is paper describes objective technical results and analysis. Any subjective views or opinions that might be expres
the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Governm

# Rendezvous algorithms for large-scale particle sim

SAND2020-1688C

Steve Plimpton (SNL) and Chris Knight (ANL)

SIAM Conference on Parallel Processing
for Scientific Computing (PP20)

Seattle - Feb 2020

CCR
Center for Computing Research

Sandia National

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

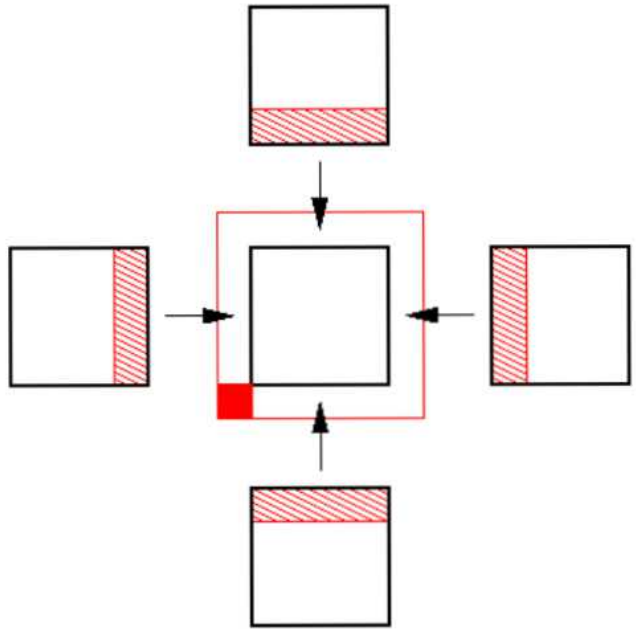
NNSA

oratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia,
neywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract

# Communication patterns

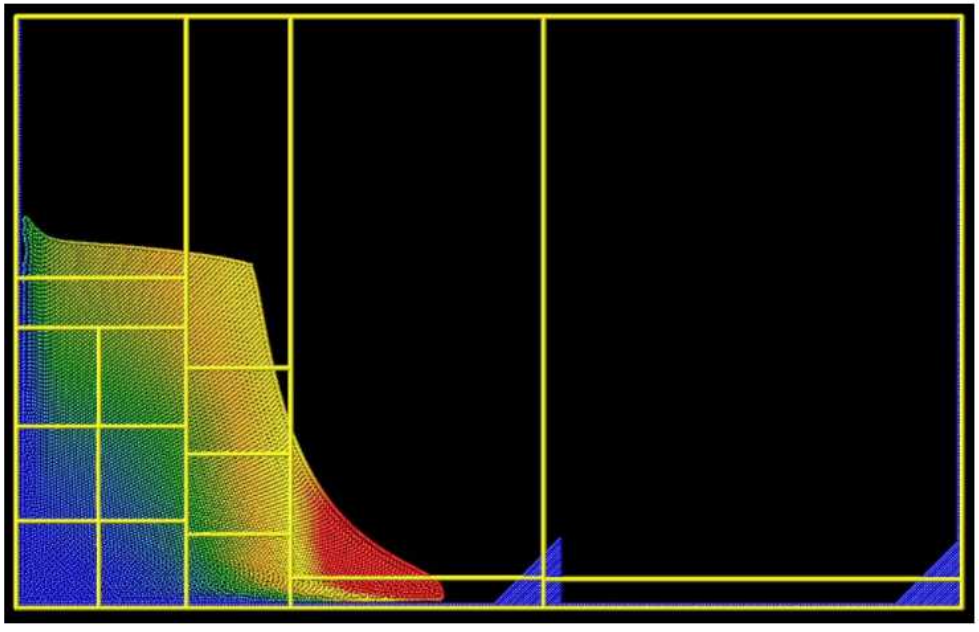
(1) Regular, often structured

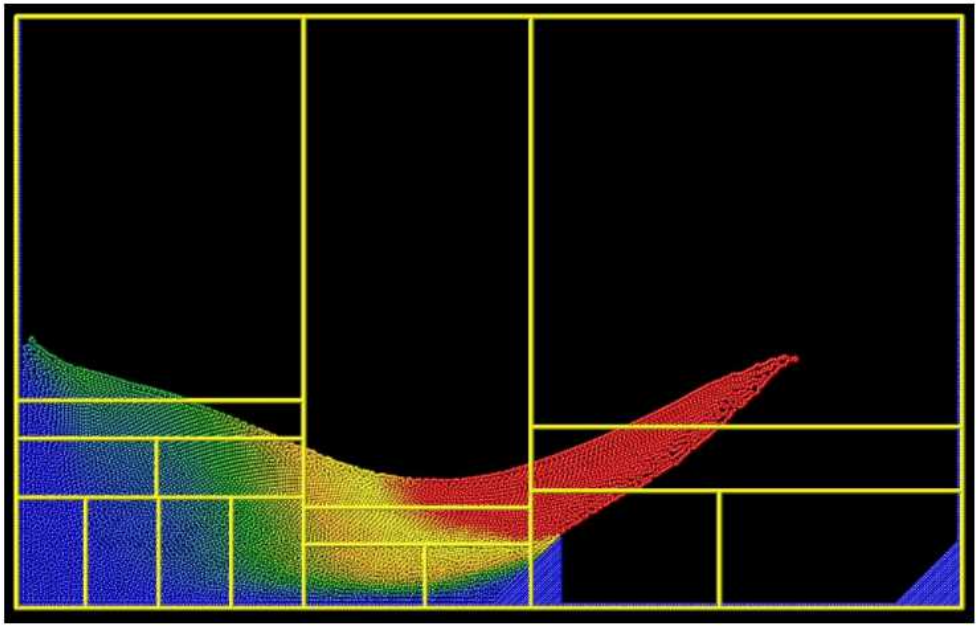
- Procs **know** who to **send** to,
  and who to **receive** from
- Domain decomp,
  halo of ghost cells or particles
- Exchange with
  4 or 8 procs in 2d, 6 or 26 in 3d
- MPI:
  Send(), Recv(), Irecv(), Sendrecv()



