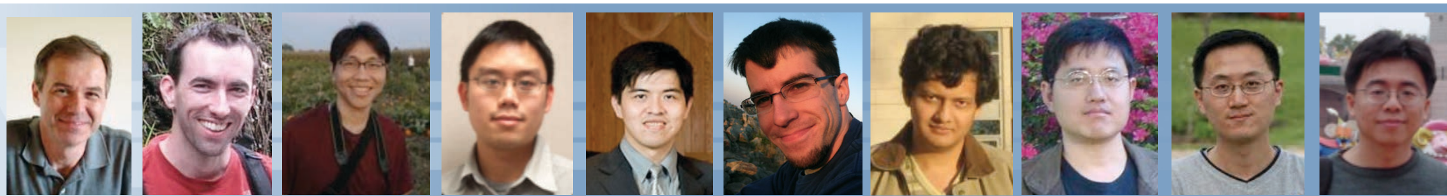
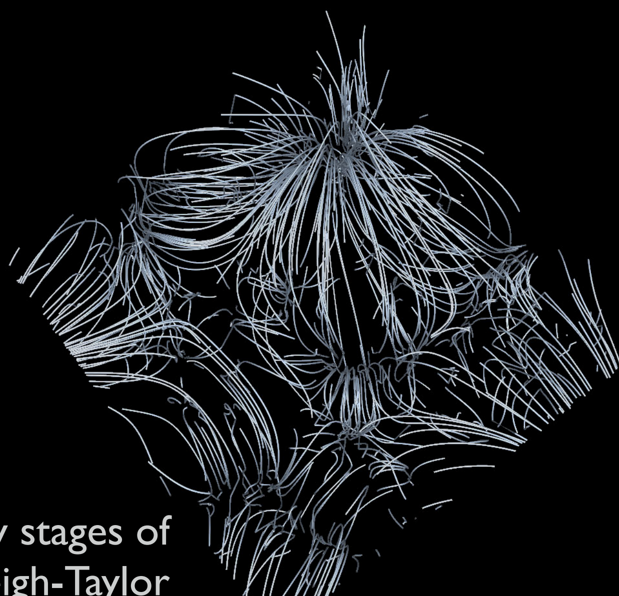


Foundations of Data-Parallel Particle Advection



Early stages of Rayleigh-Taylor Instability flow

Tom Peterka, Rob Ross *Argonne National Laboratory*

Boonth Nouanesengsey, Teng-Yok Lee, Abon Chaudhuri, Jimmy Chen, Kewei Lu, Han-Wei Shen *The Ohio State University*

Wes Kendall, Jian Huang *University of Tennessee, Knoxville*

Zhanping Liu, *Kentucky State University*

IEEE Vis 2013 Tutorial

10/13/13 Atlanta GA

Tom Peterka

tpeterka@mcs.anl.gov

Mathematics and Computer Science Division

Scientific Data Analysis and Specifically Particle Tracing

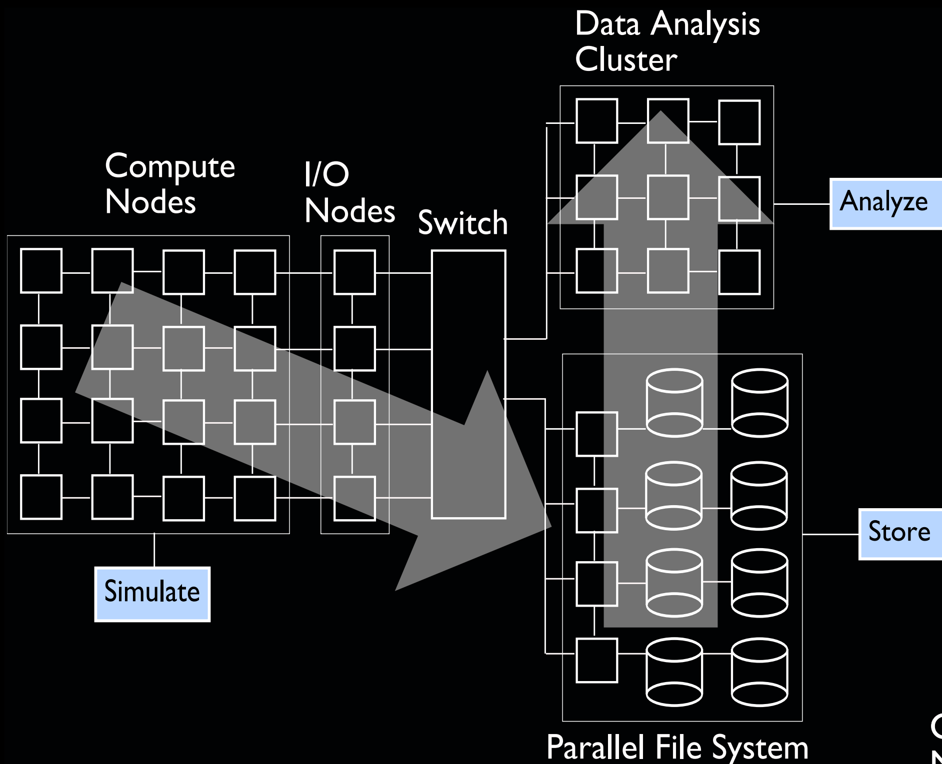
General

- Big science => HPC analysis
- Data analysis => data movement
- Parallel => distributed memory data parallel
- Most analysis algorithms are not up to the challenge
  - Either serial or shared memory parallel
  - Communication and I/O are scalability killers

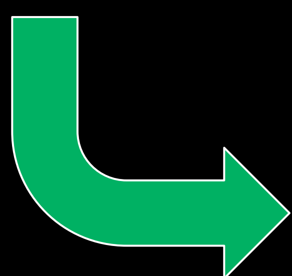
Particle Tracing

- Data sizes are large, and large numbers of particles are needed (hundreds of thousands) for accurate further analysis of field line features.
- High communication volume and data-dependent load balance make particle tracing challenging to parallelize and scale efficiently.

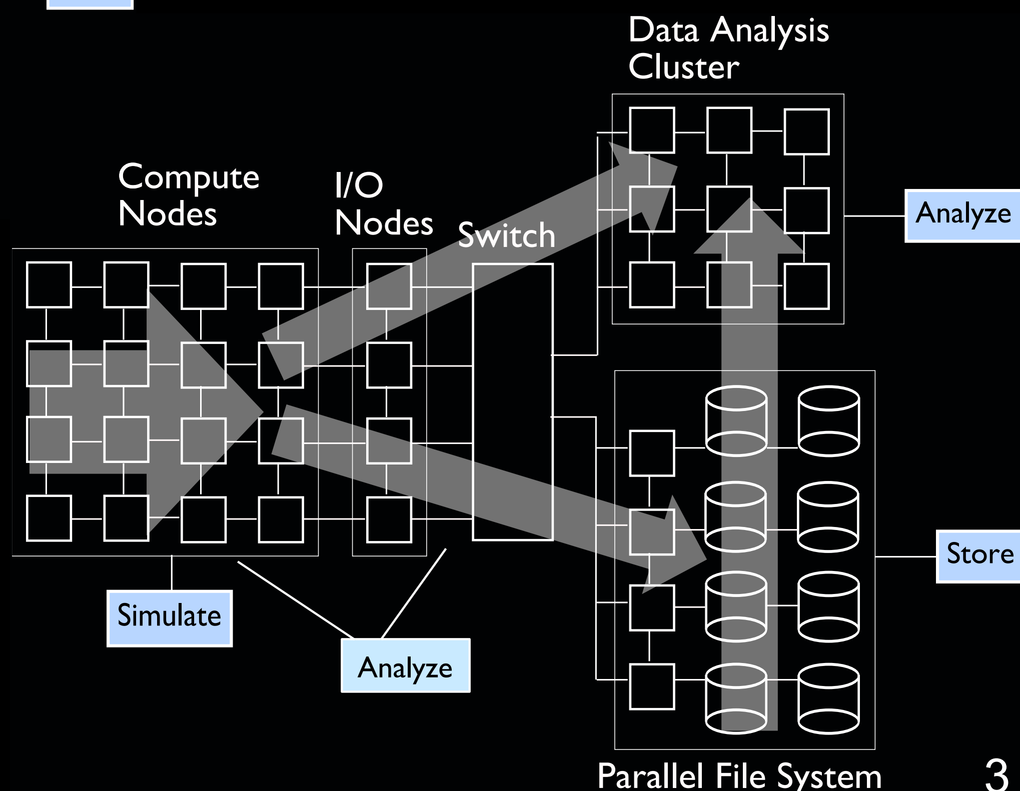
Moving from Postprocessing to Run-Time Scientific Data Analysis in HPC



Postprocessing particle tracing and visualization



Run-time particle tracing and postprocessing visualization



The Need for Parallel Particle Tracing

When data sizes are too large to move or process serially, parallel particle tracing needs to be executed on HPC machines. Results are available sooner, access to all data at full resolution is possible.

