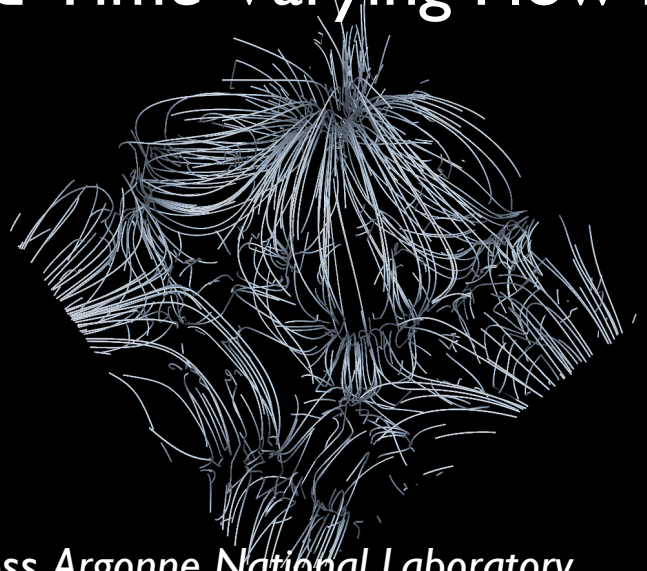
U.S. DEPARTMENT OF ENERGY

Argonne NATIONAL LABORATORY

A Study of Parallel Particle Tracing for Steady-State and Time-Varying Flow Fields

Early stages of Rayleigh-Taylor Instability flow



Tom Peterka, Rob Ross *Argonne National Laboratory*

Boonth Nouanesengsey, Teng-Yok Lee, Han-Wei Shen *The Ohio State University*

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

Tom Peterka

tpeterka@mcs.anl.gov

Mathematics and Computer Science Division

IPDPS 2011

5/18/11 Anchorage AK

The Need for Parallel Visualization and Analysis

When data sizes are too large for moving data or processing serially, parallel analysis and visualization needs to be executed on HPC machines at increasingly large scale. Results are available sooner, access to all data at full resolution is possible. Visualization / data analysis are becoming computational challenges in their own right, requiring scalable algorithms.

Rayleigh-Taylor Instability

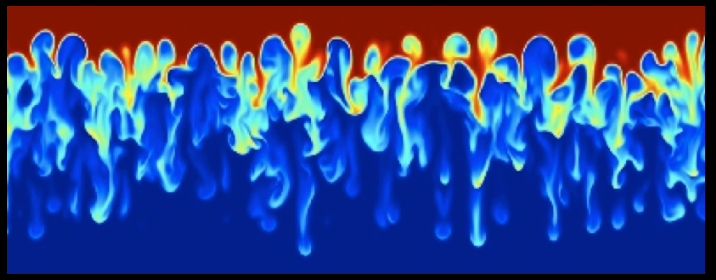


Image courtesy Mark Petersen, Daniel Livescu, LANL. Code: CFDNS

Test Data Sizes

Dataset	Grid size	Data size (GB)
MAX	512^3	1.5
MAX	1024^3	12
MAX	2048^3	98
RTI	2304 x 4096 x 4096	432
Flame	1408 x 1080 x 1100 x 32 time steps	608

MAX Experiment

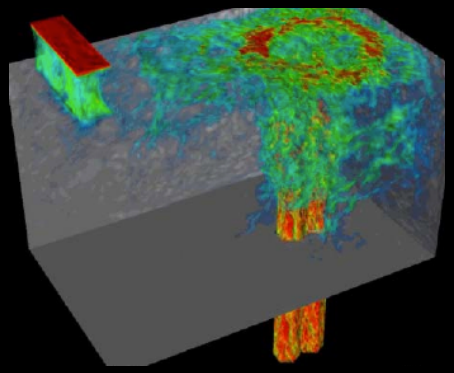


Image courtesy Paul Fischer, Aleks Obabko, ANL. Code: Nek5000

Flame Stabilization

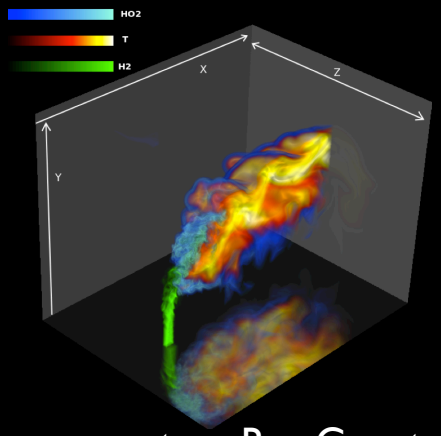


Image courtesy Ray Grout, NREL, Hongfeng Yu, Jackie Chen, SNL Code: S3D

Parallel Particle Tracing of Field Lines

Problem

Field lines require high-order iterative numerical integration to trace particles in the flow field.

Data sizes are large, as the previous slide showed, 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.

Contributions

A configurable 3D / 4D hybrid data structure enables variable size and number of blocks, and adjustable in-core parallelism / out-of-core sequencing while tracing time-varying flows.