# Communication patterns

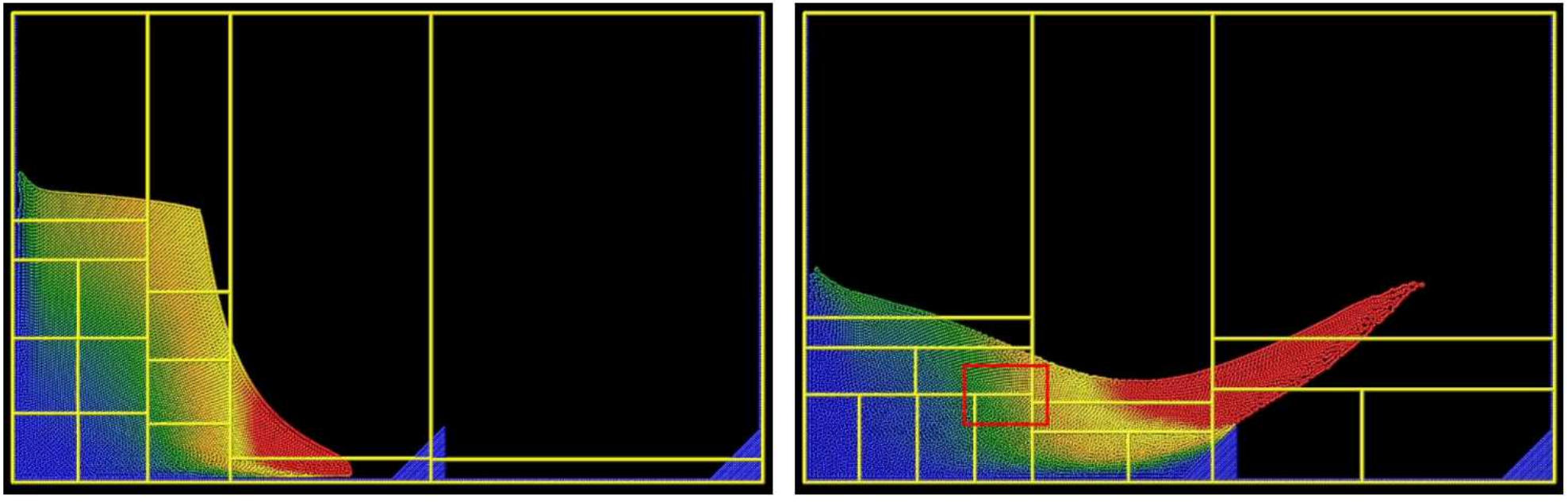
(2) Irregular, often unstructured





# Communication patterns

(2) Irregular, often unstructured



# Communication patterns

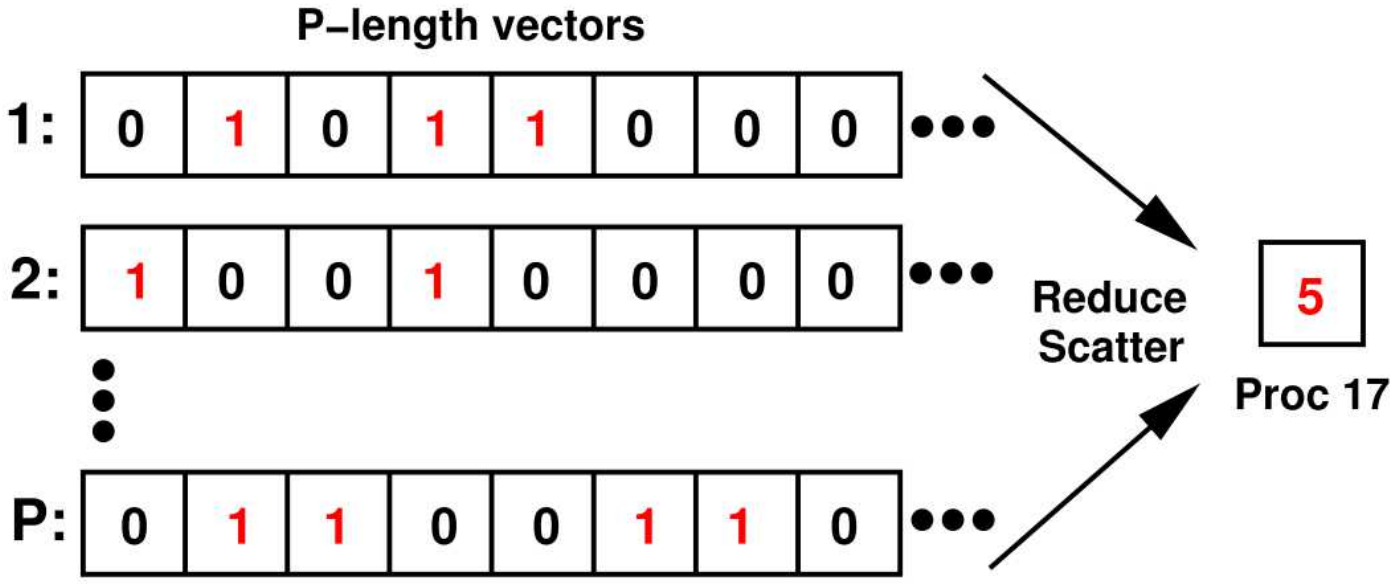
(2) Irregular, often unstructured

- Procs **know** who to **send** to, but **not** who to **receive** from
- Load rebalance: send my data to new owning procs
- Comm with small, but arbitrary # of other procs
- MPI: Reduce_scatter(), then Send(), Recv(), Irecv()

