Large-Scale DSMC Simulations ...
Part 2: Computational aspects

SAND2017-13246PE

Michael Gallis & Steve Plimpton
Sandia National Labs

NASA Ames - December 2017

Center for Computing Research

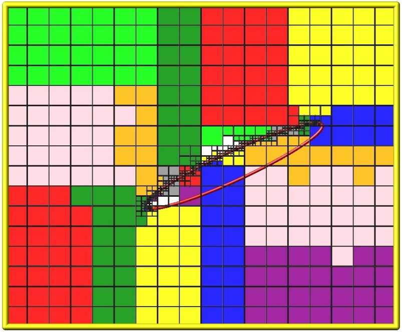
Engineering Sciences Center

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: SAND2017-12447C

National Nuclear Security Administration

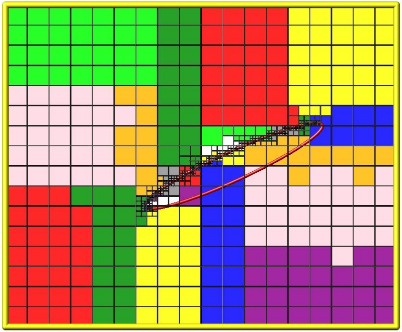
How SPARTA works in parallel

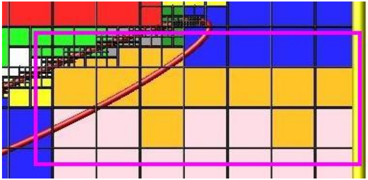
- 3 kinds of data: **particles, grid cells, surface elements**
  - hierarchical Cartesian grid with cells cut/split by surfs
  - grid cells (and their particles) are load balanced (RCB)
  - surface elements are not (yet) distributed



How SPARTA works in parallel

- 3 kinds of data: **particles, grid cells, surface elements**
  - hierarchical Cartesian grid with cells cut/split by surfs
  - grid cells (and their particles) are load balanced (RCB)
  - surface elements are not (yet) distributed





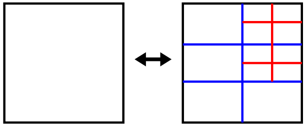
- Each proc: **clump** of child cells + **ghost cells** within cutoff
- **One-pass communication** per step, if cutoff long enough

Interesting features, current and future

- Adaptive gridding
- Run on GPU, KNL via Kokkos
- Code extensibility
- **Distributed surfaces**

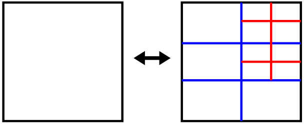
Adaptive gridding

- **Hierarchical** grid
  - top-level is single grid cell = simulation box
  - each parent cell has variable Nx by Ny by Nz child cells
  - recurse as many levels as desired (64-bit cell IDs)
  - oct-tree is 2x2x2 case, up to 15 levels
- Each child or parent cell considered **independently**



Adaptive gridding

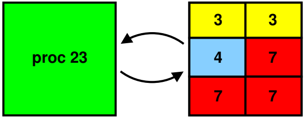
- **Hierarchical** grid
  - top-level is single grid cell = simulation box
  - each parent cell has variable Nx by Ny by Nz child cells
  - recurse as many levels as desired (64-bit cell IDs)
  - oct-tree is 2x2x2 case, up to 15 levels
- Each child or parent cell considered **independently**



- **Refinement/coarsening criteria**:
  - any per-grid value, current or time-averaged
  - geometric: nearness to upwind surface elements
  - flow: number of particles in cell,
    mean-free-path $= \lambda = \{\sqrt{2}\pi D_{\mathrm{ref}}^2 n (T_{\mathrm{ref}}/T)^{\omega-1/2}\}^{-1}$

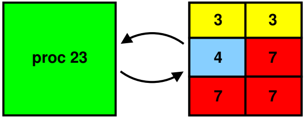
Parallel implementation

- **Refinement** is a **local** operation
  - proc that owns a child cell can decide to refine, create new cells
  - all procs can do this without communication
- **Coarsening** may **not be local** operation
  - if one proc owns all children, then local
  - if multiple procs own children, communication needed



