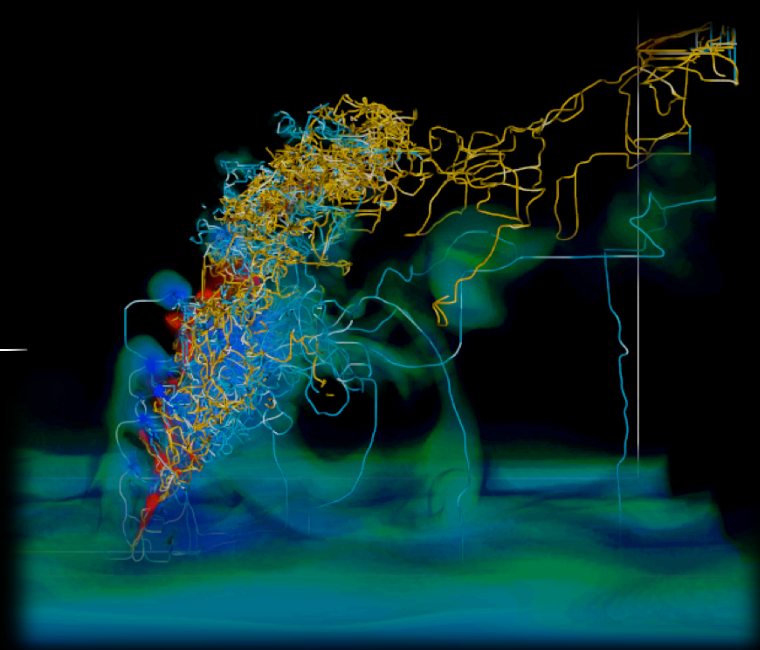
“I have had my results for a long time, but I do not yet know how I am to arrive at them.”

–Carl Friedrich Gauss, 1777-1855

In Situ Data Analysis



Morse-Smale Complex of combustion in the presence of a cross flow (image courtesy Attila Gyulassy)

Tom Peterka

tpeterka@mcs.anl.gov

Mathematics and Computer Science Division

CScADS Summer Workshop 7/26/12

Scalable Analysis & Visualization: The Data Parallel Approach

Treat analysis as any other parallel computation

-Decompose the domain

-Assign to processors

-Combine local and global operations

-Use parallel I/O, MPI, other programming models

-Balance load, minimize communication

-Measure strong, weak scaling, efficiency

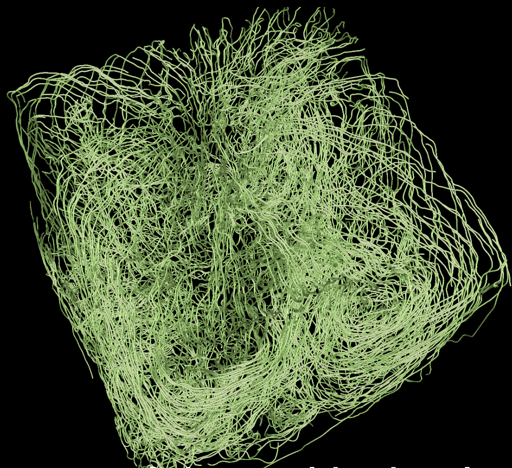
↓

Integrate with simulation

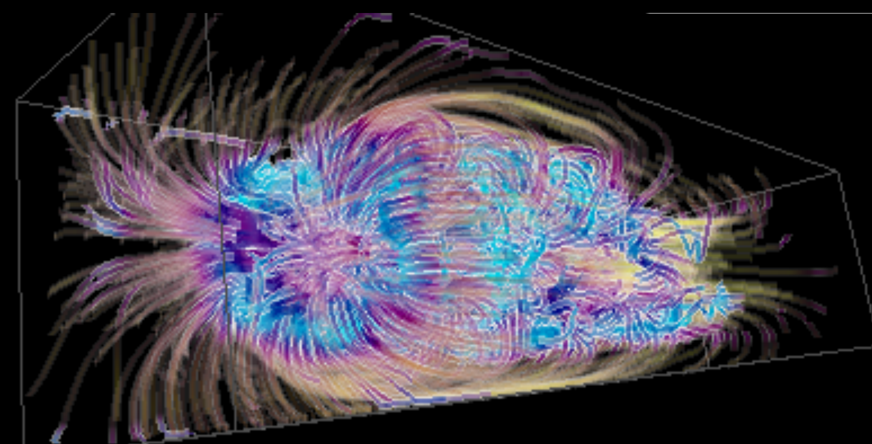
"The combination of massive scale and complexity is such that high performance computers will be needed to analyze data, as well as to generate it through modeling and simulation."

–Lucy Nowell, Scientific Data Management and Analysis at Extreme Scale, Office of Science Program Announcement LAB 10-256, 2010.

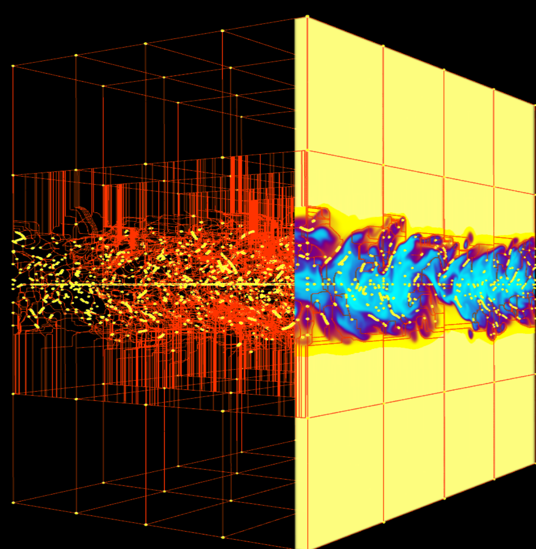
Data Analysis Comes in Many Flavors



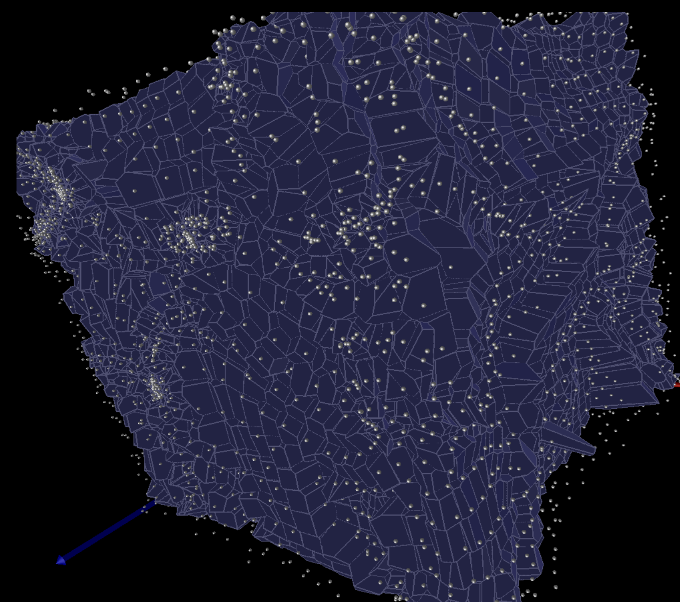
Particle tracing of thermal hydraulics flow



Information entropy analysis of astrophysics



Morse-Smale complex of combustion



Voronoi tessellation of cosmology

Separate Analysis Ops from Data Ops

Analysis	Application	Application Data Model	Analysis Data Model	Analysis Algorithm	Communication	Additional
Particle Tracing	CFD	Unstructured Mesh	Particles	Numerical Integration	Nearest neighbor	File I/O, Domain decomposition, process assignment, utilities
Information Entropy	Astrophysics	AMR	Histograms	Convolution	Global reduction, nearest neighbor	
Morse-Smale Complex	Combustion	Structured Grid	Complexes	Graph Simplification	Global reduction	
Computational Geometry	Cosmology	Particles	Tessellations	Voronoi	Nearest neighbor	

You do this yourself

Can use serial libraries such as OSUFlow, Qhull, VTK (don’t have to start from scratch)

DIY handles this

Tackling the Data-Intensive Part of Data Analysis

DIY: help the user write own data-parallel analysis algorithms.