A communication algorithm that enables adjustable synchronization

A study of load balancing with three solutions that we tested

Large-scale parallel benchmark results of both and static and time-varying scientific data

Comparison to Prior Work

Reference	Max. Grid Size	Max. processes	Max. Particles	Time Dependence	Data Structure	Load Balance
Yu et al. SC 07	864^3	256	1 M	Time-varying	4D	Preprocess
Pugmire et al. SC 09	800^3	512	22 K	Steady-state	3D	Dynamic
Peterka et al. IPDPS 11	2K x 4K x 4K	32 K	128 K	Time-varying	3D / 4D hybrid	Static

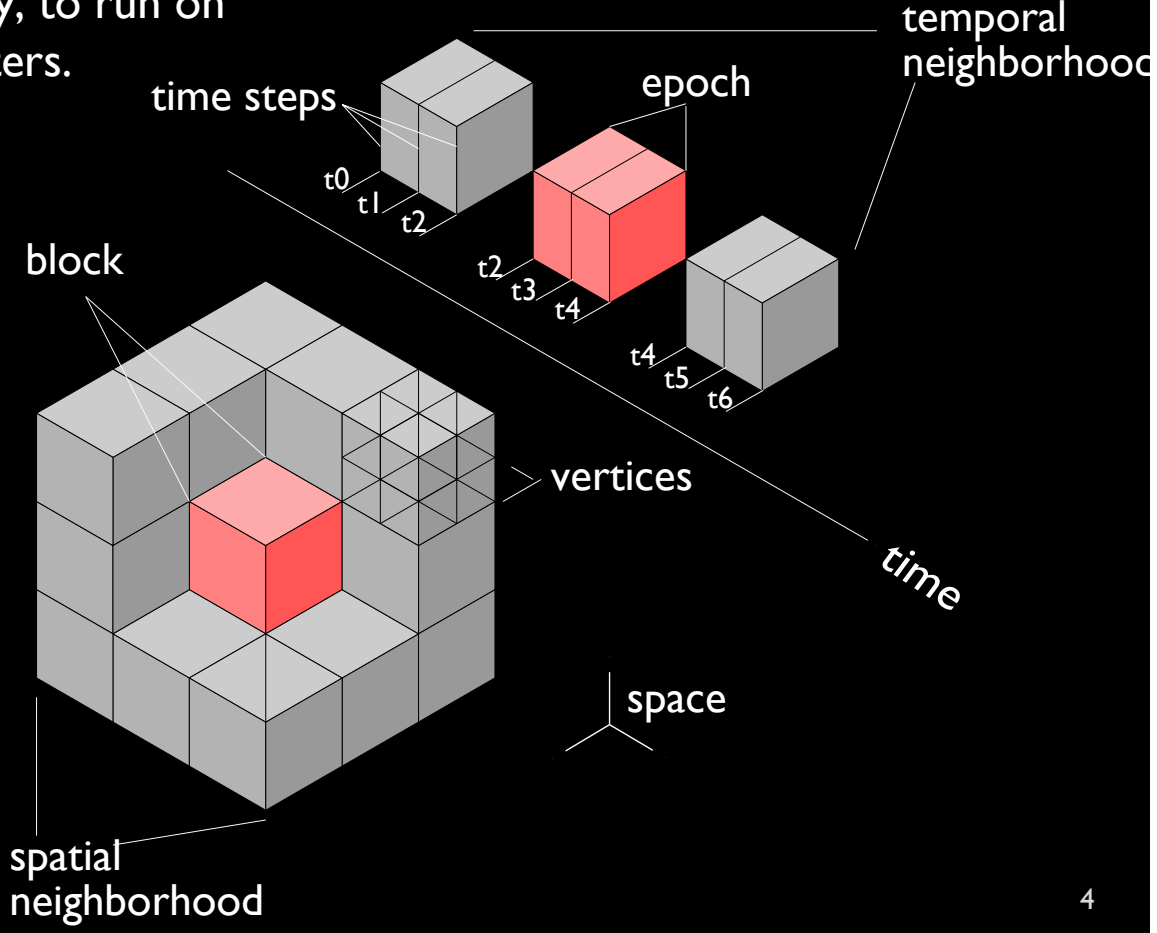
Configurable, 3D / 4D Hybrid Data Structure and Algorithm

Internally, all blocks are 4D, but we allow separate grouping in space (blocks) and time (epochs) such that we can control how much data are kept in-core with the size of the epoch. This enables time-varying data to be traced natively in 4D, without requiring the entire 4D dataset to be resident in memory, to run on desktops, clusters, and supercomputers.

