.2

Particle species

- Species command
  - species ../data/air.species O2 N2 O ...
  - see data directory for provided species files

- Sample lines from species data file (9 attributes/species):

```
# ID MolWt/Mass RotDof/Rel VibDof/Rel/T SpecWt Q
O2 32.00 5.31E-26 2 0.2 2 5.58659E-5 2256.0 1.0 0.0
N2 28.016 4.65E-26 2 0.2 2 1.90114E-5 3371.0 ...
O 16.00 2.65E-26 0 0.0 0 0.0 0.0 1.0 0.0
...
```

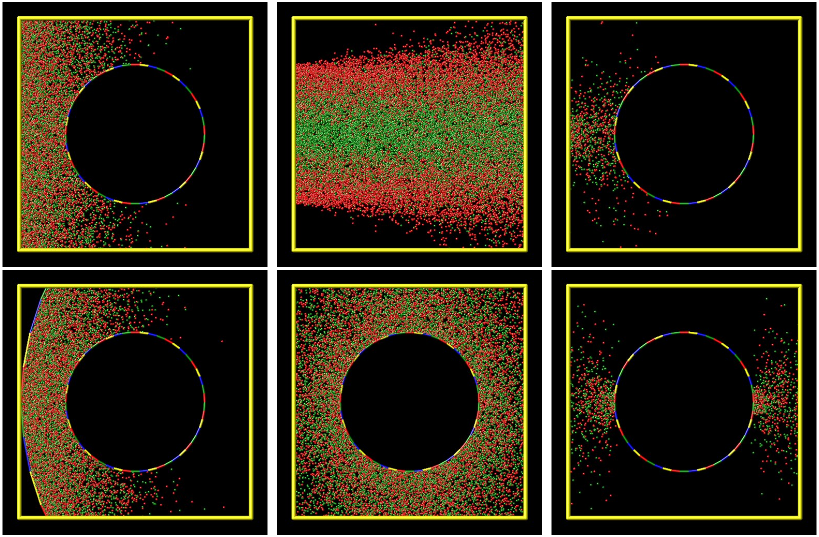
Particle mixtures

- Mixture command
  - mixture = collection of species, with attributes
  - define as many mixtures as you like
- Mixture attributes
  - global: nrho, temperature, stream vector
  - per-species: number fraction
  - groups = subsets of species within mixture
- Mixture IDs are inputs to other commands:
  - fix emit/face mix-ID: create nrho/temp/vstream particles
  - collide vss mix-ID: define collision groups
  - compute grid mix-ID: tally grid stats by group

Particle creation

- Create_particles command:
  - N particles, or according to Fnum and mixture properties
  - only in grid cells not cut by surfs
- Fix emit commands (multiple if desired):
  - emit/face = emit from one or more simulation box faces
  - emit/file = emit from face with profile from file
  - emit/surf = emit from surface elements
  - emit/surf/file (future): surf emission profile from file
  - options to limit by region, flow/normal dir, subset of surfs

Fix emit sampler - see examples/emit



Gas-phase collision models

- Collide command
  - only VSS and VHS models at this point (may add others)
  - parameters come from *.vss files (see data dir)
  - VHS is simply VSS with $\alpha = 1.0$
- Sample lines from VSS data file (8 params/species):

```
# ID diam ω Tref α Zrot T* C1 C2
O2 3.96E-10 0.77 273.15 1.4 16.5 113.5 56.5 153.5
N2 4.07E-10 0.74 273.15 1.6 18.1 91.5 9.1 220.0
O 3.0E-10 0.80 273.15 1.0 0.0 0.0 0.0 0.0
...
```

- Collide_modify command
  - options for rotational and vibrational energy tracking
  - options for computing per-grid-cell collision counts