Rayleigh-Taylor Instability

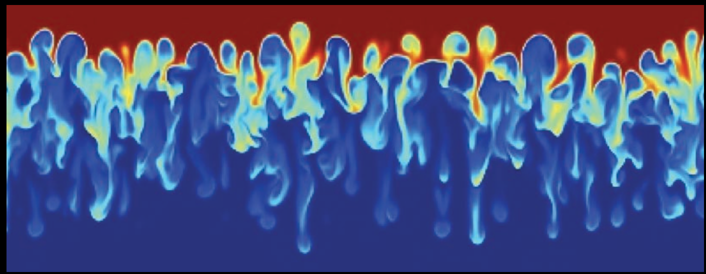


Image courtesy Mark Petersen, Daniel Livescu, LANL. Code: CFDNS

Test Data Sizes

Dataset	Grid size	Data size (GB)
MAX	2048^3	98
RTI	2304 x 4096 x 4096	432
Flame	1408 x 1080 x 1100 x 32 time steps	608

MAX Experiment

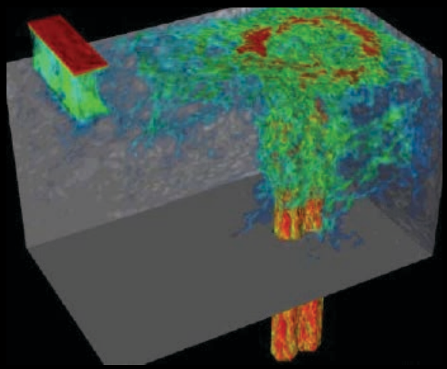


Image courtesy Paul Fischer, Aleks Obabko, ANL. Code: Nek5000

Flame Stabilization

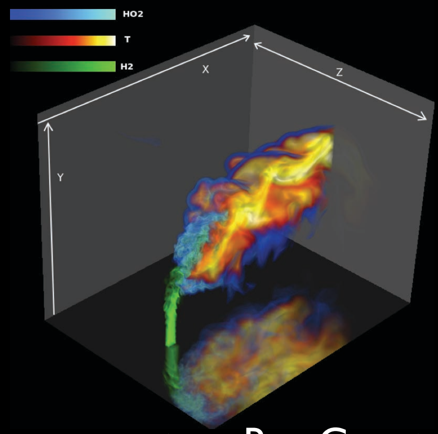
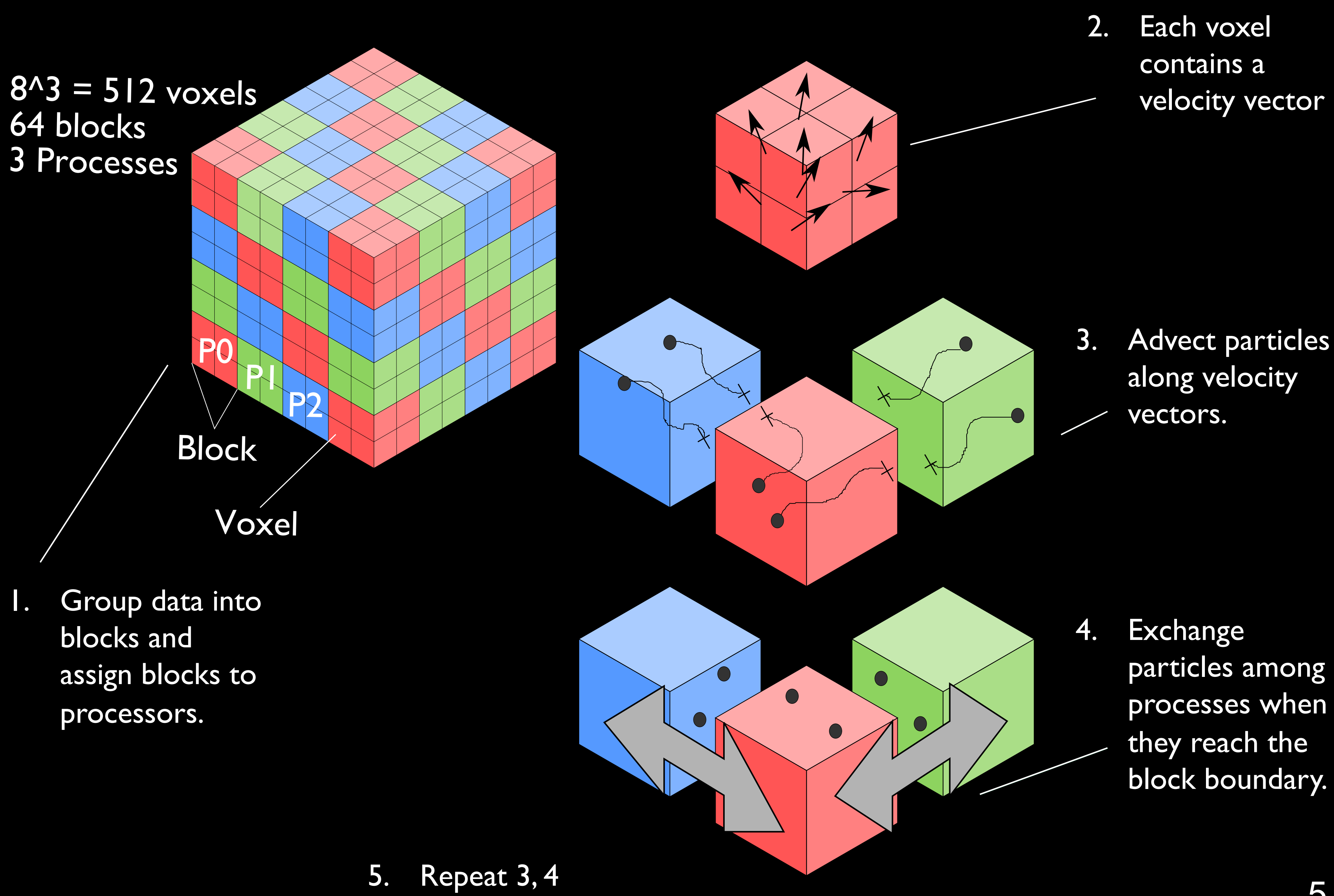


Image courtesy Ray Grout, NREL, Hongfeng Yu, Jackie Chen, SNL Code: S3D

Simple Data Parallelization



OSUFlow and DIY

OSUFlow is a library of serial / parallel particle tracing functions that is parallelized using a library called DIY that helps the user write data-parallel analysis algorithms by decomposing a problem into blocks and communicating items between blocks.