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
  }
}
```

Data structure



Communication Algorithm

Sparse collectives can be implemented using all-to-all (A2A) or point-to-point (P2P).

A2A is synchronous, while P2P allows for varying degrees of synchrony.

All-to-all implementation

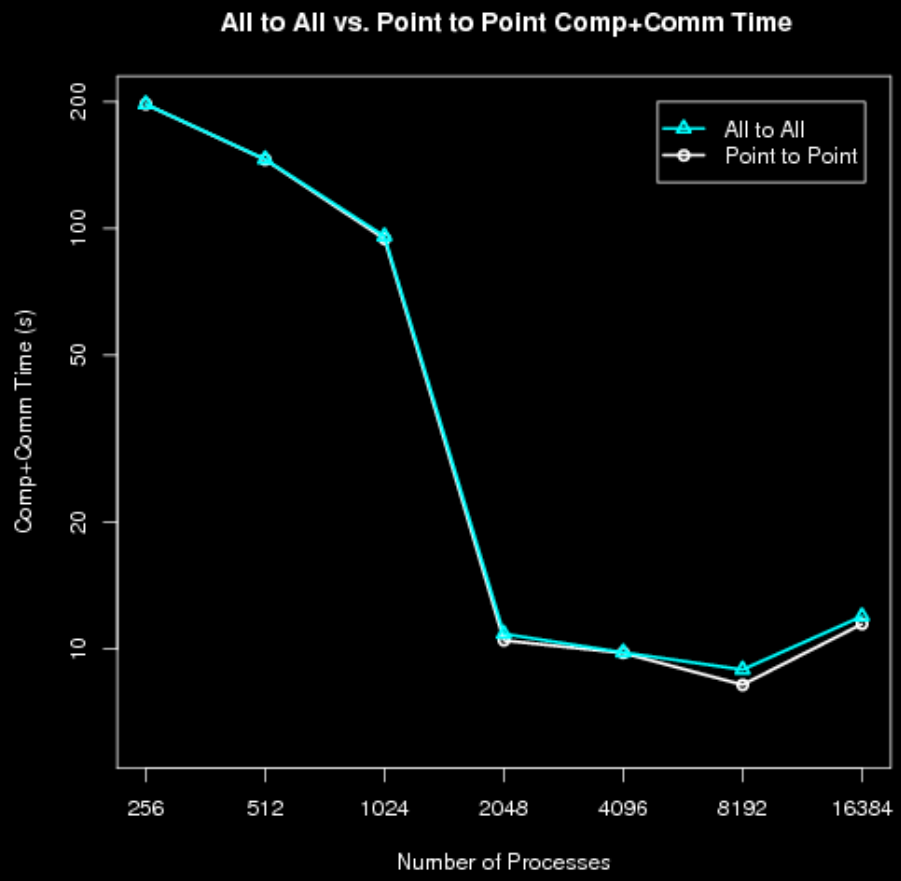
Pack vector of sending block ids, # points
Exchange point counts (MPI_Allltoallv)
Unpack vector of receiving point counts
Pack vector of sending points
Exchange points (MPI_Alltoallv)
Unpack vector of received points

Point-to-point implementation

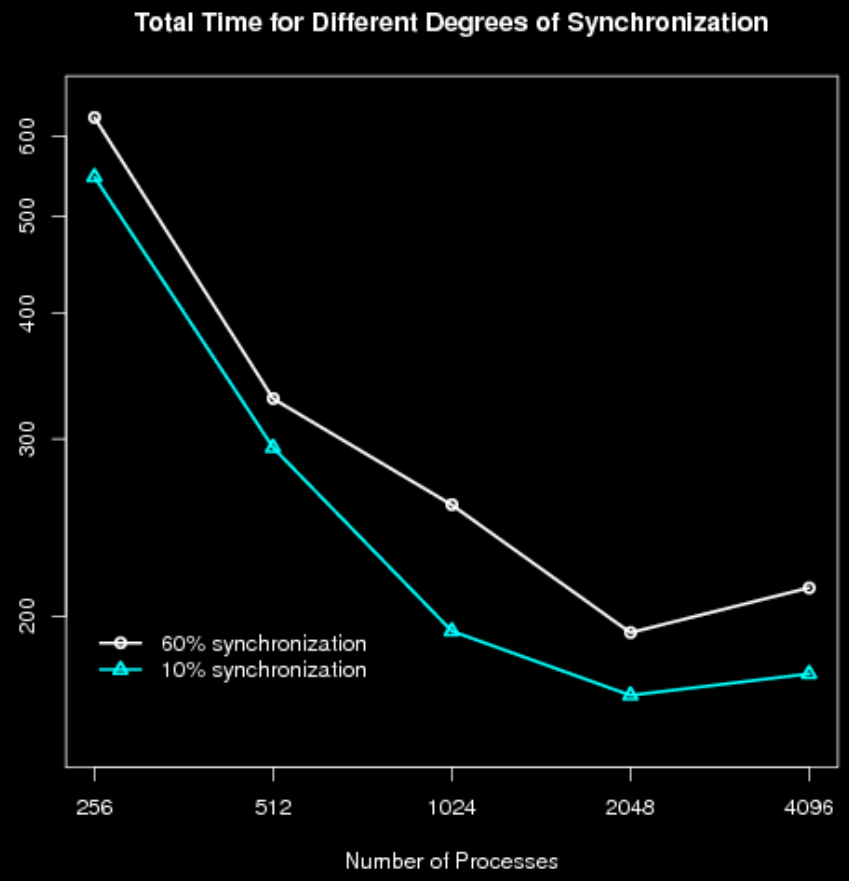
```
for (processes in my neighborhood) {
  pack message of block IDs and particle counts
  post nonblocking send
  pack message of particles
  post nonblocking send
  post nonblocking receive of IDs and counts