Gas-phase chemistry models

- React command
  - Bird TCE or QK or hybrid TCE/QK models (may add others)
  - TCE = total collision energy model
  - QK = quantum kinetic model
  - reactions come from *.tce files (see data dir)
- Dissociation:
  - O2 + N ⇒ O + O + N
  - D A 1.0 8.197e-19 1.660e-8 -1.5 -8.197e-19
- Exchange:
  - NO + O ⇒ O2 + N
  - E A 0.0 2.684e-19 1.389e-17 0.0 -2.684e-19
- Ionization:
  - N + e ⇒ N+ + e + e
  - I A 0.0 2.322e-18 4.1513E4 -3.82 -2.322e-18
- Recombination: coming soon
  - requires 3rd collision partner for energy conservation

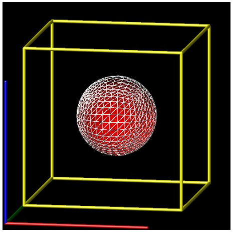
Surface element files

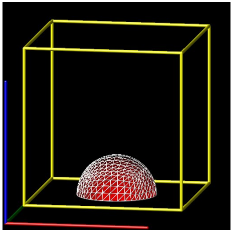
- Surface files have simple syntax: see data/sdata.shuttle
- Triangles for 3d, line segments for 2d or axisymmetric

```
# shuttle file (removed some blank lines)
310 points
616 triangles

Points
1 3.070224 -0.119728 0.996443
2 5.942016 -1.201900E-002 4.157199
...
310 -4.999492 -0.6817100 0.569242

Triangles (order of indices matters)
1 310 32 294
2 76 308 306
...
616 168 125 169
```

Reading surfaces

- Creating surface element files
  - tools/stl2surf.py converts STL files
    to SPARTA format
  - tools/surf_create.py for simple shapes:
    - sphere, circle, box, rectangle, etc
- Read_surf command:
  - can use multiple times
  - options: translate, rotate, scale, invert, ...
  - can clip to simulation box
    602 points ⇒ clipped to 321 points
    1200 triangles ⇒ clipped to 600 tris
    0.100631 min triangle edge length
  - same sphere file can be used
    in different simulations



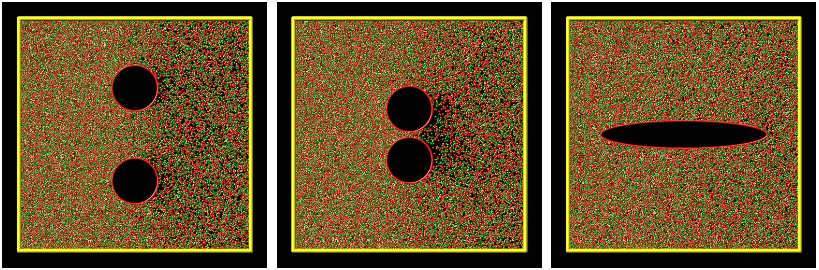


Defining surface element groups

- Assign each surface element to one or more groups
    - by type index in surface file
    - by geometric region
    - see group command
    - some commands operate on a surface element group
        - compute surf, move_surf, etc
- Surface collision and reaction models
    - surf_collide command: diffuse or specular
    - surf_react command: clone/delete or file of reactions
- Define as many models as you wish
    - assign each to a different group or surfaces
    - set different temperatures on different surf patches
    - define different reactions for different objects

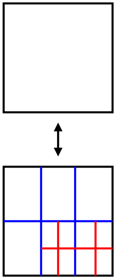
Changing surface elements during a simulation

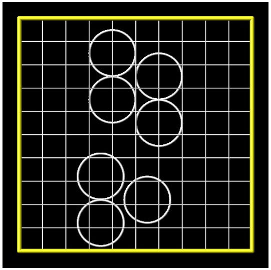
- Add surfaces via read_surf command, particle check option
- Remove_surf command
- Move/rotate surfaces via move_surf or fix move/surf
- All operations are on group (subset) of surface elements
- Allow new surf positions to be read from file (coming soon)