Main ideas and Objectives

-Large-scale parallel analysis (visual and numerical) on HPC machines

-Scientists, visualization researchers, tool builders

-In situ, coprocessing, postprocessing

-Data-parallel problem decomposition

-Scalable data movement algorithms

Benefits

-Researchers can focus on their own work, not on parallel infrastructure

-Analysis applications can be custom

-Reuse core components and algorithms for performance and productivity

Implement Data Operations in a Library with a small ℓ

Features

Parallel I/O to/from storage
- -MPI-IO, BIL

Domain decomposition
- -Decompose domain
- -Describe existing decomposition

Network communication
- -Global reduction (2 flavors)
- -Local nearest neighbor

Library

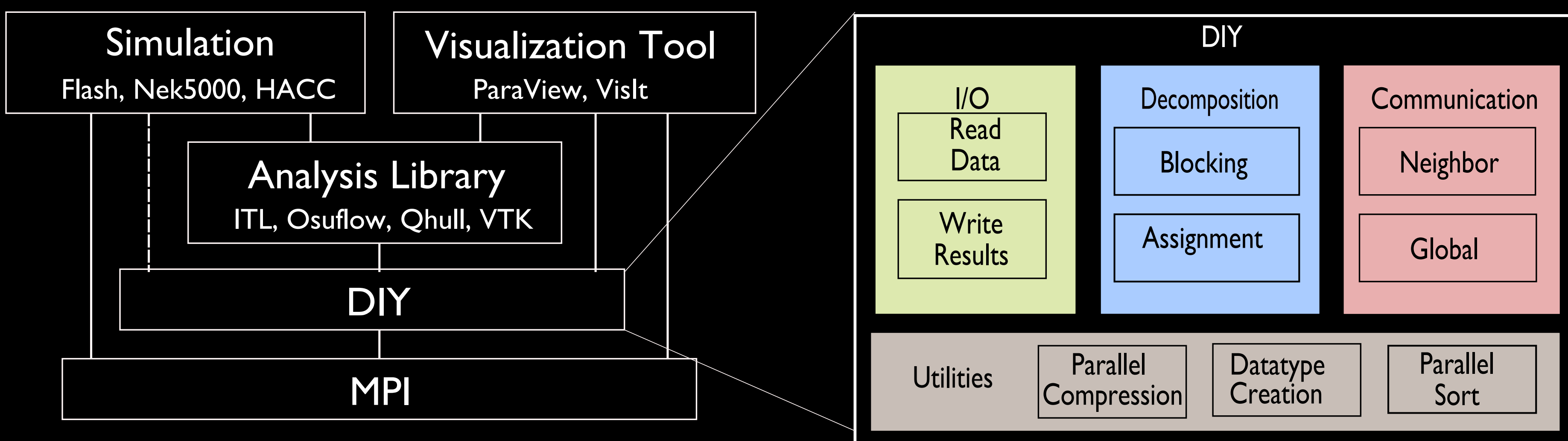
Written in C++

C bindings, future Fortran bindings

Autoconf build system (configure, make, make install)