OSUFlow Features

Static / time-varying flows

Regular / rectilinear / curvilinear / unstructured grids

Fixed / adaptive step sizes

Various integration methods

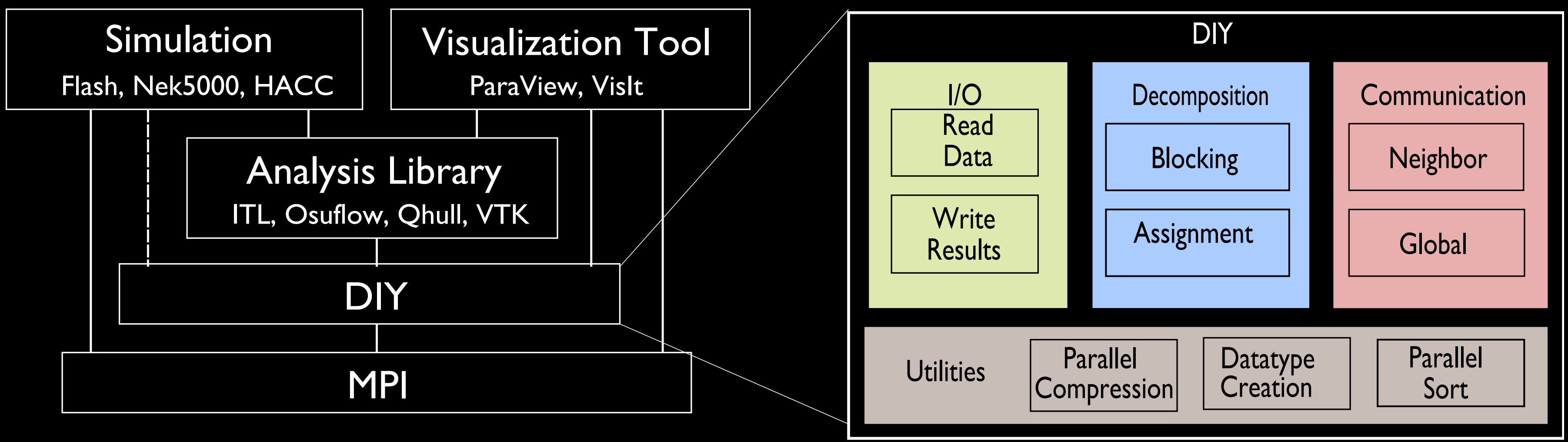
DIY Features

Parallel I/O to/from storage

Domain decomposition

Network communication

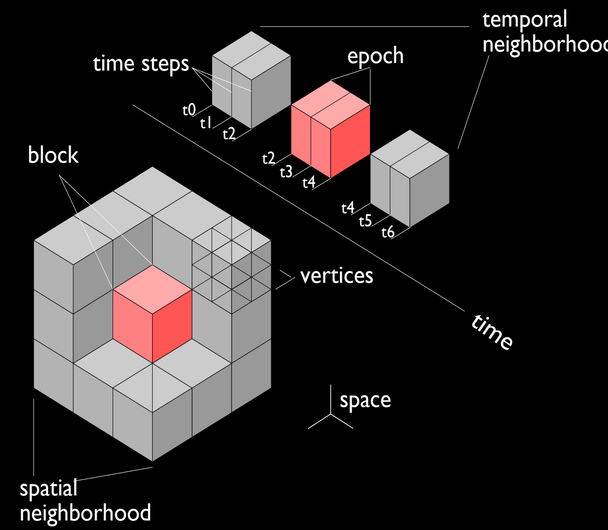
Utilities

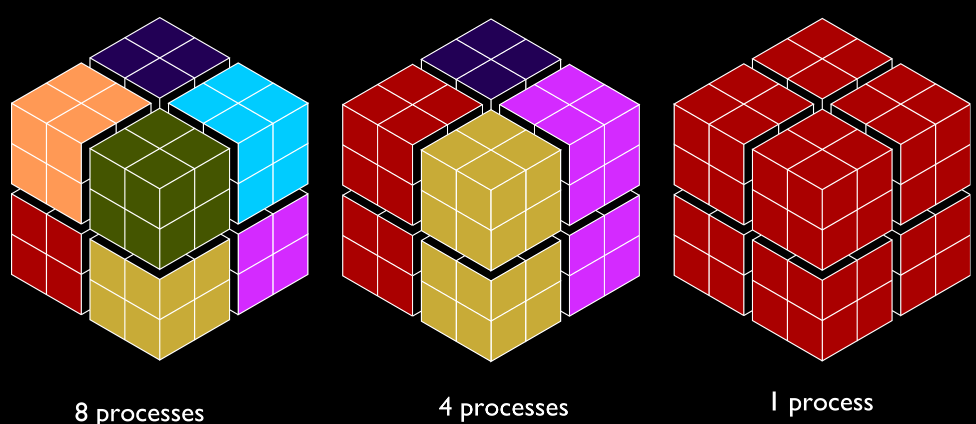


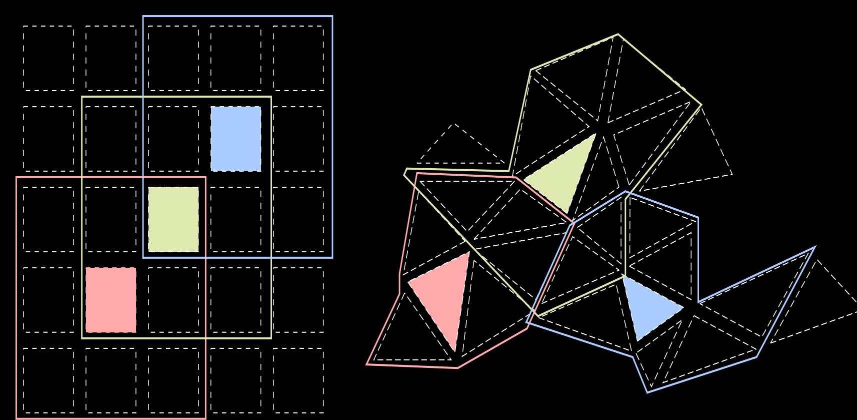
DIY usage and library organization

Nine Things That DIY Does

1. Separate analysis ops from data ops
2. Group data items into blocks
3. Assign blocks to processes
4. Group blocks into neighborhoods
5. Support multiple multiple instances of 2, 3, and 4
6. Handle time
7. Communicate between blocks in various ways
8. Read data and write results
9. Integrate with other libraries and tools



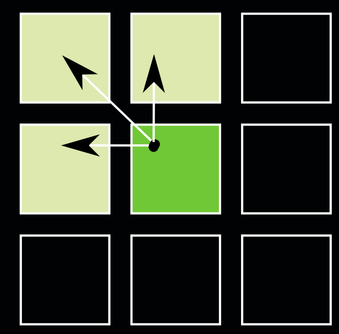




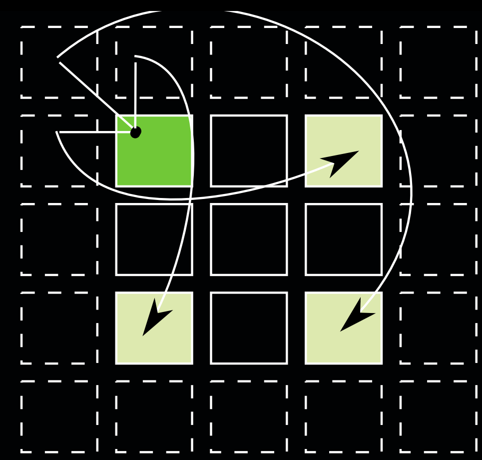
Two examples of 3 out of a total of 25 neighborhoods

Howdy Neighbor

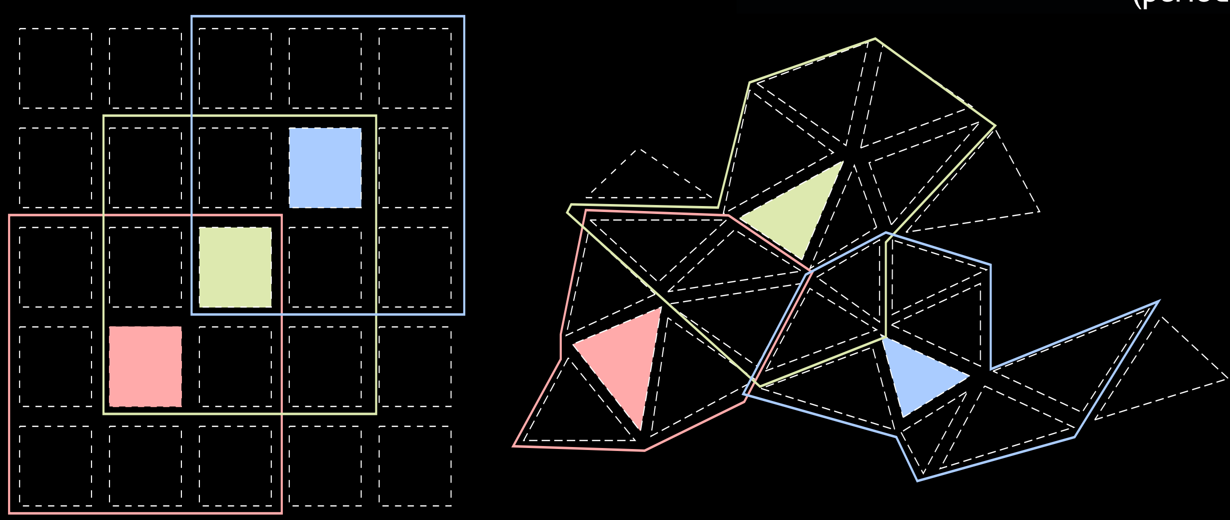
- Neighborhoods provide limited-range communication among arbitrary groupings of blocks with distributed, scalable data structures
- DIY provides different options within a neighborhood including sending an item to all neighbors near enough to receive it and periodic boundary condtions. Items are enqueued are subsequently exchanged (2 steps). Items are user-defined.



Send to all neighbors near enough to a target point



Support for wraparound neighbors (periodic boundary conditions)