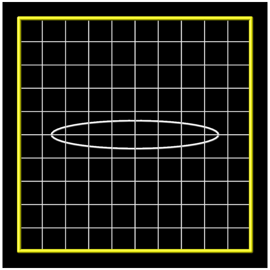


Adaptive gridding

- Adapt means refine and/or coarsen hierarchical grid
- Adapt_grid command: once, before or between runs
- Fix adapt command: on-the-fly adaptivity
- Criteria:
  - nearness to surf, number of particles, mean-free path
  - any per-grid value calculated by a compute or fix

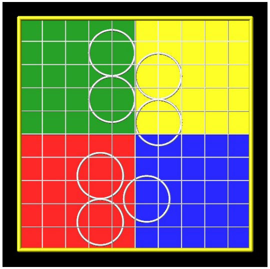


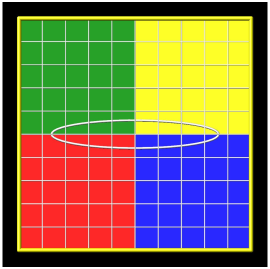




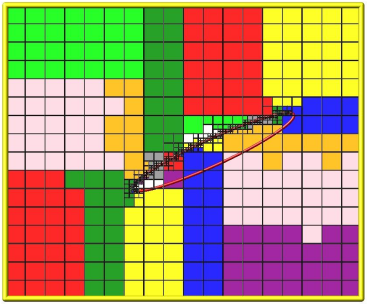
Load-balancing

- Re-assign grid cells (and their particles) to procs
- Balance_grid command: static before or between runs
- Fix balance command: dynamic load-balancing
- Criteria:
  - by blocks, cell count or particle count via RCB
  - by CPU cost (not yet)

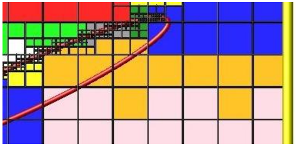




How recursive coordinate bisectioning (RCB) works



Breaks ties on cut planes



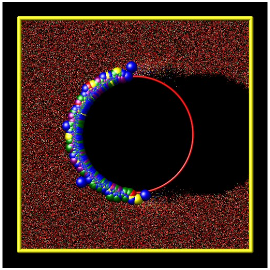
- RCB is fast
- Bigger cost is data move
- 1 billion grid cells on 1024 IBM BG/Q nodes (64K MPI tasks)
  - worst case: move all cells
  - balance time = 15 secs
  - RCB = 2, move = 12, ghosts = 1

Ambipolar approximation

- Can be used for low-density plasmas (charged particles)
- Ambipolar approximation:
  - electrons are not free particles
  - each electron stays close to parent ion (neutral gas)
  - can ignore electron's small mass and high speed (1000x)
  - use a normal molecular timestep $\Rightarrow$ efficiency win
- Implementation details:
  - ion/electron move as one particle
  - ion stores extra velocity for electron
  - when perform collisions within cell, split into two particles

Ambipolar example

- See examples/ambi and doc/Section_howto 4.11
- Fix ambipolar especies ion1 ion2 ...
- Collide_modify ambipolar yes
- React command: include charged reactions in input file

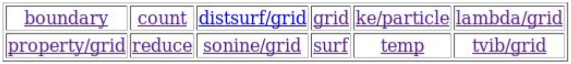


Diagnostic quantities via computes

- Compute commands calculate some property of system
- Always for the current timestep
- Other commands invoke them and access the results
  - stats output, dumps, fixes, variables
- Categories
  - global: temp, count, boundary, ...
  - particle: ke
  - grid: grid (nrho, ke, temp, etc), lambda/grid, sonine/grid, ...
  - surface: surf (count, pressure, shear stress, ke, etc)

