Advanced and new features in SPARTA and How to add your own new features

Steve Plimpton
Sandia National Labs (retired)
Temple University (adjunct)
sjplimp@gmail.com

DSMC23 Short Course
Sept 2023 - Santa Fe, NM

Topics for this talk

1. Advanced features in SPARTA
2. New features since last DSMC19 conference
3. How data is stored and how parallelism operates
4. Custom attributes for particles, grid cells, surface elements
5. How to add new features and models to the code
6. Q & A about features you might like to add

Basic features in SPARTA

See short-course talks from DSMC15 conf
Tutorials link on webpage: https://sparta.github.io/tutorials.html

- 2d, 3d, and axi-symmetric (quasi-2d) models
- Units and boundary conditions
- Particle species and mixtures
- Particle creation via boundary or surface emission
- Gas-phase collision and chemistry models
- Explicit surfaces: triangles in 3d, line segments in 2d
- Surface collision/chemistry models
- Time-averaging of particle and grid statistics
- Stats_style (logfile) and dump (snapshot) commands
- Post-processing and viz with TecPlot or ParaView

See Section_tools 9 in manual
for problem setup and post-processing tools

More advanced features

Will show a couple of slides on each of these topics
Point you to commands and doc pages with more details

- Stan: Acceleration for GPUs or OpenMP (multi-threading)
- Input script options
  - variables
  - if/then/else and looping
  - running multiple simulations
- Adaptive gridding
- Load balancing
- Ambipolar approximation for low-density plasmas
- Use SPARTA as a library

Define variables in input scripts

See variable command doc page

- Styles: index, loop, world, equal, particle, grid, surf, ...
  - variable name index run1 run2 run3 run4
  - variable i loop 100
  - variable temp world 200.0 250.0 300.0 350.0
  - variable frac equal 100.0*c_count[2]/np
  - variable vmag particle sqrt(vx*vx+vy*vy+vz*vz)
  - similar formulas for grid cells or surface elements

Define variables in input scripts

See variable command doc page

- Styles: index, loop, world, equal, particle, grid, surf, ...
  - variable name index run1 run2 run3 run4
  - variable i loop 100
  - variable temp world 200.0 250.0 300.0 350.0
  - variable frac equal 100.0*c_count[2]/np
  - variable vmag particle sqrt(vx*vx+vy*vy+vz*vz)
  - similar formulas for grid cells or surface elements
- Formulas can be complex
  - stats keywords (step, np, vol, ...)
  - math operators & functions (sqrt, log, cos, ...)
  - special functions (min, ave, trap, stride, stagger, ...)
  - particle attributes (x, vx, mass, ...)
  - grid cell attributes (location, volume)
  - surface element attributes (location, area)
  - output from computes, fixes, other variables
- Formulas can thus be spatially- and/or time-dependent

Five ways to use variables in input scripts

1. Index-style vars can be (re)set from command line
2. Substitute in any command via $x or ${myVar}
   - global fnum ${Fnum} nrho ${Nrho} vstream $v 0 0
   - read_surf sdata.${filename}
3. 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

Five ways to use variables in input scripts

1. Index-style vars can be (re)set from command line
2. Substitute in any command via $x or ${myVar}
   - global fnum ${Fnum} nrho ${Nrho} vstream $v 0 0
   - read_surf sdata.${filename}
3. 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
4. Some commands allow variables as arguments
   - surf_collide diffuse v_temp ... (equal-style)
   - surf_collide diffuse v_temp ... (surf-style)
   - dump_modify every v_lognext (output logarithmically)
   - dump image ... view v_theta v_phi ... (fly-by movie)
5. Next command increments a multi-value variable to next value

More options for input scripts

- If/then/else logic via if command
- Looping via next and jump commands
  - increment a parameter, run some more, repeat
  - loop over multiple runs (next slide)

More options for input scripts

- If/then/else logic via if command
- Looping via next and jump commands
  - increment a parameter, run some more, repeat
  - loop over multiple runs (next slide)
- Insert another script via include command
  - useful for long list of parameters
- Filename wildcard and suffix options:
  - dump.*.% for per-snapshot or per-processor output
  - read_restart old.restart.* for last file input
  - read_surf sdata.huge.gz
- 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 6.3 of manual

Example script for multiple runs

Run 8 successive simulations:

```
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      SELF
```

Example script for multiple runs

Run 8 successive simulations:

```
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       SELF
```

Run 100 simulations on 8 partitions of processors until finished:

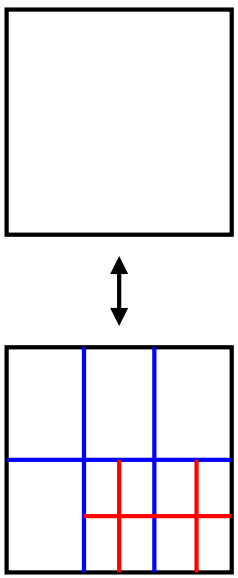
- change a and rho to universe-style variables with 100 values
- mpirun -np 64 spa_mpi -p 8x8 -in in.flow

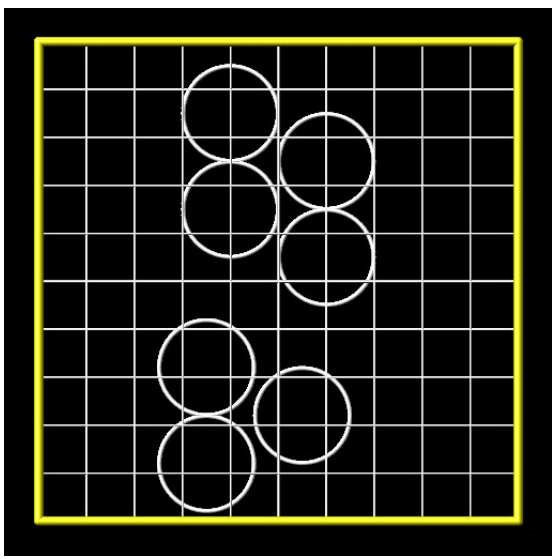
Adaptive gridding

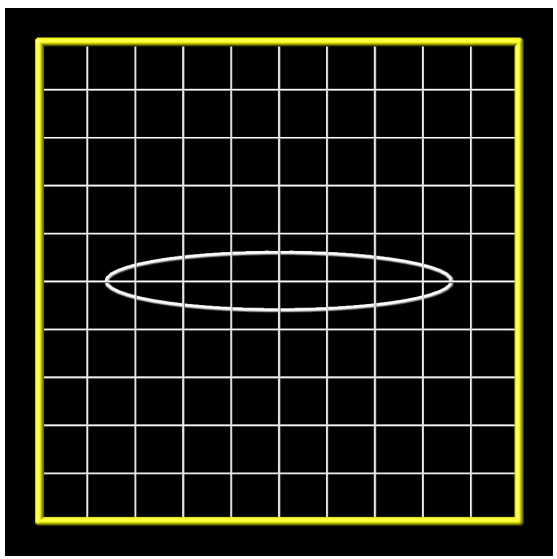
- Adapt means refine and/or coarsen hierarchical grid
- Adapt_grid command: once either before or between runs
- Fix adapt command: on-the-fly adaptivity
- Criteria:
  - nearness to surf, number of particles, mean-free path
  - any per-grid value calculated by a compute or fix

Adaptive gridding

- Adapt means refine and/or coarsen hierarchical grid
- Adapt_grid command: once either 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





