

Parallelizing Data Analysis

Tom Peterka

Big Data Get Together
10/26/12

Obvious, but worth reminding

- Big science begets big data, and
 - Big data analysis needs big science resources
- Data analysis is data intensive.
 - Data intensity = data movement.
- Big data movement is ~~hard~~ fun.
 - Embarrassingly parallel and MapReduce are limited.
- Most analysis algorithms are not up to the challenge
 - Either serial, or
 - Communication, I/O are scalability killers

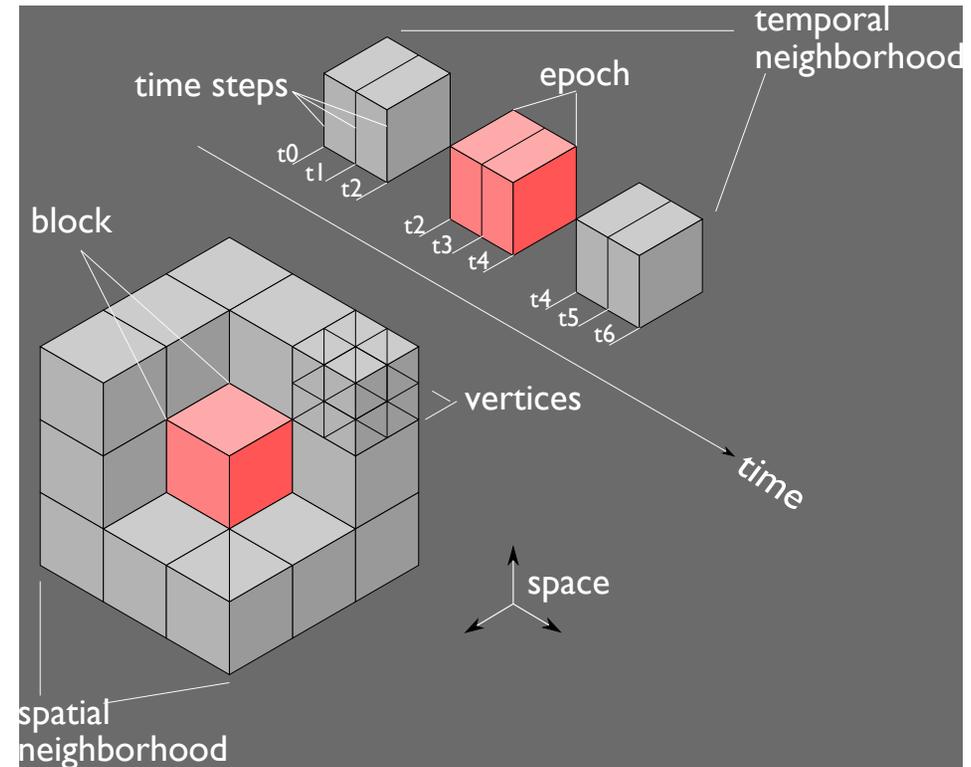
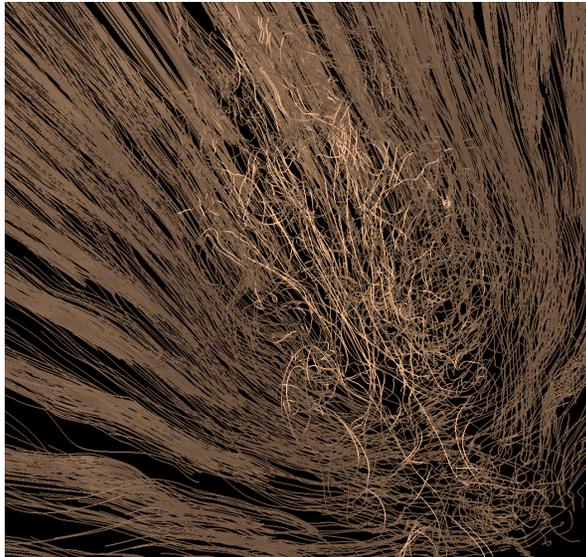
There is hope.