Diagnostic quantities via computes

- Compute commands calculate some property of system
- Always for the current timestep
- Other commands invoke them and access the results
  - stats output, dumps, fixes, variables
- Categories
  - global: temp, count, boundary, ...
  - particle: ke
  - grid: grid (nrho, ke, temp, etc), lambda/grid, sonine/grid, ...
  - surface: surf (count, pressure, shear stress, ke, etc)
- To learn what compute styles SPARTA has ...
  doc/Section_commands.html or doc/compute.html



boundary	count	distsurf/grid	grid	ke/particle	lambda/grid
property/grid	reduce	sonine/grid	surf	temp	tvib/grid

Time-averaged statistics via fixes

- Fix ave/time command: averaging of global values
- Fix ave/grid command: averaging of grid cell values
- Fix ave/surf command: averaging of surface element values
- Can time average any value a compute or variable produces
- Results can be output direct to file or via dump commands
- Running averages or within time windows

Time-averaged statistics via fixes

- Fix ave/time command: averaging of global values
- Fix ave/grid command: averaging of grid cell values
- Fix ave/surf command: averaging of surface element values
- Can time average any value a compute or variable produces
- Results can be output direct to file or via dump commands
- Running averages or within time windows
- 3 examples:

  1.
     ```
     compute 1 count all
     compute myTemp temp
     fix 1 ave/time 10 100 1000 c_myTemp c_1 file out.ave
     ```

  2.
     ```
     compute 2 grid all nrho ke temp erot
     fix 2 ave/grid 100 10 1000 c_1
     dump 2 grid all 1000 dump.grid.out id f_2
     ```

  3.
     ```
     compute 3 surf all all n px py pz ke
     fix 3 ave/surf 100 10 1000 c_3 ave running
     dump 3 surf sphere1 5000 dump.surf.out id f_3
     ```

Stats output

One line of output every N timesteps to screen and log file

- See doc/stats_style.html command
- Any scalar can be output:
    - dozens of keywords: step, np, nbound, ncoll, nreact, cpu, ...
    - any scalar output of a compute or fix: c_ID, c_ID[N], f_ID[N]
        - fix ave/time stores time-averaged quantities
    - equal-style variable: v_MyVar

Stats output

One line of output every N timesteps to screen and log file

- See doc/stats_style.html command
- Any scalar can be output:
  - dozens of keywords: step, np, nbound, ncoll, nreact, cpu, ...
  - any scalar output of a compute or fix: c_ID, c_ID[N], f_ID[N]
    - fix ave/time stores time-averaged quantities
  - equal-style variable: v_MyVar
- Can post-process via:
  - tools/log2txt.py log.sparta datafile (Step Np Ncoll ...)
  - tools/logplot.py log.sparta Step Ncollave
  - both can read stats output across multiple runs

Dump output

Snapshot of particle, grid, surface values every N timesteps

- See doc/dump.html and dump_modify commands
- P/G/S attributes, compute/fix/variable results can be output
- Can use as many dump commands as you wish
- Output to one big file, file/proc, file/timestep, in between

Dump output

Snapshot of particle, grid, surface values every N timesteps

- See doc/dump.html and dump_modify commands
- P/G/S attributes, compute/fix/variable results can be output
- Can use as many dump commands as you wish
- Output to one big file, file/proc, file/timestep, in between
- Styles:
  - particle, grid, surf
  - image: instant JPG/PNG/PPM, rendered in parallel
  - movie: image $\Rightarrow$ movie via FFmpeg
- Can limit output by group, geometric region, threshold value
  - only particles of selected species (mixture)
  - only particles with velocity $>$ vthresh
  - only grid cells in geometric region
  - only surf elements in surface group

Use SPARTA as a library

See doc/Section_howto.html 4.6 and 4.7

- C-style interface: call from C, C++, Fortran, Python
- See python and python/examples directories
- Parallel python also possible

```
% python
>>> from sparta import sparta
>>> spa = sparta()
>>> spa.file("in.flow")
>>> spa.command("run 1000")
>>> np =
     spa.extract_global("nplocal",0)
>>> temp =
     spa.extract_compute("temp",0,0)
>>> print "Np",np,
     "temperature",temp
>>> spa.close()
```