

LA-UR-13-21884

Approved for public release; distribution is unlimited.

Title: Portable Data-Parallel Visualization and Analysis Operators

Author(s): Sewell, Christopher Meyer

Intended for: GPU Technology Conference, 2013-03-20 (San Jose, California, United States)



Disclaimer:

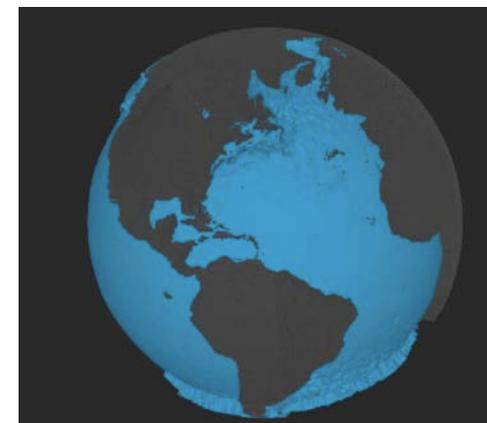
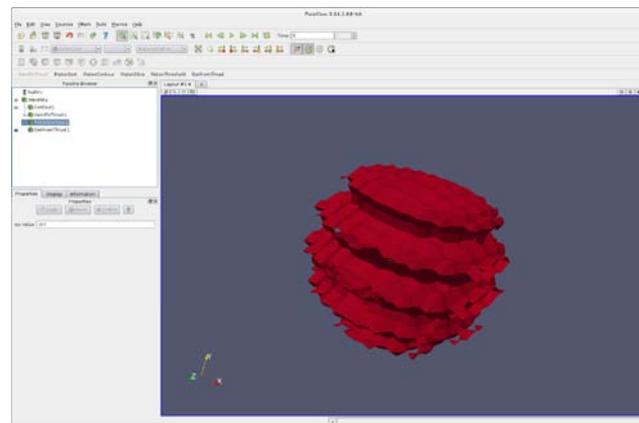
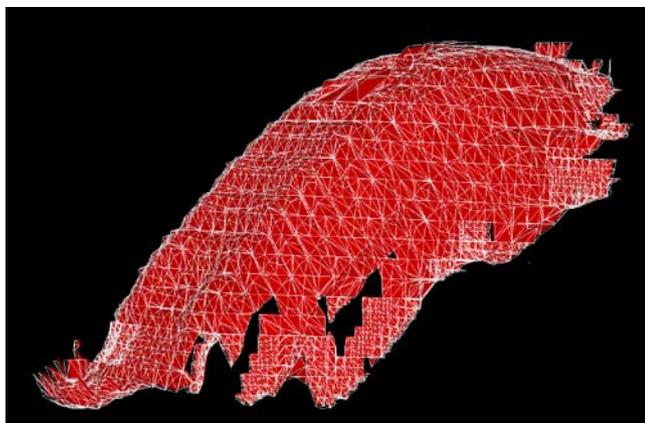
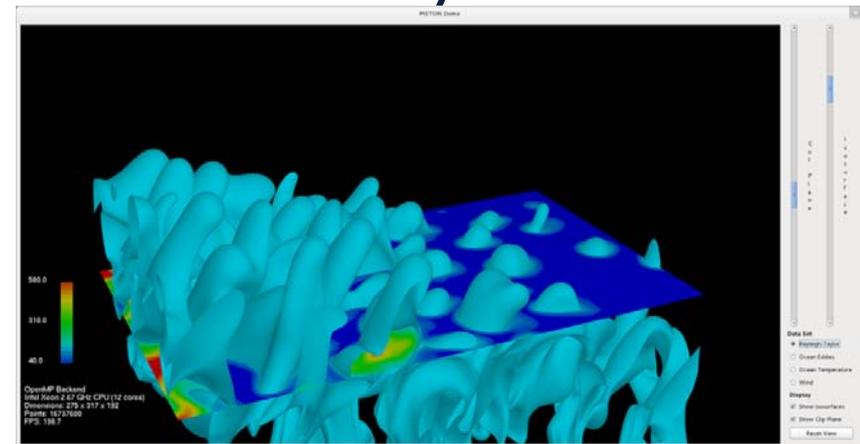
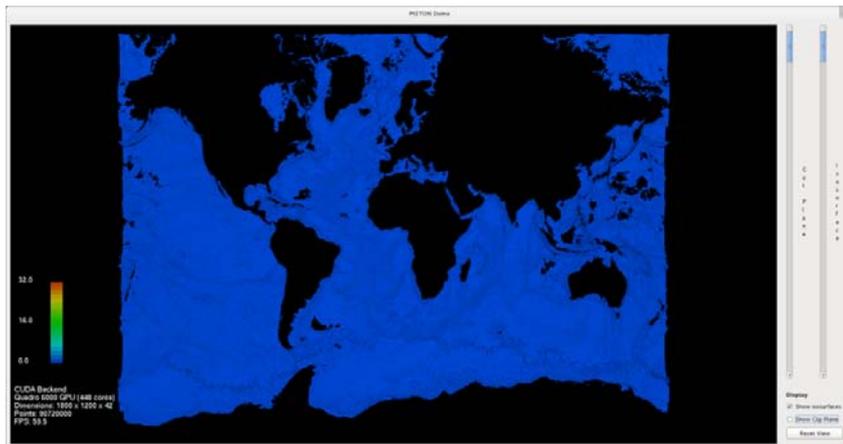
Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Abstract for “Portable Data-Parallel Visualization and Analysis Operators”

This presentation describes the overall goal of PISTON and PINION (to provide high parallel performance on current and next-generation supercomputers using portable, data-parallel code), and summarizes the work on these projects to date. It is intended for an audience at NVIDIA’s GPU Technology Conference, and thus has an emphasis on how it uses Thrust to write code that obtains good parallel performance when compiled to different backends, including CUDA.