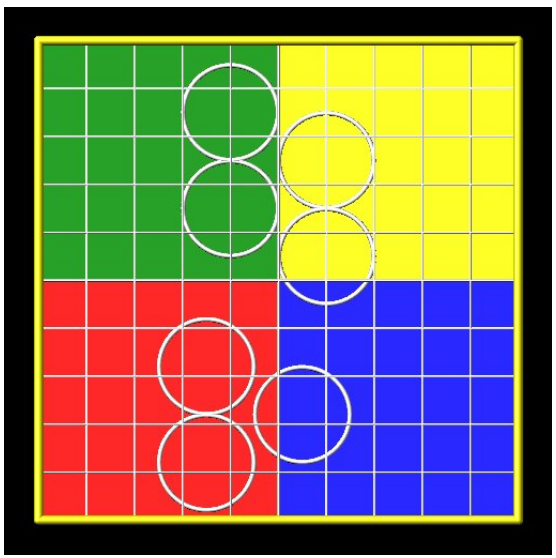


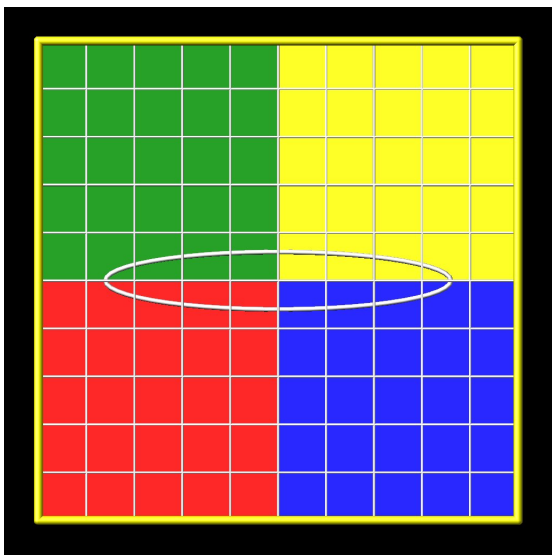
Load balancing

- Re-assign grid cells (and their particles) to procs
- Balance_grid command: static before or between runs
- Fix balance command: dynamic load-balancing
- Based on grid cell count, particle count, or CPU cost

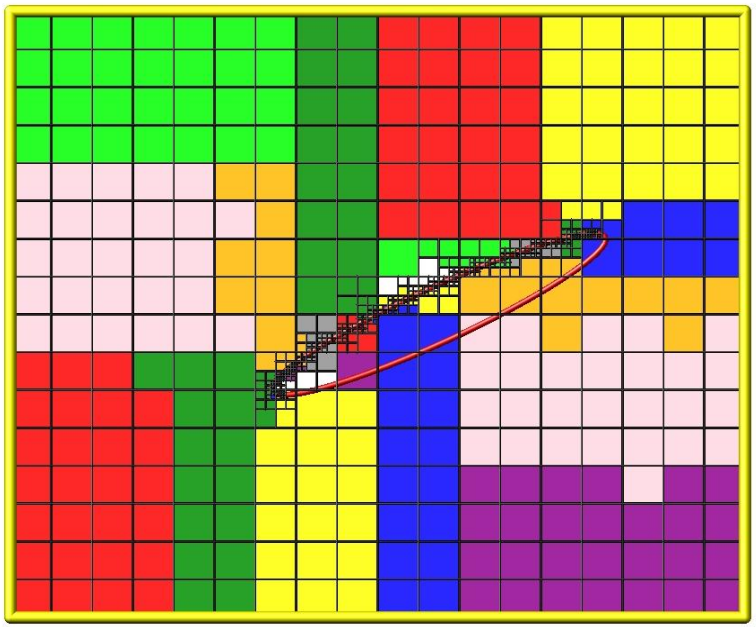
Load balancing

- Re-assign grid cells (and their particles) to procs
- Balance_grid command: static before or between runs
- Fix balance command: dynamic load-balancing
- Based on grid cell count, particle count, or CPU cost

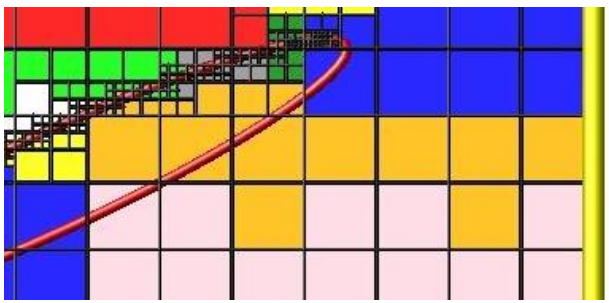




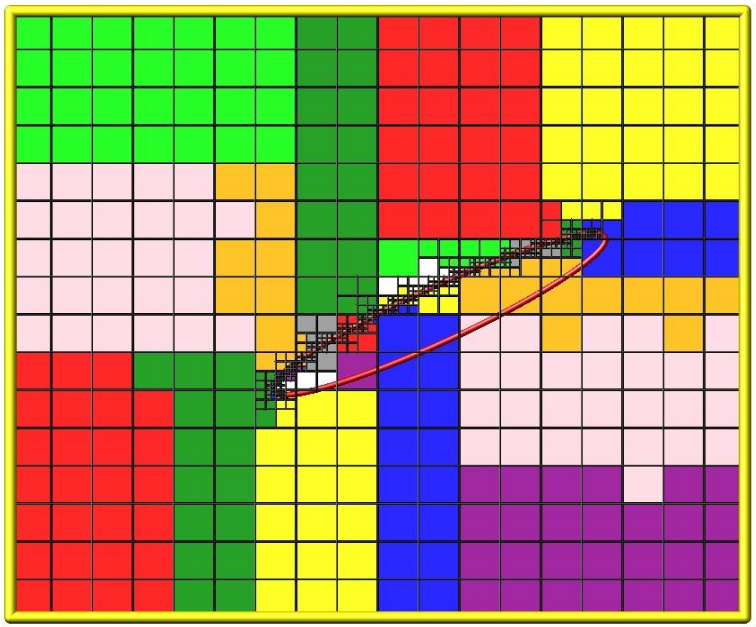
How recursive coordinate bisectioning (RCB) works



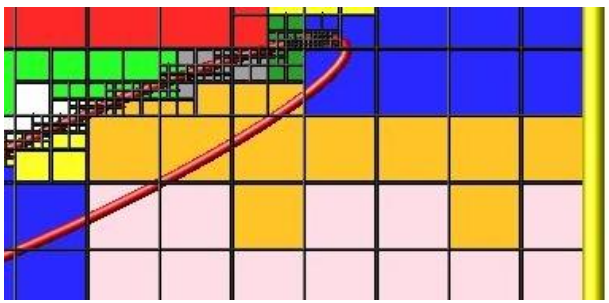
Breaks ties on cut planes



How recursive coordinate bisectioning (RCB) works



Breaks ties on cut planes



- RCB itself is fast
- Bigger cost is data migration
- 1 billion grid cells on 64K MPI tasks (IBM BG/Q machine)
  - worst case scenario: migrate all cells (and their particles)
  - re-balance time = 15 secs
  - RCB = 2, migrate = 12, acquire ghosts = 1

Ambipolar model for ionized systems

- Can be used for low-density (weak) plasmas
- 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

Ambipolar model for ionized systems

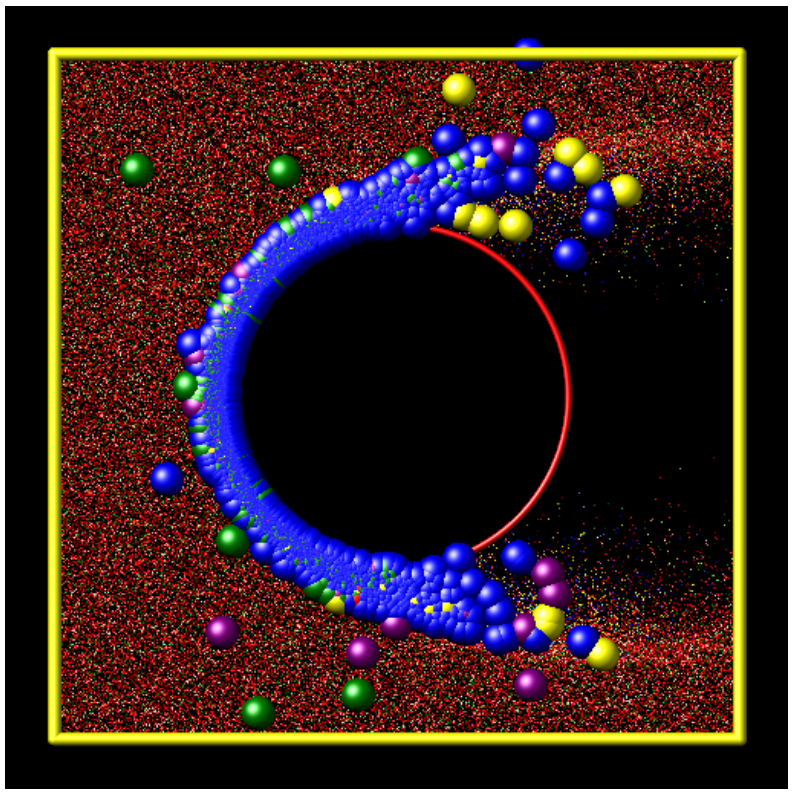
- Can be used for low-density (weak) plasmas
- Ambipolar approximation:
  - electrons are not free particles
  - each electron stays close to parent ion (neutral gas)
  - can ignore electron's small mass and high speed (1000x)
  - use a normal molecular timestep $\Rightarrow$ efficiency win
- Implementation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