}
wait for enough IDs and counts to arrive
for (IDs and counts that arrived) {
  post blocking receive for particles
}
```

Communication Performance

Changing from all-to-all (A2A) to point-to-point (P2P) and waiting for all messages to arrive offers little improvement, but dialing down the percentage of messages for which to wait helps significantly.



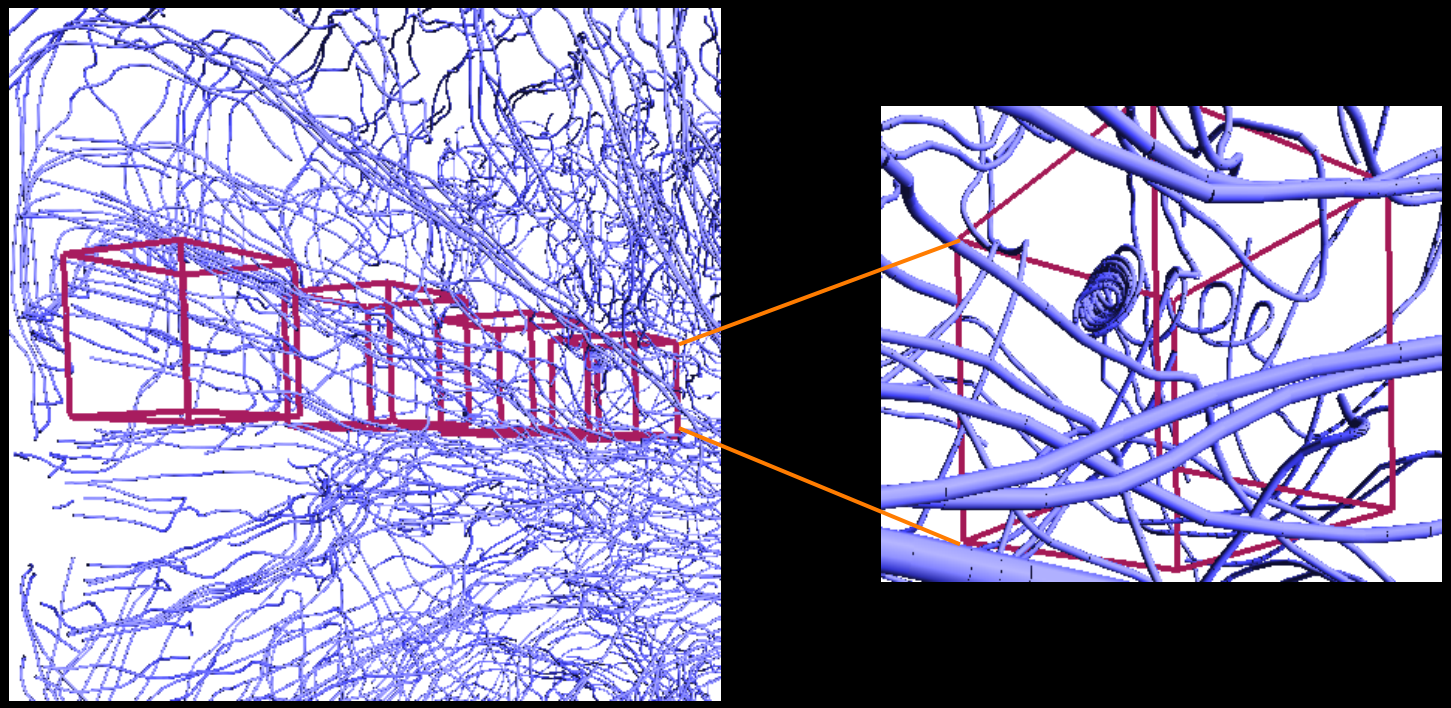
MAX experiment data. A2A is virtually the same as 100% synchronized P2P



Flame stabilization data. Less synchronization (waiting for a smaller percentage of messages) improves performance.

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.

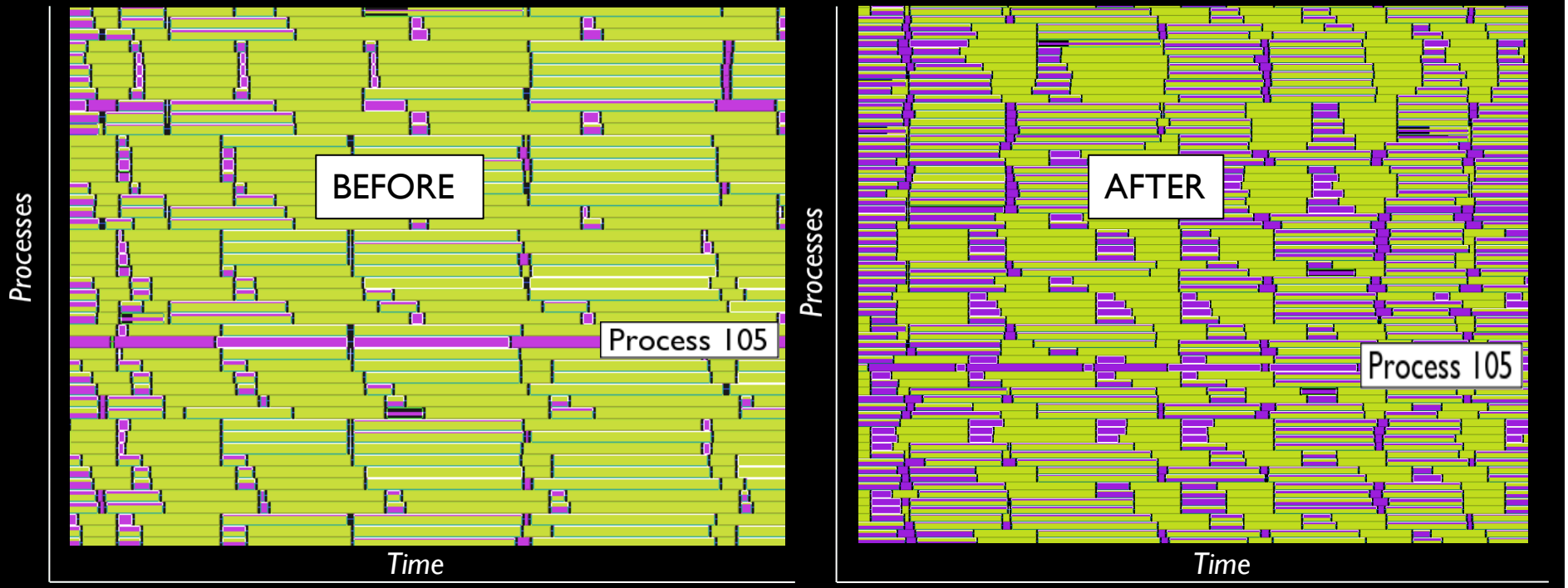


One process containing 4 blocks, with one block containing a vortex, can affect the load balance of the entire program execution.