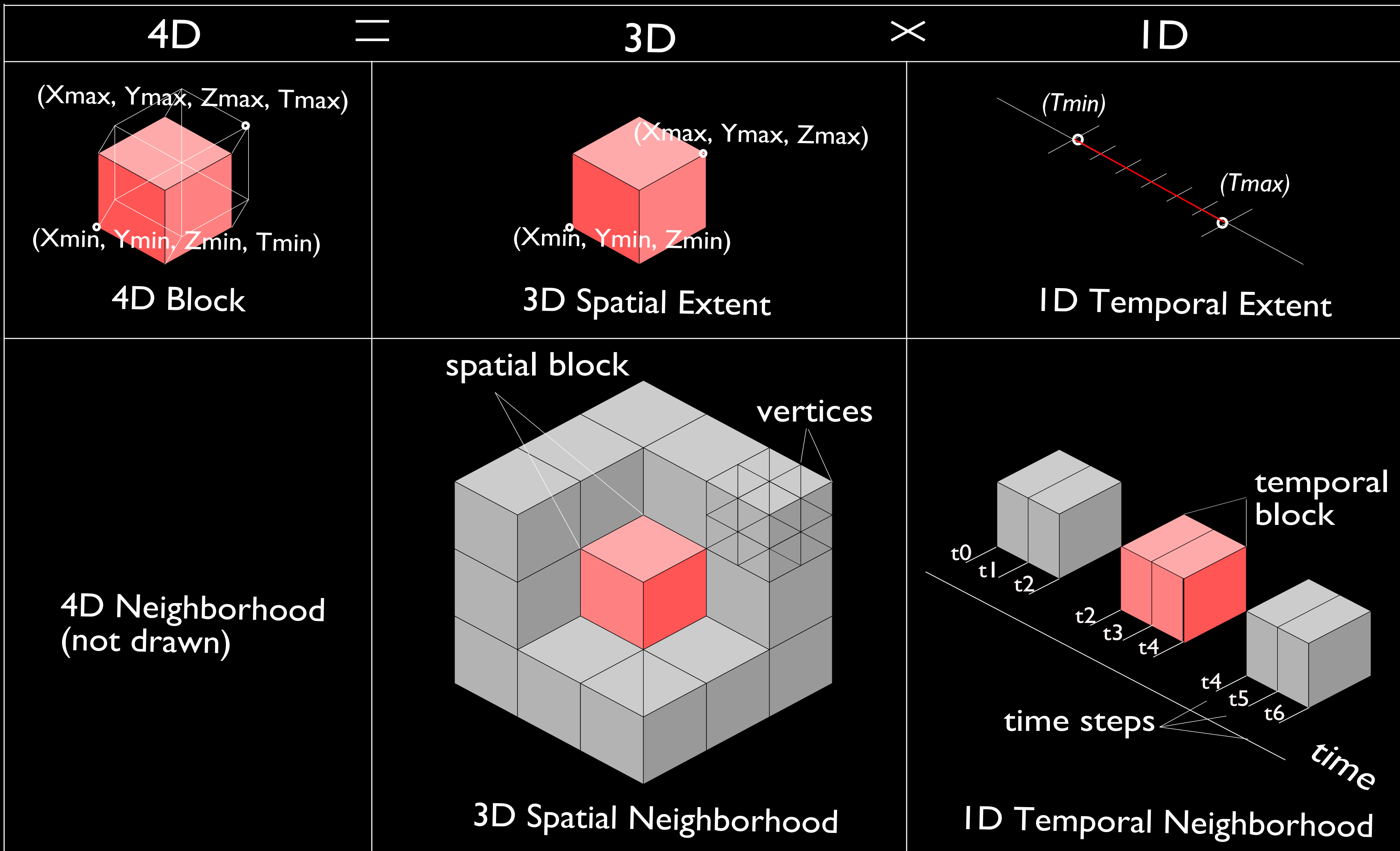


Two examples of 3 out of a total of 25 neighborhoods

It’s About Time

-Time often goes forward only

-Usually do not need all time steps at once

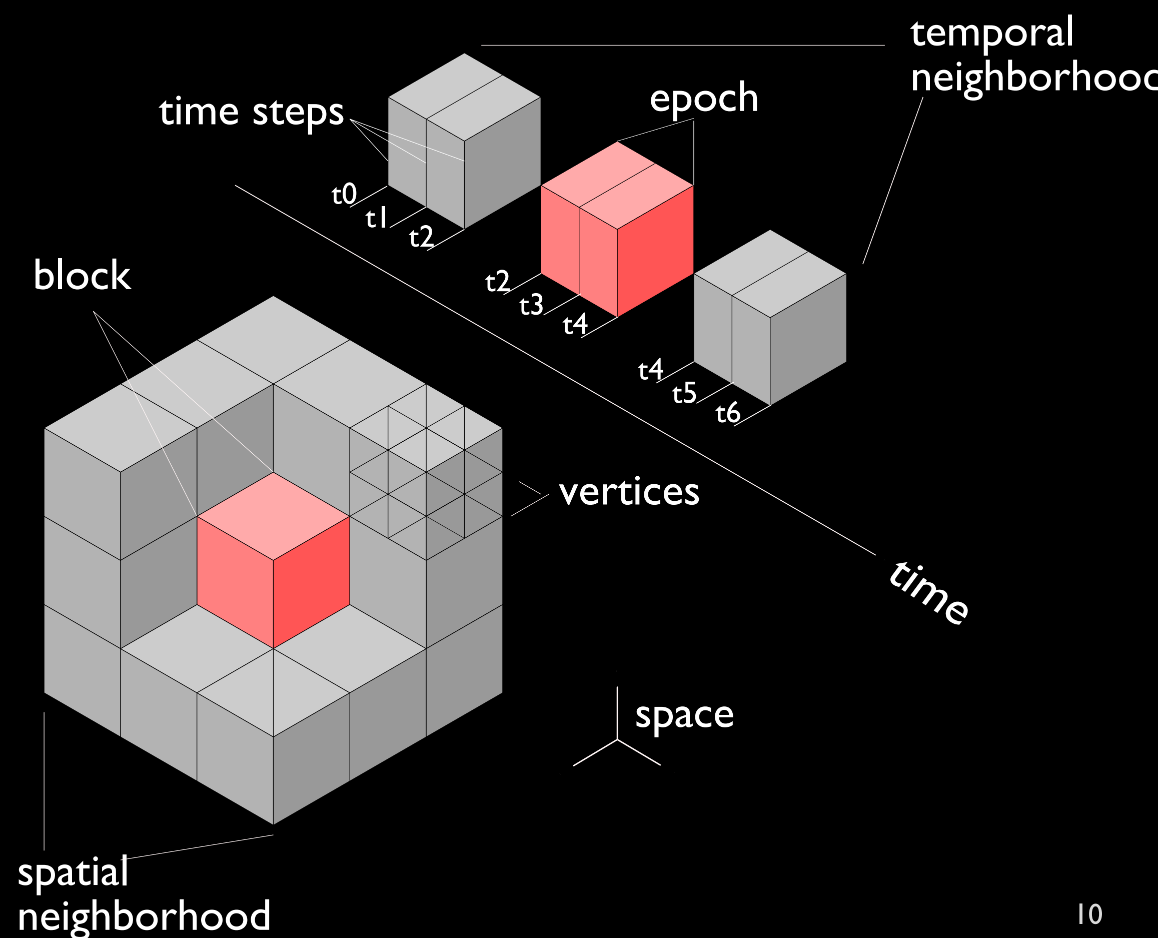


Hybrid 3D/4D time-space decomposition. Time-space is represented by 4D blocks that can also be decomposed such that time blocking is handled separately.

Configurable 3D / 4D Hybrid Algorithm

Internally, all blocks are 4D, but we allow separate grouping in space (blocks) and time (epochs) to control how much data are kept in-core in each epoch. This enables time-varying data to be traced natively in 4D, without requiring the entire 4D dataset to be resident in memory.

Data structure



Algorithm

```
decompose domain into blocks
  and assign blocks to processes
for (epochs) {
  read my process' data blocks
  for (rounds) {
    for (my blocks) {
      advect particles
    }
    exchange particles
  }
}
```

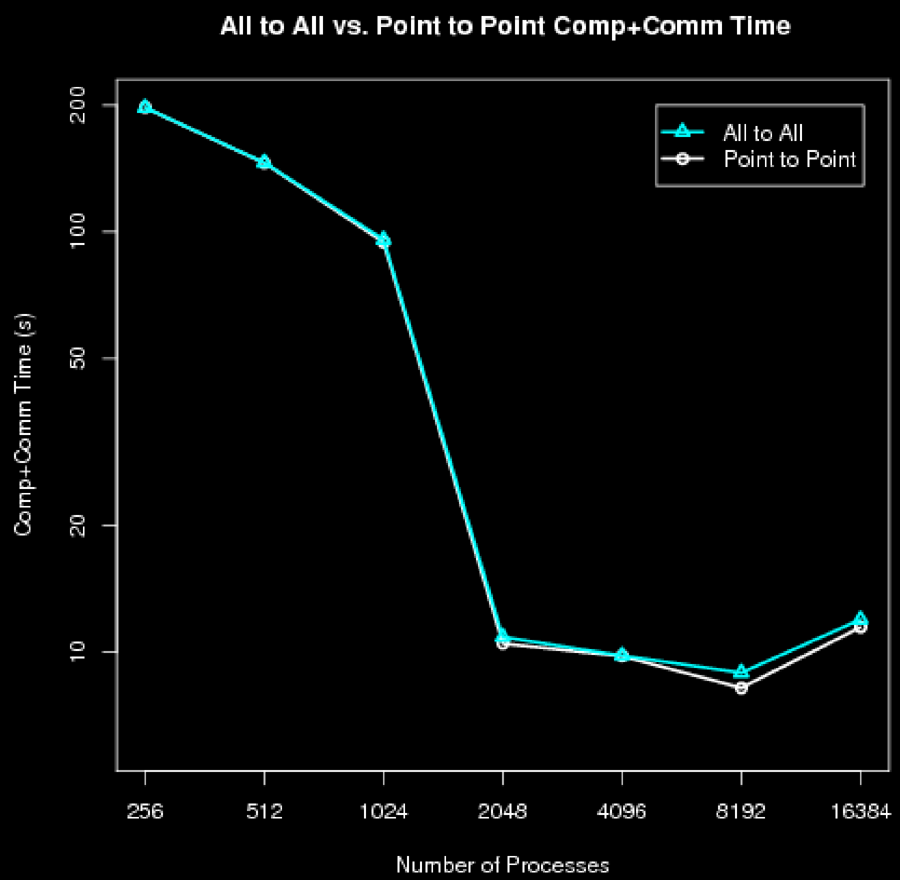
Adjustable Synchronization Communication Algorithm

Wait factor: the fraction of items for which to wait is adjustable. Typically we use 0.1 (wait for 10% of pending items to arrive in each round).

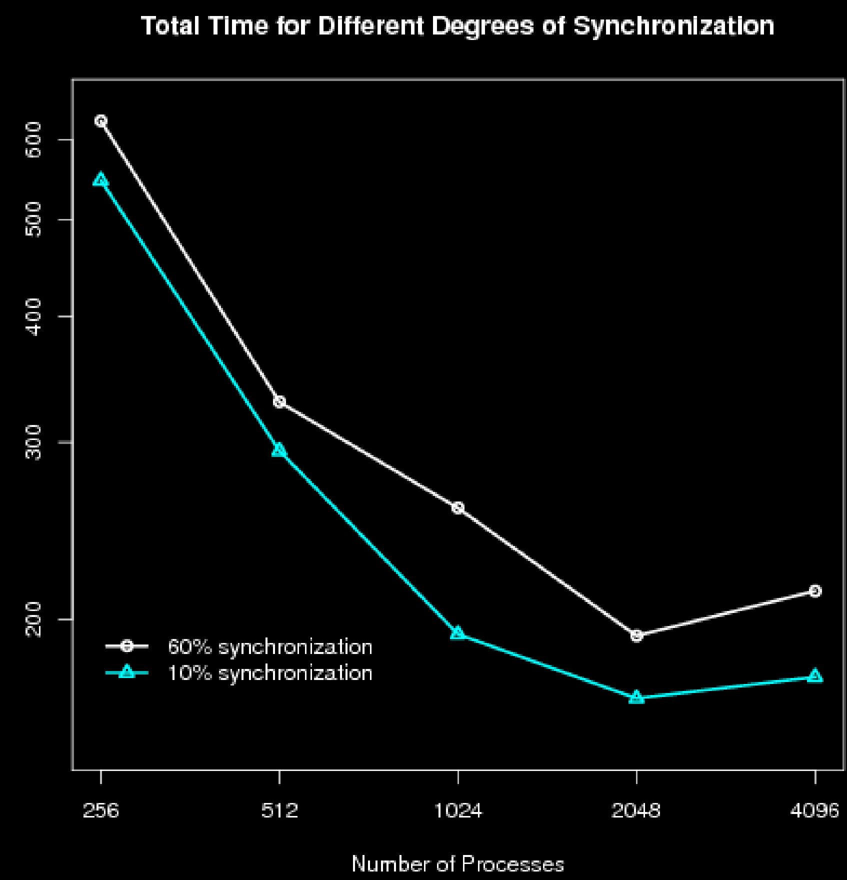
```
for (blocks in my neighborhood) {

  pack and send messages of block IDs and
            particle counts
  pack and send messages of particles

}
wait for enough IDs and counts to arrive
for (IDs and counts that arrived) {

  receive particles

}
```

Wait Factor Communication Performance

Nonblocking point-to-point and waiting for all messages to arrive (wait factor = 1.0) offers little improvement over all-to-all communication, but dialing down the wait factor helps significantly.