# Communication patterns

(2) Irregular, often unstructured

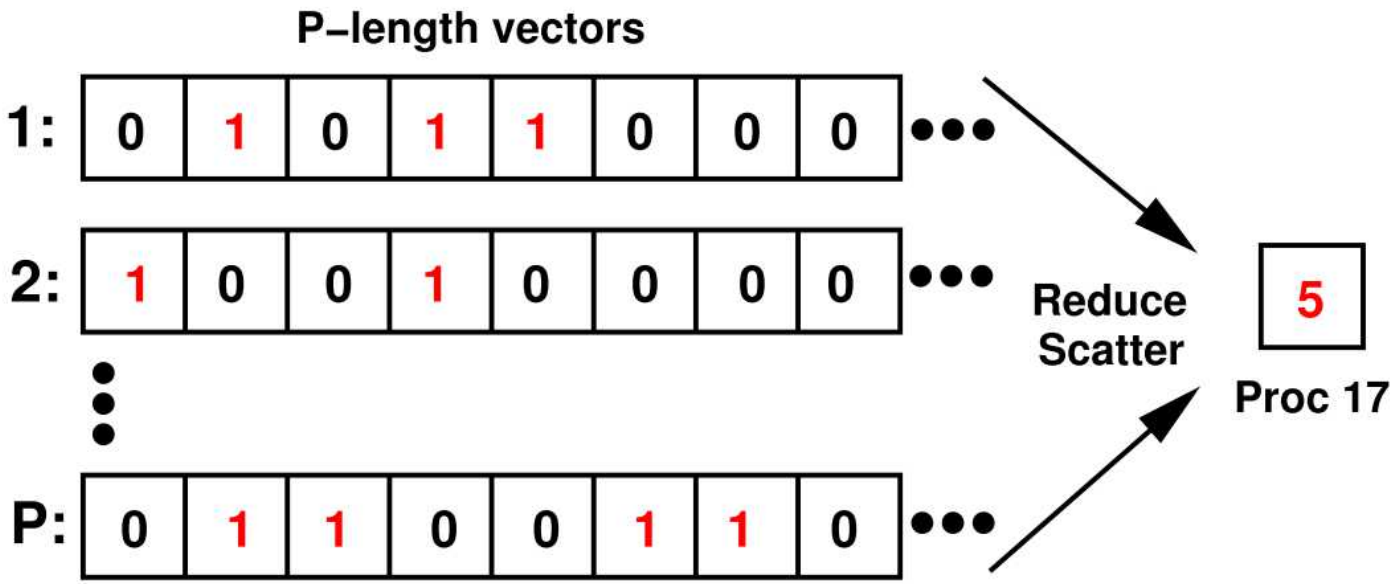
- Procs **know** who to **send** to, but **not** who to **receive** from
- Load rebalance: send my data to new owning procs
- Comm with small, but arbitrary # of other procs
- MPI: Reduce_scatter(), then Send(), Recv(), Irecv()



# Communication patterns

(2) Irregular, often unstructured

- Procs **know** who to **send** to, but **not** who to **receive** from
- Load rebalance: send my data to new owning procs
- Comm with small, but arbitrary # of other procs
- MPI: Reduce_scatter(), then Send(), Recv(), Irecv()




- Trilinos and Zoltan packages use this pattern

# Another communication pattern

(3) Rendezvous

- Procs **don't know** who to **send** to, **nor** who to **receive** from
- **Key idea**: create an **intermediate decomposition**
- Procs know who to send to (in Rvous decomp)
- Procs know who to receive from (sent from Rvous decomp)
- Often random comm with a few or nearly all procs
- Randomization **load balances** the comm and comp
- MPI: Reduce_scatter() or All2all()

# Another communication pattern

(3) Rendezvous

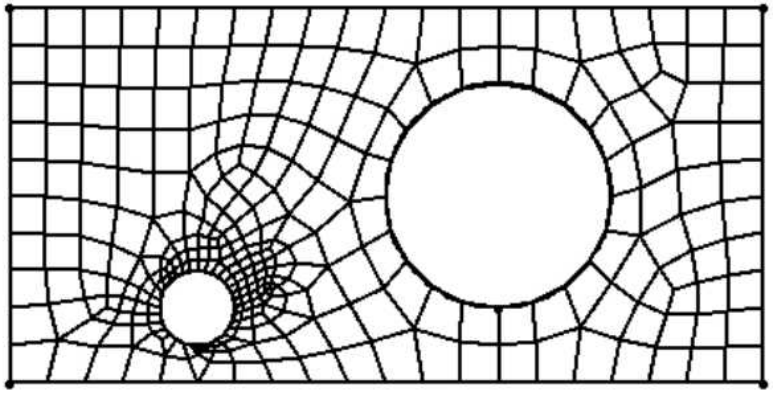
- Procs **don't know** who to **send** to, **nor** who to **receive** from
- **Key idea**: create an **intermediate decomposition**
- Procs know who to send to (in Rvous decomp)
- Procs know who to receive from (sent from Rvous decomp)
- Often random comm with a few or nearly all procs
- Randomization **load balances** the comm and comp
- MPI: Reduce_scatter() or All2all()

Why call it a **rendezvous algorithm**?