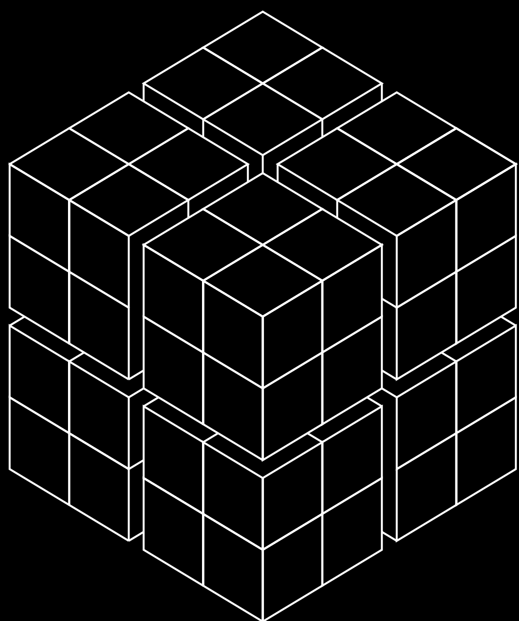
Lightweight: libdiy.a 800KB

Maintainable: ~15K lines of code, including examples

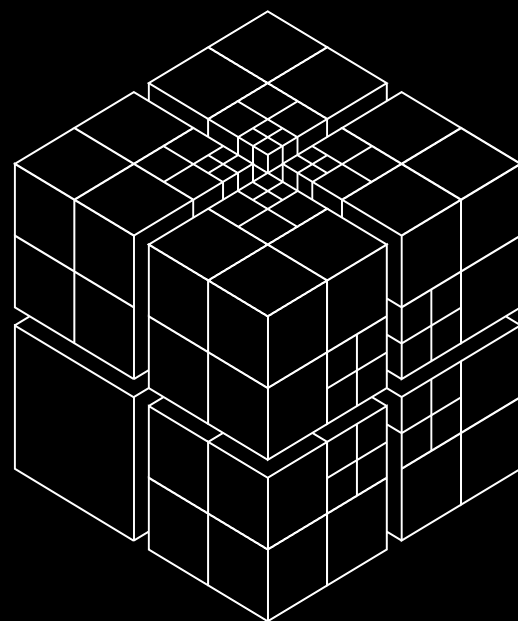


DIY usage and library organization

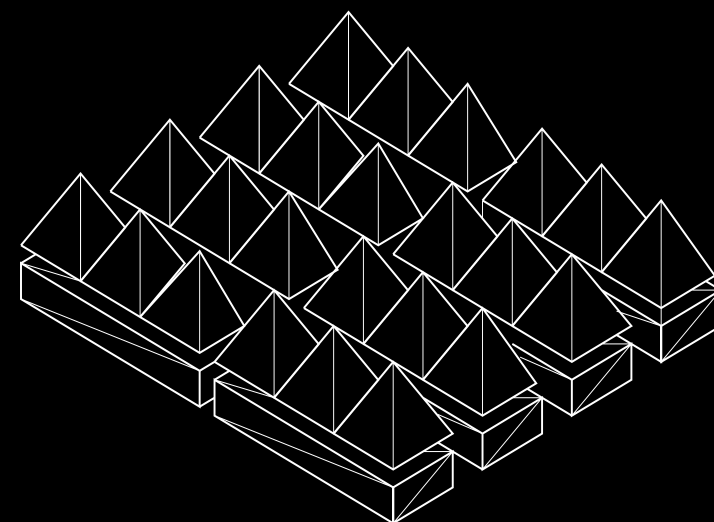
Group Data Items Into Blocks



Structured Grid



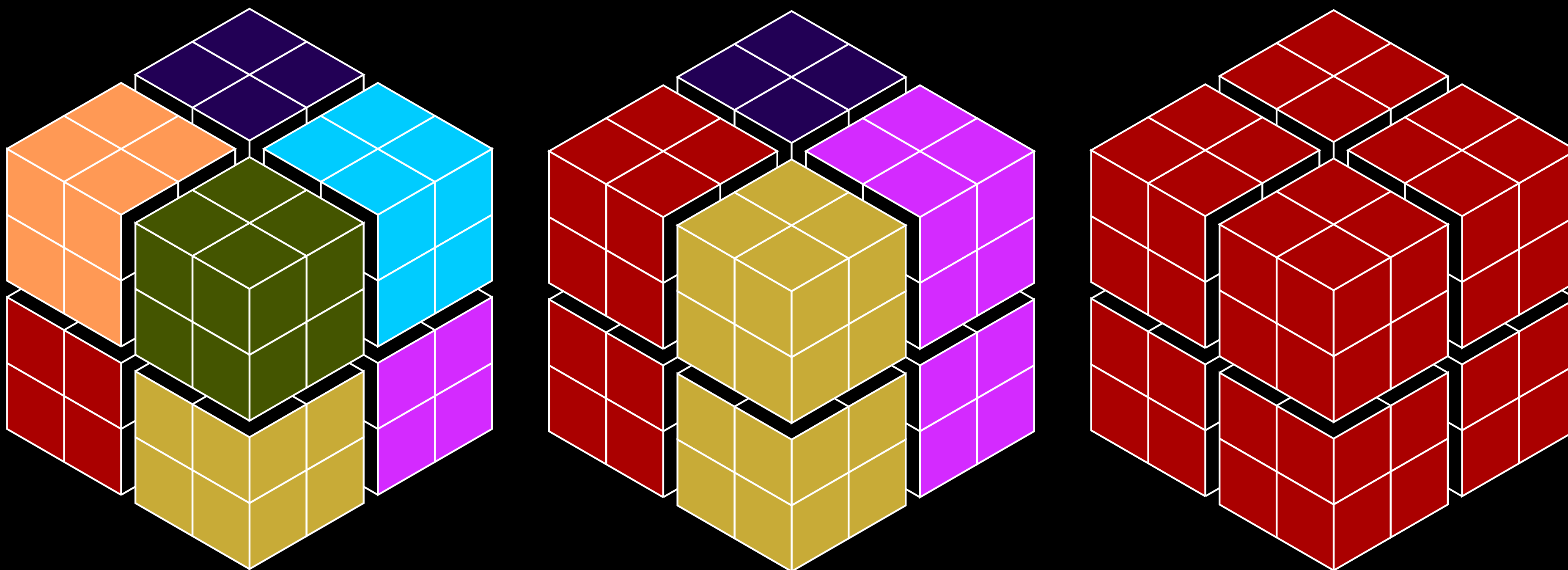
AMR Grid



Unstructured Mesh

The block is DIY's basic unit of data. Original dataset is decomposed into generic subsets called blocks, and associated analysis items live in the same blocks. Blocks contain one or more instances of the data type described earlier.

Block ≠ Process



8 processes

4 processes

1 process

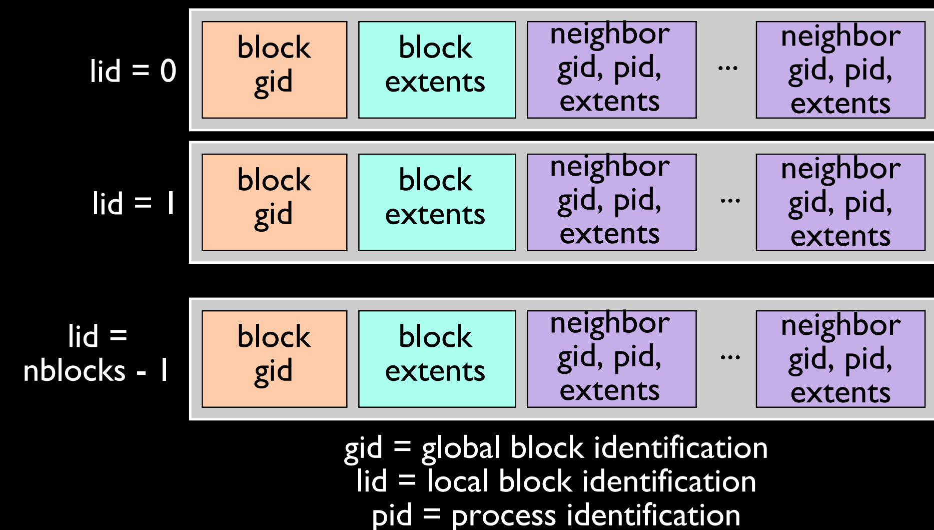
All data movement operations are per block; blocks exchange information with each other using DIY's communication algorithms. DIY manages and optimizes exchange between processes based on the process assignment. This allows for flexible process assignment as well as easy debugging.