Portable Data-Parallel Visualization and Analysis Operators

Chris Sewell, Li-Ta Lo, and James Ahrens
Los Alamos National Laboratory



Overview

- Goal: Portability and performance for visualization and analysis operators on current and next-generation supercomputers
- Main idea: Write operators using only data-parallel primitives (scan, reduce, etc.)
- Requires architecture-specific optimizations for only for the small set of primitives
- PISTON is built on top of NVIDIA's Thrust Library

Motivation

- Current production visualization software does not take full advantage of acceleration hardware and/or multi-core architecture
- Research on accelerating visualization operations is mostly hardware-specific; few were integrated in visualization software
- Standards such as OpenCL may allow program to run cross-platform, but usually still requires many architecture specific optimizations to run well

The Data Parallel Programming Model

- Data parallelism: independent processors performs the same task on different pieces of data
- A seminal work: Guy Blelloch's "Vector Models for Data Parallel Computing")
- Due to the massive data sizes we expect to be simulating we expect data parallelism to be a good way to exploit parallelism on current and next generation architectures

| | | | | | |
|------------------------|---|----|----|----|----|
| input | 4 | 5 | 2 | 1 | 3 |
| ----- | | | | | |
| transform(+1) | 5 | 6 | 3 | 2 | 4 |
| inclusive_scan(+) | 4 | 9 | 11 | 12 | 15 |
| exclusive_scan(+) | 0 | 4 | 9 | 11 | 12 |
| exclusive_scan(max) | 0 | 4 | 5 | 5 | 5 |
| transform_inscan(*2,+) | 8 | 18 | 22 | 24 | 30 |
| for_each(-1) | 3 | 4 | 1 | 0 | 2 |
| sort | 1 | 2 | 3 | 4 | 5 |
| copy_if(n % 2 == 1) | 5 | 1 | 3 | | |
| reduce(+) | | | | | 15 |
| | | | | | |
| input1 | 0 | 0 | 2 | 4 | 8 |
| input2 | 3 | 4 | 1 | 0 | 2 |
| ----- | | | | | |
| upper_bound | 3 | 4 | 2 | 2 | 3 |
| permutation_iterator | 4 | 8 | 0 | 0 | 2 |

Example data-parallel operations

Challenge: Write operators in terms of these primitives only
Reward: Efficient, portable code

NVIDIA's Thrust Library



- Thrust is an open-source C++ template library developed by NVIDIA
- It allows the user to write CUDA programs using an STL-like interface, without having to know CUDA-specific syntax or functions
- In addition to CUDA, it has backends for OpenMP and Intel TBB, and can be extended to support additional backends
- It implements many data-parallel primitives, with user-defined functors
- It provides `thrust::host_vector` and `thrust::device_vector`, simplifying memory management and data transfer between the host and device

```
#include <thrust/transform_reduce.h>
#include <thrust/functional.h>
#include <thrust/device_vector.h>
#include <thrust/host_vector.h>
#include <cmath>

// square<T> computes the square of a number f(x) -> x*x
template <typename T>
struct square
{
    __host__ __device__
    T operator()(const T& x) const {
        return x * x;
    }
};

int main(void)
{
    // initialize host array
    float x[4] = {1.0, 2.0, 3.0, 4.0};

    // transfer to device
    thrust::device_vector<float> d_x(x, x + 4);

    // setup arguments
    square<float> unary_op;
    thrust::plus<float> binary_op;
    float init = 0;

    // compute norm
    float norm = std::sqrt(thrust::transform_reduce(d_x.begin(),
                                                    d_x.end(), unary_op, init, binary_op));

    std::cout << norm << std::endl;

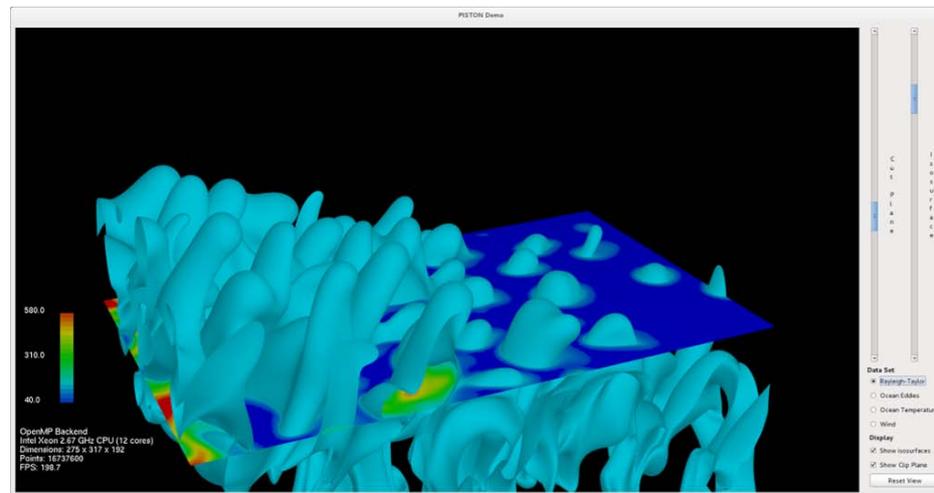
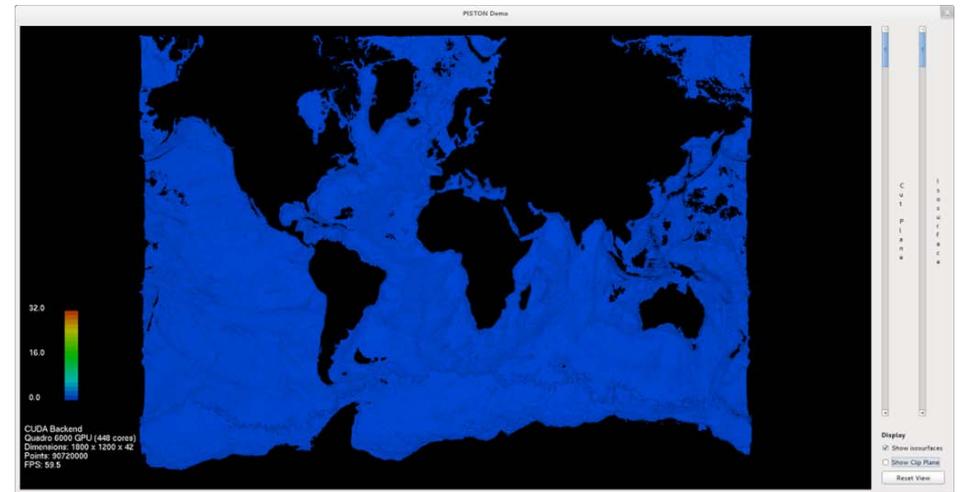
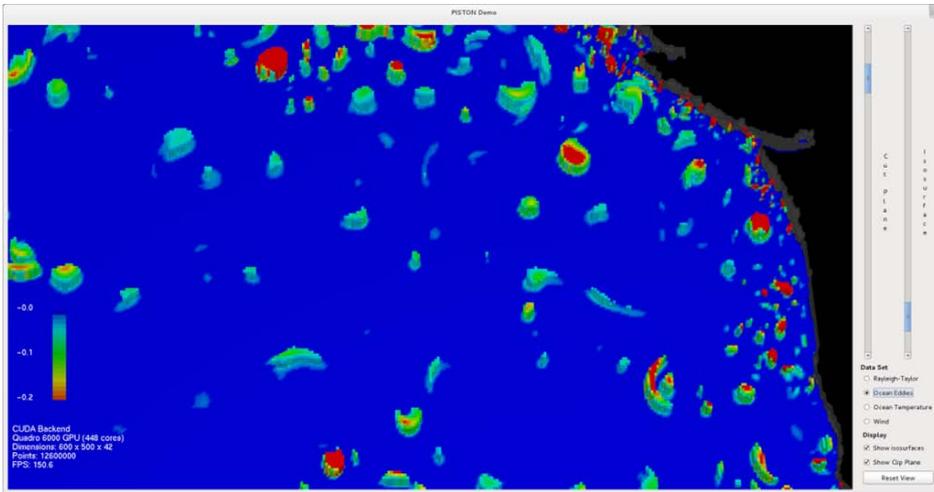
    return 0;
}
```