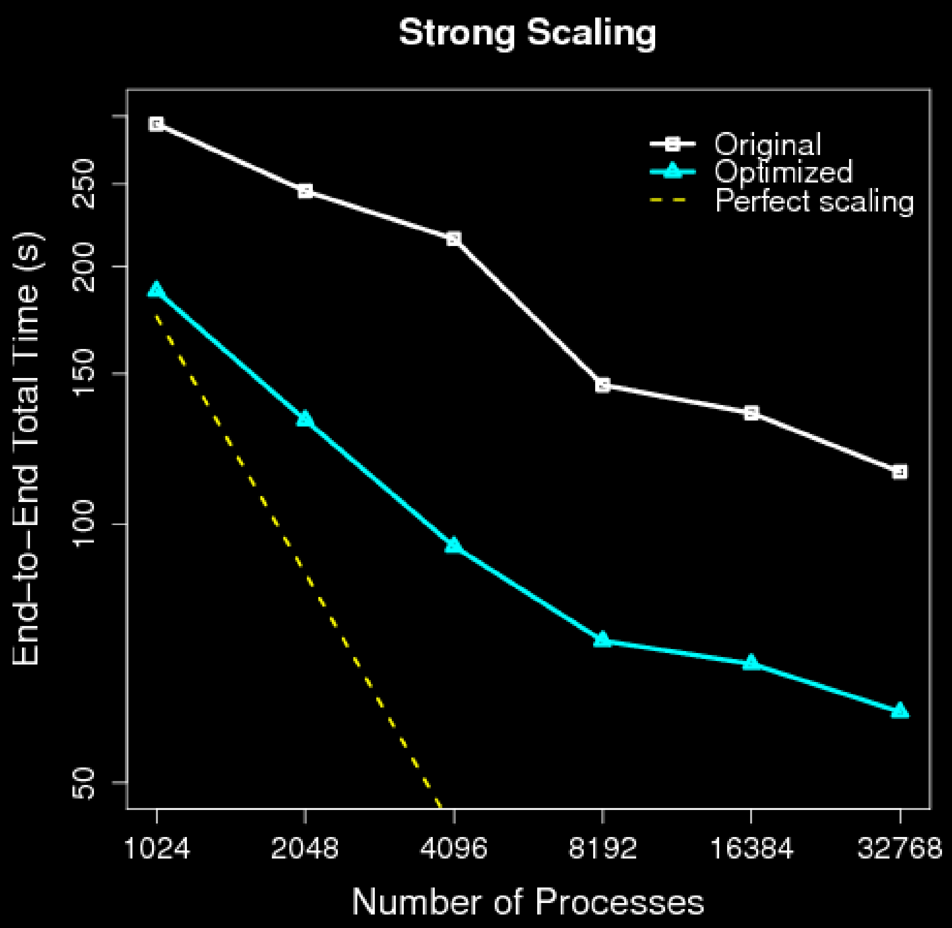


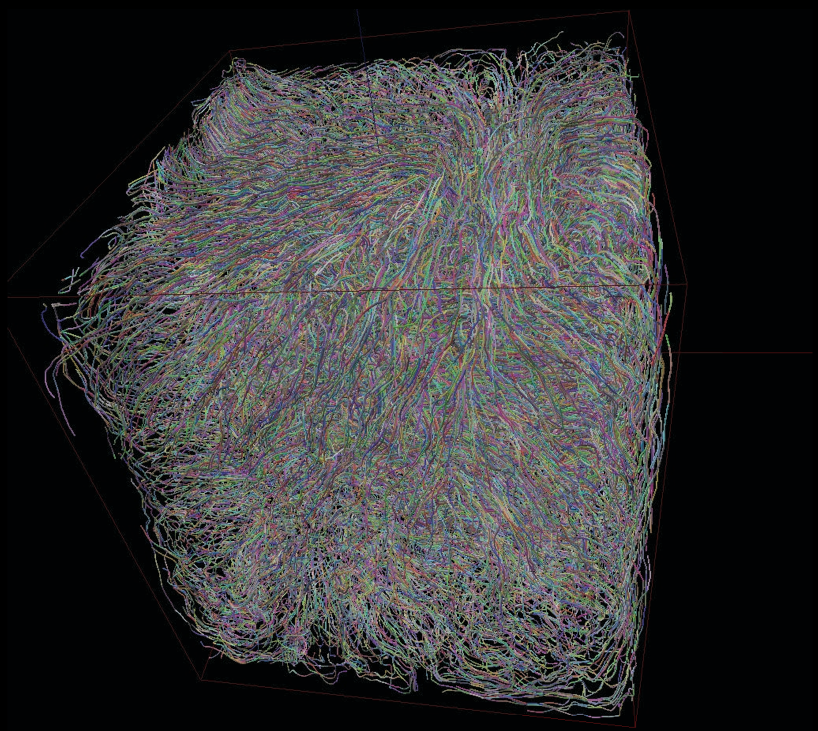
MAX experiment data. Point to point with wait factor = 1.0 is virtually the same as all to all.



Flame stabilization data. Less synchronization (wait factor = 0.1) improves performance.

Communication Performance at Scale



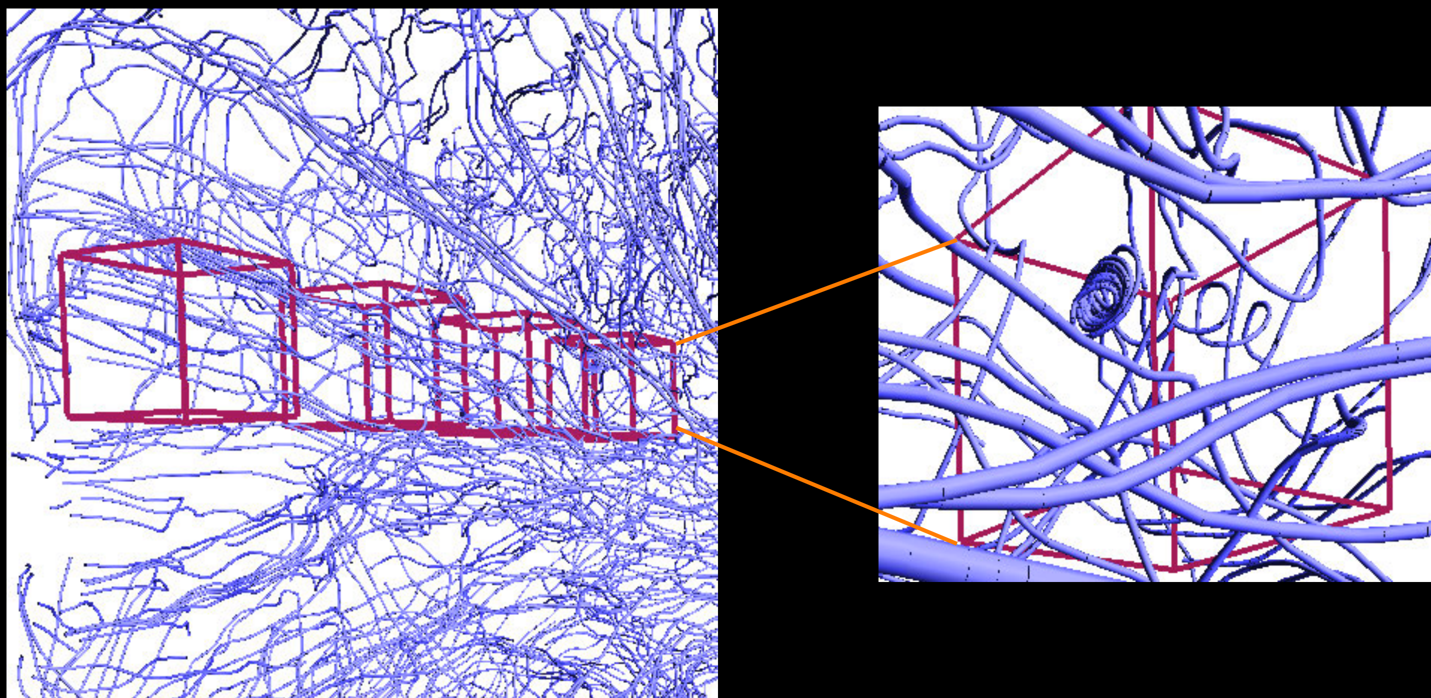


Platform: IBM Blue Gene/P

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. Most of this improvement comes from the wait factor. The left plot includes end-to-end time, including I/O, computation, and communication. The right image shows 8 thousand particles, much fewer than were actually tested.

The Problem of Load Balancing

Computational load is data dependent: data blocks containing vortices (sinks) attract particles and have high angular frequency requiring thousands more advection steps to compute than blocks with homogeneous flow. In the following slides, we evaluate three solutions: particle termination, multiblock assignment, and dynamic block re-assignment.

