The SDAV Software Frameworks for Visualization and Analysis on Next-Generation Multi-core Architectures

Presentation by Chris Sewell, Los Alamos National Laboratory

PISTON
Chris Sewell, Li-ta Lo, James Ahrens

DAX
Ken Moreland

EAVL
Jeremy Meredith

DIY
Tom Peterka

Productization support provided by Kitware
SDAV VTK-m Frameworks

- **Objective:** Enhance existing multi/many-core technologies in anticipation of in situ analysis use cases with LCF codes

- **Benefit to scientists:** These frameworks will make it easier for domain scientists’ simulation codes to take advantage of the parallelism available on a wide range of current and next-generation hardware architectures, especially with regards to visualization and analysis tasks

- **Projects**
  - EAVL, Oak Ridge National Laboratory
  - DAX, Sandia National Laboratory
  - DIY, Argonne National Laboratory
  - PISTON, Los Alamos National Laboratory

- Work on integrating these projects with VTK is on-going, in collaboration with Kitware
EAVL: Extreme-scale Analysis and Visualization Library

- Targets approaching hardware/software ecosystem:
  - Update traditional data model to handle modern simulation codes and a wider range of data.
  - Investigate how an updated data and execution model can achieve the necessary computational, I/O, and memory efficiency.
  - Explore methods for visualization algorithm developers to achieve these efficiency gains and better support exascale architectures.

http://ft.ornl.gov/eavl
https://github.com/jsmeredith/EAVL
An Efficient Data Model in EAVL

- More efficiently support existing data types with more flexible mesh structures
- Better support non-physical and new types of data (high-order, high-dim)
- Algorithms execute faster due to fewer data transformations.
A Traditional Data Set Model

**Data Set**

**Rectilinear**
- Dimensions
- 3D Axis Coordinates
- Cell Fields
- Point Fields

**Structured**
- Dimensions
- 3D Point Coordinates
- Cell Fields
- Point Fields

**Unstructured**
- Connectivity
- 3D Point Coordinates
- Cell Fields
- Point Fields
The EAVL Data Set Model

- Data Set
  - Cells[]
  - Points[]
  - Fields[]

- CellSet
  - Explicit Connectivity
  - Structured Dimensions
  - QuadTree Tree
  - Subset CellList

- Field
  - Name
  - Association
  - Values

- Coords
  - FieldName
  - Component
EAVL Example: Elevating a Structured Grid

- No problem-sized data modifications.
  - Interleaved and separated coordinates can be used simultaneously.

```
<table>
<thead>
<tr>
<th>eavlStructuredCellSet</th>
<th>eavlStructuredCellSet</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegularStructure: 30 40</td>
<td>RegularStructure: 30 40</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>eavlCoordinates</th>
<th>eavlCoordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldName: “c” “c”</td>
<td>FieldName: “c” “c” “val”</td>
</tr>
<tr>
<td>Component: 0 1</td>
<td>Component: 0 1 0</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>eavlField#0</th>
<th>eavlField#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: “c”</td>
<td>Name: “val”</td>
</tr>
<tr>
<td>Association: Points</td>
<td>Association: Points</td>
</tr>
<tr>
<td>Value[2*npts]</td>
<td>Values[npts]</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>eavlField#0</th>
<th>eavlField#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: “c”</td>
<td>Name: “val”</td>
</tr>
<tr>
<td>Association: Points</td>
<td>Association: Points</td>
</tr>
<tr>
<td>Value[2*npts]</td>
<td>Values[npts]</td>
</tr>
</tbody>
</table>
```
Productive Algorithm Development in EAVL

- Topological iterators encapsulate data-parallel patterns
- Functors provide optimized execution on CPU and GPU
- Transparent heterogeneous memory space support

```cpp
struct PolyNormalFunctor {
    void operator()(float *x, float *y, float *z, float *n) {
        // get two adjacent edge vectors
        float ax = x[1]-x[0], ay = y[1]-y[0], az = z[1]-z[0];
        float bx = x[2]-x[1], ay = y[2]-y[1], az = z[2]-z[1];
        // calculate their cross product
        n[0] = ay*bz - az*by;  n[1] = az*bx - ax*bz;   n[2] = ax*by - ay*bx;
    }
};

void FaceNormalFilter::Execute(...) {
    executor->AddOperation(new NodeToCellOp3(xcoord, ycoord, zcoord, outputnormals, inputcells, PolyNormalFunctor()));
}
```
Dax: A Toolkit for Analysis and Visualization at Extreme Scale