Sample Thrust code to compute vector norm

How PISTON/PINION Leverage Thrust

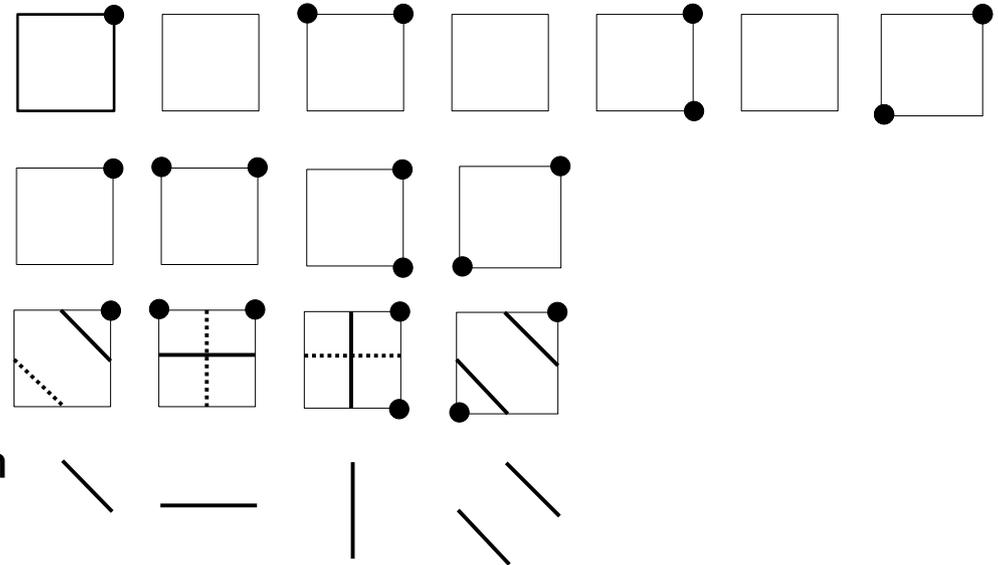
- Thrust provides:
 - An STL-like interface for memory management (host/device vectors) and data-parallel algorithms
 - Backend implementations of the data-parallel algorithms for CUDA, as well as slightly less-developed implementations for OpenMP and TBB
- PISTON/PINION intend to provide:
 - A library of visualization and analysis operators implemented using Thrust
 - A data model simulation meshes (e.g., VTK structured grids, unstructured grids, AMR)
 - Simulation operators (e.g., advection, interface reconstruction, etc.)
- PISTON/PINION intend to enhance:
 - Non-CUDA backends (e.g., OpenCL prototype, optimize OpenMP for Xeon Phi, etc.)
 - Interface to support distributed memory operations

Videos of PISTON in Action



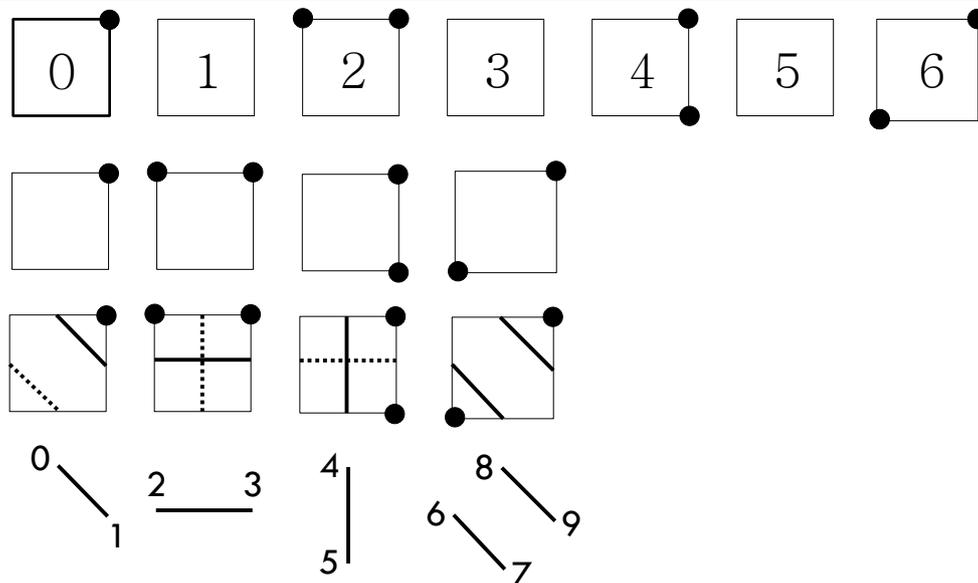
Isosurface with Marching Cube – the Naive Way

- Classify all cells by *transform*
- Use *copy_if* to compact valid cells.
- For each valid cell, generate same number of geometries with flags.
- Use *copy_if* to do stream compaction on vertices.
- This approach is too slow, more than 50% of time was spent moving huge amount of data in global memory.
- Can we avoid calling *copy_if* and eliminate global memory movement?