Ambipolar example

- See examples/ambi and Section_howto 6.11 of manual
- Fix ambipolar especies ion1 ion2 ...
- Collide_modify ambipolar yes
- React command: include charged reactions in input file

Ambipolar example

- See examples/ambi and Section_howto 6.11 of manual
- Fix ambipolar especies ion1 ion2 ...
- Collide_modify ambipolar yes
- React command: include charged reactions in input file



Use SPARTA as a library

See Section_howto 6.6 and 6.7 of manual

- C-style interface:
  call from
  C, C++, Fortran,
  Python
- See python and
  python/examples
  directories
- Parallel python
  possible via mpi4py
- Library API can be
  extended

Use SPARTA as a library

See Section_howto 6.6 and 6.7 of manual

- C-style interface: call from C, C++, Fortran, Python
- See python and python/examples directories
- Parallel python possible via mpi4py
- Library API can be extended

```
% 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()
```

Topic #2

1. Advanced features in SPARTA
2. New features since last DSMC19 conference
3. How data is stored and how parallelism operates
4. Custom attributes for particles, grid cells, surface elements
5. How to add new features and models to the code
6. Q & A about features you might like to add

New features since last DSMC19 conference

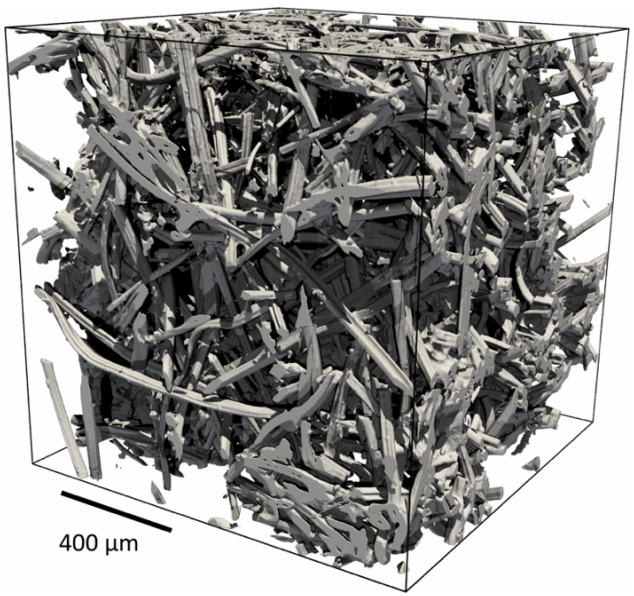
Again, will show a slide or two on each of these topics
Point you to commands and doc pages with more details
Some of these were contributed by SPARTA users !

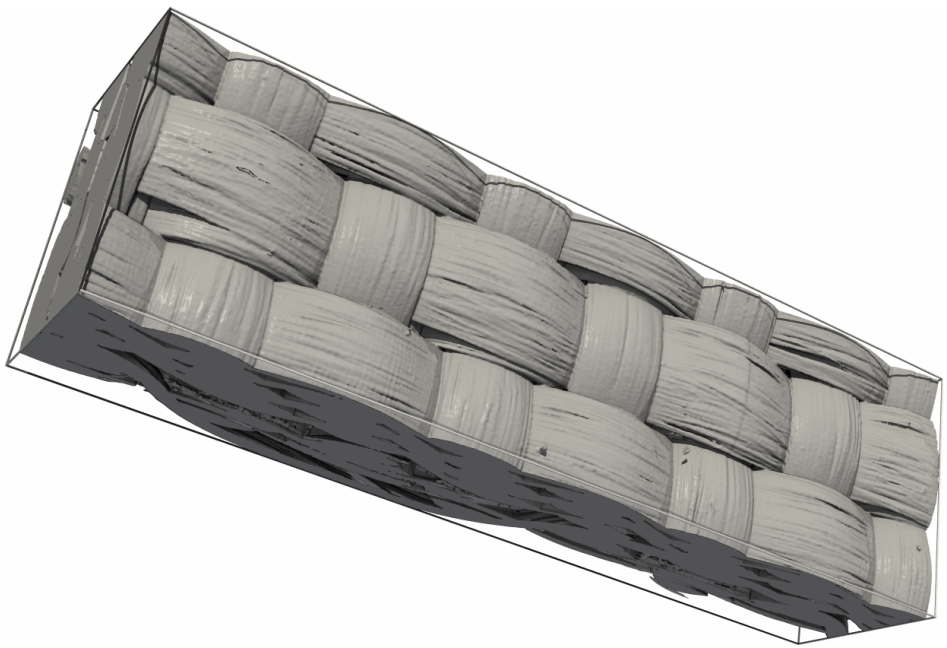
- Implicit surfaces
- Ablation of implicit surfaces
- Vibrational energy states for particles
- Transparent surfaces for flow statistics
- Four new surface collision models
- On-surface chemistry model for explicit surfs
- Similar on-surface chemistry model for implicit surfs (WIP)
- Two options for thermostatting particles
- Create_particles cut option for cut/split cells
- External global field options
- Couple surface temperatures to flow conditions

Implicit surfaces

Work with Arnaud Borner (NASA Ames)

- NASA FiberForm^{TM} porous carbon fiber material
  0.52-mm^3, $V_{frac} = 14\%$, 800^3 grid, 57.4M tris
- 6 ply ADEPT weave
  1-mm^2 xsec, $V_{frac} = 75\%$, 168x220x744 grid, 4.5M tris