Parallel Time-Varying Flow Analysis

Collaboration with the Ohio State University

Approach

- In core / out of core processing of time steps
- Simple load balancing (multiblock assignment, early particle termination)
- Adjustable synchronization communication

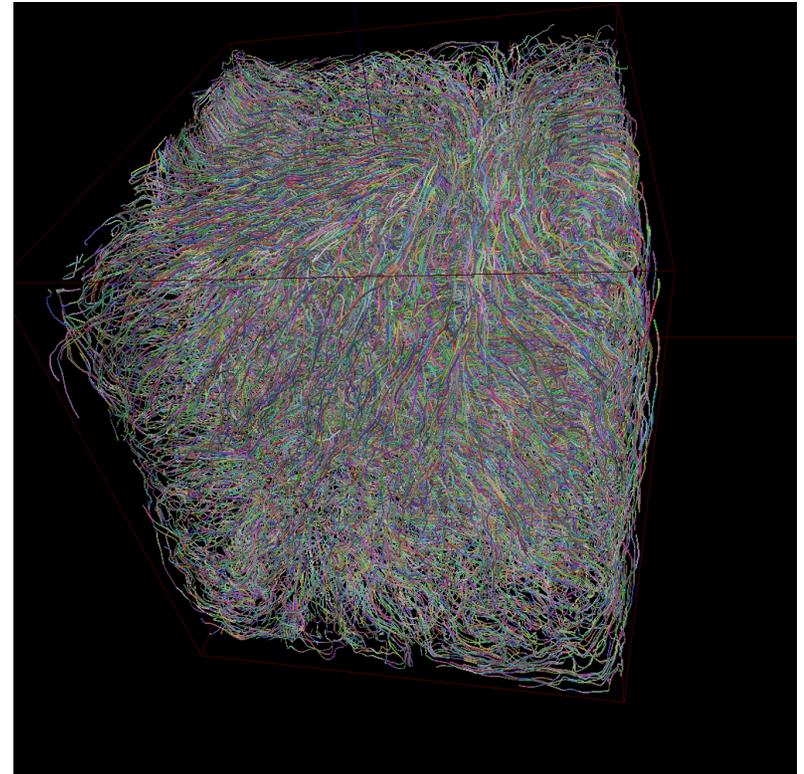
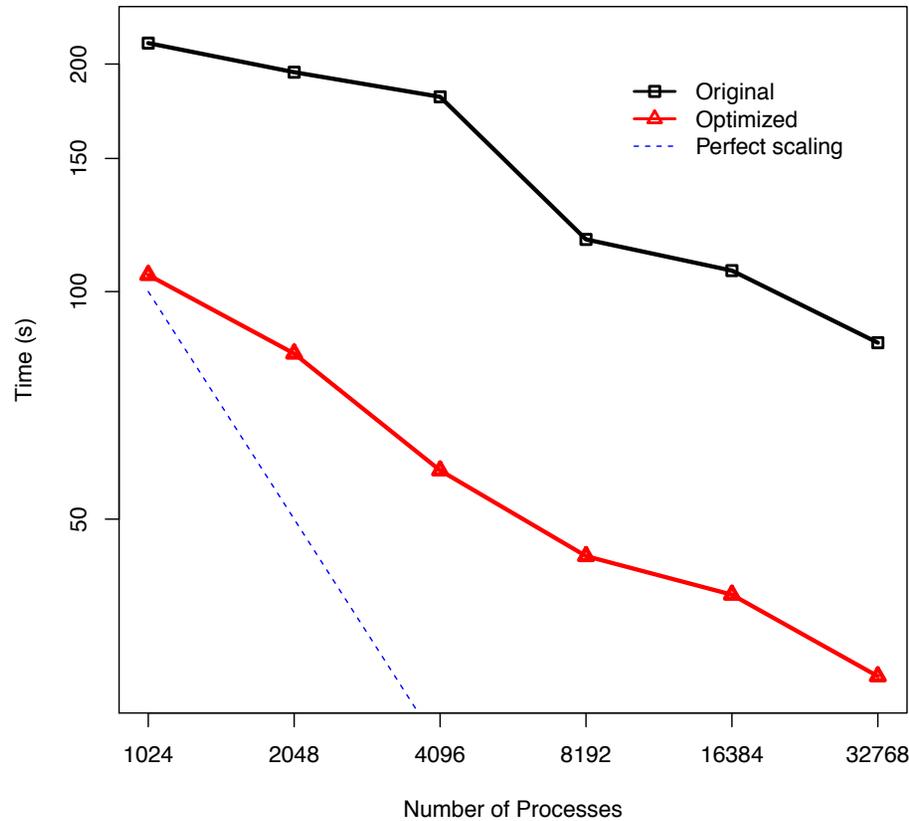


Pathline tracing of 32 time-steps of combustion in the presence of a cross-flow in S3D (in collaboration with Jackie Chen, Hongfeng Yu, Janine Bennett, Ray Grout)

Parallelization within epochs and serialization across epochs adds greater flexibility.