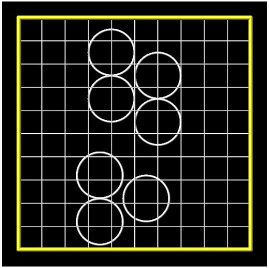
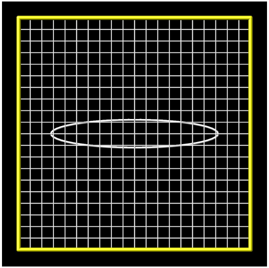
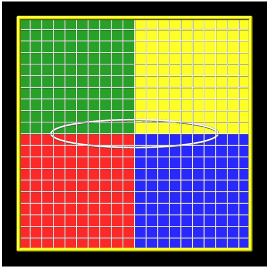
Parallel implementation

- **Refinement** is a **local** operation
  - proc that owns a child cell can decide to refine, create new cells
  - all procs can do this without communication
- **Coarsening** may **not be local** operation
  - if one proc owns all children, then local
  - if multiple procs own children, communication needed

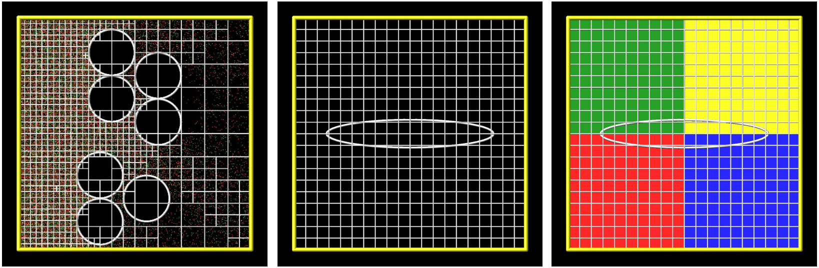


- Use **rendezvous** algorithm for non-local coarsening
  - owner of each child may not know who owns other children
  - everyone knows rendezvous proc, send child info to it
- **Rendezvous processor**
  - owner of parent cell, assigned in round-robin fashion
  - gathers info from all children, decides whether to coarsen
  - communicates result to all procs owning child cells

Simple 2d examples

Simple 2d examples

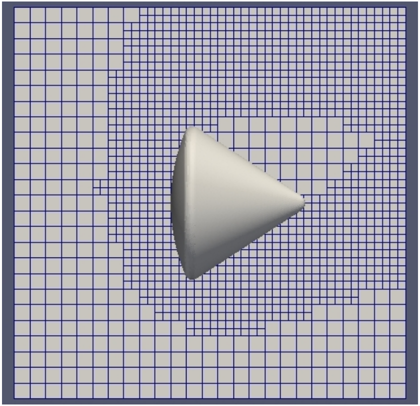


For moving surfaces:

- remove particles from any cell cut by element
- assume move is slow enough to allow particles to re-equilibrate

3d flow around Apollo capsule

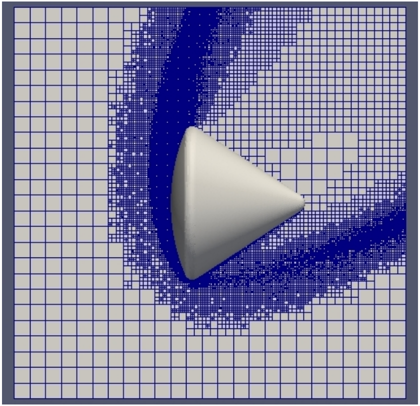
- 74 million particles, initial coarse grid $= 25^3 = 16\text{K}$ cells



- Final 5-level refined grid $= 6.5$ million child cells
- Uniform grid at fully refined scale $= 800^3 = 512\text{M}$ cells

3d flow around Apollo capsule

- 74 million particles, initial coarse grid $= 25^3 = 16\text{K}$ cells



- Final 5-level refined grid $=$ 6.5 million child cells
- Uniform grid at fully refined scale $= 800^3 = 512\text{M}$ cells

Run SPARTA on GPU, KNL via Kokkos

- Work by Stan Moore and Dan Ibanez
- Kokkos: **https://github.com/kokkos**
  - one solution for MPI+X programming model
  - goal: (re)write application kernels only once,
    run efficiently on variety of current/future hardware