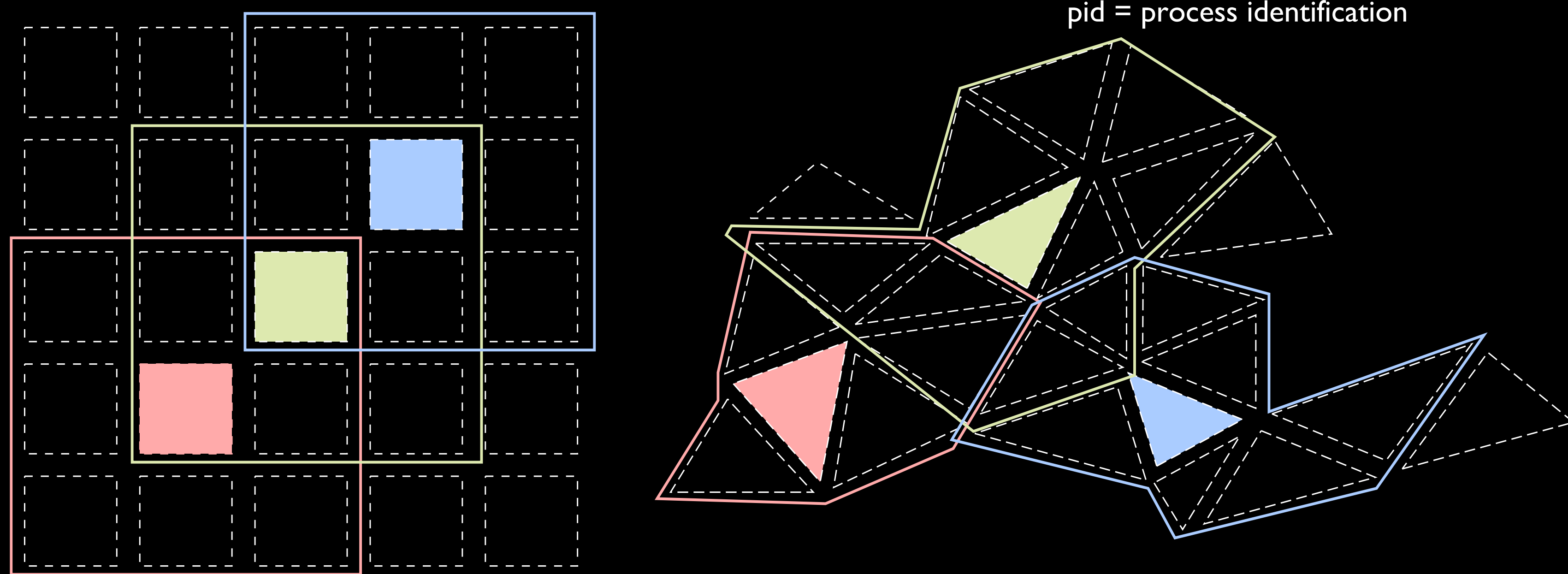
Group Blocks into Neighborhoods

-Limited-range communication

-Allow arbitrary groupings

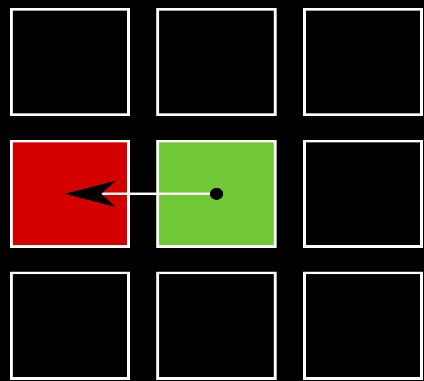
-Distributed, local data structure and knowledge of other blocks (not master-slave global knowledge)



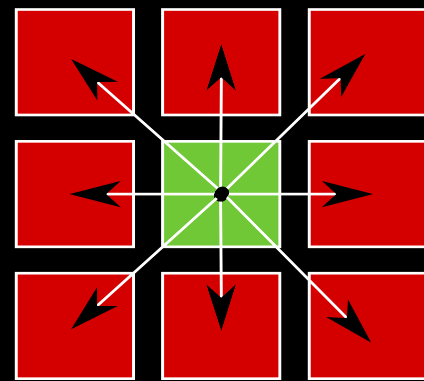


Two examples of 3 out of a total of 25 neighborhoods

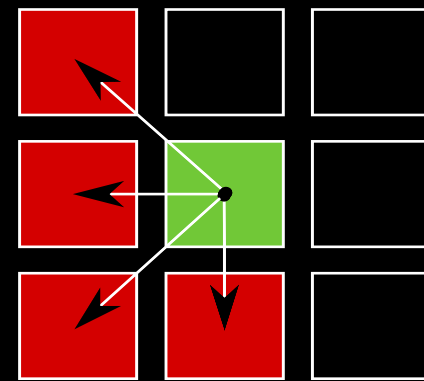
Provide Different Neighborhood Communication Patterns



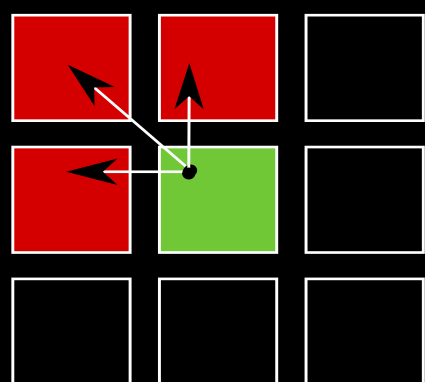
DIY_Enqueue_item_pt()
DIY_Enqueue_item_mask()



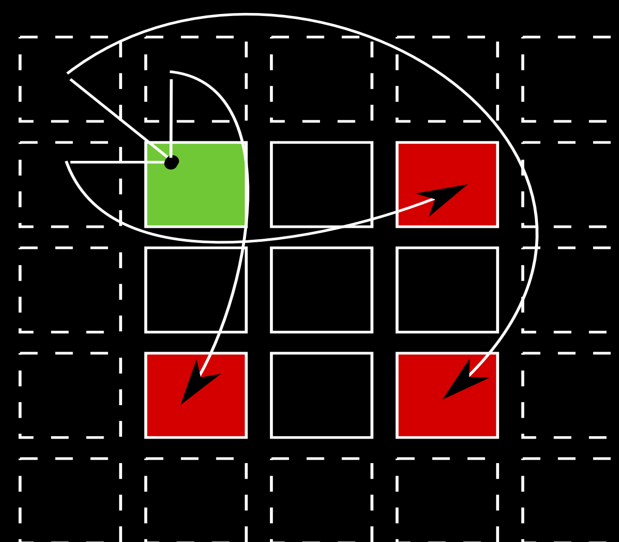
DIY_Enqueue_item_all()



DIY_Enqueue_item_half()



DIY_Enqueue_item_all_near()
DIY_Enqueue_item_half_near()



Support for wraparound neighbors
(repeating boundary conditions)

DIY provides point to point and different varieties of collectives within a neighborhood via its enqueue_item mechanism. Items are enqueued are subsequently exchanged (2 steps).