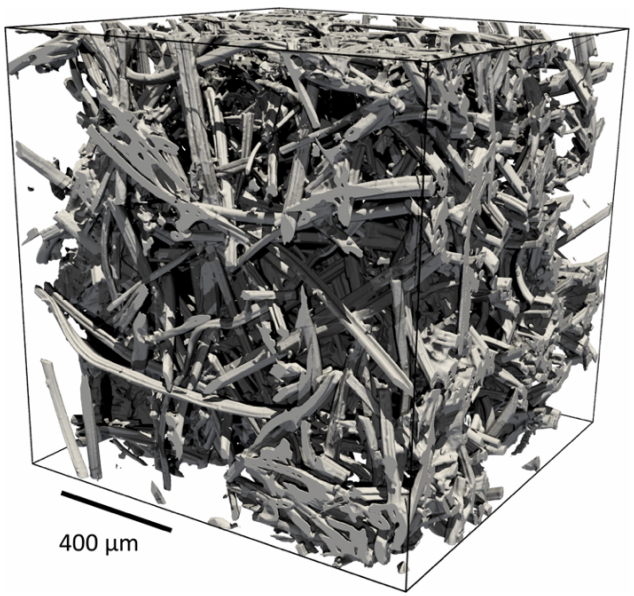


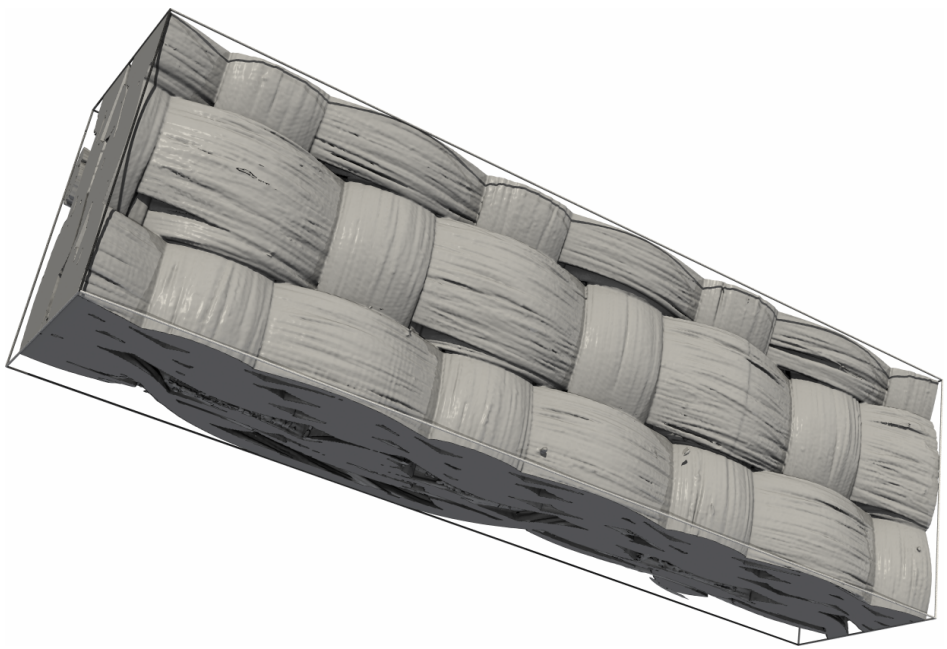


Implicit surfaces

Work with Arnaud Borner (NASA Ames)

- NASA FiberForm^{TM} porous carbon fiber material
  0.52-mm^3, $V_{frac} = 14\%$, 800^3 grid, 57.4M tris
- 6 ply ADEPT weave
  1-mm^2 xsec, $V_{frac} = 75\%$, 168x220x744 grid, 4.5M tris





By contrast, 50000 explicit triangles for entire Mir space station

Implicit surfaces in SPARTA

See Section_howto 6.13 of manual

Start with data file
e.g. experimental 3d tomographic image of heat shield material

Implicit surfaces in SPARTA

See Section_howto 6.13 of manual

Start with data file
e.g. experimental 3d tomographic image of heat shield material

- Image voxels ⇒ corner point values on DSMC grid
- Read_isurf command reads voxels, assigns to grid cells
- Choose threshold value between 0 and 255 for surface
- Marching cubes algorithm creates triangles (squares ⇒ lines)
  - each triangle wholly contained in a grid cell
  - multiple triangles per cell to capture surface
  - highly parallel, compute tris for each grid cell independently
- Processor owning a grid cell also owns its triangles

Implicit surfaces in SPARTA

See Section_howto 6.13 of manual

Start with data file
e.g. experimental 3d tomographic image of heat shield material

- Image voxels ⇒ corner point values on DSMC grid
- Read_isurf command reads voxels, assigns to grid cells
- Choose threshold value between 0 and 255 for surface
- Marching cubes algorithm creates triangles (squares ⇒ lines)
  - each triangle wholly contained in a grid cell
  - multiple triangles per cell to capture surface
  - highly parallel, compute tris for each grid cell independently
- Processor owning a grid cell also owns its triangles

By contrast, one explicit triangle can span many grid cells

Ablation of implicit surfaces

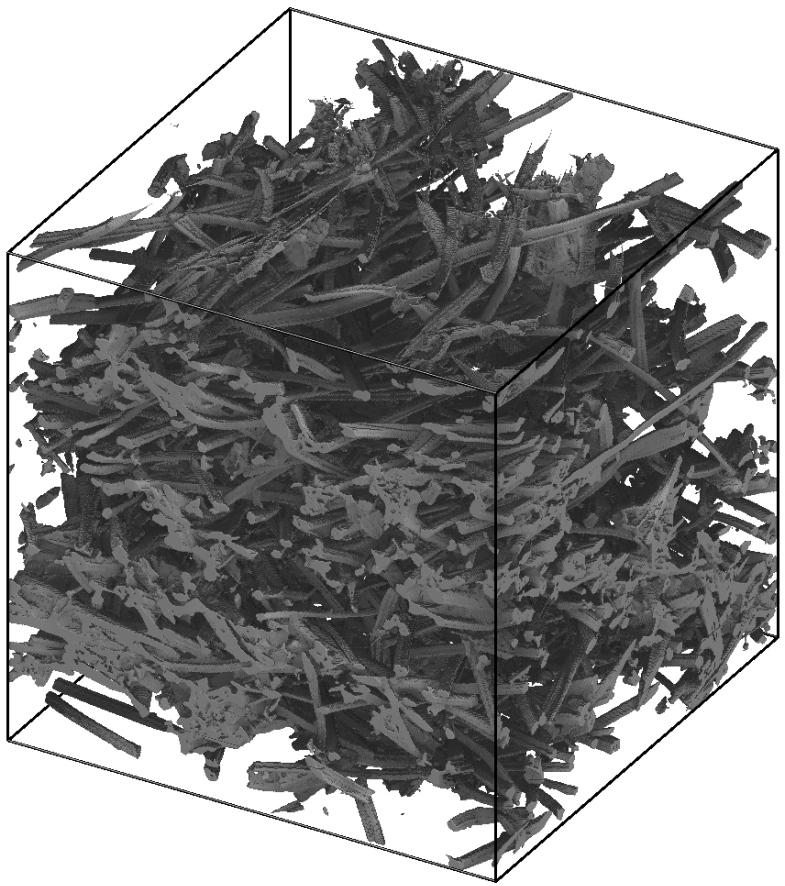
Also work with Arnaud Borner (NASA Ames)
See talk at DSMC19 conference

- Fix ablation command
- Hooked to per-grid compute or fix which tallies damage
- Damage = energy deposition or chemical reactions
- Decrement grid corner point values due to damage
- Re-triangulate periodically
- Surface effectively recedes, i.e. ablates

See examples/ablation and Section_howto 6.14 of manual

Ablation movie

- 800^3 grid, 57M triangles, 60M particles, 22M surf collide/step
- Monatomic O at 2000K, each collision $\Rightarrow$ oxidation reaction
- On-the-fly imaging (not ParaView quality)



Vibrational energy states for particles

- Useful for high temperature models with molecular species
- Vibrational energy can be discretized across multiple modes
- Vibrational energy can be exchanged in gas phase collisions

- Input per-species vibfile with info on discrete states
- Collide_modify command options
  - vibrate = no or smooth or discrete
  - rotate = no or smooth
- Use fix vibmode command to store extra per-particle vector

See examples/vibrate and Section_howto 6.12 of manual

Transparent surfaces for flow statistics

- Group of triangles can be flagged as transparent
- Particles pass through them unaffected
- Transparent surf objects do not need to be watertight
  - e.g. can define a plane
  - can also intersect non-transparent surfs or each other
- Surf_collide transparent allows tally of count/mass/energy flux
- Only one side of surface triggers tallying

See examples/circle/in.circle.transparent and
Section_howto 6.15 of manual

Five new surface collision models

First three from Krishnan Gopalan (NASA Ames)
Fourth from Tim Teichmann (KIT)

1. cll
   - model of Cercignani, Lampis, and Lord
   - accommodation coeffs for normal, tangential, rot & vib energy
2. td
   - thermal desorption model of Krishnan Gopalan
   - scattering with thermal Maxwell-Boltzmann distribution
3. impulsive
   - complex collision model (8 params) of Krishnan Gopalan
   - appropriate for high-energy beam of particles
   - scattering can be highly anisotropic

Five new surface collision models

First three from Krishnan Gopalan (NASA Ames)
Fourth from Tim Teichmann (KIT)

1. cll
   - model of Cercignani, Lampis, and Lord
   - accommodation coeffs for normal, tangential, rot & vib energy
2. td
   - thermal desorption model of Krishnan Gopalan
   - scattering with thermal Maxwell-Boltzmann distribution
3. impulsive
   - complex collision model (8 params) of Krishnan Gopalan
   - appropriate for high-energy beam of particles
   - scattering can be highly anisotropic
4. adiabatic
   - model of Mohammadzadeh, Rana, and Struchtrup
   - isotropic scattering with conserved velocity magnitude
   - no energy transfer between particles and surface

