Title: The PISTON Software Framework for Visualization and Analysis on Next-Generation Multi-core Architectures

Author(s): Christopher Sewell
Li-ta Lo
James Ahrens

Intended for: Visualization Frameworks for Multi-Core and Many-Core Architectures Panel, The International Conference for High Performance Computing, Networking, Storage, and Analysis, November 14, 2012
The PISTON Software Framework for Visualization and Analysis on Next-Generation Multi-core Architectures

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 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).
Questions for the Panel

● What fundamental problem are you trying to solve?
  ● Portable performance for visualization and analysis operators

● What are your plans to deal with exascale-specific issues (massive concurrency, distributed memory, memory overhead, fault tolerance)?
  ● Data-parallel programming model; requires architecture-specific optimizations for a limited set of “embarrassingly parallel” primitive operators

● What is your philosophy for dealing with ambiguity of the exascale architecture (multiple swim lanes, heterogeneous architectures)?
  ● Data-parallel programming model; requires architecture-specific optimizations for a limited set of “embarrassingly parallel” primitive operators

● How is your technology implemented?
  ● Extension of NVIDIA’s Thrust library

● What is the long-term result for this effort? (Production software? Research prototype?)
  ● Open-source repository (available on-line now); integrated into VTK/ParaView (available on-line now)
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

<table>
<thead>
<tr>
<th>Input</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>transform(+1)</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>inclusive_scan(+)</td>
<td>4</td>
<td>9</td>
<td>11</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>exclusive_scan(+)</td>
<td>0</td>
<td>4</td>
<td>9</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>exclusive_scan(max)</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>transform_inscan(*2,+</td>
<td>8</td>
<td>18</td>
<td>22</td>
<td>24</td>
<td>30</td>
</tr>
<tr>
<td>for_each(-1)</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>sort</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>copy_if(n % 2 == 1)</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reduce(+)</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input1</th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

| upper_bound | 3 | 4 | 2 | 2 | 3 |
| permutation_iterator | 4 | 8 | 0 | 0 | 2 |
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

# 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

![Graph showing CUDA compute rates with grid size equivalent (cubed) on the x-axis and frames per second on the y-axis. The graph compares NVIDIA Native CUDA Demo (Quadro 448 cores) and PISTON CUDA Backend (Quadro 448 cores).]

3D Isosurface Generation: CPU Compute Rates

![Graph showing CPU compute rates with grid size equivalent (cubed) on the x-axis and frames per second on the y-axis. The graph compares PISTON OMP Backend (Opteron 48 cores), Parallel VTK (Opteron 48 cores), and VTK (Opteron 1 core).]
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

Developed with Dave DeMarle and Utkarsh Ayachit at Kitware
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
  - Current repository: [https://github.com/losalamos/PISTON](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