Isosurface with Marching Cube – Optimization

- Inspired by HistoPyramid
- The filter is essentially a mapping from input cell id to output vertex id
- Is there a “reverse” mapping?
- If there is a reverse mapping, the filter can be very “lazy”
- Given an output vertex id, we *only* apply operations on the cell that would generate the vertex
- Actually for a range of output vertex ids



Isosurface with Marching Cubes Algorithm

1. input

transform(classify_cell)

2. caseNums

3. numVertices

transform_inclusive_scan(is_valid_cell)

4. validCellEnum

5. CountingIterator

upper_bound

6. validCellIndices

make_permutation_iterator

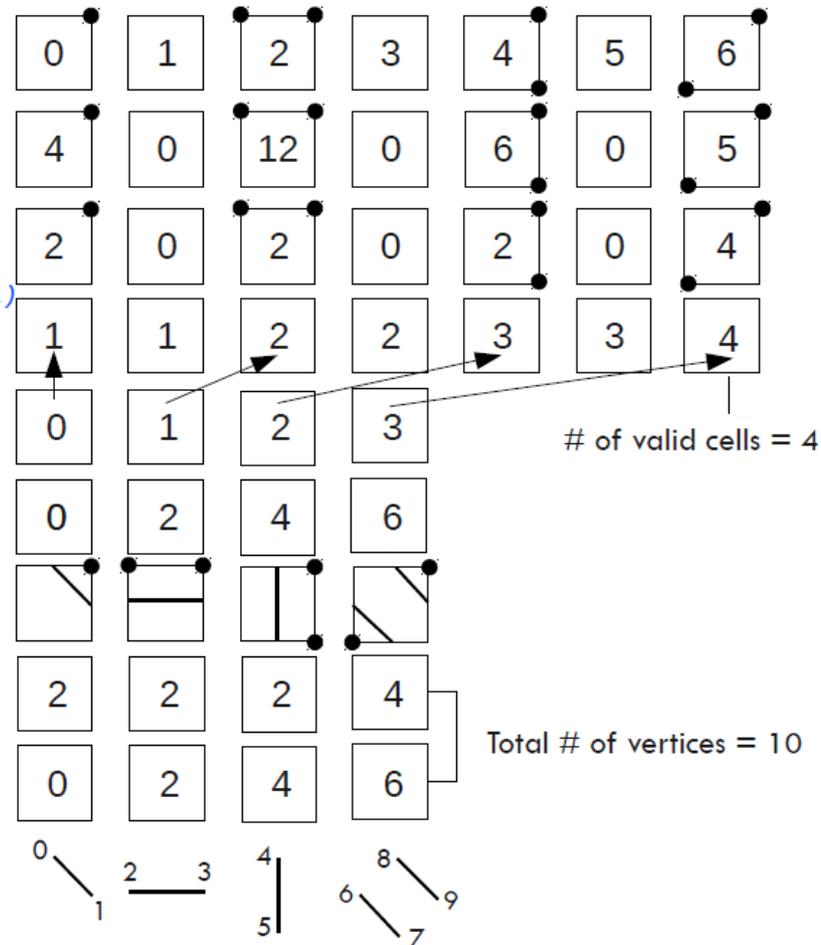
7. numVerticesCompacted

exclusive_scan

8. numVerticesEnum

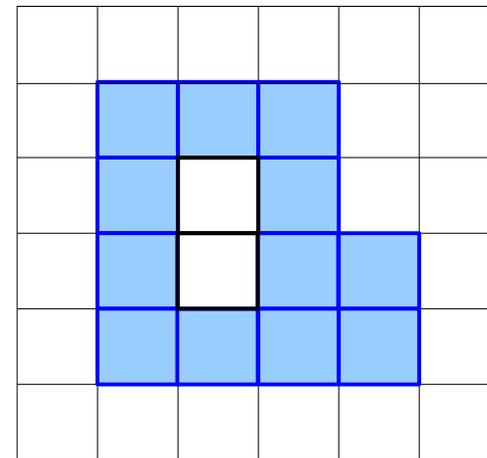
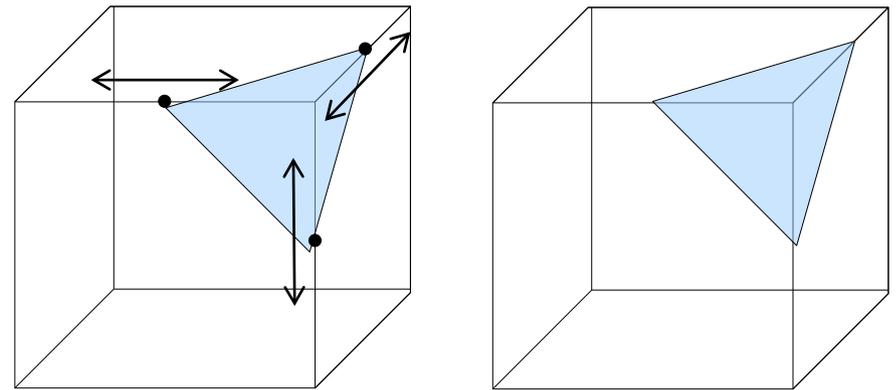
for_each(isosurface_functor)

9. outputVertices



Variations on Isosurface: Cut Surfaces and Threshold

- Cut surface
 - Two scalar fields, one for generating geometry (cut surface) the other for scalar interpolation
 - Less than 10 LOC change, negligible performance impact to isosurface
 - One 1D interpolation per triangle vertex
- Threshold
 - Classify cells, this time based on whether value at each vertex falls within threshold range, then stream compact valid cells and generate geometry for valid cells
 - Additional pass of cell classification and stream compaction to remove interior cells



Additional Operators

Blelloch's "Vector Models for Data-Parallel Computing"