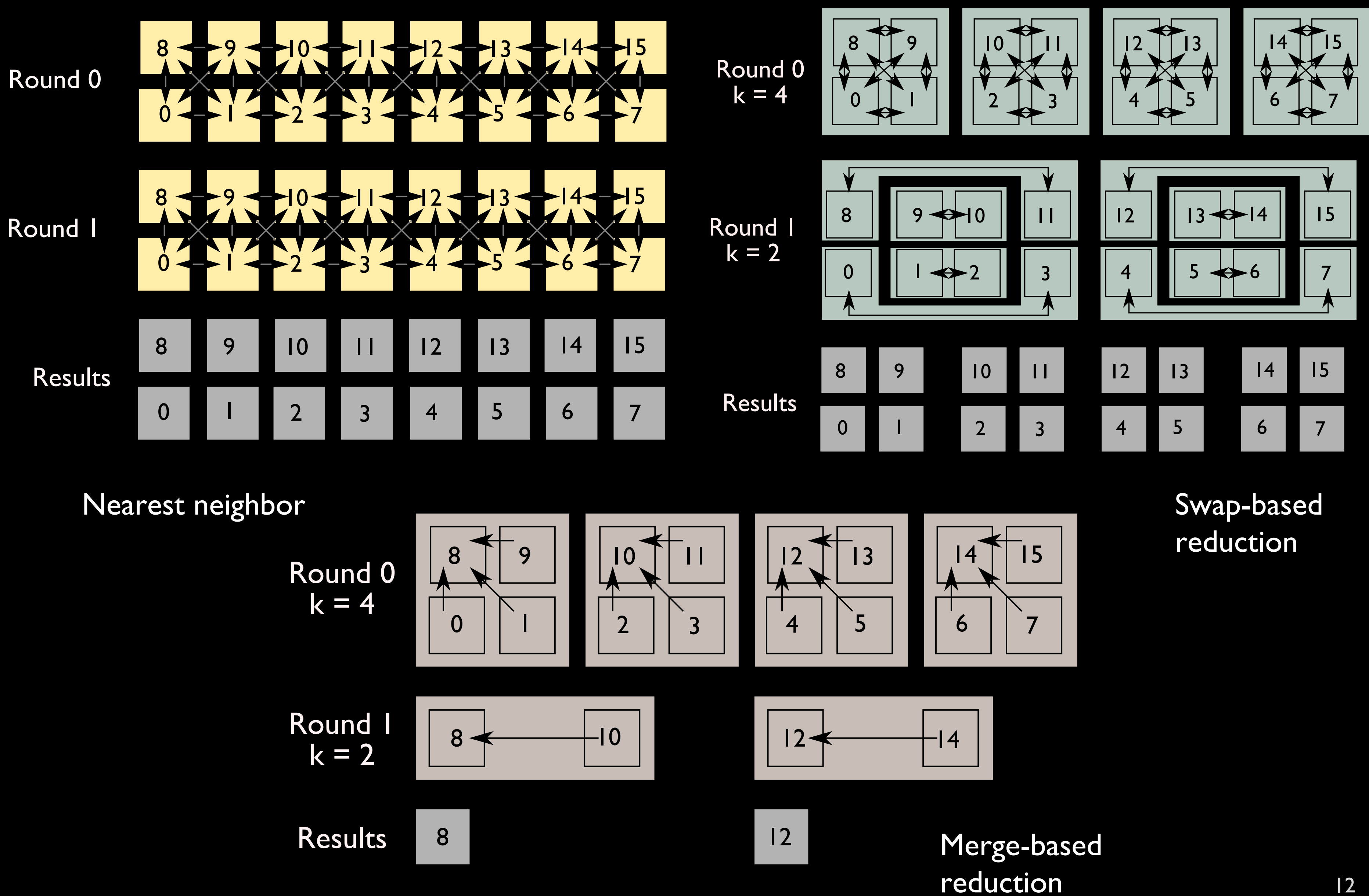
Make Global and Neighborhood Communication Fast and Easy

Analysis	Communication
Particle Tracing	Nearest neighbor
Global Information Entropy	Merge-based reduction
Point-wise Information Entropy	Nearest neighbor
Morse-Smale Complex	Merge-based reduction
Computational Geometry	Nearest neighbor
Region growing	Nearest neighbor
Sort-last rendering	Swap-based reduction

Factors to consider when selecting communication algorithm:
-associativity
-number of iterations
-data size vs. memory size
-homogeneity of data

DIY provides 3 efficient scalable communication algorithms on top of MPI. May be used in any combination.

3 Communication Patterns



Data Input

Multiblock and Multifile I/O

-Application-level two-phase I/O

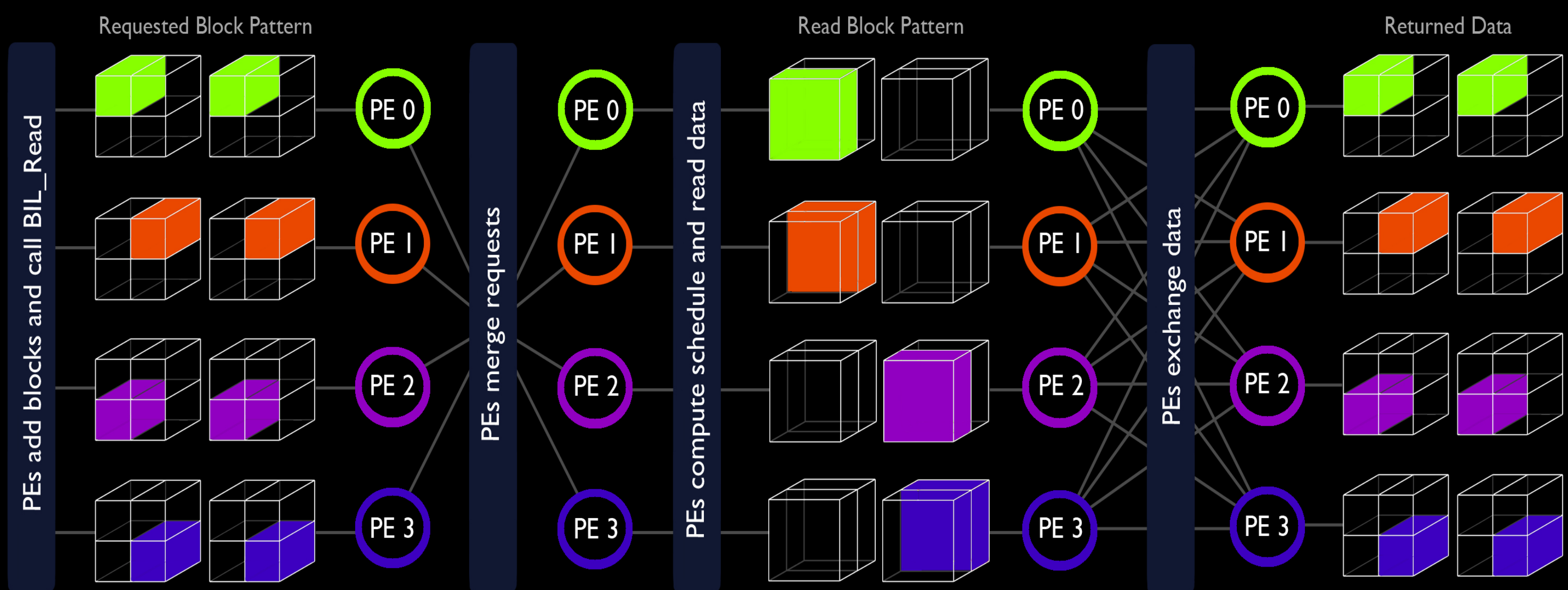
-Reads raw, netCDF, HDF5 (future)

-Read requests sorted and aggregated into large contiguous accesses

-Data redistributed to processes after reading

-Single and multi block/file domains

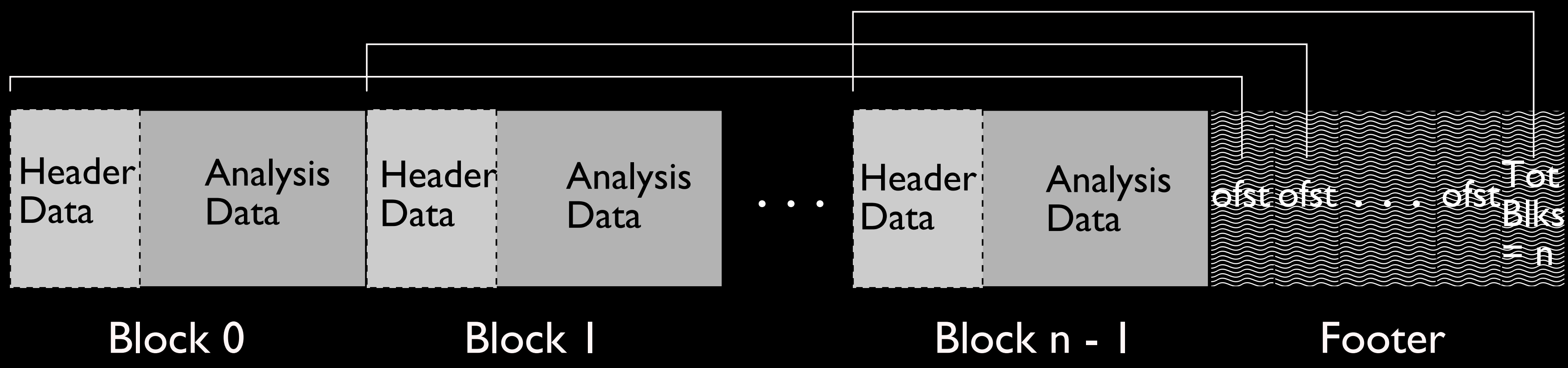
-75% of IOR benchmark on actual scientific data