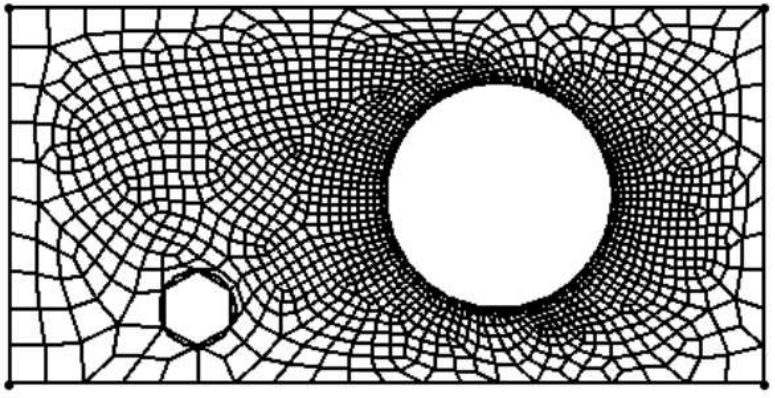
Because datums from different procs **rendezvous** at a single proc, so that proc can perform a computation that needs all the datums

# Rendezvous algorithm for grid transfer operation

*Plimpton, Hendrickson, Stewart, "A Parallel Rendezvous Algorithm for Interpolation Between Multiple Grids", J Parallel and Distributed Computing, 64, 266-276 (2004).*



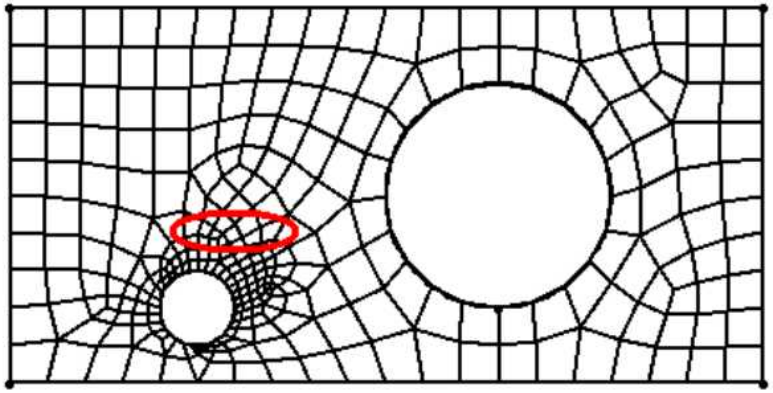
**thermal**



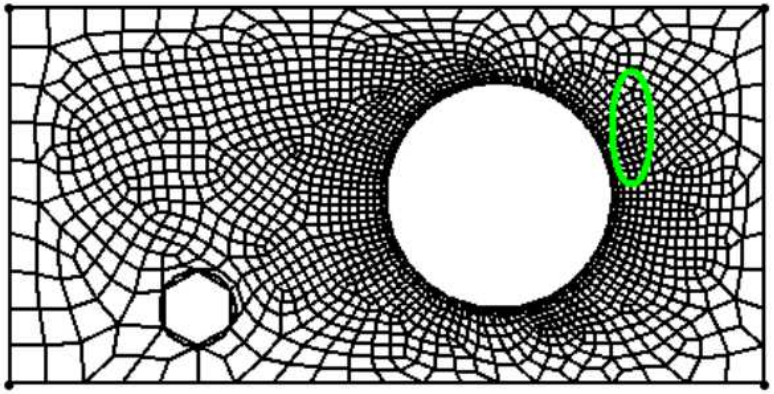
**stress**

# Rendezvous algorithm for grid transfer operation

*Plimpton, Hendrickson, Stewart, "A Parallel Rendezvous Algorithm for Interpolation Between Multiple Grids", J Parallel and Distributed Computing, 64, 266-276 (2004).*



**thermal**

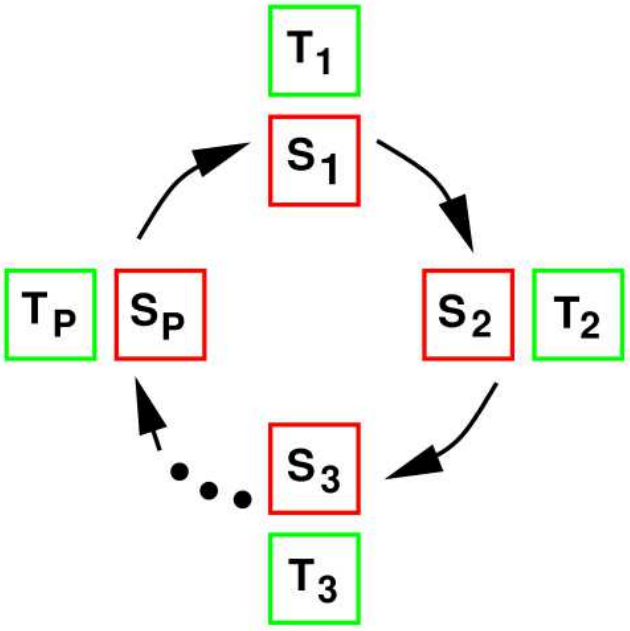


**stress**

- Two grids overlay same physical domain
- Refined & decomposed independently for physics & efficiency
- Each proc owns a random sub-domain in **both** decomps
- Interpolate between 2 grids, back-and-forth each timestep
- T⇒S requires data from all T procs which overlap a S proc
- But procs **know nothing** about either global decomposition