Parallel Particle Tracing

Strong Scaling Performance

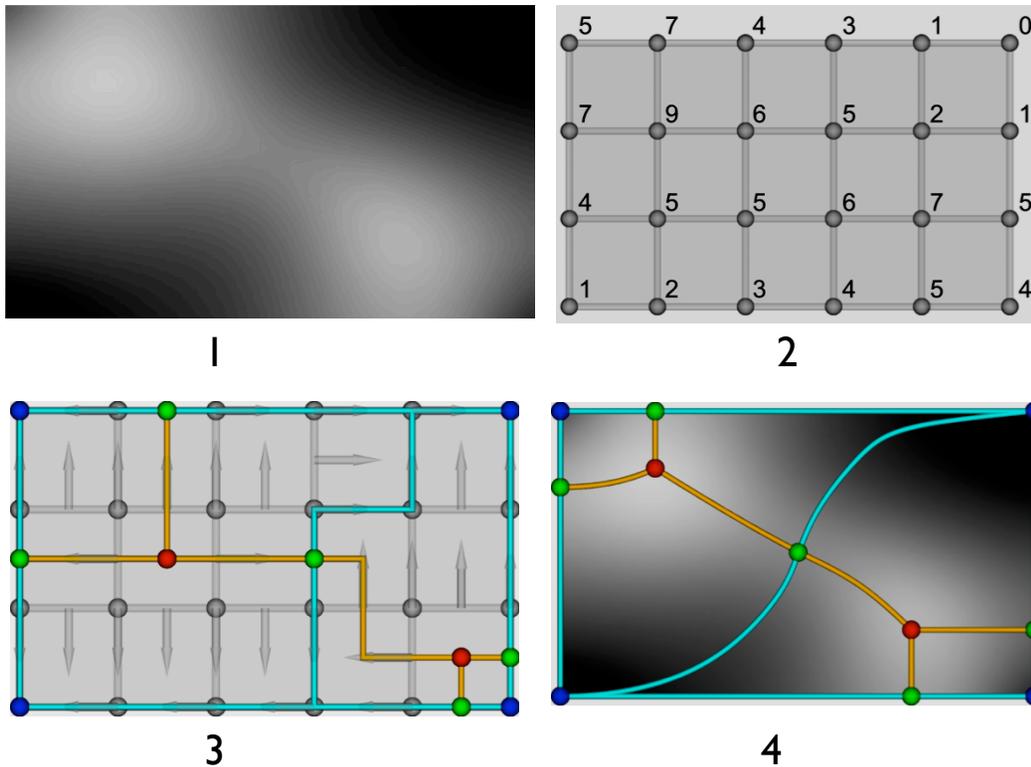


Particle tracing of $\frac{1}{4}$ million particles in Nek5000 2048³ thermal hydraulics dataset results in strong scaling to 32K processes and an overall improvement of 2X over earlier algorithms (In collaboration with Paul Fischer and Aleks Obabko)

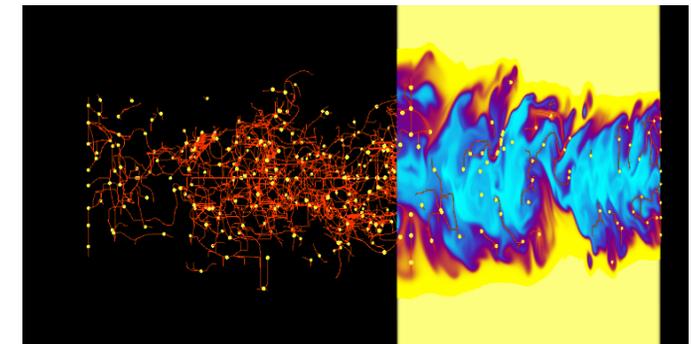
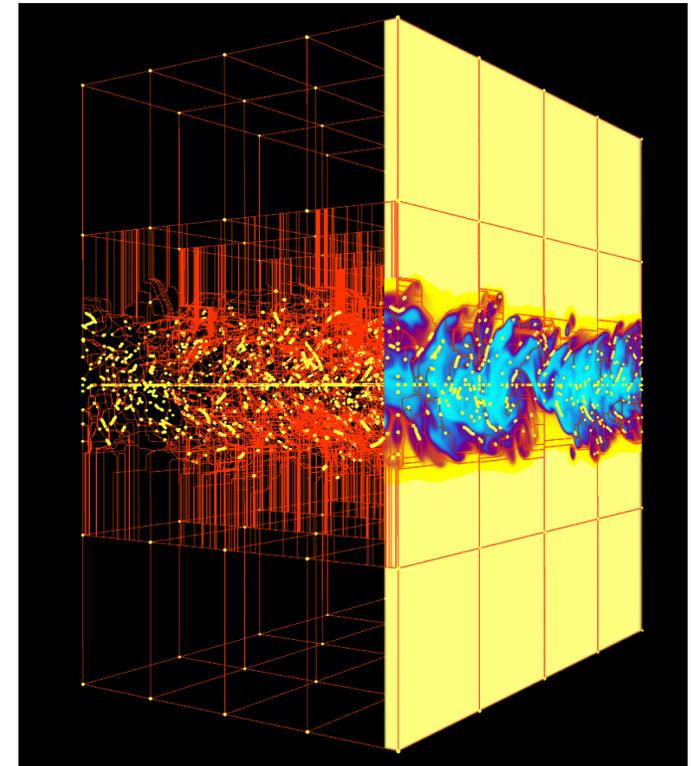
Parallel Topological Analysis