Input algorithm

Kendall et al., Towards a General I/O Layer for Parallel Visualization Applications, CG&A ‘11

Analysis Output



Output file format

Features

Binary

General header/data blocks

Footer with indices

Application assigns semantic value to DIY blocks

Written efficiently in parallel

Parallel block-wise compression

Example Usage

```
// initialize
int dim = 3; // number of dimensions in the problem
int tot_blocks = 8; // total number of blocks
int data_size[3] = {10, 10, 10}; // data size
MPI_Init(&argc, &argv); // init MPI before DIY
DIY_Init(dim, ROUND_ROBIN_ORDER, tot_blocks, &nblocks,
    data_size, MPI_COMM_WORLD);
```

```
// decompose domain
int share_face = 0; // whether adjoining blocks share the same face
int ghost = 0; // additional layers of ghost cells
int ghost_dir = 0; // ghost cells apply to all or some sides of a block
int given[3] = {0, 0, 0}; // constraints on blocking (none)
DIY_Decompose(share_face, ghost, ghost_dir, given);
```

```
// read data
 for (int i = 0; i < nblocks; i++) {
   DIY_Block_starts_sizes(i, min, size);
   DIY_Read_add_block_raw(min, size, infile, MPI_INT, (void**)&(data[i]));
}
DIY_Read_blocks_all();
```

Example API Continued

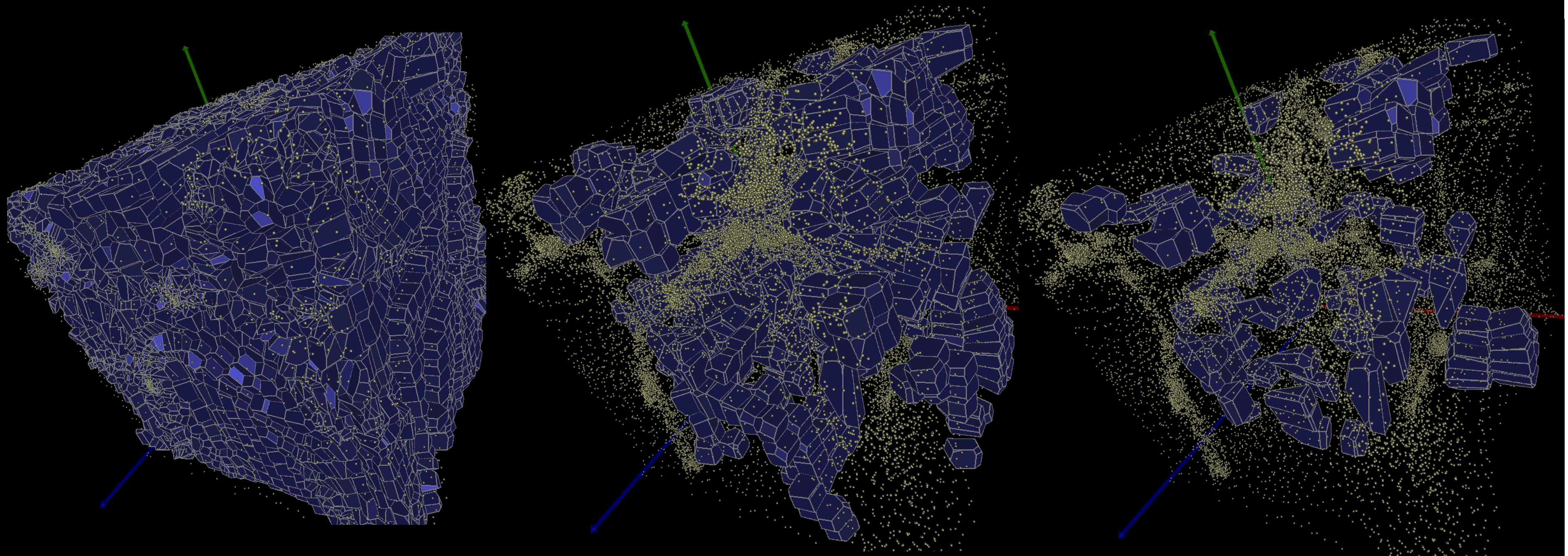
```
// your own local analysis


// merge results, in this example
// could be any combination / repetition of the three communication patterns
int rounds = 2; // two rounds of merging
int kvalues[2] = {4, 2}; // k-way merging, eg 4-way followed by 2-way merge
int nb_merged; // number of output merged blocks
DIY_Merge_blocks(in_blocks, hdrs, num_in_blocks, out_blocks, num_rounds, k_values,
&MergeFunc, &CreateItemFunc, &DeleteItemFunc, &CreateTypeFunc, &num_out_blocks);
```

```
// write results
DIY_Write_open_all(outfile);
DIY_Write_blocks_all(out_blocks, num_out_blocks, datatype);
DIY_Write_close_all();
```

```
// terminate
DIY_Finalize(); // finalize DIY before MPI
MPI_Finalize();
```