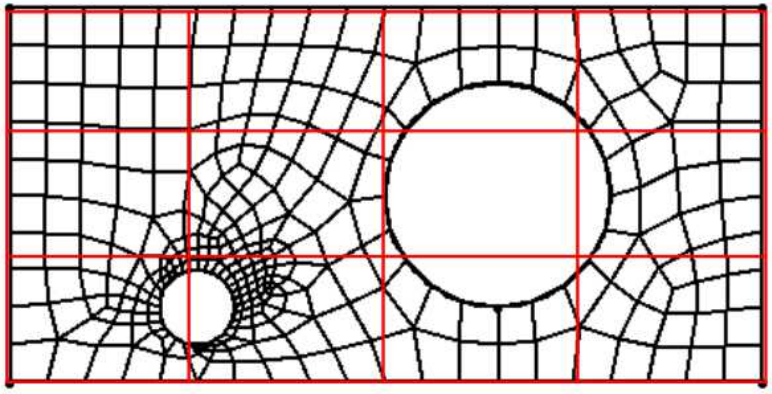
# Brute force solution

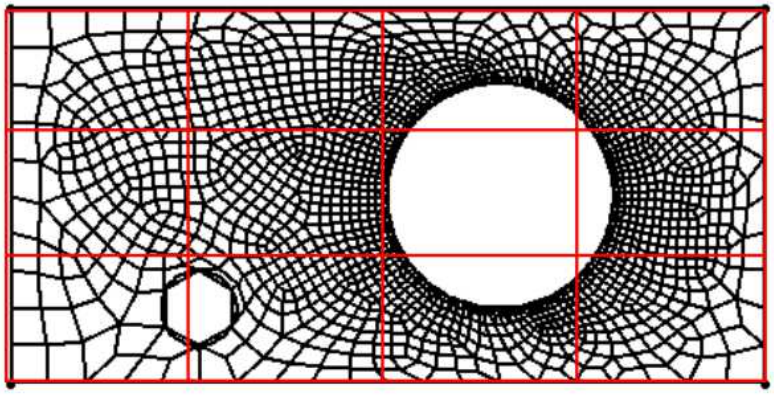
**Ring** communication:




- **Circulate** my S decomp grid cells **around ring to all procs**
- Each receive: keep S cells that overlap my T cells
- After P steps, each T proc can now perform interpolation
- Comp scaling: examine all N grid cells $\Rightarrow$ O(N)
- Comm scaling: P messages of size N/P $\Rightarrow$ O(N)
- **If occasional**, simple and actually OK for modest N and P
- But not for **huge N or P** or every step

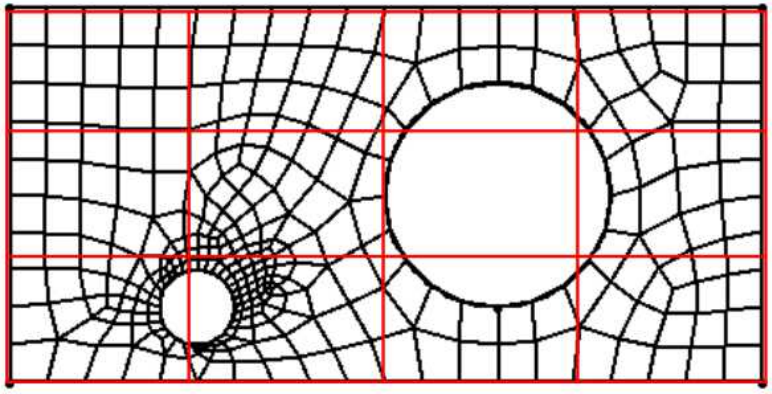
# Rendezvous solution

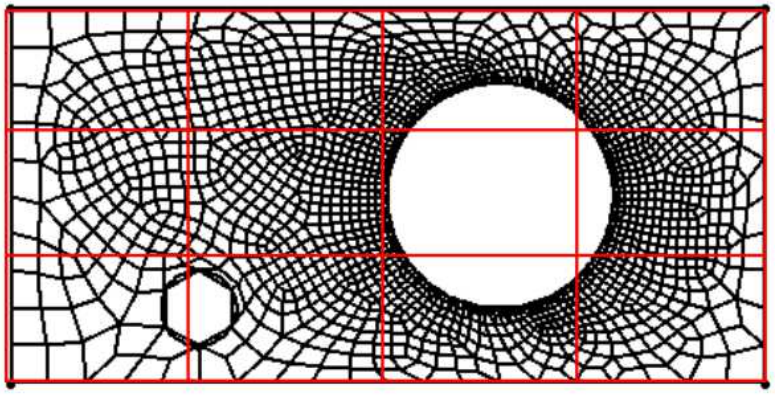




- Create a geometrically-based **rendezvous decomp** R
- Regular grid of procs, or RCB (for load balancing)
- **Important**: all procs are part of all 3 decomps: S,T,R
- T procs: send info for each grid cell to owning R proc
- S procs: ditto (at same time)
- R procs: find cell/cell overlaps, perform interpolation
- R procs: send final results to S or T procs

# Rendezvous solution



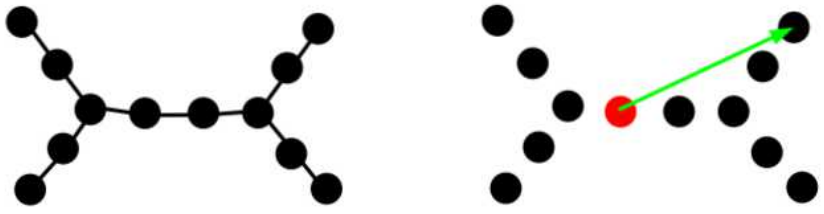


- Create a geometrically-based **rendezvous decomp** R
- Regular grid of procs, or RCB (for load balancing)
- **Important**: all procs are part of all 3 decomps: S,T,R
- T procs: send info for each grid cell to owning R proc
- S procs: ditto (at same time)
- R procs: find cell/cell overlaps, perform interpolation
- R procs: send final results to S or T procs

