• A single place for the visualization community to collaborate, contribute, and leverage massively threaded algorithms.
• Reduce the challenges of writing highly concurrent algorithms by using data parallel algorithms
• Make it easier for simulation codes to take advantage of these parallel visualization and analysis tasks on a wide range of current and next-generation hardware.
VTK-m Architecture

- Data Parallel Algorithms
- Arrays
- Filters
- Control DataModel
- Execution
- Worklets
- Execution DataModel
- Control
VTK-m Filters

- Cell Average
- Clean Grid
- Clip by Field or Implicit Function
- External Faces
- Extract Geometry, Points, Structured
- Gradient
  - Divergence, Vorticity, and Qcriterion
  - Fast paths for ‘StructuredGrids’, and ‘ImageData’
- Histogram
- Marching Cubes
  - Hex and Voxel only
VTK-m Filters

- Mask Points
- Point Average
- Point Elevation
- Surface Normals
  - Faceted
  - Smooth
- Tetrahedralize
- Threshold
- Triangulate
- Vertex Clustering
  - Surface Simplification
VTK-m Execution Performance

- Portability must come with good performance to be advantageous
- VTK-m looks at performance from two perspectives
  - Algorithmic performance
  - Data parallel primitive performance
Gradient Performance

- To improve the performance of gradients on uniform grids VTK-m has added the point neighborhood worklet type
  - Allows for execution on a per point location with neighborhood

![Graph showing performance comparison](image)

- Data Size
  - 512^3
  - 256^3

- VTK-m Before
- VTK-m After

2x Intel(R) Xeon(R) CPU E5-2620 v3
Level of Detail Performance

- Level Of Detail is an algorithm that ParaView uses to facilitate interactivity with high triangle count models
Level of Detail Performance

• VTK-m is being used to develop a Level of Detail filter that will support higher resolution data
• For this to occur we had to make sure the VTK-m parallel primitives scaled well
• For Level of Detail this meant improving the performance of:
  – Copy and CopyIf
  – Reduce and ReduceByKey
  – Unique and UniqueByKey
Level of Detail Performance

Copy on 128.00 MiB of values (Longer is better)

vtkm::Vec< vtkm::Float32, 3 >

vtkm::Int64

0 5 10 15 20

Copy Speed in GiB/s

VTK-m Before
VTK-m After

2x Intel(R) Xeon(R) CPU E5-2620 v3
Level of Detail Performance

Reduce By Key on 2MiB of values with 25% unique

<table>
<thead>
<tr>
<th>Data Type</th>
<th>VTK-m Before</th>
<th>VTK-m After</th>
</tr>
</thead>
<tbody>
<tr>
<td>vtkm::Vec&lt;vtkm::Float32, 3&gt;</td>
<td>0.0015</td>
<td>0.0005</td>
</tr>
<tr>
<td>vtkm::Int64</td>
<td>0.0005</td>
<td>0.0005</td>
</tr>
</tbody>
</table>

2x Intel(R) Xeon(R) CPU E5-2620 v3
### Level of Detail Performance

Unique on 2MiB of values with 5% unique

- **vtkm::Vec< vtkm::Float32, 3 >**
  - 0.0004 seconds before
  - 0.0002 seconds after

- **vtkm::Int64**
  - 0.0006 seconds before
  - 0.0004 seconds after

2x Intel(R) Xeon(R) CPU E5-2620 v3
Level of Detail Performance

512x512x512
InPts: 14027872
InTri: 28055742

VTK: 9.58s
NumPts: 1003638
NumTri: 2011232

VTK-m: 1.85s
NumPts: 1003653
NumTri: 2044767

VTK-m: 1.37s
NumPts: 1003653
NumTri: 2044767

2x Intel(R) Xeon(R) CPU E5-2620 v3
On Going Work

• Implementing a TBB radix sort for int/float/long/double data types for better performance and scaling

• Redesign of TBB worklet execution to promote compiler vectorization.
  – Automatic transformation of aos to soa
  – Automatic block reads and writes
Acknowledgments

This research was supported by the Exascale Computing Project (http://www.exascaleproject.org), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.

Project Number: 17-SC-20-SC