Collaboration with SCI Institute, University of Utah

- Transform discrete scalar field into Morse-Smale complex
- Nodes are minima, maxima, saddle points of scalar values
- Arcs represent constant-sign gradient flow
- Used to quickly see topological structure



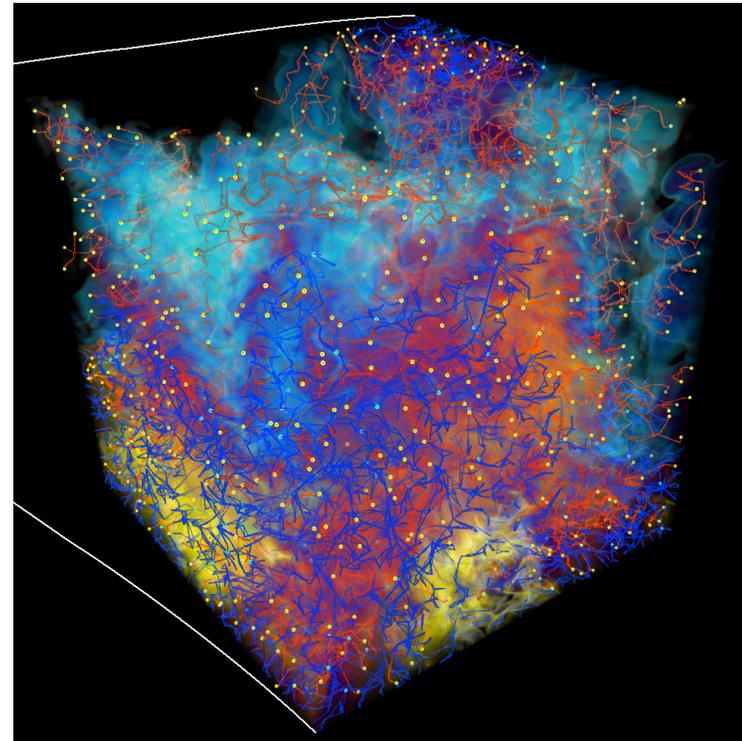
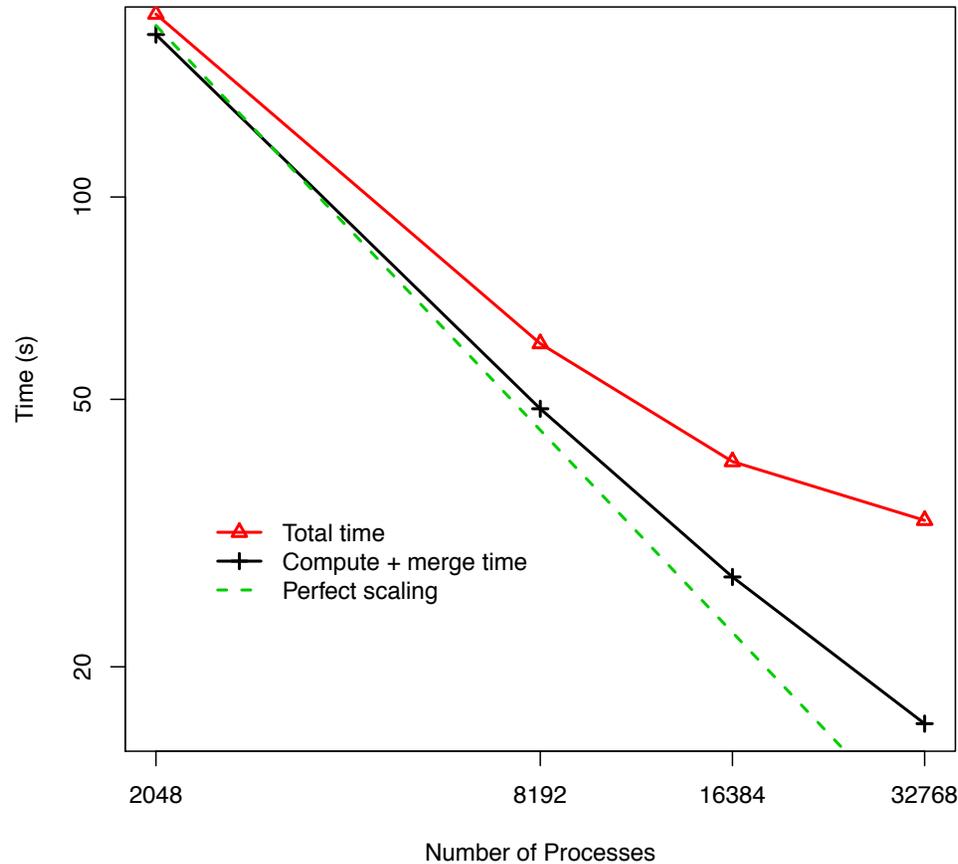
Example of computing discrete gradient and Morse-Smale Complex