Data Structures

Graphs: Neighbor reducing, distributing excess across edges

Trees: Leafix and rootfix operations, tree manipulations

Multidimensional arrays

Computational Geometry

Generalized binary search

k-D tree

Closest pair

Quickhull

Merge Hull

Graph Algorithms

Minimum spanning tree

Maximum flow

Maximal independent set

Numerical Algorithms

Matrix-vector multiplication

Linear-systems solver

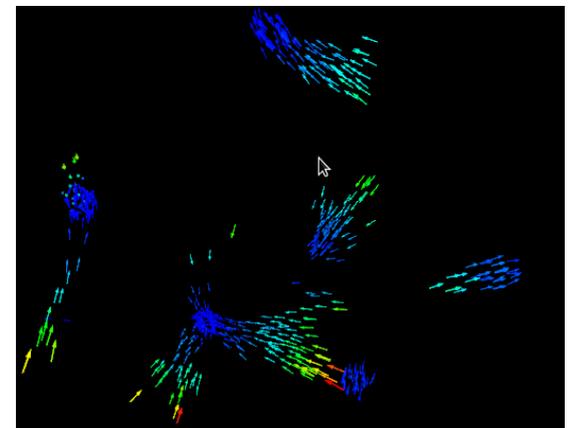
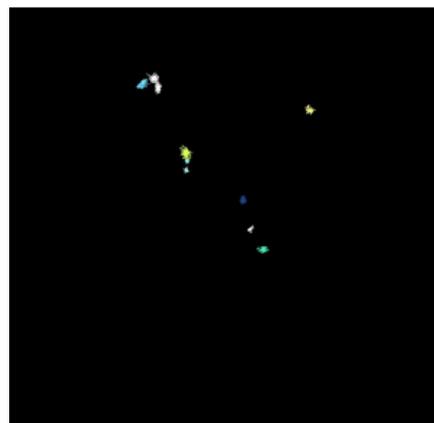
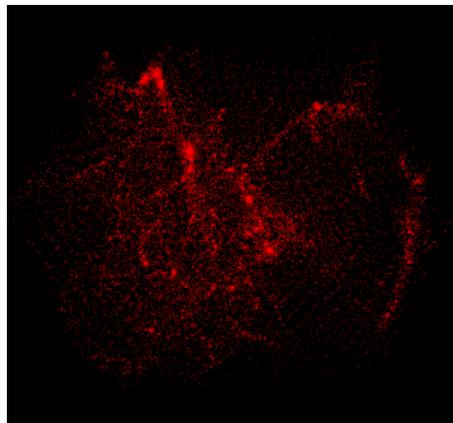
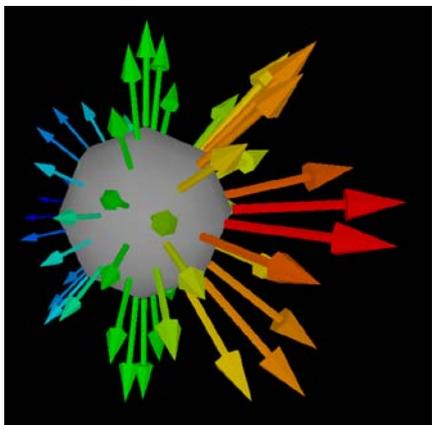
Simplex

Outer product

Sparse-matrix multiplication

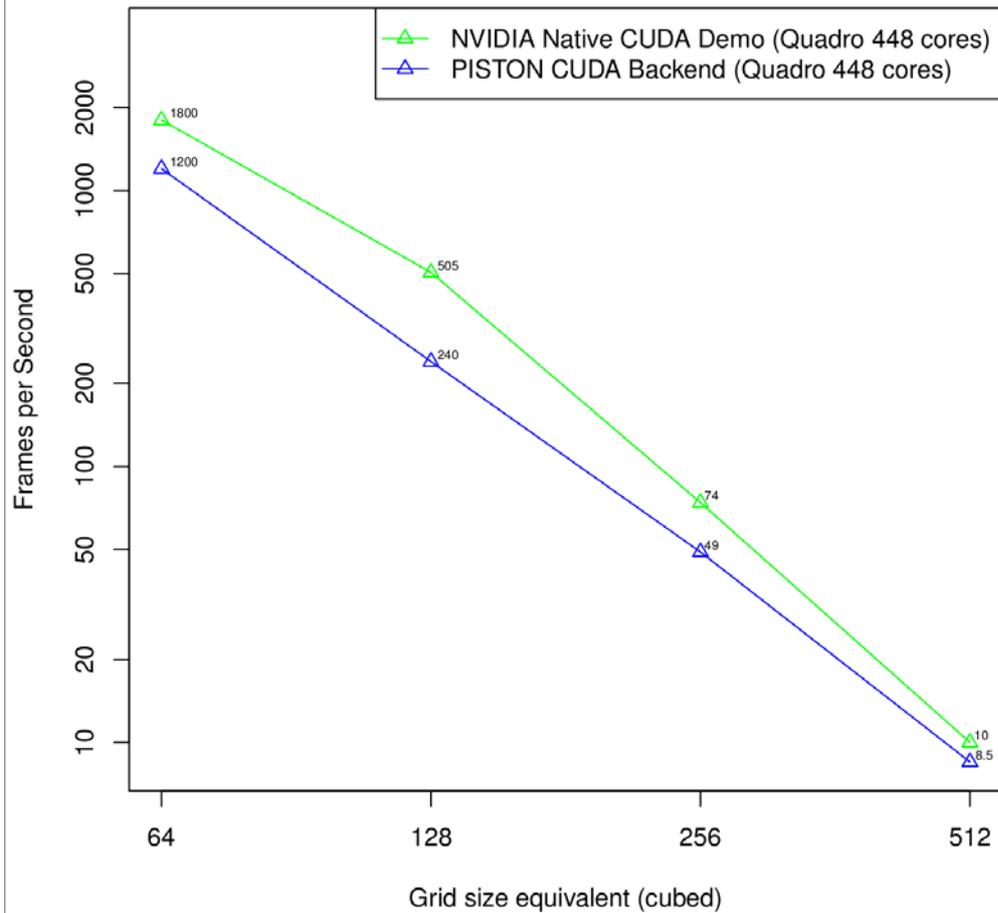
Current prototypes

- Glyphs
- Halo finder for cosmology simulations
- "Boid" simulation (flocking birds)