Five new surface collision models

First three from Krishnan Gopalan (NASA Ames)
Fourth from Tim Teichmann (KIT)

1. cll
   - model of Cercignani, Lampis, and Lord
   - accommodation coeffs for normal, tangential, rot & vib energy
2. td
   - thermal desorption model of Krishnan Gopalan
   - scattering with thermal Maxwell-Boltzmann distribution
3. impulsive
   - complex collision model (8 params) of Krishnan Gopalan
   - appropriate for high-energy beam of particles
   - scattering can be highly anisotropic
4. adiabatic
   - model of Mohammadzadeh, Rana, and Struchtrup
   - isotropic scattering with conserved velocity magnitude
   - no energy transfer between particles and surface
5. specular noslip option = reflect all components

On-surface chemistry model for explicit surfs

Also from Krishnan Gopalan (NASA Ames)

- Surf_react adsorb command
- Supports both gas/surf (GS) and surf/surf (PS) models
- Input reaction file defines GS and PS reactions of various kinds
- For PS, gas particles can adsorb/desorb to/from surface
- For PS, define on-surface chemical species
- For PS, each triangle maintains state
  - per-species and total counts of adsorbed particles
- For PS, chemical reactions performed every Nsync steps
  - each triangle advances its state by (dt * Nsync)
  - network of on-surf chemical reactions invoked probabilistically
  - time counter algorithm used

See examples/surf_react_adsorb folder

Similar on-surface chemistry model for implicit surfs

Work by Victoria Arias from Kelly Stephani group at UIUC
Work in progress - not yet released in public SPARTA

- New surf_react implicit command
- Same model and GS/PS options as for
  surf_react adsorb command for explicit surfaces
- But for implicit surfaces
- Surface state is now per-grid cell, for all triangles in cell
- Area of surfs within a grid cell is now a dynamic quantity
- Enables ablation to be driven by GS + PS chemistry

Two options for thermostatting particles

Thermostatting is useful to keep aggregate temperature of particles near a specified target temperature

- Can ramp target T up or down during a simulation
- Can compensate for energy added to or lost from the system

Two options for thermostatting particles

Thermostatting is useful to keep aggregate temperature of particles near a specified target temperature

- Can ramp target T up or down during a simulation
- Can compensate for energy added to or lost from the system

Fix temp/global/rescale command

- Rescale particle velocities for entire system every N steps
- Only applied to translational degrees of freedom
- Assumes net COM velocity of system is zero

Two options for thermostatting particles

Thermostatting is useful to keep aggregate temperature of particles near a specified target temperature

- Can ramp target T up or down during a simulation
- Can compensate for energy added to or lost from the system

Fix temp/global/rescale command

- Rescale particle velocities for entire system every N steps
- Only applied to translational degrees of freedom
- Assumes net COM velocity of system is zero

Fix temp/rescale command

- Rescale particle velocities within each grid cell every N steps
- Only adjusts thermal temperature
- Per-grid-cell COM velocity is subtracted before rescaling

Two options for thermostatting particles

Thermostatting is useful to keep aggregate temperature of particles near a specified target temperature

- Can ramp target T up or down during a simulation
- Can compensate for energy added to or lost from the system

Fix temp/global/rescale command

- Rescale particle velocities for entire system every N steps
- Only applied to translational degrees of freedom
- Assumes net COM velocity of system is zero

Fix temp/rescale command

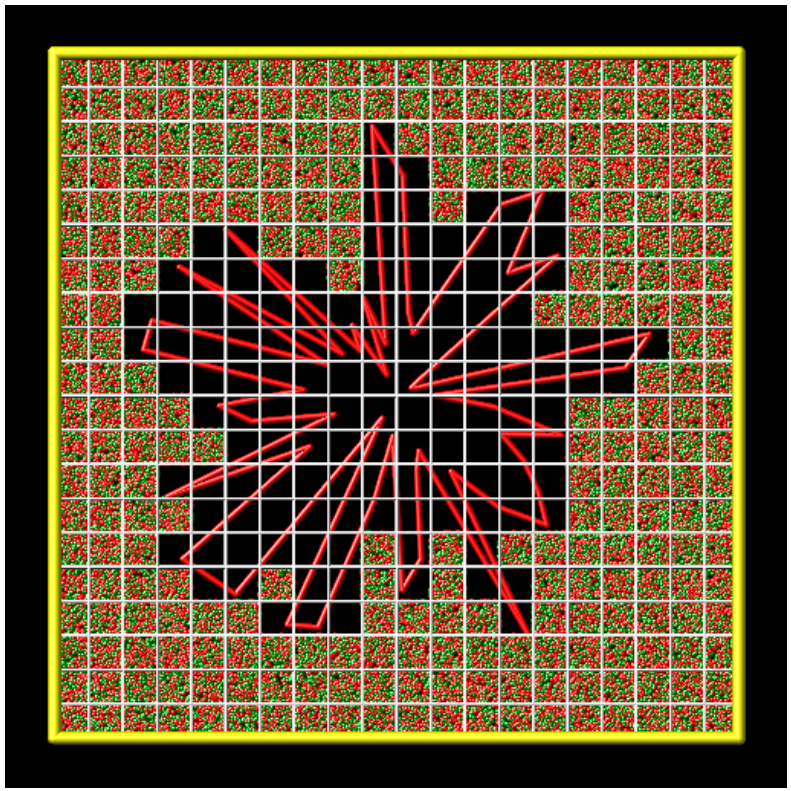
- Rescale particle velocities within each grid cell every N steps
- Only adjusts thermal temperature
- Per-grid-cell COM velocity is subtracted before rescaling

See examples/thermostat folder

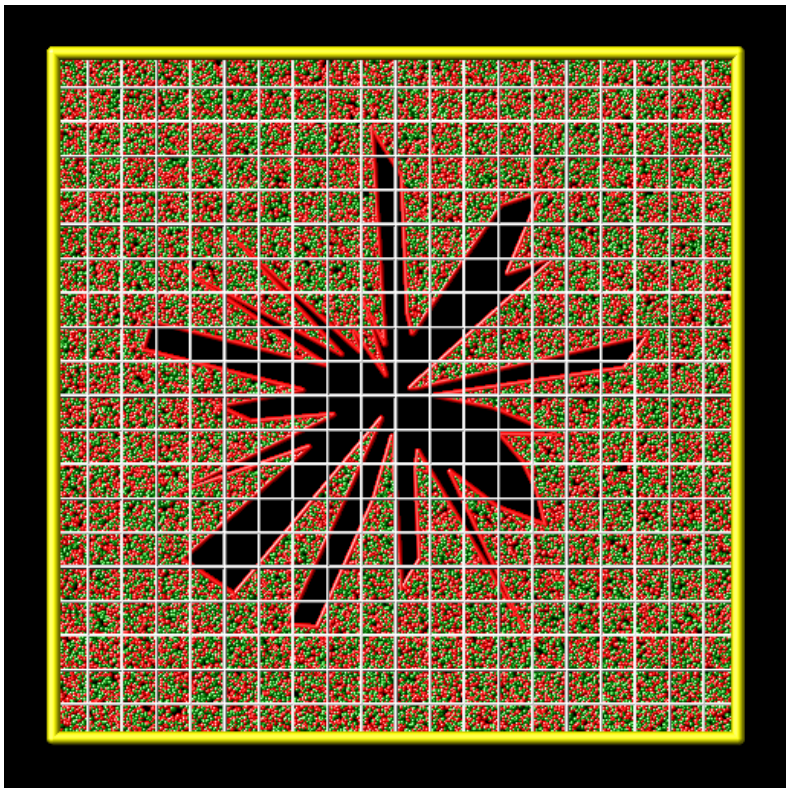
Create_particles cut option for cut/split cells

2d test problem from examples/spiky

cut no



cut yes (new default)