The primitives necessary to design finely-threaded algorithms

- “Worklets” ease design in serial, scheduled in parallel
- Basic visualization design objects (think VTK for many-core)
- Communicative operations provide neighborhood-wide operations without exposing read/write hazards

http://daxtoolkit.org

Contour with subsequent vertex welding, coarsening, subdivision, and curvature estimation

Extracted cells of large gradient and compacted points

Streamlines (preliminary work)
struct Normal: dax::exec::WorkletMapField
{

typedef void ControlSignature(Field(In),Field(Out));

typedef _2 ExecutionSignature(_1);

template<typename T>
T operator()(const T& coord) const
{
    dax::Scalar dot = dax::dot(coord,coord);
    return coord * dax::math::RSqrt(dot);
}
};
Example Dax Control Code

```cpp
int main()
{
    using namespace dax::cont;

    std::vector<dax::Vector3> coords(10);
    for(int i=0; i < 10; i++)
    {
        const dax::Scalar x(1.0f + i);
        coords[i] = dax::Vector3(dax::math::Sin(x)/i+1,
                                  1/(x*x),
                                  0);
    }

    //make a dax array handle to the coordinates
    ArrayHandle<dax::Vector3> coordHandle = make_ArrayHandle(coords);

    //make a dax array handle to store the results
    ArrayHandle<dax::Vector3> normals;

    Schedule<> scheduler;
    //note two parameters passed to scheduler like the control
    //signature requests
    scheduler(Normal(), coordHandle, normals);

    std::vector<dax::Vector3> results(normals.GetNumberOfValues());
    normals.CopyTo(results.begin());
}
```
DIY (Do-It-Yourself): Overview

Main Ideas and Objectives
- Large-scale parallel analysis (visual and numerical) on HPC machines
- For scientists, visualization researchers, tool builders
- In situ, coprocessing, postprocessing
- Data-parallel problem decomposition
- 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++
- C bindings, can be called from Fortran, C, C++
- Autoconf build system
- Lightweight: libdiv.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
DIY: Applications

- Particle tracing of thermal hydraulics flow
- Information entropy analysis of astrophysics
- Morse-Smale complex of combustion
- Voronoi tessellation of cosmology
PISTON: A Portable Data-Parallel Visualization and Analysis Framework

- 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 and Background

- Current production visualization software does not take full advantage of acceleration hardware and/or multi-core architecture
- Research on accelerating visualization operations are 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
- Data parallelism: independent processors performs the same task on different pieces of data (see Blelloch, “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
- Thrust is a NVidia C++ template library for CUDA. It can also target other backends such as OpenMP, and allows you to program using an interface similar the C++ Standard Template Library (STL)
Videos of PISTON in Action
Brief Introduction to Data-Parallel Programming and Thrust

What algorithms does Thrust provide?

- Sorts
- Transforms
- Reductions
- Scans
- Binary searches
- Stream compactions
- Scatters / gathers

Challenge: Write operators in terms of these primitives only

Reward: Efficient, portable code
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
7. numVerticesCompacted
   exclusive_scan
8. numVerticesEnum
   for_each(isosurface_functor)
9. outputVertices

# of valid cells = 4
Total # of vertices = 10
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: Leaffix 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
PISTON Open-Source Release

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

- [link](http://sdav-scidac.org/)

- Panel at SC12: “Visualization Frameworks for Multi-Core and Many-Core Architectures” Hank Childs, Jeremy Meredith, Patrick McCormick, Christopher Sewell, Kenneth Moreland
  - Wednesday, November 14, 3:30 – 5:00, 355-BC

- The SciDAC Institute of Scalable Data Management, Analysis and Visualization (SDAV) is funded by the DOE Office of Science through the Office of Advanced Scientific Computing Research.
  - SciDAC Institute Director: Arie Shoshani
  - Visualization Project Chairs: James Ahrens, Wes Bethel

- Related PISTON projects also funded by ASC Program, ASCR, LANL LDRD