Two levels of simplification of the Morse-Smale complex for jet mixture fraction.

Parallel Morse-Smale Complex

Total & Compute+Merge Time For Rayleigh-Taylor Mixing



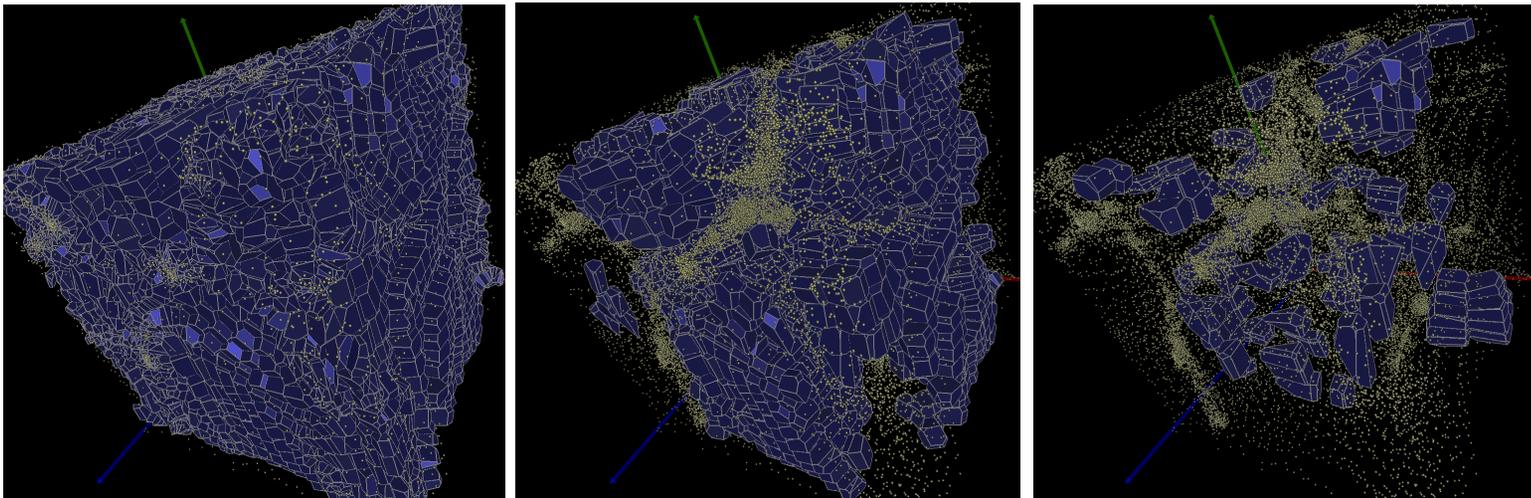
Computation of Morse-Smale complex in 1152^3 Rayleigh-Taylor instability data set results in 35% end-to-end strong scaling efficiency, including I/O.