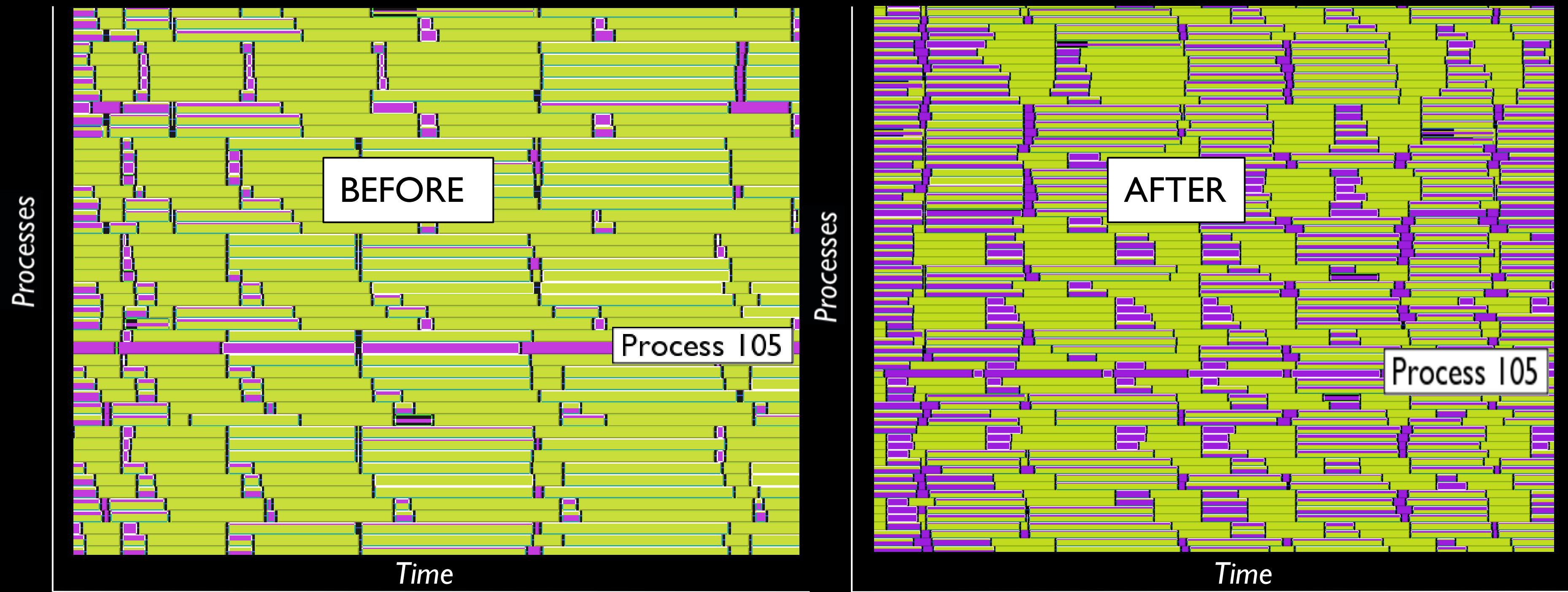


One process containing 4 blocks, with one block containing a vortex, can affect the load balance of the entire program execution.

Particle Termination

Problem: A busy process causes others to wait, which propagates throughout the system.

Solution: Particles that don't exit the current block after one round are terminated. There is no loss of information because these particles have near-zero velocity.

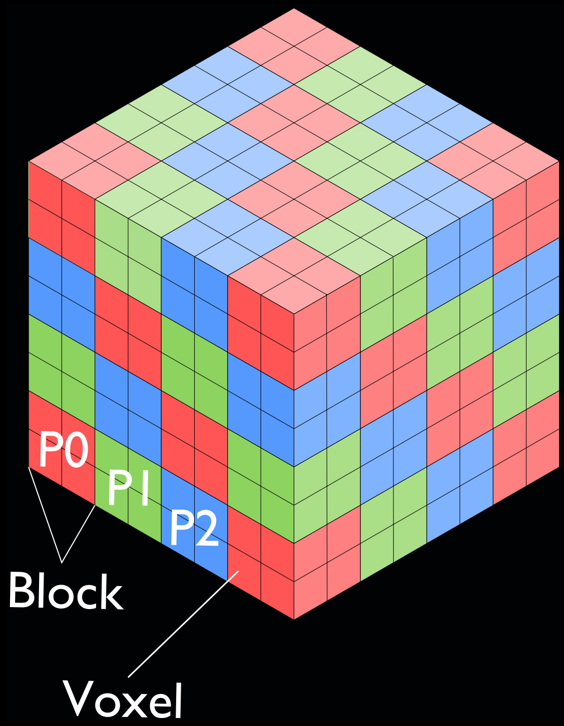


	Without Particle Termination	With Particle Termination
Max. Computation Time	243 s	55 s
Total Execution Time	256 s	67 s

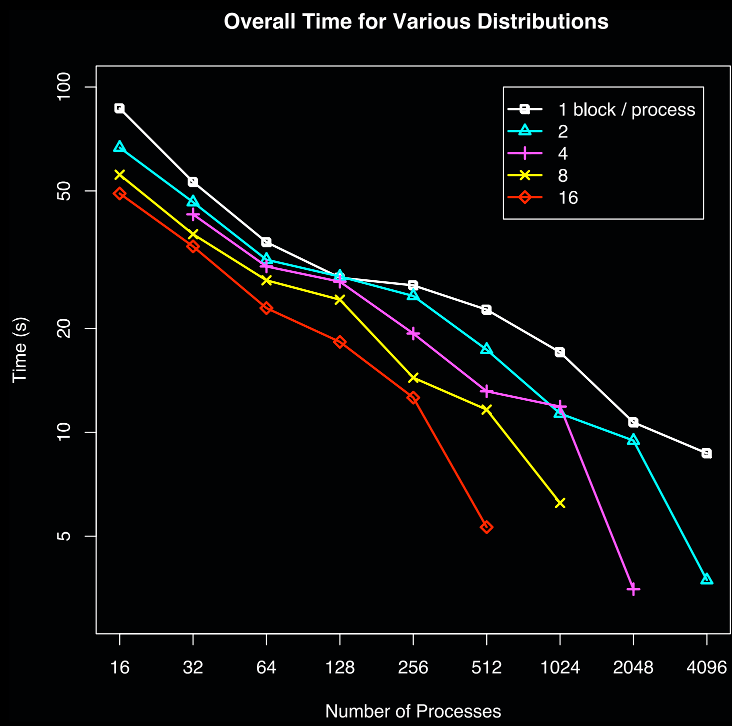
Jumpshots of 128 processes: process 105 is computation-bound and causes all others to wait. Terminating particles that do not leave the current block reduces maximum computation time and overall time.

Multiblock Assignment

Decomposing the domain into a larger number of smaller blocks helps, to a limit. Computational hot-spots are more likely to be amortized over a greater number of processes. Limiting factor: smaller blocks incur less computation and more communication because surface area / volume increases.



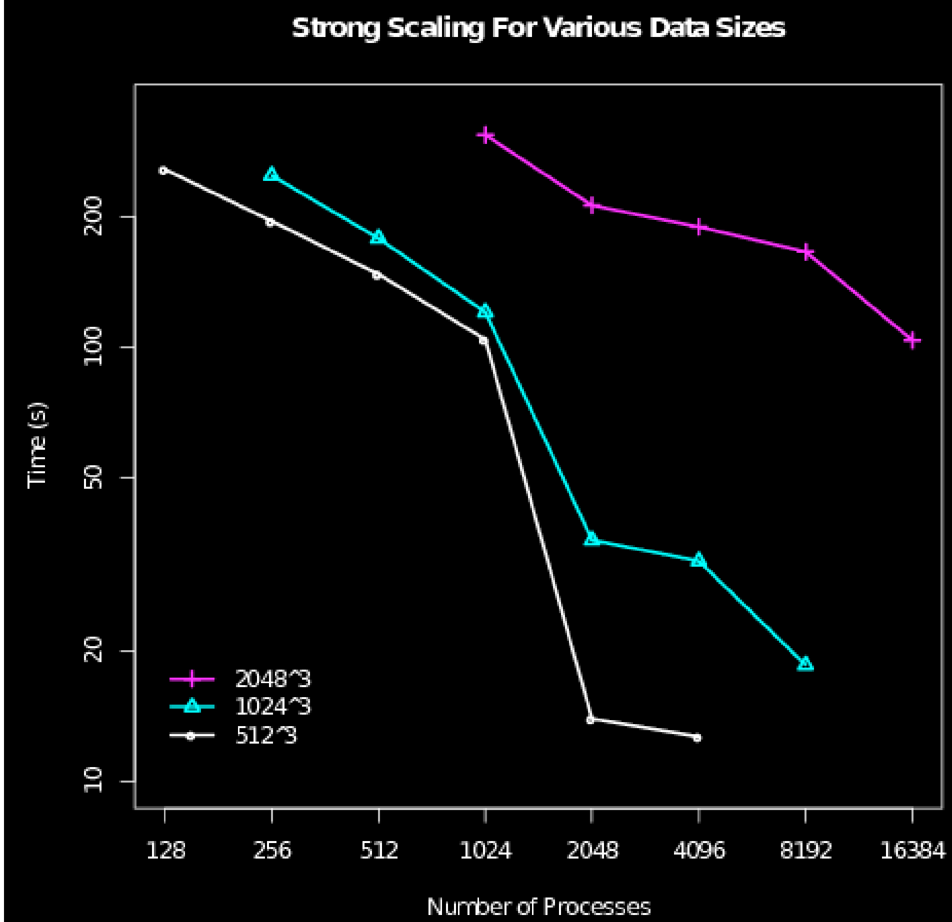
Example of 512 voxels decomposed into 64 blocks and assigned to 3 processes. Each process contains 21 or 22 blocks.