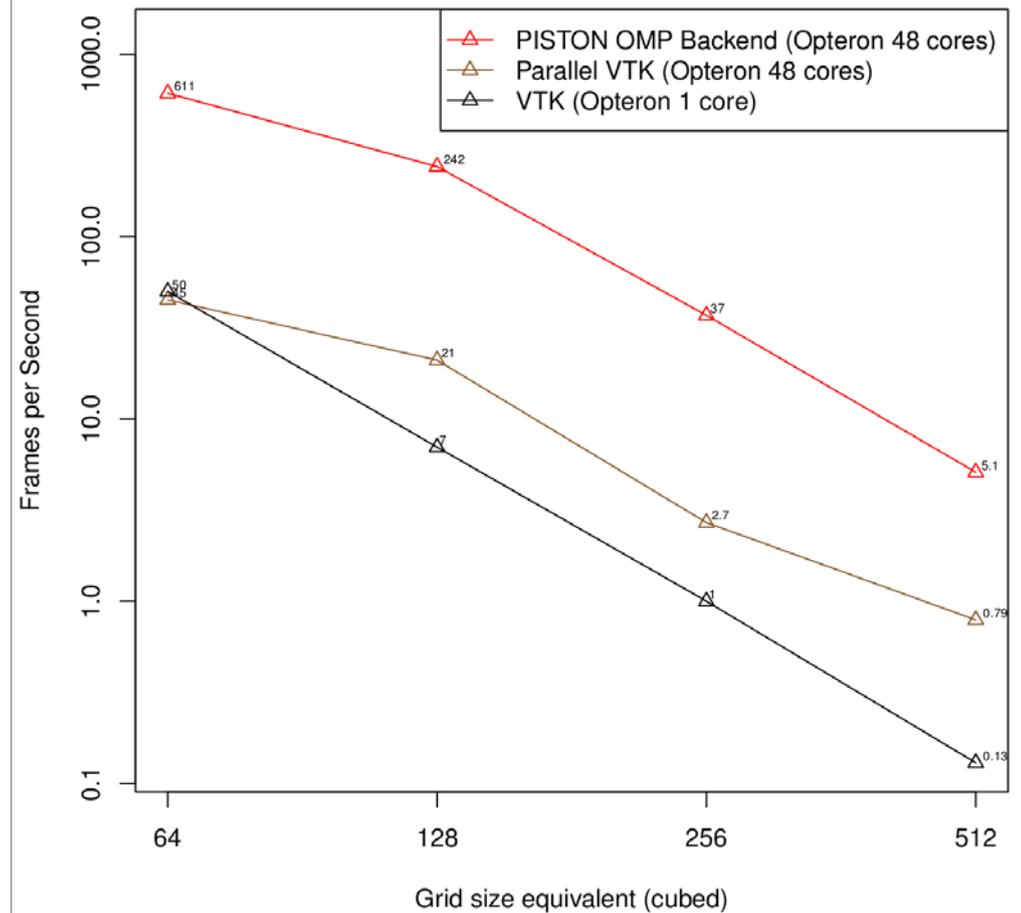


PISTON Performance

3D Isosurface Generation: CUDA Compute Rates

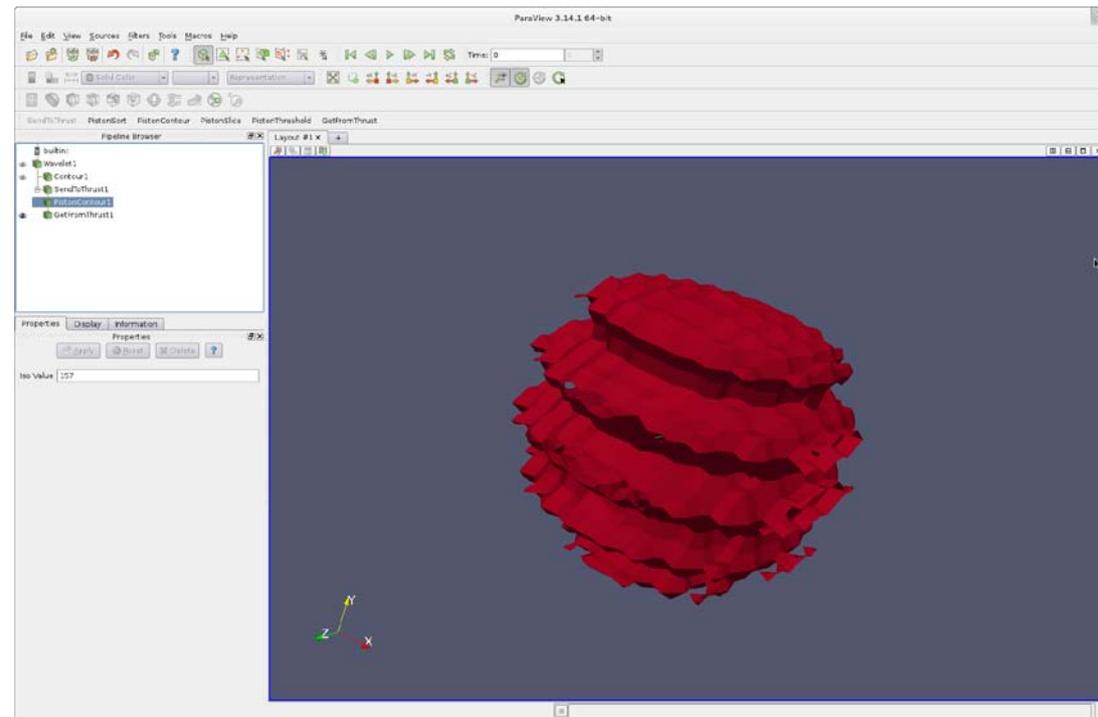
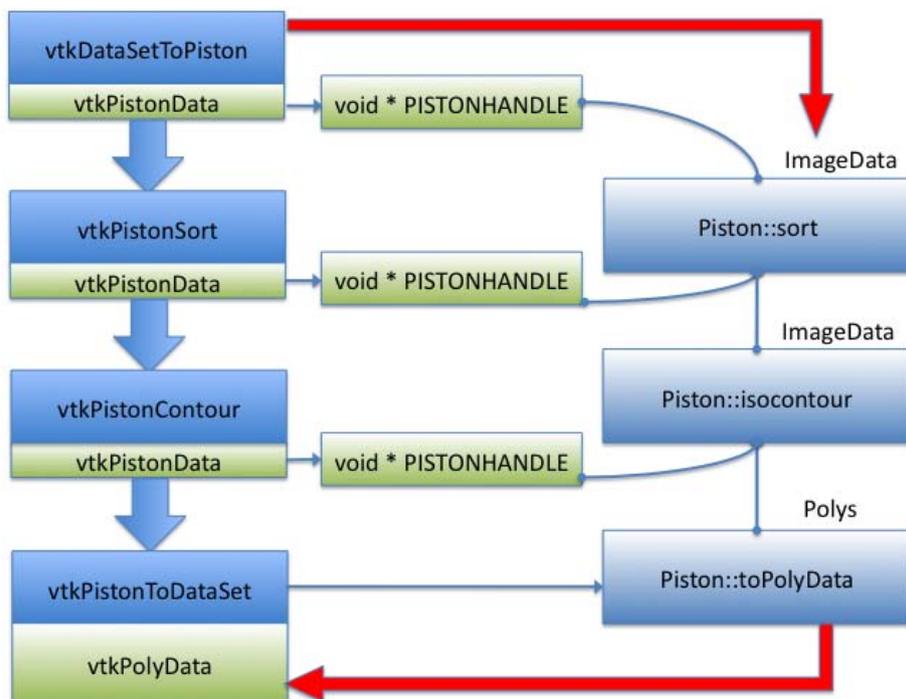