Parallel Voronoi Tessellation

Collaboration with Salman Habib, Katrin Heitmann, and the HACC cosmology group (HEP, MCS, ALCF divisions of

ANL)

Deriving a dense mesh representation from a sparse N-body particle simulation allows scientists to conduct novel analyses not possible with the original data, because particle density is now a continuous function that can be computed everywhere in the field. The Voronoi tessellation is ideal for cosmological data because it self-adapts to widely varying particle distributions.

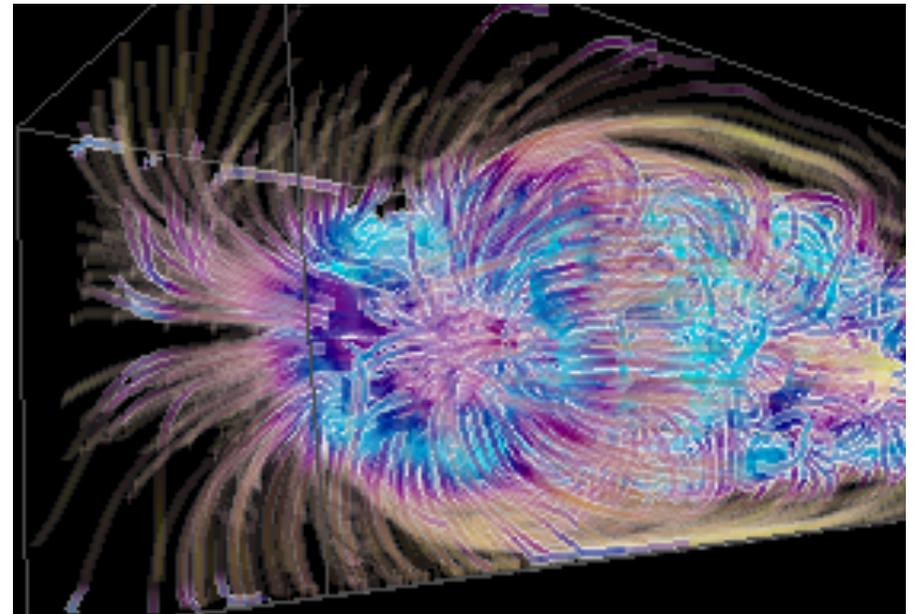
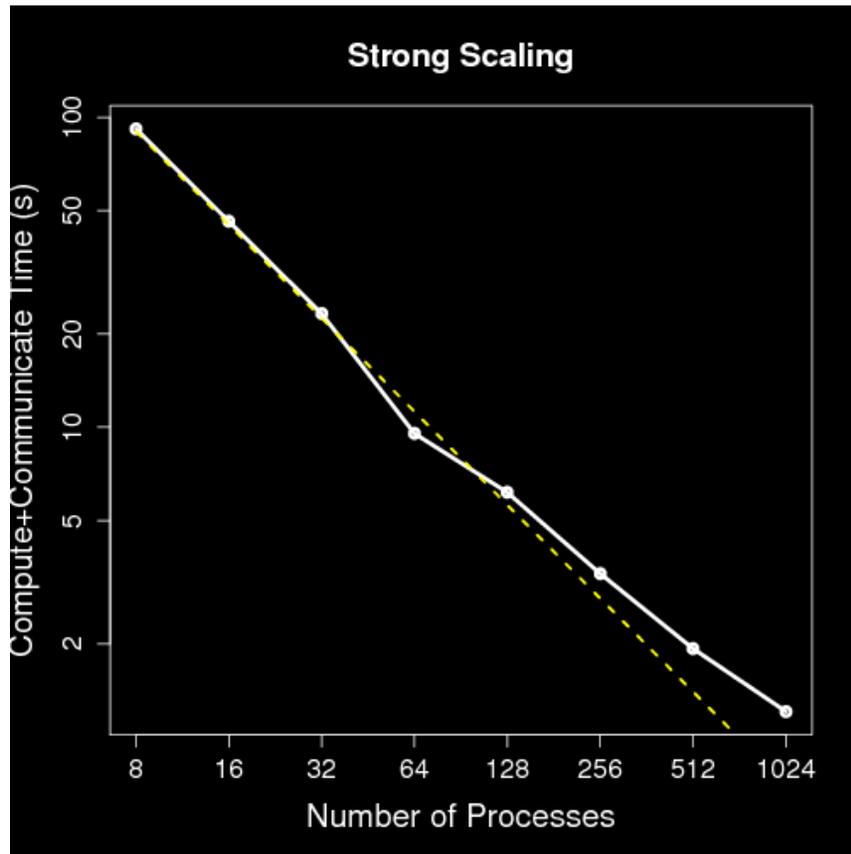


Applying threshold filtering on cell volume and forming connected components of remaining Voronoi cells locates cosmological voids.