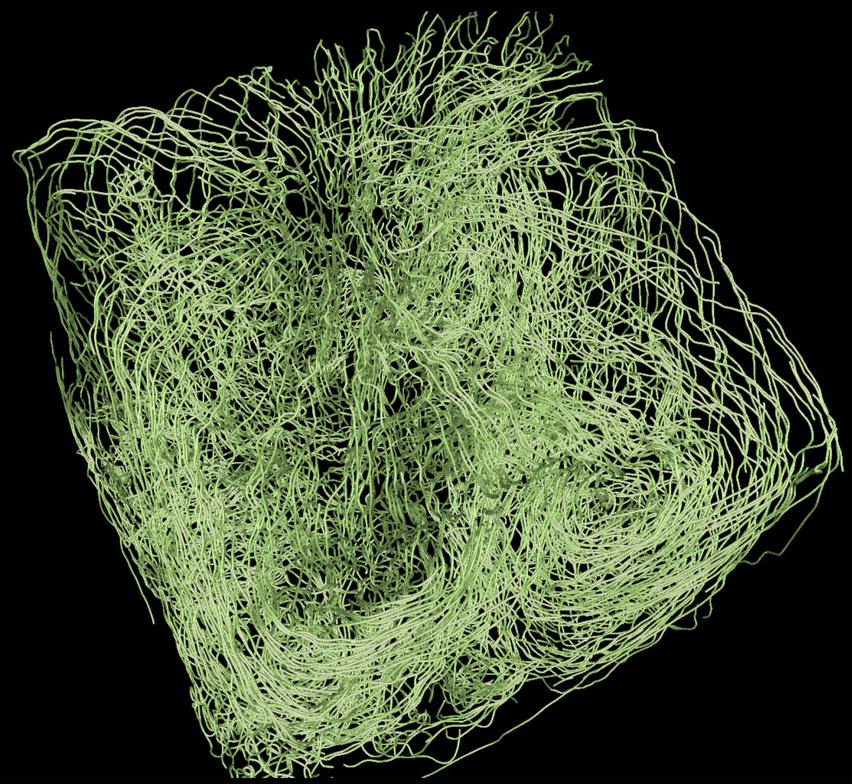


Decompositions of 1, 2, 4, 8, and 16 blocks per process in the MAX dataset, 512^3, 8K particles. Higher block numbers reduce the overall execution time. Early particle termination not applied in these tests.