Applications

Parallel Voronoi Tessellation

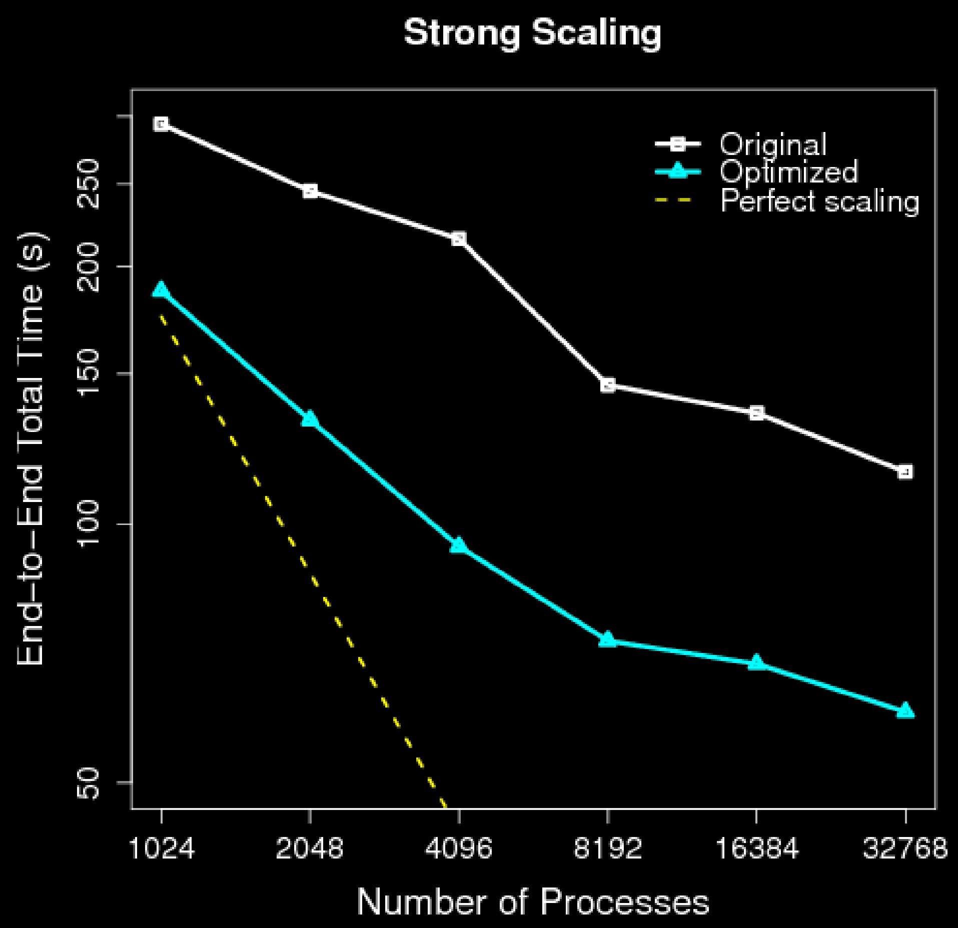


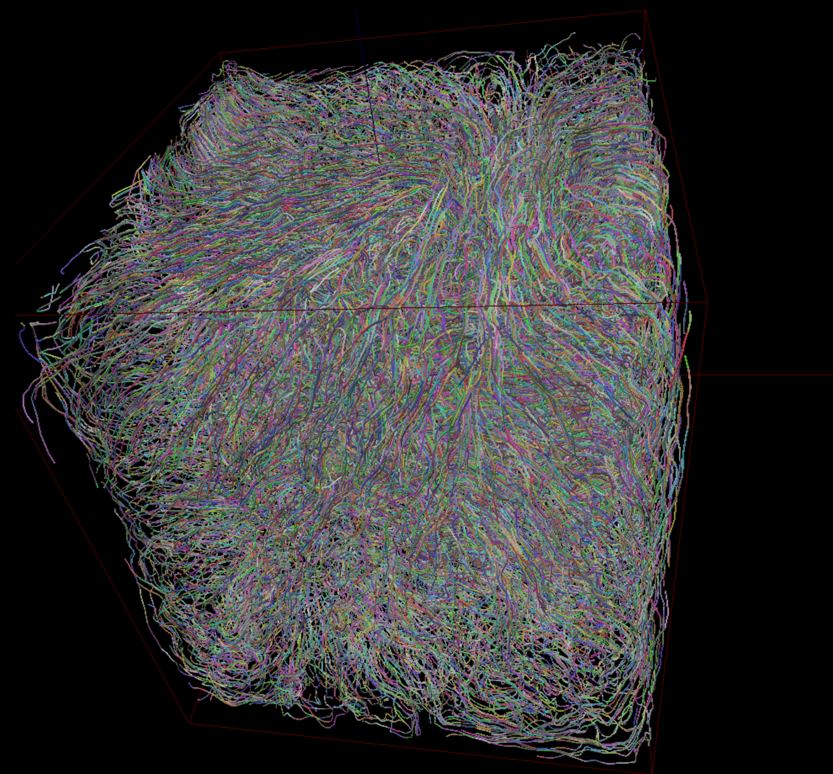
Thresholding cell volume to reveal cosmological voids

Particles	Processes	Total Time (s)	Simulation Time (s)	Tessellation Time (s)
512^3	2048	3852	3684	167
	4192	2008	1918	89
	8096	1784	1722	62
	16384	1406	1344	61

Subset of strong and weak scaling test results shows good scalability and relatively small fraction of total run time for in situ analysis