Approach

- Compute parallel Voronoi tessellation in situ
- Compute connected components of threshold-filtered Voronoi cells to identify cosmological voids
- Time-varying tessellations help scientists understand feature evolution

Parallel Information Entropy



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

Black Box Parallelism

Fortunately, a variety of different parallel analyses have common data needs.

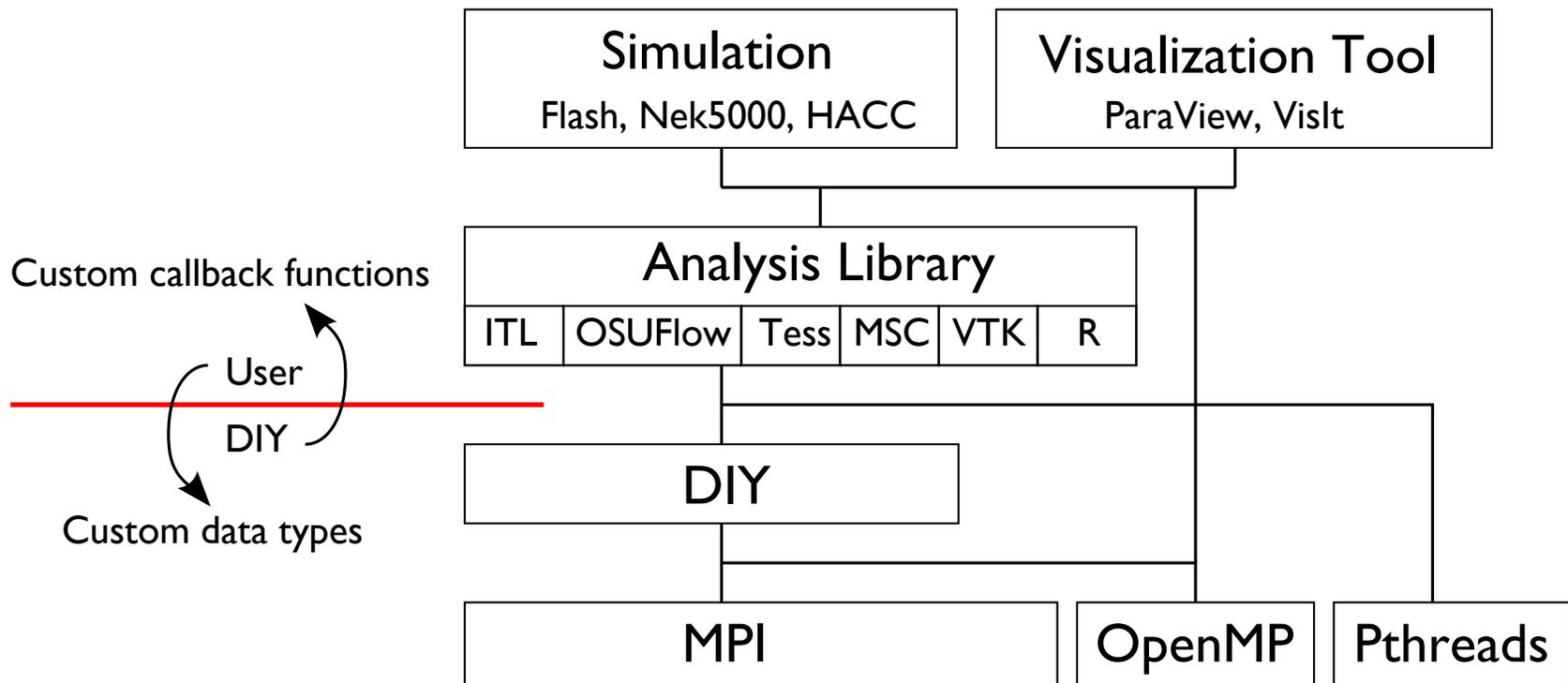
Analysis	Application	Application Data Model	Analysis Algorithm	Communication	Additional
Particle Tracing	CFD	Unstructured Mesh	Numerical Integration	Nearest neighbor	File I/O, Domain decomposition, process assignment, utilities
Information Entropy	Astrophysics	AMR	Convolution	Global reduction, nearest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

You do this yourself with the help of serial libraries such as OSUFlow, Qhull, VTK (don't have to start from scratch)

DIY does this for you

Draw a line between user space and data space.