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.

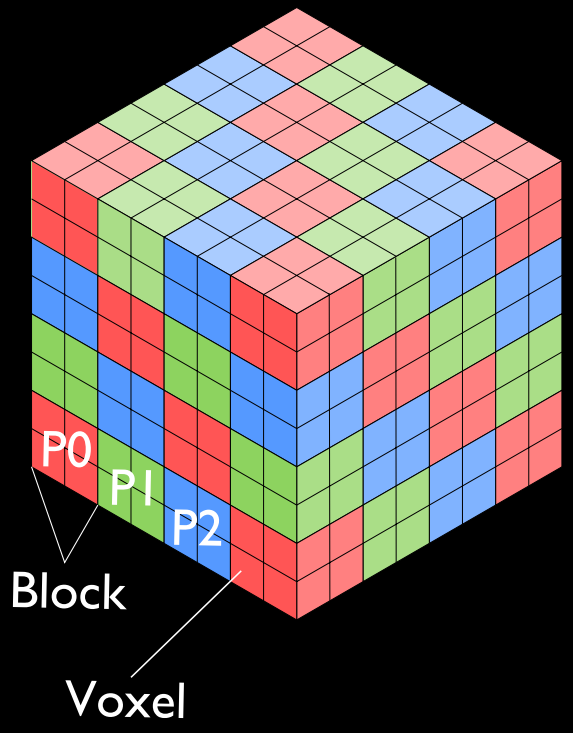


	Without Particle Termination	With Particle Termination
Max. Computation Time	243 s	55 s
Total Execution Time	256 s	67 s

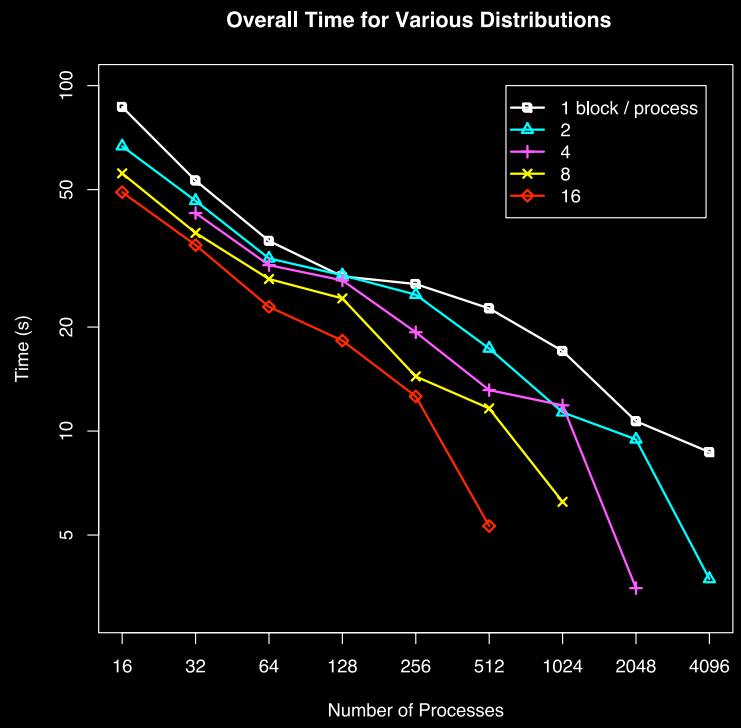
Jumpshots of 128 processes: process 105 is computation-bound and causes all others to wait. Terminating particles that do not leave the current block reduces maximum computation time and overall time.

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.

Dynamic Block Re-assignment

Re-assignment is performed with the help of the Zoltan Parallel Data Services Toolkit. Granularity is block-level, and the weighting function is the number of advection steps per block. Reassignment is performed between epochs in time-varying flame dataset.

Dynamic re-assignment algorithm