More robust particle initialization

External global field options - see examples/bfield

- Specify an external field which accelerates particles
- Can be constant or time-varying or spatially-varying
- Global field command with 3 options:
- Constant magnitude x y z $\Longrightarrow$ gravity
  - static field with magnitude in (x,y,z) direction

External global field options - see examples/bfield

- Specify an external field which accelerates particles
- Can be constant or time-varying or spatially-varying

- Global field command with 3 options:
- Constant magnitude x y z $\Longrightarrow$ gravity
  - static field with magnitude in (x,y,z) direction
- Particle fix-ID $\Longrightarrow$ magnetic field
  - fix-ID for fix field/particle command
  - field with 3 components applied to each particle
  - each component is a particle-style variable (formula)
  - can depend on timestep and particle x, v, mass, mu

External global field options - see examples/bfield

- Specify an external field which accelerates particles
- Can be constant or time-varying or spatially-varying

- Global field command with 3 options:
- Constant magnitude x y z $\Longrightarrow$ gravity
  - static field with magnitude in (x,y,z) direction
- Particle fix-ID $\Longrightarrow$ magnetic field
  - fix-ID for fix field/particle command
  - field with 3 components applied to each particle
  - each component is a particle-style variable (formula)
  - can depend on timestep and particle x, v, mass, mu
- Grid fix-ID Nfreq $\Longrightarrow$ turbulence driver
  - fix-ID for fix field/grid command
  - field with 3 components applied to each particle
  - each component is a grid-style variable (formula)
  - can depend on timestep and grid cell position
  - Nfreq = how often grid-cell field is re-evaluated

External global field options - see examples/bfield

- Specify an external field which accelerates particles
- Can be constant or time-varying or spatially-varying

- Global field command with 3 options:
- Constant magnitude x y z $\Longrightarrow$ gravity
  - static field with magnitude in (x,y,z) direction
- Particle fix-ID $\Longrightarrow$ magnetic field
  - fix-ID for fix field/particle command
  - field with 3 components applied to each particle
  - each component is a particle-style variable (formula)
  - can depend on timestep and particle x, v, mass, mu
- Grid fix-ID Nfreq $\Longrightarrow$ turbulence driver
  - fix-ID for fix field/grid command
  - field with 3 components applied to each particle
  - each component is a grid-style variable (formula)
  - can depend on timestep and grid cell position
  - Nfreq = how often grid-cell field is re-evaluated
- Grid option is computationally cheaper than particle option

Couple surface temperatures to flow conditions

From Arnaud Borner (NASA Ames)

- Fix surf/temp command
- Takes a compute or fix ID which calculates per-surf heat flux
- Uses Stefan-Boltzmann law for a gray-body
- $Q_{wall} = \sigma \ \epsilon_{surf} \ {T_{surf}}^4$
- Resets surface temperature as function of heat flux
- Can re-compute surf temperatures every N steps
- Surface temperature affects future particle/surf collisions

See examples/adjust_temp folder

Topic #3

1. Advanced features in SPARTA
2. New features since last DSMC19 conference
3. How data is stored and how parallelism operates
4. Custom attributes for particles, grid cells, surface elements
5. How to add new features and models to the code
6. Q & A about features you might like to add

Three flavors of data in DSMC

Particles, grid cells, and surface elements (triangles)

100 bytes per particle

- ID, species index, grid cell, x, v, rotate & vibrate energy

200 bytes per grid cell

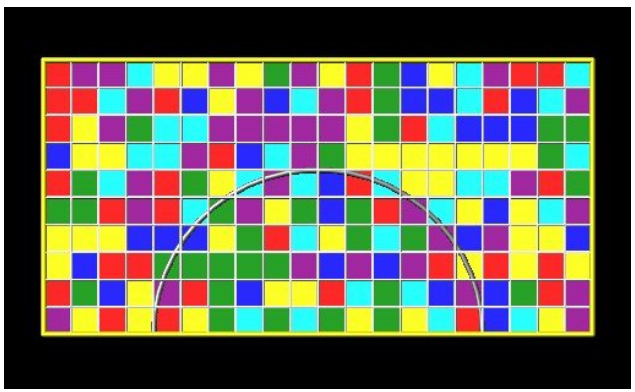
- ID, level in grid hierarchy, lo/hi corner points, volume, flags
- list of neighboring grid cells
- list of intersecting surfs, more info if a split cell

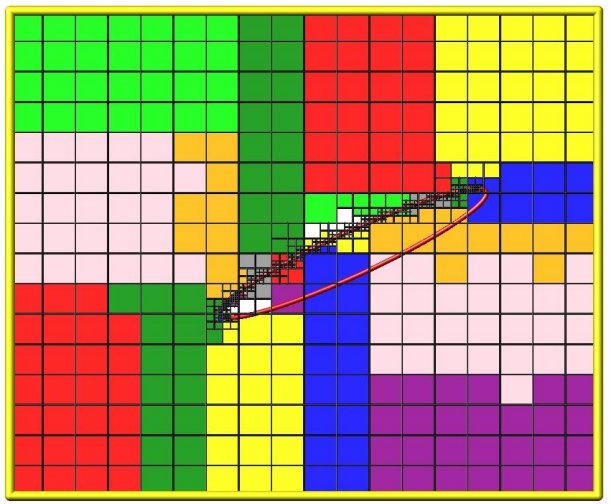
125 bytes per triangle

- ID, flags, indices of collision and reaction models
- coords of 3 corner points, normal vector

Grid decomposition in SPARTA

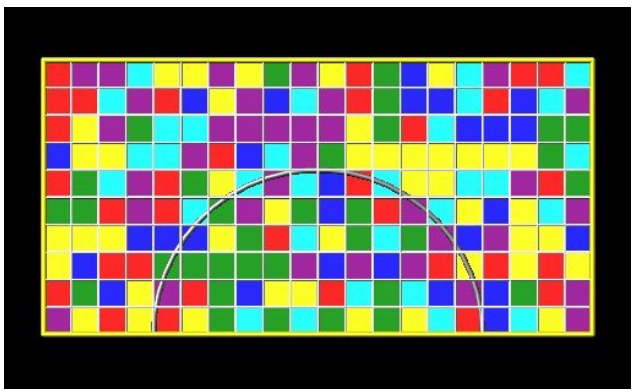
- Grid can be hierarchical (adapted)
- Each processor owns a unique subset of the grid cells
- Can be scattered or clumped

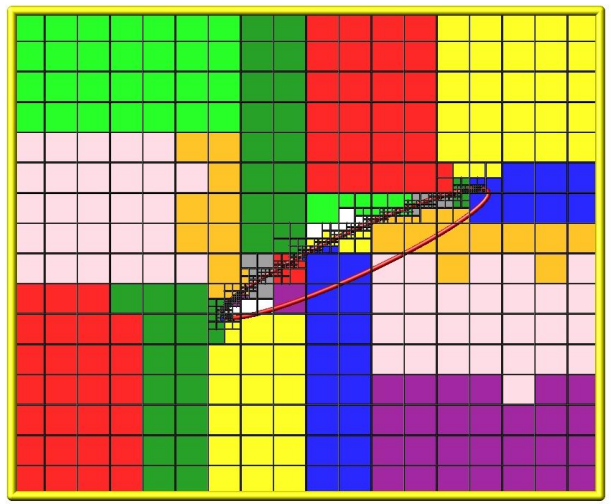




Grid decomposition in SPARTA

- Grid can be hierarchical (adapted)
- Each processor owns a unique subset of the grid cells
- Can be scattered or clumped





- For scattered, each proc stores ghost copy of all grid cells
- For clumped, each proc stores only ghost cells within a cutoff
  - command: global gridcut 0.1
  - good setting = max particle move distance for 1 timestep
- Ghost cells enable tracking particle moves to end of timestep

Particle and surface element decomposition in SPARTA

Particles:

- Each processor owns only the particles in its owned grid cells
- No ghost particles

Particle and surface element decomposition in SPARTA

Particles:

- Each processor owns only the particles in its owned grid cells
- No ghost particles

Surface elements (triangles):