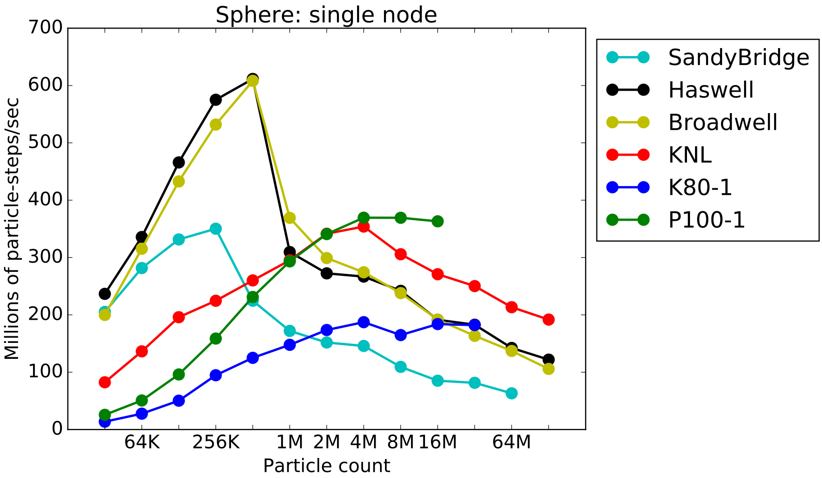
Run SPARTA on GPU, KNL via Kokkos

- Work by Stan Moore and Dan Ibanez
- Kokkos: **https://github.com/kokkos**
  - one solution for MPI+X programming model
  - goal: (re)write application kernels only once,
    run efficiently on variety of current/future hardware
- Two major components:
  1. Data access abstraction via **Kokkos arrays**
     - optimal layout & access pattern for each device
     - CPU, GPU, KNL, etc
  2. **Parallel dispatch** of small chunks of work
     - auto-mapped onto back-end languages
     - CUDA, OpenMP, etc

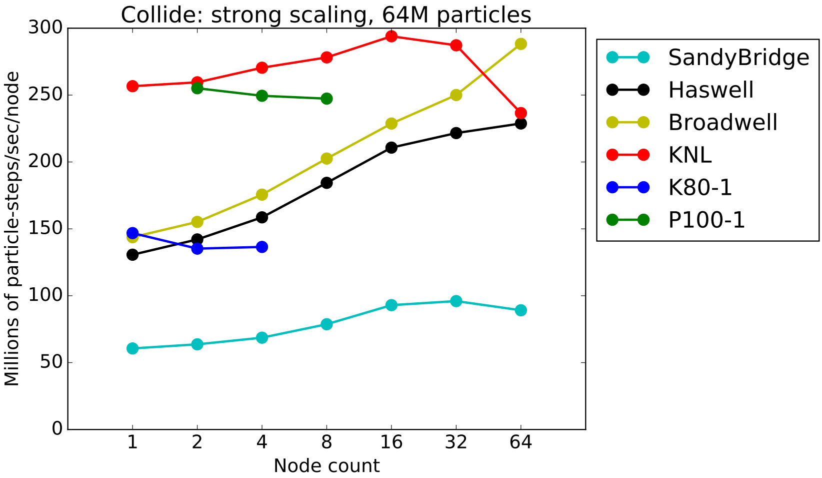
Run SPARTA on GPU, KNL via Kokkos

- Work by Stan Moore and Dan Ibanez
- Kokkos: **https://github.com/kokkos**
  - one solution for MPI+X programming model
  - goal: (re)write application kernels only once,
    run efficiently on variety of current/future hardware
- Two major components:
  1. Data access abstraction via **Kokkos arrays**
     - optimal layout & access pattern for each device
     - CPU, GPU, KNL, etc
  2. **Parallel dispatch** of small chunks of work
     - auto-mapped onto back-end languages
     - CUDA, OpenMP, etc
- Incremental rewrite of SPARTA **kernels** so they:
  - use Kokkos data structures, fine granularity, thread-safe
  - status: move, sort, collide, **hierarchical grid, surfs**
  - not yet: reactions, many diagnostics, BC, etc

Single-node SPARTA performance on CPU, GPU, KNL



Strong-scaling SPARTA performance on CPU, GPU, KNL



SPARTA designed to be easy to extend

- Virtual parent class defines interface to a **capability**
- Many child classes (**features**) can implement the interface
- Currently, $\sim$50% of SPARTA code base is add-on child classes

SPARTA designed to be easy to extend

- Virtual parent class defines interface to a **capability**
- Many child classes (**features**) can implement the interface
- Currently, $\sim$50% of SPARTA code base is add-on child classes

Examples:

- collision & reaction models = VSS, VHS, QK, TCE, hybrid
- surface collision models = specular, diffuse, piston, vanish
- surface reaction models = global, list of reactions/probabilities
- **diagnostics** (16) = various per-particle, per-grid, per-surf
- **fixes** (14) (operate while timestepping) = particle emission, time averaging, load-balancing, move surfs, etc
- fixes can define additional per-particle attributes: **ambipolar**, polyatomic vibrational **energy levels**