```
start with default round-robin partition
for (epochs) {
  read data for current epoch
    with current block assignment
  advect particles
  compute weighting function
  compute new block assignment
}
```

Next steps

-Optimize re-assignment

-Test in steady-state flow

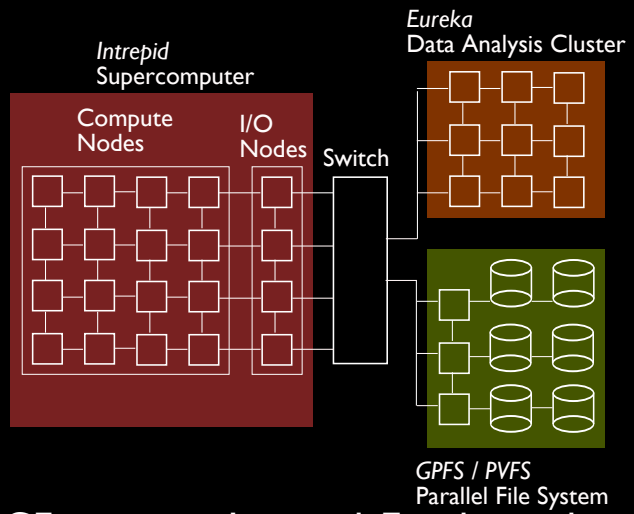
		Std. Dev. Total Advection Steps		Max. Compute Time	
Time steps	Epochs	Static	Dynamic	Static	Dynamic
16	4	31	20	0.15	0.14
16	8	57	28	1.03	0.99
32	4	71	42	0.18	0.16
32	8	121	52	1.12	1.06
64	4	172	103	0.27	0.21
64	8	297	109	1.18	1.09

compare compare

Dynamic re-assignment compared to static assignment, measured by standard deviation in number of steps (blue columns) and maximum compute time (red columns)

Large-Scale Benchmark Results: Test Environment

Tests were run on the Argonne Leadership Facility's *Intrepid* Machine at up to 32 K processes in virtual node mode (one MPI process per core).



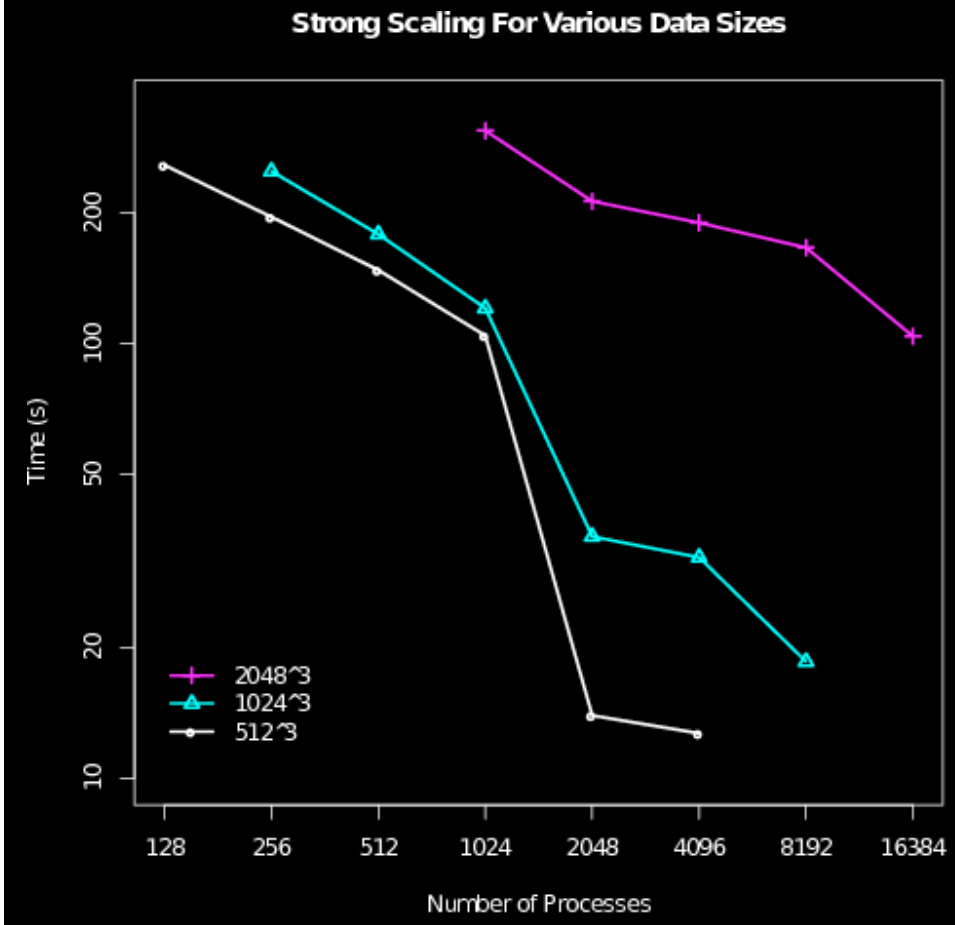
ALCF operates Intrepid, Eureka, and a parallel file system connected to the same network. Our tests were run on Intrepid.

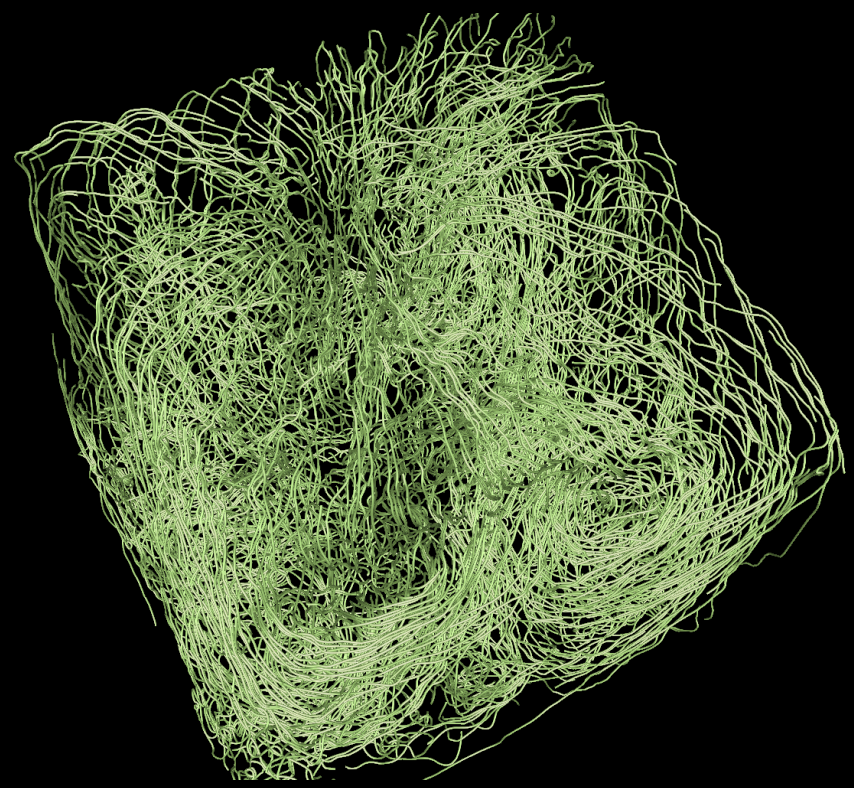
The Blue Gene/P features a highly scalable compute architecture composed of 160,000 compute cores. Peak performance is 557 TF.