JPDC paper showed rendezvous comm can be **quite fast**

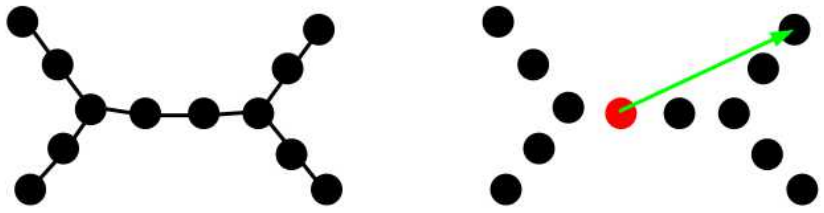
# Rigid body setup in LAMMPS

Identify central particle per body, bcast ID to other particles



# Rigid body setup in LAMMPS

Identify central particle per body, bcast ID to other particles



Rendezvous algorithm:

- Rvous decomp: each proc owns **random subset** of body IDs
- Send one datum per particle: particle ID, coords, proc ID
- Rvous proc receives all particles in body, computes info
- Send one datum per particle: particle ID, center particle ID

# Weak scaling results for rigid body setup

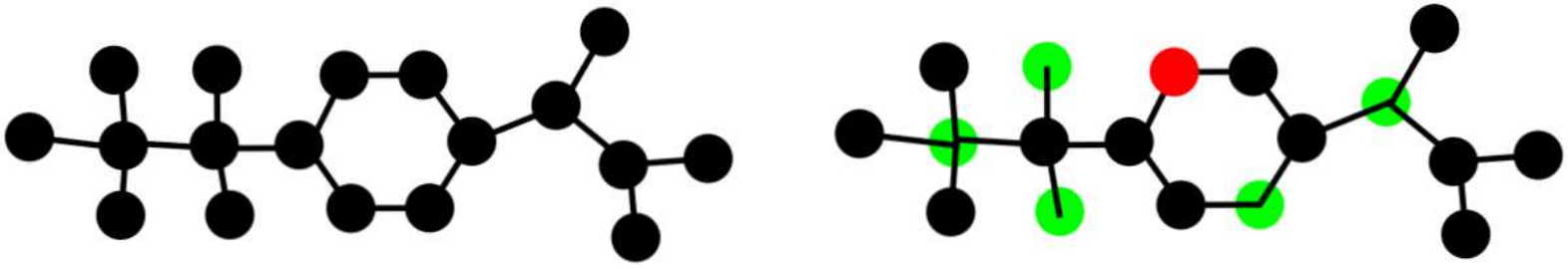
Mira BG/Q at ALCF: 1 node to 48K nodes, upto **9B** particles

| Nodes | 1 | 64 | 256 | 1K | 4K | 16K | 48K |
|---|---|---|---|---|---|---|---|
| MPI | 16 | 1K | 4K | 16K | 64K | 256K | 768K |
| Bodies | 5K | 336K | 1.3M | 5.4M | 22M | 86M | 258M |
| Atoms | 184K | 12M | 47M | 188M | 752M | 3.0B | 9.0B |
| Ring | **0.121** | 7.56 | 31.0 | 127 | 497 | *2000 | ***6000** |
| Rvous | **0.027** | 0.028 | 0.033 | 0.066 | 0.27 | 1.2 | **3.5** |

- Ring: each proc scans all the datums
- Rvous: each proc receives exactly the N/P data it needs
- Asterisk timings are estimates
- 4x to 1700x advantage for rendezvous algorithm
- Don't care on 256 nodes with 47M atoms, but do at scale

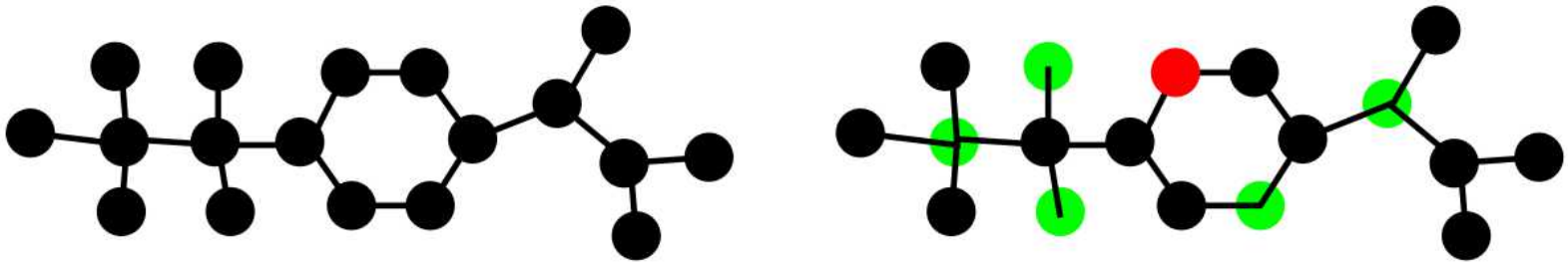
# Bond walking setup in LAMMPS

Molecules as graphs: find 1st, 2nd, 3rd, etc neighs of each atom



# Bond walking setup in LAMMPS

Molecules as graphs: find 1st, 2nd, 3rd, etc neighs of each atom



Rendezvous algorithm: **once for each level** of neighbors
Example: find 2nd neighs from 1st neighs