- Let users do what they do best
ie, custom serial analysis
- Let DIY do what it does best
ie, data parallelization



DIY is that common data library.

Main Ideas and Objectives

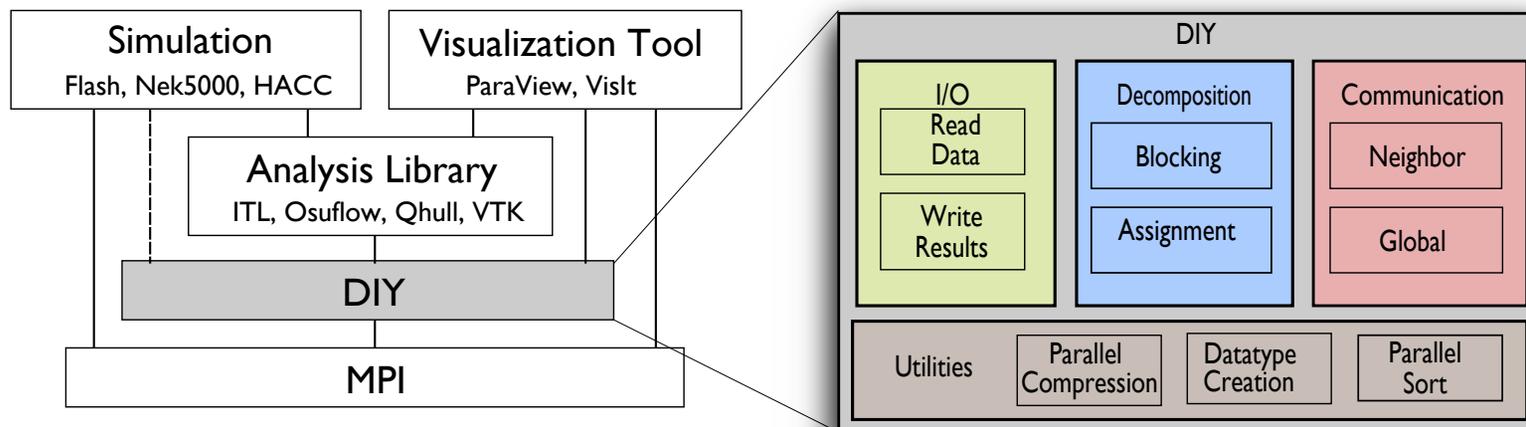
- Large-scale data-parallel analysis (visual and numerical)
- For scientists, visualization researchers, tool builders
- In situ, coprocessing, postprocessing
- MPI + threads hybrid parallelism
- Scalable data movement algorithms
- Runs on Unix-like platforms, from laptop to supercomputer (including all IBM and Cray HPC leadership machines)

Features

- Parallel I/O to/from storage
- Domain decomposition
- Network communication
- Written in C++ w/ C bindings, called from Fortran, C, C++
- Autoconf build system
- Lightweight: libdiy.a 800KB
- Maintainable: ~15K lines of code

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



DIY usage and library organization

Usage

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();
```

Summary

Benefits

- **Productivity**
 - Express complex algorithms
 - Multiple blocks per process
 - Neighbor inclusion
 - Complete / partial reductions
 - Neighborhood communication pattern
 - Simplify existing tasks
 - Custom data type creation
 - Compression
- **Performance**
 - Published scalability
 - Configurable algorithms

To Do: Research Directions

- Advanced decomposition
 - Block groups
- More / better low-level communication algorithms
 - Less synchronous, more overlap with computation
 - Point to point between blocks
- High-level communication operations
 - Ghost cell exchange
 - Kernel convolution (stencil)
- Load balancing and Resiliency
 - Block overloading
 - Dynamic reassignment
- Keeping up with new developments
 - Programming models
 - MPI + X, MPI-3
 - Architectures
 - Mira, Titan

Resources

- Software
 - <https://svn.mcs.anl.gov/repos/diy/trunk>
- Publications
 - Peterka et al., Scalable Parallel Building Blocks for Custom Data Analysis, LDAV'11.
 - Peterka et al., T., Versatile Communication Algorithms for Data Analysis, IMUDI'12.
 - Gyulassy et al., The Parallel Computation of Morse-Smale Complexes, IPDPS'12.
 - Peterka et al., A Study of Parallel Particle Tracing for Steady-State and Time-Varying Flow Fields, IPDPS'11.
- More info
 - tpeterka@mcs.anl.gov
 - www.mcs.anl.gov/~tpeterka