- Decomposition can be global or distributed
- For global, each proc owns copy of all triangles
- Typically best if a modest number of triangles (10K or less)
- For distributed, each proc owns copy of only triangles which overlap its owned + ghost grid cells
- Triangles for ghost cells needed to track particles
- Implicit triangles are always distributed (with grid cells)

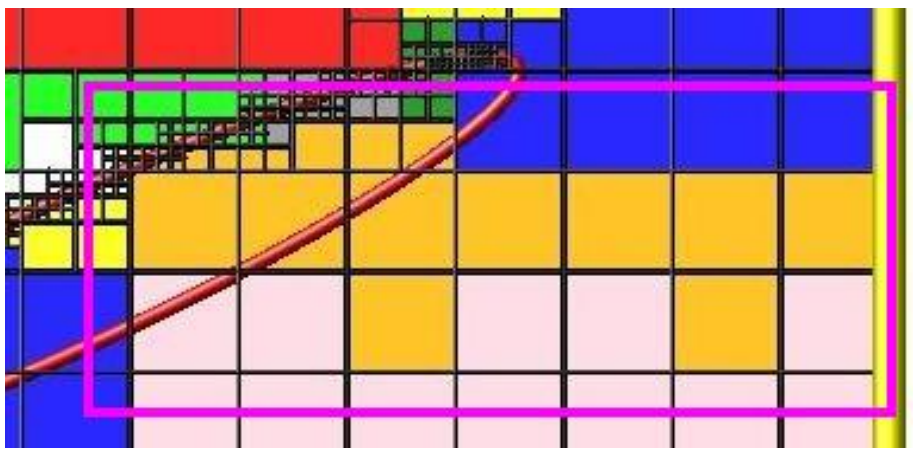
Parallel communication each timestep

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

Parallel communication each timestep

- Each timestep:
  1. each particle is ray-traced thru one or more grid cells
  2. can bounce off triangles, 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



Topic #4

1. Advanced features in SPARTA
2. New features since last DSMC19 conference
3. How data is stored and how parallelism operates
4. Custom attributes for particles, grid cells, surface elements
5. How to add new features and models to the code
6. Q & A about features you might like to add

Custom attributes for particles, grid cells, surface elements

Motivation:

- New models and features may require new kinds of data
- Either for particles, grid cells, or surface elements (triangles)
- Beyond the stored data previously summarized

Custom attributes for particles, grid cells, surface elements

Motivation:

- New models and features may require new kinds of data
- Either for particles, grid cells, or surface elements (triangles)
- Beyond the stored data previously summarized

Input script can define various custom attributes:

- Each is either for particles, grid cells, or triangles
- Each attribute has a name, so can be referenced elsewhere
- Each has a data type (integer or floating point)
- Each has a size
  - custom vector = one value per particle/cell/surf
  - custom array = multiple values per particle/cell/surf

Custom attributes for particles, grid cells, surface elements

Motivation:

- New models and features may require new kinds of data
- Either for particles, grid cells, or surface elements (triangles)
- Beyond the stored data previously summarized

Input script can define various custom attributes:

- Each is either for particles, grid cells, or triangles
- Each attribute has a name, so can be referenced elsewhere
- Each has a data type (integer or floating point)
- Each has a size
  - custom vector = one value per particle/cell/surf
  - custom array = multiple values per particle/cell/surf

Parts of what is described here are already available in SPARTA

- Other parts are new enhancements
- Currently in GitHub pull request #428, will be released soon

Commands which define or use custom attributes

All 3 flavors of attributes: (particle, grid cell, or triangle)

- custom command - create and/or set values of an attribute
  - set via corresponding variable style
  - particle-style, grid-style, or new surf-style variables
- compute reduce - reduce attribute to a scalar value
- dump - output attributes to a particle/grid/surf dump file
- variable command - use attribute in a variable formula

Commands which define or use custom attributes

All 3 flavors of attributes: (particle, grid cell, or triangle)

- custom command - create and/or set values of an attribute
  - set via corresponding variable style
  - particle-style, grid-style, or new surf-style variables
- compute reduce - reduce attribute to a scalar value
- dump - output attributes to a particle/grid/surf dump file
- variable command - use attribute in a variable formula

Per-particle custom attributes:

- fix ambipolar - uses a vector and array for ambipolar quantities

Commands which define or use custom attributes

Per-surf custom attributes:

- fix surf/temp - vector to calculate/store surface temps
- surf_react adsorb - vectors and array to store chemical state
- surf_collide - vector temperature for particle/surf collisions
- fix ave/surf - time-average an attribute
- read_surf - define and initialize attributes
- write_surf - write attributes to a surf data file

Commands which define or use custom attributes

Per-surf custom attributes:

- fix surf/temp - vector to calculate/store surface temps
- surf_react adsorb - vectors and array to store chemical state
- surf_collide - vector temperature for particle/surf collisions
- fix ave/surf - time-average an attribute
- read_surf - define and initialize attributes
- write_surf - write attributes to a surf data file

Per-grid custom attributes:

- surf_react implicit - vectors and array to store chemical state
- fix ave/grid - time-average an attribute
- read_grid - define and initialize attributes
- write_grid - write attributes to a grid data file

Additional functionality of custom attributes

- Per-surf attributes for either global or distributed surfs
- Per-grid attributes optionally stored with ghost grid cells
  - likewise with surfs that overlap ghost grid cells
- Migrate with owning particle/grid/surf when load-balancing
- Stored in restart files

Topic #5

1. Advanced features in SPARTA
2. New features since last DSMC19 conference
3. How data is stored and how parallelism operates
4. Custom attributes for particles, grid cells, surface elements
5. How to add new features and models to the code
6. Q & A about features you might like to add

SPARTA is designed to be extensible

- Enabled by C++ object orientation and SPARTA styles
- See Section_modify of manual for overview
- Also discussed in 2019 Phys Fluids SPARTA overview paper

SPARTA is designed to be extensible

- Enabled by C++ object orientation and SPARTA styles
- See Section_modify of manual for overview
- Also discussed in 2019 Phys Fluids SPARTA overview paper
- Before you start writing code you may want to ask:
  - can SPARTA already do this ?
  - how hard would it be to implement ?
  - is my plan a good way to implement this idea ?

SPARTA is designed to be extensible

- Enabled by C++ object orientation and SPARTA styles
- See Section_modify of manual for overview
- Also discussed in 2019 Phys Fluids SPARTA overview paper
- Before you start writing code you may want to ask:
  - can SPARTA already do this ?
  - how hard would it be to implement ?
  - is my plan a good way to implement this idea ?
- Three ways to ask these Qs:
  - post a message to the mail list
  - post an issue on the GitHub site
  - email the developers (Stan, Michael, Steve)
- We can give you some feedback/advice on your idea

Idea #1 - Couple SPARTA to another code

Not really modifying SPARTA, other code has new functionality

Idea #1 - Couple SPARTA to another code

Not really modifying SPARTA, other code has new functionality

Other code calls SPARTA

- see Section_howto 6.6: 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
- example: umbrella Python script can invoke SPARTA and other code, pass info between them

Idea #1 - Couple SPARTA to another code

Not really modifying SPARTA, other code has new functionality

Other code calls SPARTA

- see Section_howto 6.6: 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
- example: umbrella Python script can invoke SPARTA and other code, pass info between them

SPARTA calls other code

- see Section_howto 6.7: Coupling SPARTA to other codes
- wrap the other code in a compute or fix
- pass appropriate SPARTA data (e.g. particle or grid data)
- other code returns new data (e.g. to alter BC)
- when build SPARTA, link with the other code

Idea #2 - Write code for a new style

A style is a child class derived from a parent class

$\sim$55% of SPARTA code base is add-on styles

Majority of recent new features mentioned were add-on styles

Idea #2 - Write code for a new style

A style is a child class derived from a parent class

~55% of SPARTA code base is add-on styles

Majority of recent new features mentioned were add-on styles

9 kinds of styles:

- surf_collide style = surface collision models
- surf_react style = surface reaction models
- compute style = diagnostics
- fix style = operations within timestep
- collide style = gas collision models
- react style = gas reaction models
- region style = geometric regions
- dump style = output of snapshots
- command style = added input script commands
  - create_box, balance_grid, run, ...