MAX Experiment Results

Strong scaling, 512^3, 1024^3, 2048^3 data, 128K particles, 1 time-step



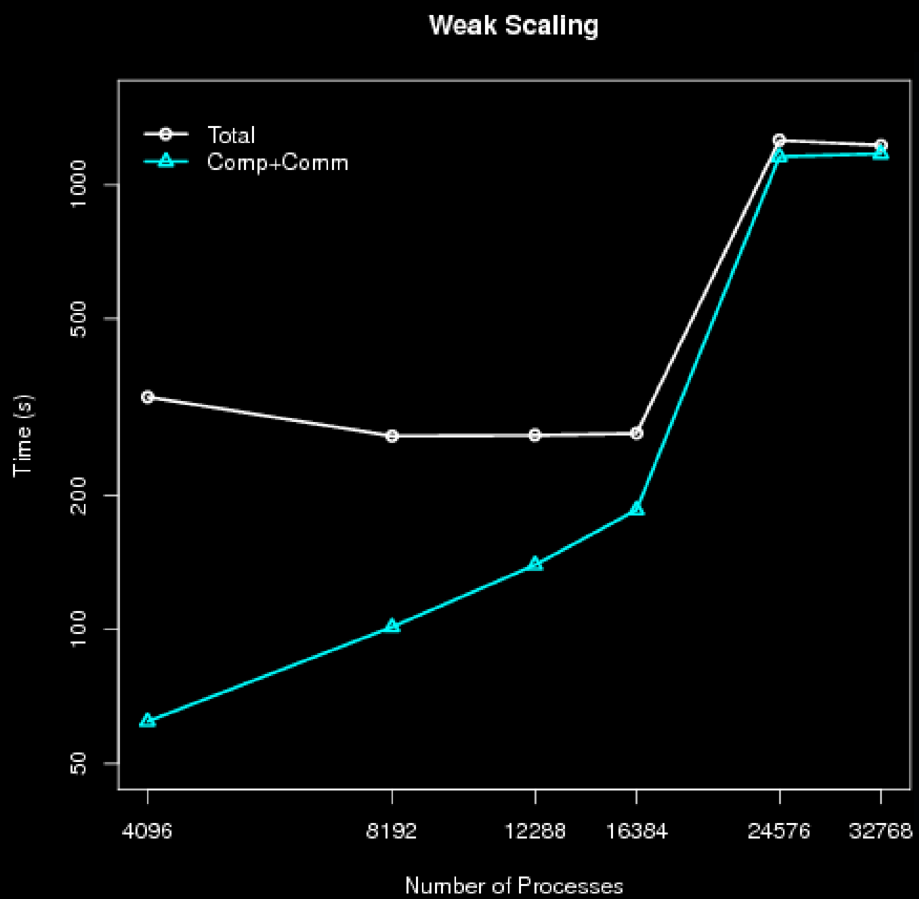


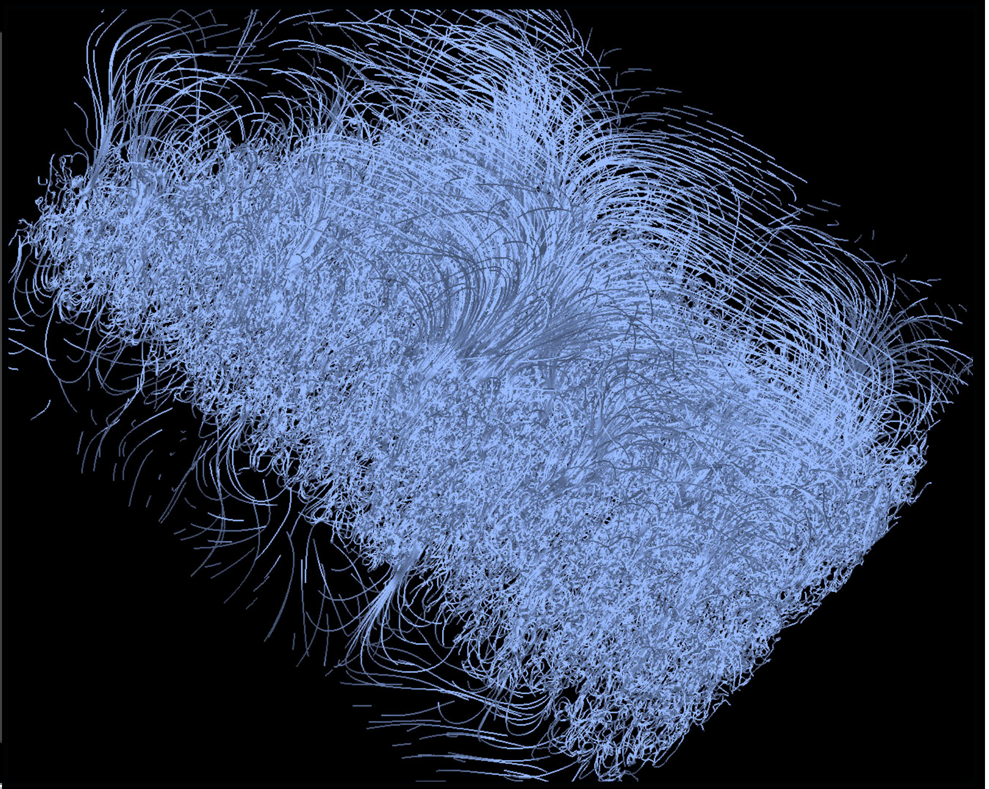
Platform: IBM Blue Gene/P

Data courtesy Aleks Obabko and Paul Fischer, ANL

Rayleigh-Taylor Results

Weak scaling, 2304 x 4096 x 4096 data, 16K to 128K particles, 1 time-step



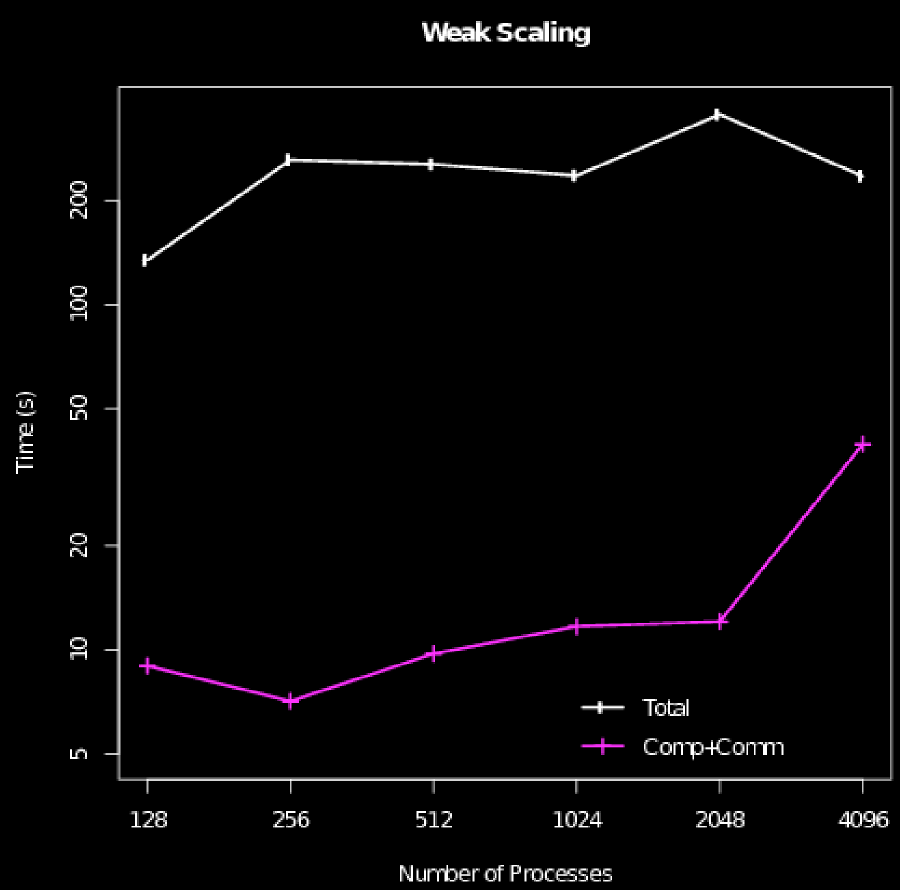


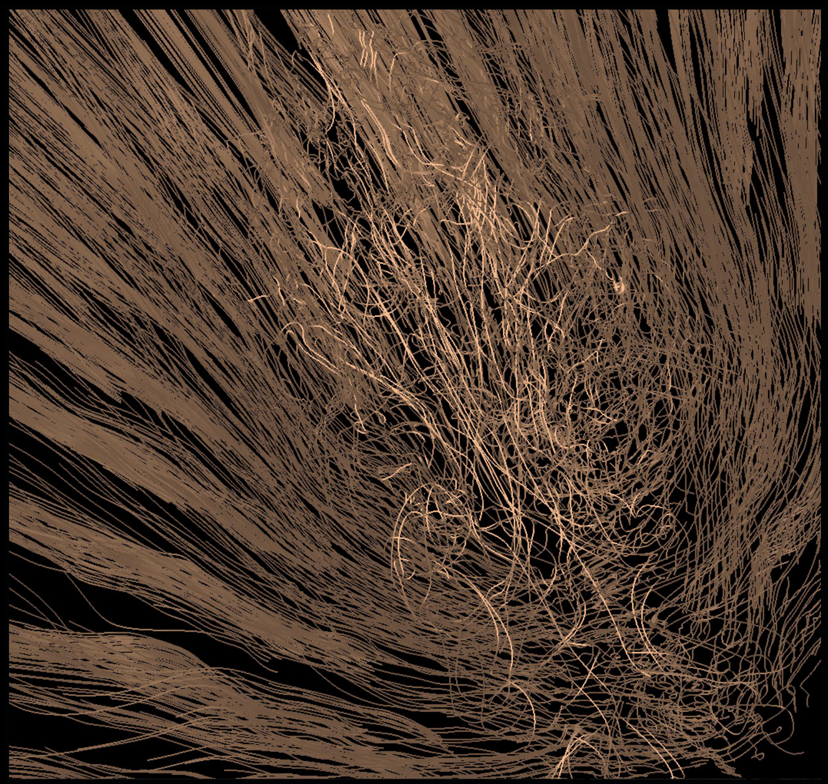
Platform: IBM Blue Gene/P

Data courtesy Mark Petersen and Daniel Livescu, LANL

Flame Stabilization Results

Weak scaling, 1408 x 1080 x 1100 data, 512 to 16K particles,1 to 32 time-steps





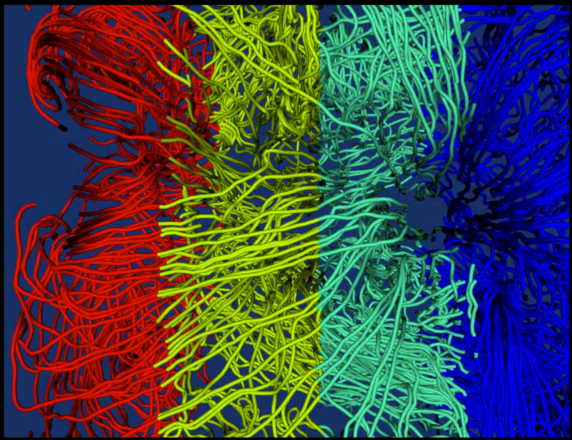
Platform: IBM Blue Gene/P

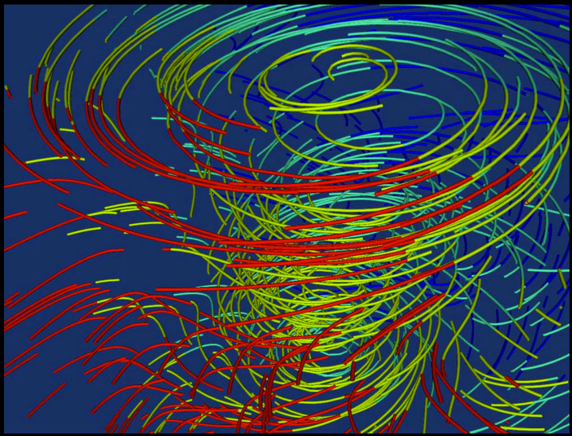
Data courtesy Ray Grout, NREL and Jackie Chen, SNL

VTK Integration

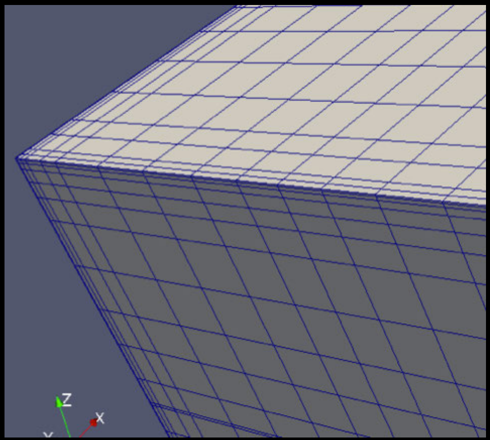
Courtesy Zhanping Liu and Jimmy Chen

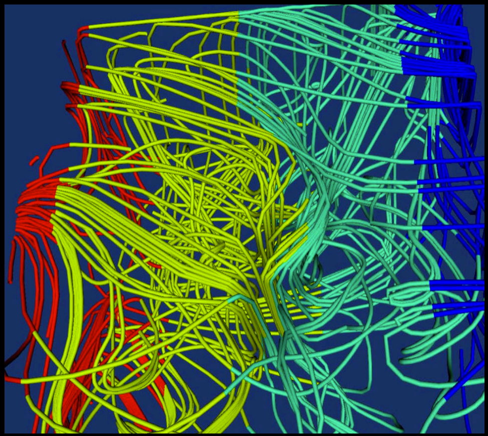
VTK Parallel Reader — Adapter — VTK Parallel Streamline / Pathline Filter (OSUFlow, DIY) — Adapter — VTK Renderer



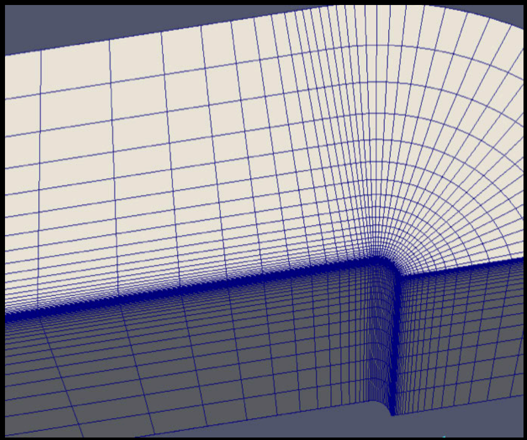


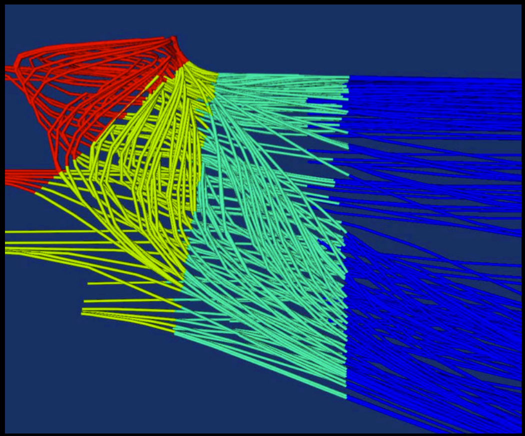
Top: Streamlines of thermal hydraulics. Bottom: Pathlines of tornado





Top: Mesh for office airflow. Bottom: streamlines for office airflow





Top: Mesh for blunt fin. Bottom: streamlines for blunt fin

Summary

Keys to Successes

Configurable time-space data structure with variable size epochs and blocks

- Load as many time steps into memory as possible

Communication algorithm with adjustable synchronization

- Less synchronization is better, eg., wait for 10% of pending messages

Simple load balancing strategies

- Multiple blocks per process, particle termination

Ongoing / future work

Continuing to study dynamic load balancing and prediction using graph methods

AMR and unstructured grid parallelization

VTK integration

Hybrid messaging / threading parallel approaches

Recommended Reading

DIY

- Peterka, T., Ross, R., Kendall, W., Gyulassy, A., Pascucci, V., Shen, H.-W., Lee, T.-Y., Chaudhuri, A.: Scalable Parallel Building Blocks for Custom Data Analysis. Proceedings of Large Data Analysis and Visualization Symposium (LDAV'11), IEEE Visualization Conference, Providence RI, 2011.
- Peterka, T., Ross, R.: Versatile Communication Algorithms for Data Analysis. 2012 EuroMPI Special Session on Improving MPI User and Developer Interaction IMUDI'12, Vienna, AT.

Particle Tracing Applications

- Peterka, T., Ross, R., Nouanesengsey, B., Lee, T.-Y., Shen, H.-W., Kendall, W., Huang, J.: A Study of Parallel Particle Tracing for Steady-State and Time-Varying Flow Fields. Proceedings IPDPS'11, Anchorage AK, May 2011.
- Kendall, W., Wang, J., Allen, M., Peterka, T., Huang, J., Erickson, D.: Simplified Parallel Domain Traversal. Proceedings of SC11, Seattle WA, 2011.
- Nouanesengsy, B., Lee, T.-Y., Lu, K., Shen, H.-W., Peterka, T.: Parallel Particle Advection and FTLE Computation for Time-Varying Flow Fields. Proceedings of SC12, Salt Lake, UT, 2012.
- Kendall, W., Huang, J., Peterka, T.: Geometric Quantification of Features in Large Unsteady Flow. Computer Graphics and Applications Special Issue on Extreme Scale Visual Analytics, Vol. 32, No. 4, 2012.
- Pugmire, D., Garth, C., Childs, H., Peterka, T. Parallel Integral Curves. Book chapter in High Performance Visualization. Bethel, E. W., Childs, H., Hansen, C., editors, 2012.

U.S. DEPARTMENT OF ENERGY

Argonne NATIONAL LABORATORY

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

–Richard Hamming, 1962

Foundations of Data-Parallel Particle Advection

Thank You

Facilities

Argonne Leadership Computing Facility (ALCF)

Oak Ridge National Center for Computational Sciences (NCCS)

Funding

US DOE SciDAC SDAV Institute

Subversion repositories

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

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

Tom Peterka

tpeterka@mcs.anl.gov

Mathematics and Computer Science Division