SPARTA can be used as a library

At least **two purposes**:

- Use SPARTA as a tool in a higher-level **workflow**
  - can invoke multiple instances of SPARTA
- Enable **multi-physics** or **multi-scale** models
  - couple to continuum or molecular collision models

SPARTA can be used as a library

At least **two purposes**:

- Use SPARTA as a tool in a higher-level **workflow**
  - can invoke multiple instances of SPARTA
- Enable **multi-physics** or **multi-scale** models
  - couple to continuum or molecular collision models

Software details:

- C-style interface, callable from C++/C/Fortran/**Python**
- Parallel python via mpi4py
- Lib interface is easy to extend:
  - add functions to library.cpp/h
  - add wrapper methods to sparta.py for Python
- Umbrella code/script can invoke SPARTA and another code, pass info between them

SPARTA can be used as a library

At least **two purposes**:

- Use SPARTA as a tool in a higher-level **workflow**
  - can invoke multiple instances of SPARTA
- Enable **multi-physics** or **multi-scale** models
  - couple to continuum or molecular collision models

Software details:

- C-style interface, callable from C++/C/Fortran/**Python**
- Parallel python via mpi4py
- Lib interface is easy to extend:
  - add functions to library.cpp/h
  - add wrapper methods to sparta.py for Python
- Umbrella code/script can invoke SPARTA and another code, pass info between them

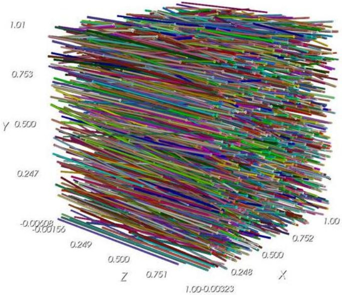
Caveat: not solving the hard coupling problem, just software issue

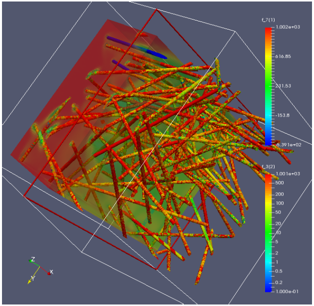
Distributed surface elements

50K triangles is good enough for the Mir space station, but ...

Distributed surface elements

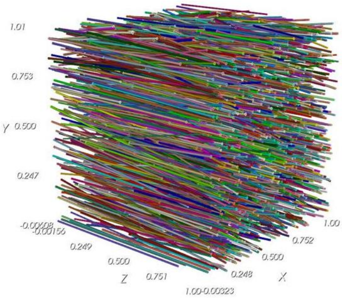
50K triangles is good enough for the Mir space station, but ...
Some **crazy users** want to model 10-100Ms of surface elements!

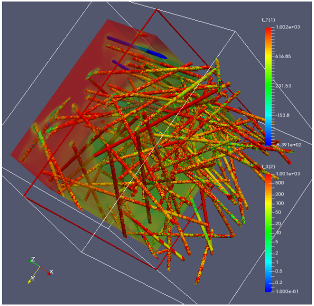




Distributed surface elements

50K triangles is good enough for the Mir space station, but ...
Some **crazy users** want to model 10-100Ms of surface elements!





- Currently each proc stores all surfs:
  - easier to not worry when a surf overlaps many cells/procs
- One surf = $\sim$80 bytes + tallies $\Rightarrow$ **100s GBs**
- **New option** for proc to only store surfs in owned+ghost cells

Complicating factors

- Load-balancing
  - surfs now have to **migrate** with cells/particles
- Per-surf-element **statistics**
  - currenty tallying is via MPI_Allreduce()
  - will need to do something more clever
- Surface **erosion**:
  - how to maintain **topology**
  - more careful nearby particle deletion/creation
- Do we need to store per-surf-element **state** ?
  - multiple procs may contribute to one element's state
  - how to insure state is **up-to-date**

Thanks and links

- **Funding** support: DOE/NNSA ASC and ATDM programs

- **http://sparta.sandia.gov**
- SPARTA short course:
  http://sparta.sandia.gov/**tutorials.html**

- Stan Moore and Dan Ibanez (Sandia): Kokkos work
- **https://github.com/kokkos**

- Looking to **collaborate** on new ideas for SPARTA
- magalli@sandia.gov, sjplimp@sandia.gov