Steps to write a new style

Section_modify 10 in manual: Modifying & Extending SPARTA

Steps to write a new style

Section_modify 10 in manual: Modifying & Extending SPARTA

- Examine the parent *.h file which defines style interface
  - class variables the child class uses
  - methods a child class must define (pure virtual)
  - optional methods a child class can define (virtual)

Steps to write a new style

Section_modify 10 in manual: Modifying & Extending SPARTA

- Examine the parent *.h file which defines style interface
  - class variables the child class 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

Steps to write a new style

Section_modify 10 in manual: Modifying & Extending SPARTA

- Examine the parent *.h file which defines style interface
  - class variables the child class 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

Steps to write a new style

Section_modify 10 in manual: Modifying & Extending SPARTA

- Examine the parent *.h file which defines style interface
  - class variables the child class 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 or reaction model

Surface collision models - see surf_collide.h

- define a collide() method, called when particle hits triangle
- wrapper() and flags_and_coeffs() methods allow surf_react adsorb to emit particles from surface
- each style can define its own input script arguments
  - parsed in constructor
- surf_collide specular and diffuse are simplest to study

Adding a new surface collision or reaction model

Surface collision models - see surf_collide.h

- define a collide() method, called when particle hits triangle
- wrapper() and flags_and_coeffs() methods allow surf_react adsorb to emit particles from surface
- each style can define its own input script arguments
  - parsed in constructor
- surf_collide specular and diffuse are simplest to study

Surface reaction models - see surf_react.h

- define a react() method, called from SurfCollide
- each style can define its own input script arguments
  - parsed in constructor
  - can read its own file of enumerated reactions
- surf_react prob is simplest to study

Diagnostic calculations via computes

- Compute commands calculate some property of system
- Always for the current timestep
- Other commands invoke computes and access their results
  - stats (logfile), dumps, fixes, variables

Diagnostic calculations via computes

- Compute commands calculate some property of system
- Always for the current timestep
- Other commands invoke computes and access their results
  - stats (logfile), dumps, fixes, variables
- Computes produce these flavors of output data:
  - global: temp, count, boundary, ...
  - particle: ke/particle
  - grid: grid (nrho, ke, temp, etc), lambda/grid, sonine/grid, ...
  - explicit surfaces: surf (count, pressure, shear stress, ke, etc)
  - implicit surfaces: isurf/grid (same quantities)

Diagnostic calculations via computes

- Compute commands calculate some property of system
- Always for the current timestep
- Other commands invoke computes and access their results
  - stats (logfile), dumps, fixes, variables
- Computes produce these flavors of output data:
  - global: temp, count, boundary, ...
  - particle: ke/particle
  - grid: grid (nrho, ke, temp, etc), lambda/grid, sonine/grid, ...
  - explicit surfaces: surf (count, pressure, shear stress, ke, etc)
  - implicit surfaces: isurf/grid (same quantities)

To learn what compute styles SPARTA has ...
Commands link on webpage or doc/compute.html (k) = Kokkos

boundary (k)	count (k)	distsurf/grid (k)	eflux/grid (k)	fft/grid	grid (k)
isurf/grid	ke/particle (k)	lambda/grid (k)	pflux/grid (k)	property/grid (k)	react/boundary
react/surf	react/isurf/grid	reduce	sonine/grid (k)	surf (k)	thermal/grid (k)
temp (k)	tvib/grid				

Adding a new compute

- For global output:
  - define compute_scalar(), compute_vector(), and/or compute_array() methods
  - store result in scalar, vector[i], array[i][j] (see compute.h)
  - example: compute_temp.cpp
    - loop over particles
    - MPI_Allreduce of KE $\Rightarrow$ scalar temperature

Adding a new compute

- For global output:
  - define compute_scalar(), compute_vector(), and/or compute_array() methods
  - store result in scalar, vector[i], array[i][j] (see compute.h)
  - example: compute_temp.cpp
    - loop over particles
    - MPI_Allreduce of KE $\Rightarrow$ scalar temperature
- For per-particle output:
  - define a compute_per_particle() method
  - store result in vector_particle[i], array_particle[i][j]

Adding a new compute

- For global output:
  - define compute_scalar(), compute_vector(), and/or compute_array() methods
  - store result in scalar, vector[i], array[i][j] (see compute.h)
  - example: compute_temp.cpp
    - loop over particles
    - MPI_Allreduce of KE $\Rightarrow$ scalar temperature
- For per-particle output:
  - define a compute_per_particle() method
  - store result in vector_particle[i], array_particle[i][j]
- Similar for per-grid or per-surf output:
  - per-grid styles loop over particles, tally to its grid cell
  - per-surf styles have surf_tally() method, called when a particle hits triangle
  - explicit surf styles are more complex, because one triangle can span multiple procs

Adding a new fix

- Fix commands can insert operations into the timestep loop
- Via start_of_step() and end_of_step() methods
- Define a setmask() method:

  mask |= START_OF_STEP;

Loop over timesteps:

move particles
communicate particles
gas collisions and reactions

output to screen and files

Adding a new fix

- Fix commands can insert operations into the timestep loop
- Via start_of_step() and end_of_step() methods
- Define a setmask() method:

  mask |= START_OF_STEP;

Loop over timesteps:

fix start-of-step — emit/face, emit/face/file, emit/surf, ...

move particles

communicate particles

gas collisions and reactions

fix end-of-step — ave/time, balance, adapt, move/surf, ...

output to screen and files

Other operations fixes can perform

- Invoke & access output from computes or variables
  - fix ave/time, fix ave/grid, fix ave/surf
- Generate output, similar to computes
  - global, per-particle, per-grid-cell, per-surf vectors/arrays
  - fix ave/time, fix ave/grid, fix ave/surf

Other operations fixes can perform

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

Other operations fixes can perform

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

To learn what fix styles SPARTA has ...
Commands link on webpage or doc/fix.html (k) = Kokkos

ablate	adapt (k)	ambipolar (k)	ave/grid (k)	ave/histo (k)	ave/histo/weight (k)
ave/surf	ave/time	balance (k)	emit/face (k)	emit/face/file	emit/surf
field/grid	field/particle	grid/check (k)	move/surf (k)	print	surf/temp
temp/global/rescale	temp/rescale (k)	vibmode (k)			

Contribute your new code to public SPARTA

- Why release your code as part of SPARTA ?
  - open source philosophy
  - fame & fortune, name on author webpage and in source code
  - attract users to your feature
    - find and fix bugs
    - extend its functionality
    - become collaborators

Contribute your new code to public SPARTA

- Why release your code as part of SPARTA ?
  - open source philosophy
  - fame & fortune, name on author webpage and in source code
  - attract users to your feature
    - find and fix bugs
    - extend its functionality
    - become collaborators
- All new code is submitted thru GitHub
  - https://github.com/sparta/sparta
  - clone or fork repo
  - create new branch with your feature
  - submit a pull request

Contribute your new code to public SPARTA

- Why release your code as part of SPARTA ?
  - open source philosophy
  - fame & fortune, name on author webpage and in source code
  - attract users to your feature
    - find and fix bugs
    - extend its functionality
    - become collaborators
- All new code is submitted thru GitHub
  - https://github.com/sparta/sparta
  - clone or fork repo
  - create new branch with your feature
  - submit a pull request
- Key points for a speedy release:
  - doc pages for new commands, in SPARTA format (doc/*.txt)
  - new examples folder if useful (small and quick runs)
  - avoid changes (if possible) to core SPARTA files
  - ask developers ahead if you think changes are necessary

Final topic #6

1. Advanced features in SPARTA
2. New features since last DSMC19 conference
3. How data is stored and how parallelism operates
4. Custom attributes for particles, grid cells, surface elements
5. How to add new features and models to the code
6. Q & A about features you might like to add

Q & A time

Thanks again for your interest in SPARTA !

Remaining time is for you to ask Qs about:

1. This talk (or others)
2. SPARTA generally
3. Suggestion for a new feature you wish SPARTA had
4. Idea for a new feature you may want to add to the code