- Rvous decomp: each proc owns every Pth atom ID
- Send one datum (I, proc) per owned atom, to I owner
- Double loop over 1st neighbors of each atom I:
  - send **two datums** (J,K) to J owner and (K,J) to K owner
  - only comm atom pairs with a ghost atom
- Rvous proc re-sends (J,K) datums to J owner
- 3rd neighs: double loop over 1st neighs and 2nd neighs

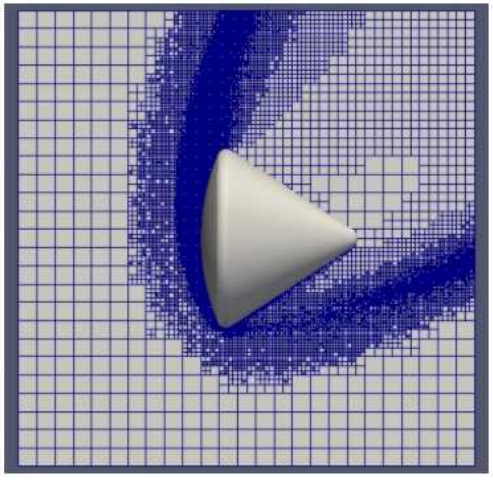
# Weak scaling results for bond walking setup

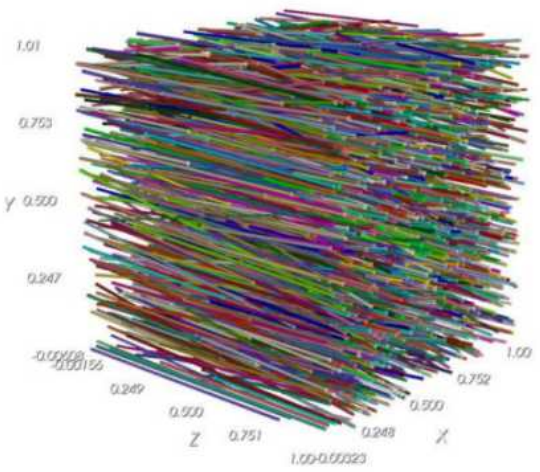
Mira BG/Q at ALCF: 64 nodes to 48K nodes, upto **37B** atoms

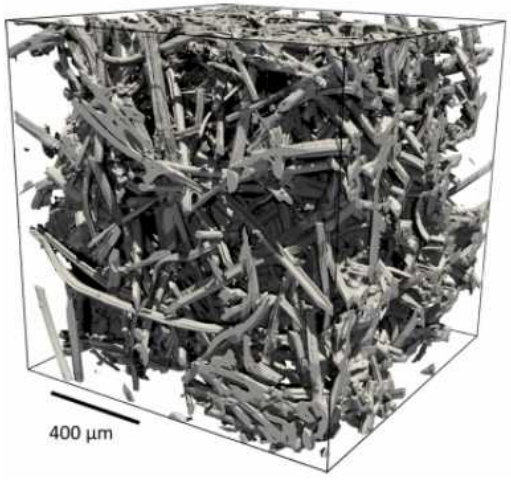
| Nodes | 64 | 256 | 1K | 4K | 16K | 32K | 48K |
|---|---|---|---|---|---|---|---|
| MPI | 1K | 4K | 16K | 64K | 256K | 512K | 768K |
| Atoms | 48M | 192M | 768M | 3.1B | 12B | 25B | 37B |
| Ring | **91.5** | 366 | 1465 | *6120 | *24500 | *49000 | ***73500** |
| Rvous | **0.912** | 0.942 | 1.14 | 1.82 | 4.95 | 9.15 | **13.2** |

- Again, asterisk timings are estimates
- 100x to 5000x advantage for rendezvous algorithm

# Surface element to grid cell mapping in SPARTA (DSMC)

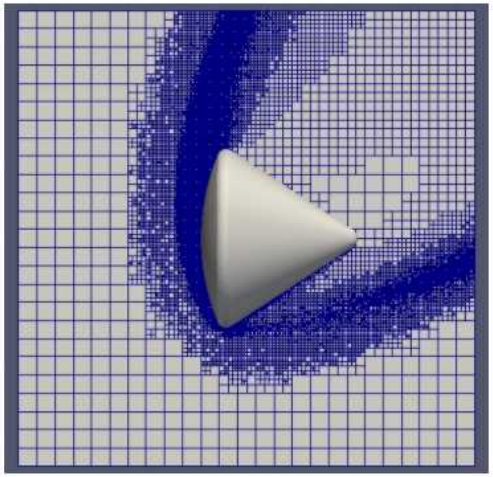


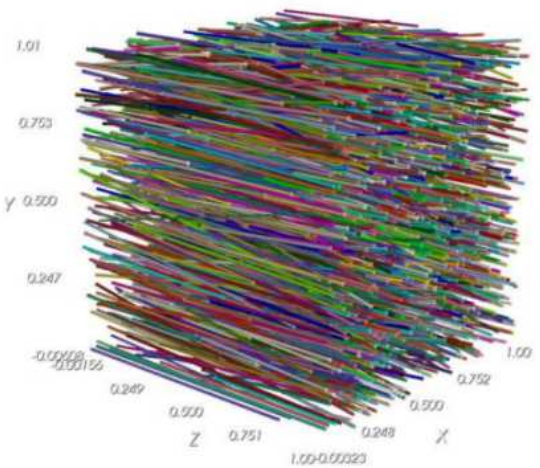



- Hierarchical grid, triangulated surfaces
- Each proc owns cluster of grid cells
- Each grid cell needs list of intersecting surface elements