Specifications		
Clock frequency	850 MHz	
Total cores	160 K	4 cores / node
3D Torus network	5 us latency	3.4 Gb/s per link
Tree network	5 us latency	6.8 Gb/s per link
Memory per MPI process	512 MB vn mode	2 GB smp mode
Total memory	80 TB	
System software	MPI, CNK	IBM xlcxx

MAX Experiment Results

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

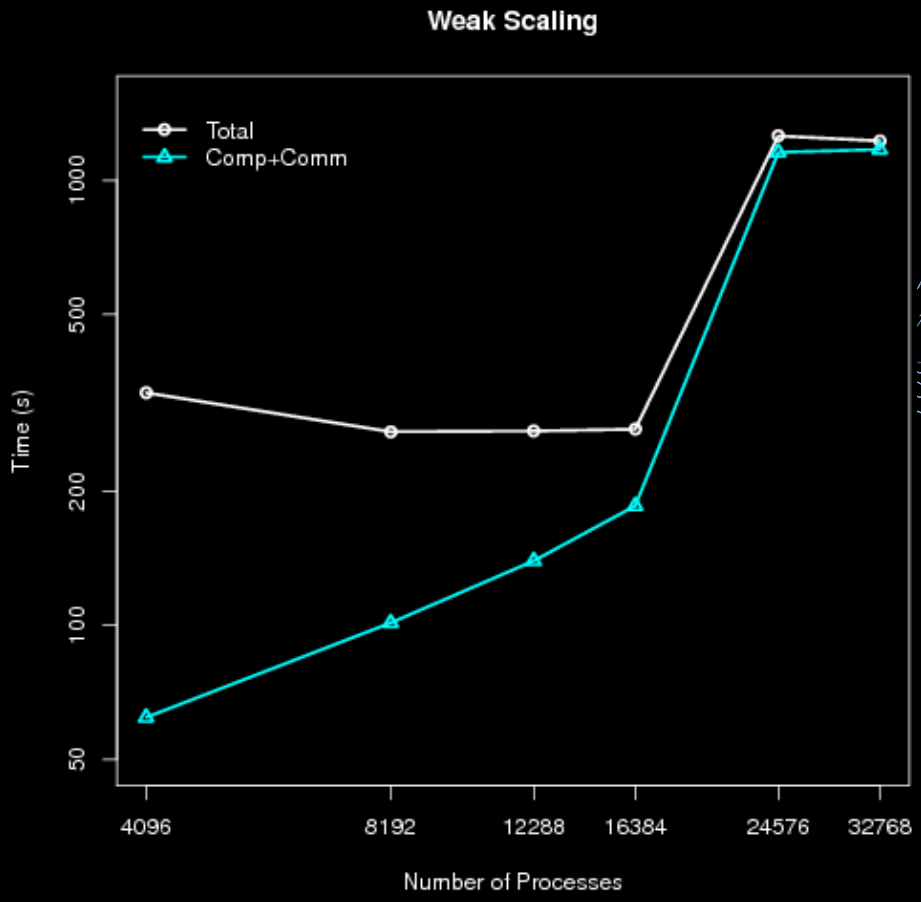


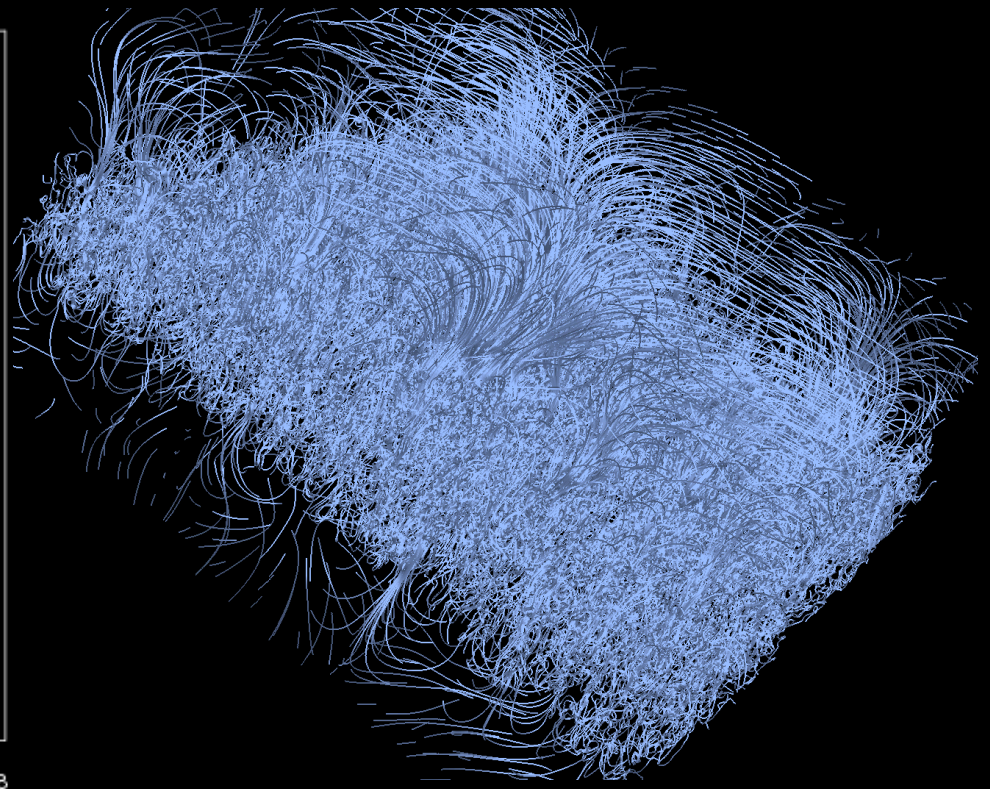


Data courtesy Aleks Obabko and Paul Fischer, ANL

Rayleigh-Taylor Results

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

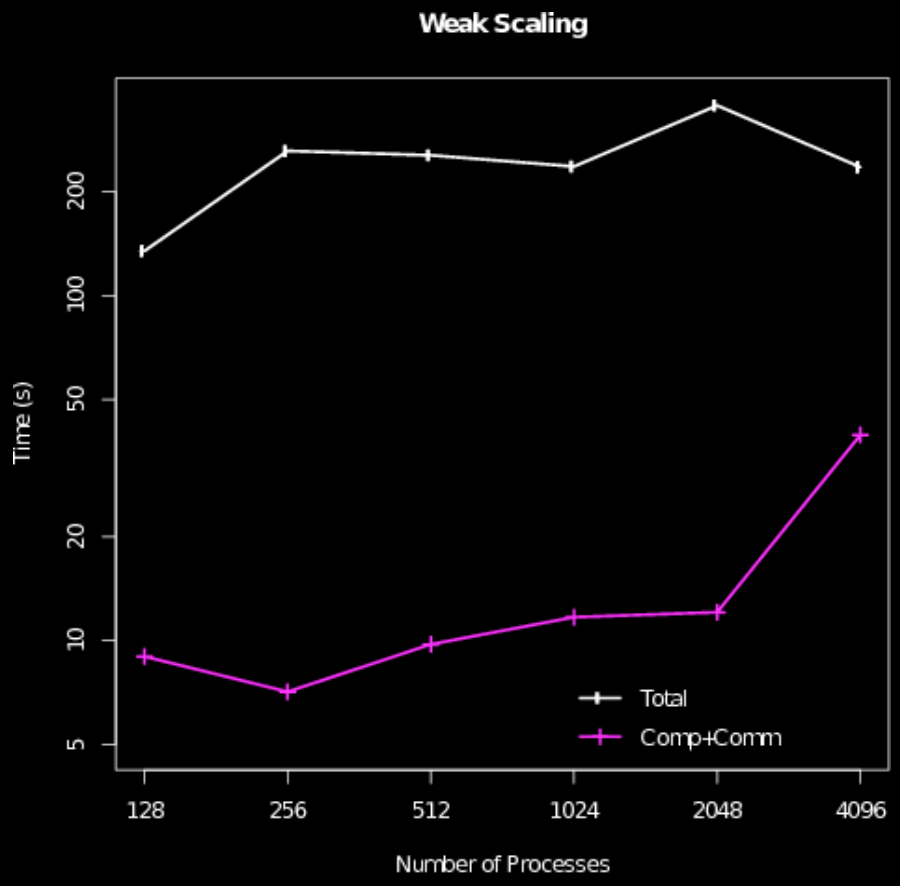


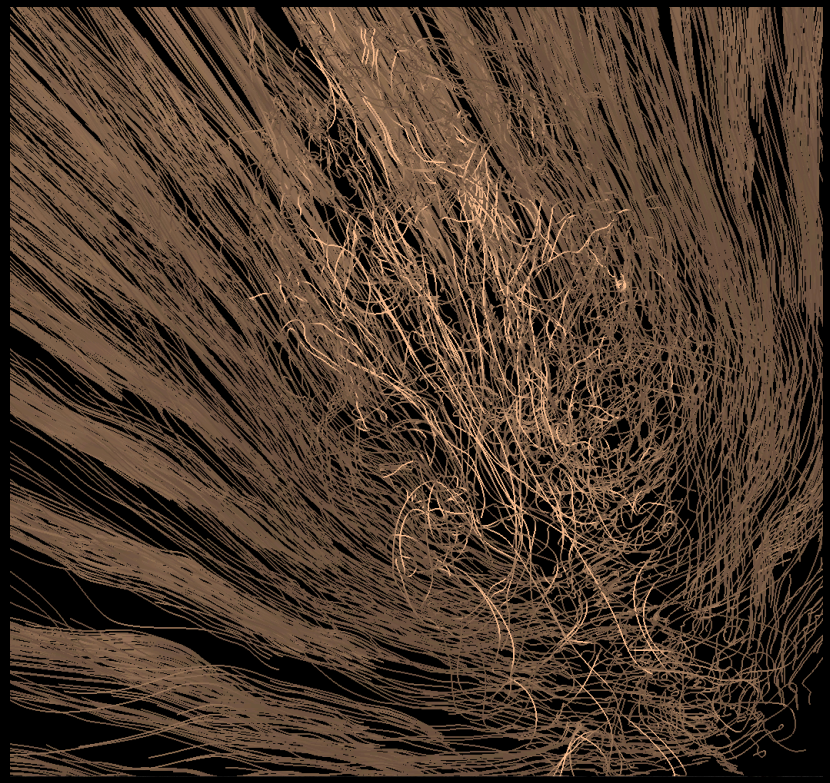


Data courtesy Mark Petersen and Daniel Livescu, LANL

Flame Stabilization Results

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





Data courtesy Ray Grout, NREL and Jackie Chen, SNL

Summary

Successes

A configurable time-space data structure with variable size epochs and blocks

A communication algorithm with adjustable synchronization

A study of load balancing that tested three solutions

Large-scale parallel benchmark results of both and static and time-varying scientific data

Conclusions

Our data structure enables us to load as many time-steps into memory as possible.

Less synchronization in communication is better, eg. waiting for 10% of pending messages.

In practice, we use particle termination and multiblock assignment for load balancing.

Dynamic re-assignment is a promising avenue for further research.

Demonstrated some of the largest parallel results to date (grid size, system size, # particles).

Ongoing / future work

Continuing to study load balancing, including preprocessing using graph partitioning

Continuing to develop prototype AMR grid version

Planning to tackle unstructured grid problems

Investigating hybrid messaging / threading parallel approaches

U.S. DEPARTMENT OF ENERGY

Argonne NATIONAL LABORATORY

A Study of Parallel Particle Tracing for Steady-State and Time-Varying Flow Fields

Thank You

Facilities
Argonne Leadership Computing Facility (ALCF)
Oak Ridge National Center for Computational Sciences (NCCS)

Funding
US DOE SciDAC UltraVis Institute

People
Rob Latham, Dave Goodell, Paul Fischer, Aleks Obabko, Mark Petersen, Daniel Livescu, Ray Grout, Jackie Chen, Paul Ricker, Matt Sutter

Tom Peterka

tpeterka@mcs.anl.gov

Mathematics and Computer Science Division

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