3D Isosurface Generation: CPU Compute Rates



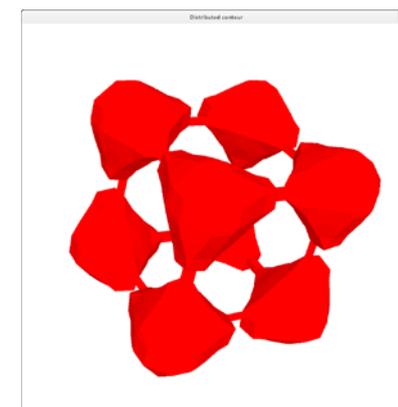
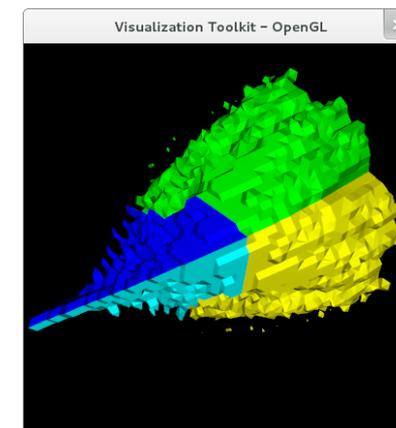
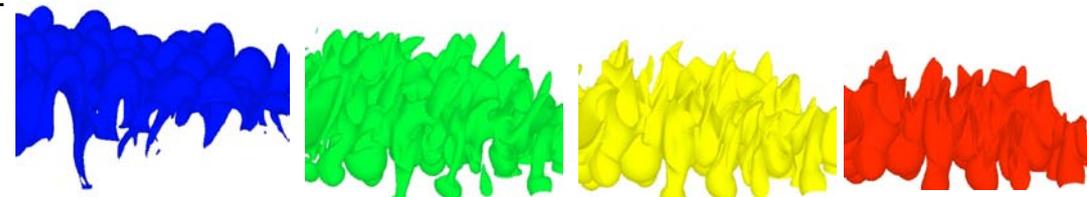
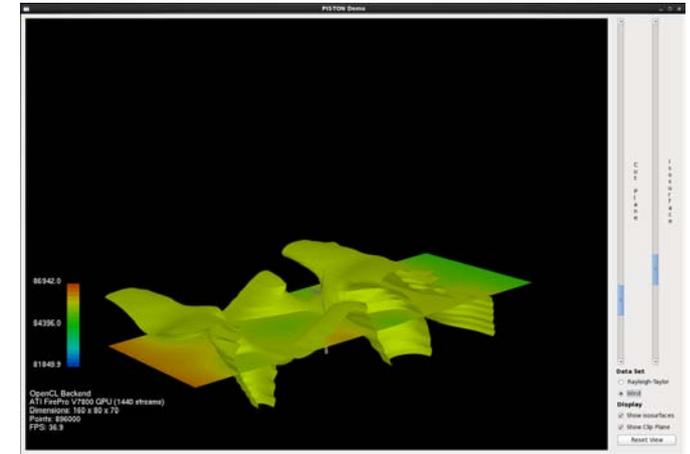
Integration with VTK and ParaView

- Filters that use PISTON data types and algorithms integrated into VTK and ParaView
- Utility filters interconvert between standard VTK data format and PISTON data format (thrust device vectors)
- Supports interop for on-card rendering



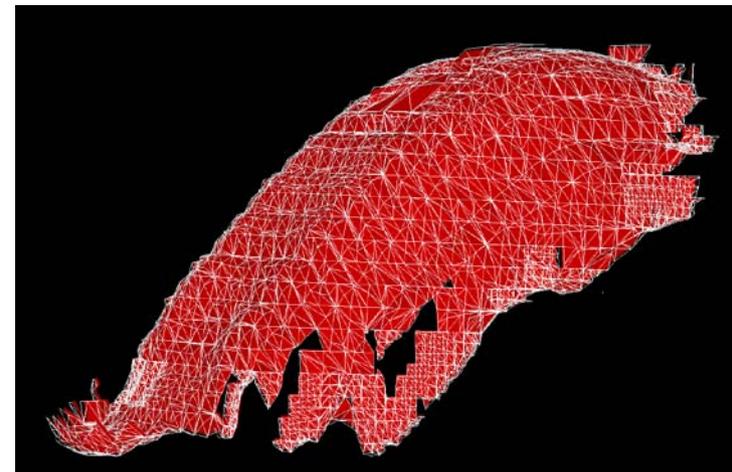
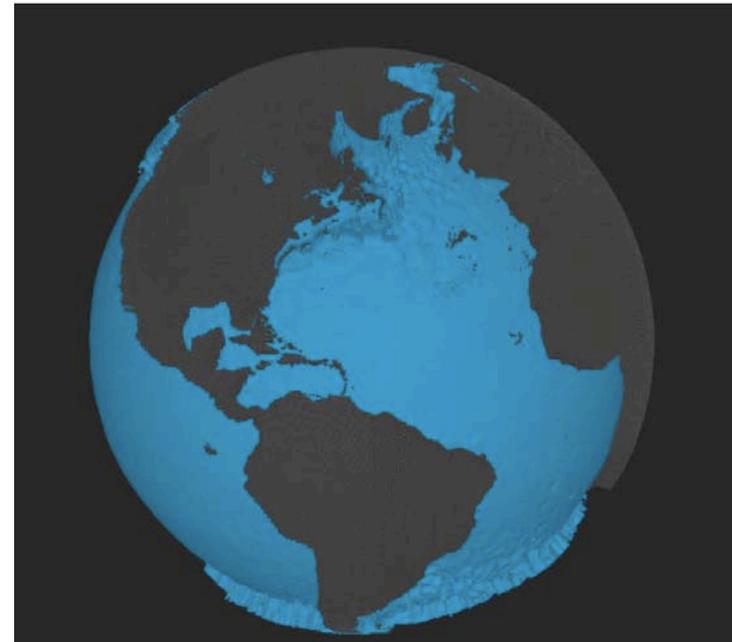
Extending PISTON's Portability: Architectures

- Prototype OpenCL backend
 - Successfully implemented isosurface and cut plane operators in OpenCL with code almost identical to that used for the Thrust-based CUDA and OpenMP backends
 - With interop on AMD FirePro V7800, we can run at about 6 fps for 256^3 data set (2 fps without interop)
- Renderer
 - Allows generation of images on systems without OpenGL
 - Rasterizing and ray-casting versions (using K-D Tree)
- Inter-node parallelism
 - VTK Integration
 - Domain partitioned by VTK's MPI libraries
 - Each node uses PISTON filters to compute results for its portion of domain
 - Results combined by VTK's compositors
 - Distributed implementations of Thrust primitives using MPI (in progress)