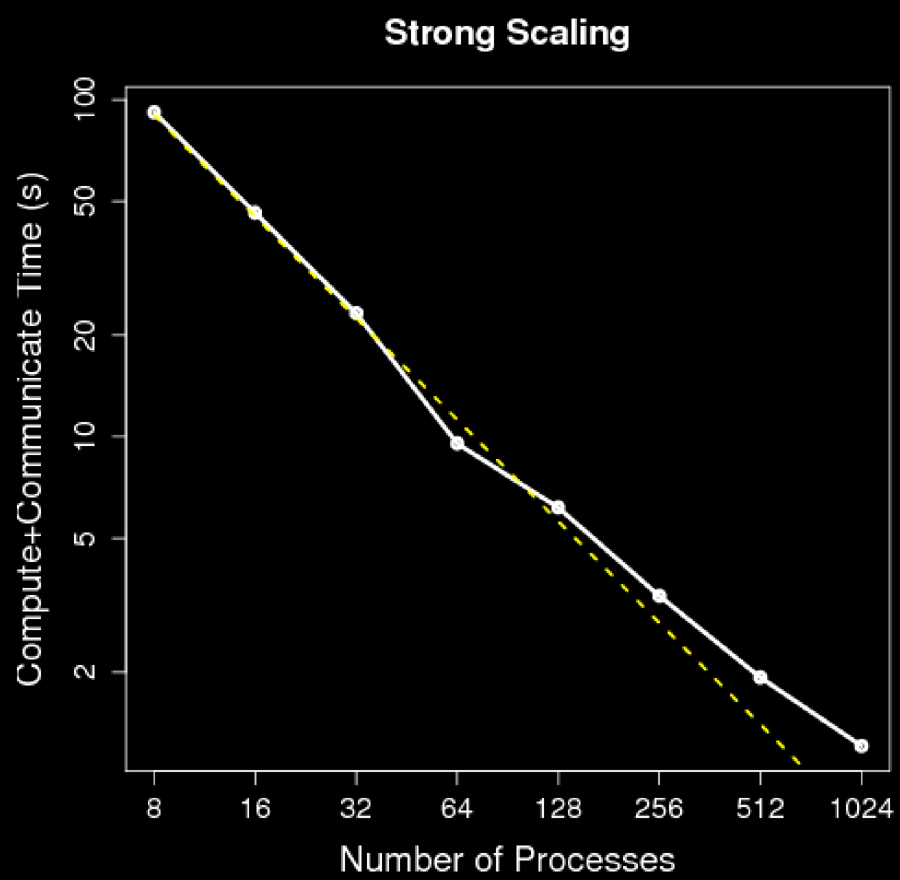
Parallel Particle Tracing

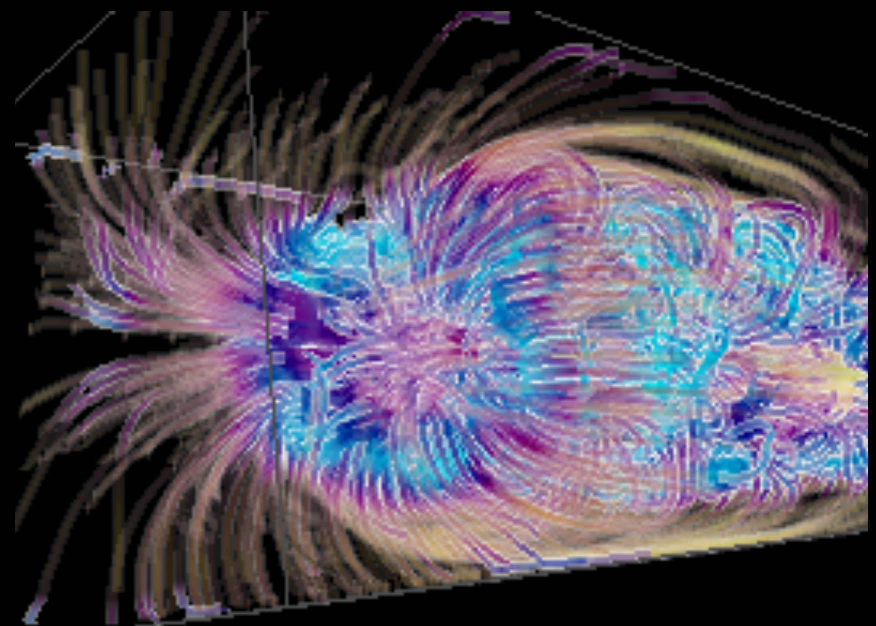




Particle tracing of ¼ million particles in a 2048^3 thermal hydraulics dataset results in strong scaling to 32K processes and an overall improvement of 2X over earlier algorithms

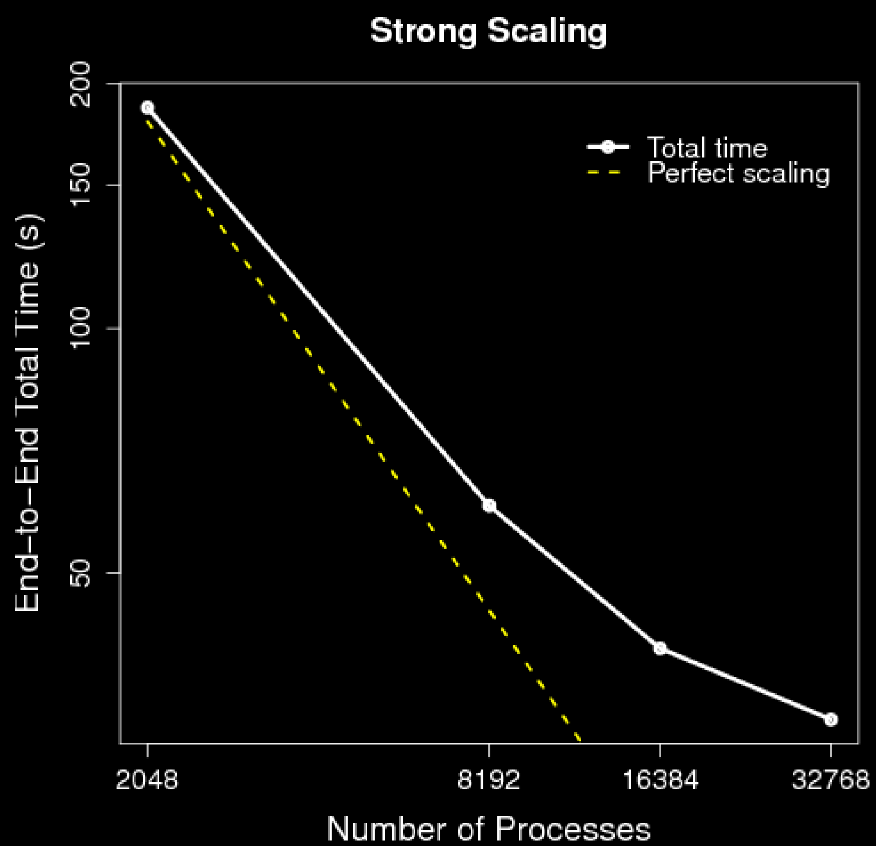
Information Entropy Performance and Scalability

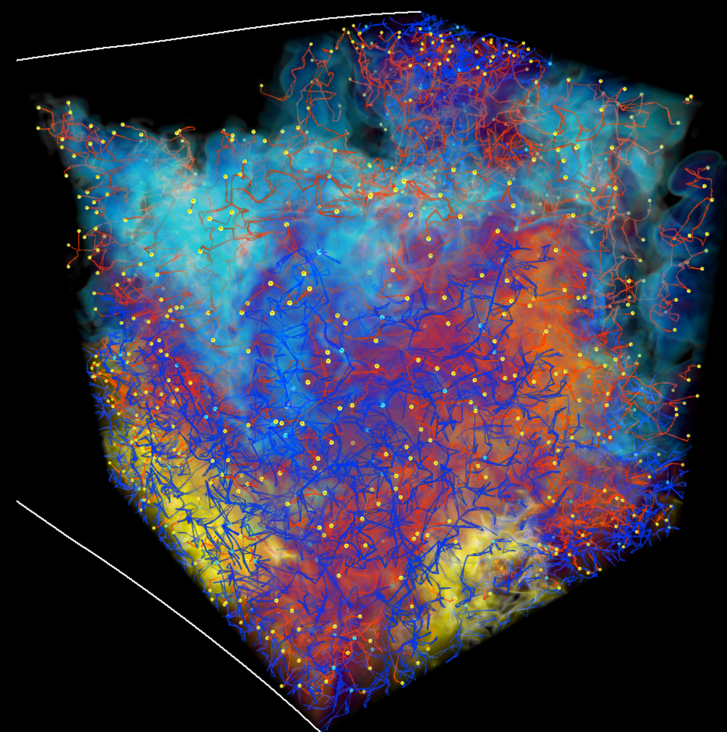




Computation of information entropy in 126x126x512
solar plume dataset shows 59% strong scaling efficiency.

Morse-Smale Complex Performance and Scalability





Computation of Morse-Smale complex in 1152^3 Rayleigh-Taylor instability data set results in 35% end-to-end strong scaling efficiency, including I/O.

Summary

-Consider data and data movement as first-class citizens

-Tools needed both for run-time as well as postprocessing analysis

-Analysis is any sequence of operations on data that hopefully reduces its size and/or improves its understandability

-Much more work to be done!

U.S. DEPARTMENT OF ENERGY

Argonne NATIONAL LABORATORY

"The purpose of computing is insight, not numbers."

–Richard Hamming, 1962

Acknowledgments:

Facilities

Argonne Leadership Computing Facility (ALCF)

Oak Ridge National Center for Computational Sciences (NCCS)

Funding

DOE SDMAV Exascale Initiative

DOE Exascale Codesign Center

http://www.mcs.anl.gov/~tpeterka/software.html

https://svn.mcs.anl.gov/repos/diy/trunk

Tom Peterka

tpeterka@mcs.anl.gov

Mathematics and Computer Science Division