# Surface element to grid cell mapping in SPARTA (DSMC)







- Hierarchical grid, triangulated surfaces
- Each proc owns cluster of grid cells
- Each grid cell needs list of intersecting surface elements
- For small triangle counts:

  faster to let each grid cell check all surfs
- For huge triangle counts: **faster to flip it**
  - each proc loops over N/P surfs
  - for each surf, identify grid cells in bounding box
  - create list of grid cells that intersect each surf
  - **But**: results are stored opposite of how needed

# Rendezvous converts cells-per-surf to surfs-per-cell

- Rvous decomp: each proc owns **random subset** of grid IDs
- Send one datum per owned grid cell: grid ID, proc ID
- Send one datum per intersection: grid ID, triangle ID
- Rvous proc re-sends (grid,triangle) datums to grid owner

# Rendezvous converts cells-per-surf to surfs-per-cell

- Rvous decomp: each proc owns **random subset** of grid IDs
- Send one datum per owned grid cell: grid ID, proc ID
- Send one datum per intersection: grid ID, triangle ID
- Rvous proc re-sends (grid,triangle) datums to grid owner

| Nodes | 64 | 64 | 256 | 256 | 1K | 1K | 4K | 16K |
|---|---|---|---|---|---|---|---|---|
| MPI | 1K | 1K | 4K | 4K | 16K | 16K | 64K | 256K |
| Surfs | 1.3M | 1.3M | 1.3M | 1.3M | 1.3M | **2M** | 1.3M | 1.3M |
| GCells | 1M | 8M | 8M | 64M | 1M | 128M | 64M | 256M |
| Old | **105** | 813 | 204 | 1600 | 104 | **1260** | 104 | 105 |
| Rvous | **3.2** | 3.5 | 3.0 | 3.3 | 2.7 | **7.5** | 2.9 | 3.9 |

- 1.3M or 2M triangles, vary grid cells and nodes
- 33x to 170x advantage for new inverted algorithm
- Rendezvous comm itself is small fraction of total

# Black-box implemention of rendezvous communication

rvous (N_in, indata, sendprocs1, callback()):

N_rvous = **MPI_All2allv**() of indata ⇒ indata_rvous

**callback**(N_rvous,indata_rvous,outdata_rvous,sendprocs2)

N_out = **MPI_All2all**() of outdata_rvous ⇒ outdata

return N_out, outdata

# Black-box implemention of rendezvous communication

rvous (N_in, indata, sendprocs1, callback()):

N_rvous = **MPI_All2allv**() of indata $\Rightarrow$ indata_rvous

**callback**(N_rvous,indata_rvous,outdata_rvous,sendprocs2)

N_out = **MPI_All2all**() of outdata_rvous $\Rightarrow$ outdata

return N_out, outdata

- 2 All2all comm ops, sandwiching a callback to process data
- Callback allows each proc to compute on Rvous data
- Can use **irregular** comm operation instead
- Irregular is faster if each proc sends to a few procs
- All2all is faster if each proc sends to (nearly) all procs

# One more communication pattern

MapReduce - from Google and 


- MapReduce is not just for data, also for scientific computing
- A mapper reads/creates datums and sends them to reducers

# One more communication pattern

MapReduce - from Google and 


- MapReduce is not just for data, also for scientific computing
- A mapper reads/creates datums and sends them to reducers
- Fundamental MapReduce communication operation:
  - Hadoop calls it a **shuffle**
  - conceptually identical to MPI_All2all()

# One more communication pattern

MapReduce - from Google and 


- MapReduce is not just for data, also for scientific computing
- A mapper reads/creates datums and sends them to reducers
- Fundamental MapReduce communication operation:
  - Hadoop calls it a **shuffle**
  - conceptually identical to MPI_All2all()
- A rendezvous alg is a subset of more general MapReduce
  - out-of-core, stream processing, Python wrappers, etc
  - cute elephant swag

# One more communication pattern

MapReduce - from Google and 


- MapReduce is not just for data, also for scientific computing
- A mapper reads/creates datums and sends them to reducers
- Fundamental MapReduce communication operation:
  - Hadoop calls it a **shuffle**
  - conceptually identical to MPI_All2all()
- A rendezvous alg is a subset of more general MapReduce
  - out-of-core, stream processing, Python wrappers, etc
  - cute elephant swag

MapReduce lib on top of MPI: **http://mapreduce.sandia.gov**
We used it for graph algorithms: e.g. connected components

# Conclusions

**Attributes** of rendezvous algorithms:

- Useful when don't know how to move data where needed
- Can auto-load-balance if randomly spread the data
- Leverages huge bisection message bandwidth of big computers
- Someone paid for it, why not use it !

# Conclusions

**Attributes** of rendezvous algorithms:

- Useful when don't know how to move data where needed
- Can auto-load-balance if randomly spread the data
- Leverages huge bisection message bandwidth of big computers
- Someone paid for it, why not use it !

**Lesson learned** (for Nth time):

- Better to load balance individual stages of a computation
- Pay the extra cost for comm in between
- Rather than perform all stages with poor load-balancing

# Conclusions

**Attributes** of rendezvous algorithms:

- Useful when don't know how to move data where needed
- Can auto-load-balance if randomly spread the data
- Leverages huge bisection message bandwidth of big computers
- Someone paid for it, why not use it !

**Lesson learned** (for Nth time):

- Better to load balance individual stages of a computation
- Pay the extra cost for comm in between
- Rather than perform all stages with poor load-balancing

What about **GPUs** ?

- There are no more machines with 1M+ MPI tasks
- Still useful at scale for ops that don't map to GPUs
- Occasional ops: setup, rebalance, grid-adaptation, etc