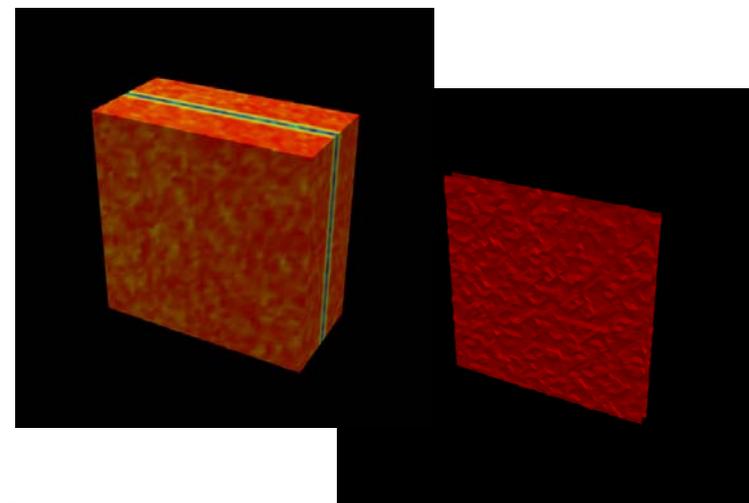
Extending PISTON's Portability: Data Types

- Curvilinear coordinates
 - Multiple layers of coordinate transformations
 - Due to kernel fusion, very little performance impact
- Unstructured / AMR data
 - Tetrahedralize uniform grid or unstructured grid (e.g., AMR mesh)
 - Generate isosurface geometry based on look-up table for tetrahedral cells
 - Next step: Develop PISTON operator to tetrahedralize grids, and/or to compute isosurface directly on AMR grid

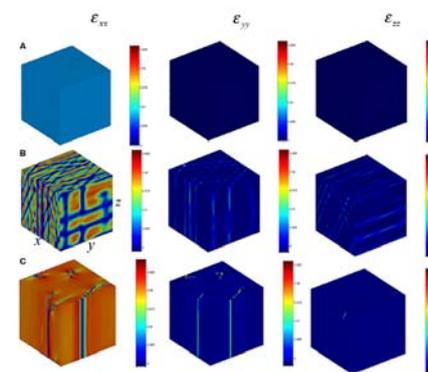


PISTON In-Situ

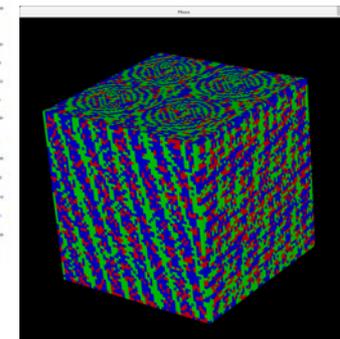
- VPIC (Vector Particle in Cell) Kinetic Plasma Simulation Code
 - Implemented first version of an in-situ adapter based on Paraview CoProcessing Library (Catalyst)
 - Three pipelines: vtkDataSetMapper, vtkContourFilter, vtkPistonContour
- CoGL
 - Stand-alone meso-scale simulation code developed as part of the Exascale Co-Design Center for Materials in Extreme Environments
 - Studies pattern formation in ferroelastic materials using the Ginzburg–Landau approach
 - Models cubic-to-tetragonal transitions under dynamic strain loading
 - Simulation code and in-situ viz implemented using PISTON



Output of vtkDataSetMapper and vtkPistonContour filters on Hhydro charge density at one timestep of VPIC simulation

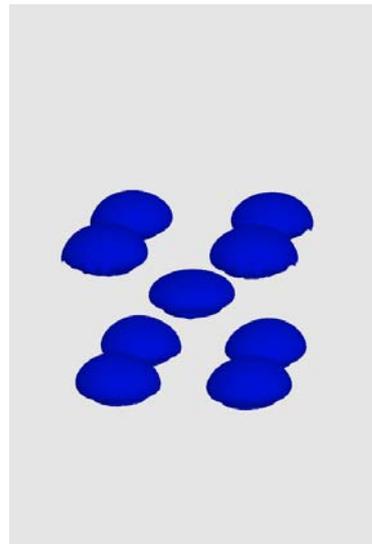
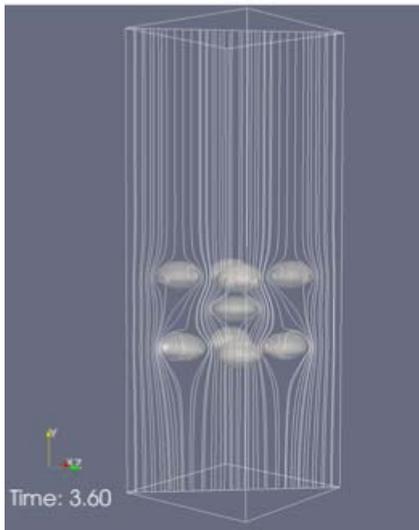


Strains in x,y,z (above); PISTON in-situ visualization (right)



PISTON's New Companion Project: PINION

- A portable, data-parallel software framework for physics simulations
 - Data structures that allow scientists to program in a way that maps easily to the problem domain rather than dealing directly with 1D host/device vectors
 - Operators that provide data-parallel implementations of analysis and computational functions often used in physics simulations
 - Backends that optimize implementations of data parallel primitives for one or two emerging supercomputer hardware architectures

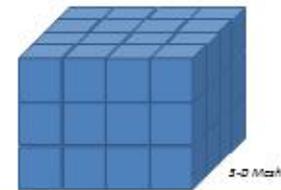


Physics

$$\frac{\partial \mathbf{f}}{\partial t} + \nabla(\bar{\mathbf{u}}\mathbf{f}) = 0 \quad \text{Advection}$$

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f} \quad \text{Navier-Stokes}$$

Physics-Based
Data Model
and Operators



output = interface_reconstruct(input)
output = advect(input) Operators
output = gradient(input)
...

Thrust API for
Data-Parallel
Primitives



transform(inVector, outVector, functor)
scan(inVector, outVector, functor)
value = reduce(inVector, functor)
...

Data-Parallel Primitives

Thrust Backend
Implementations
of Primitives



Hardware
Architectures
and Compilers



PISTON Open-Source Release

- Open-source release
 - Stable tarball: <http://viz.lanl.gov/projects/PISTON.html>
 - Current repository: <https://github.com/losalamos/PISTON>

Acknowledgments

- The SciDAC Institute of Scalable Data Management, Analysis and Visualization (SDAV), funded by the DOE Office of Science through the Office of Advanced Scientific Computing Research.
- NNSA ASC CCSE Program
- The Exascale Co-Design Center for Materials in Extreme Environments, funded by the DOE Office of Advanced Scientific Computing Research (ASCR)
- Los Alamos Laboratory Directed Research and Development Program