Moving HPC Scientific Visualization Forward

Robert Maynard
Principal Engineer, Kitware
● 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.
Filters

- Cell Average
- Clean Grid
- **Clip by Field or Implicit Function**
- Contour Trees for Uniform Grids
- External Faces
- Extract Geometry, Points, Structured
- **Gradient**
- Histogram and Entropy
- Marching Cubes
  - Hex and Voxel only
- Mask Points
- Point Average
- Point Elevation
- **Probe**
- Streamlines
- **Surface Normals**
  - Faceted
  - Smooth
- Surface Simplification
- Tetrahedralize
- Threshold
- Triangulate
VTK-m now contains point and cell locators

- Optimized Point Locator for Uniform Grids
- General Purpose KD-Tree Point Locator
- Cell Locator implemented using two level Uniform Grid

Has enabled us to write filters such as streamlines and probe
CUDA: NVIDIA GP100 TBB: 2x Intel Xeon CPU E5-2620 v3 [24 cores]
Probe Performance

Probing input disk.ex2 [7472 cells]

- **VTK TBB**: 0.0455 seconds
- **VTK-m TBB**: 0.0136 seconds
- **VTK-m CUDA**: 0.0098 seconds

CUDA: NVIDIA GP100 TBB: 2x Intel Xeon CPU E5-2620 v3 [24 cores]
VTK-m now contains a Gradient Filter
- Supports all linear 3D cell types
- Supports Divergence, Vorticity, and QCriterion

CUDA: NVIDIA GP100 TBB: 2x Intel Xeon CPU E5-2620 v3 [24 cores]
To improve the performance of gradients on image and structured grids VTK-m has added a point neighborhood worklet type.
Gradient Performance

Point Gradient on Image Data

<table>
<thead>
<tr>
<th>Size</th>
<th>VTK TBB</th>
<th>VTK-m TBB</th>
<th>VTK-m CUDA</th>
<th>VTK-m CUDA (kernels only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512^3</td>
<td>0.2937</td>
<td>0.2153</td>
<td>0.53</td>
<td>0.0113</td>
</tr>
<tr>
<td>768^3</td>
<td>0.5087</td>
<td>0.7646</td>
<td>1.9873</td>
<td>0.0354</td>
</tr>
</tbody>
</table>

CUDA: NVIDIA GP100 TBB: 2x Intel Xeon CPU E5-2620 v3 [24 cores]
VTK-m Color Table is aimed to support the common use cases of ParaView and VisIt.

- RGB, HSV, LAB, Diverging Color Spaces
- Independent Opacity controls
- Supports sampling through a lookup table
Coloring 12.7M Points using 'Inferno' [LAB space]

CUDA: NVIDIA GP100 TBB: 2x Intel Xeon CPU E5-2620 v3 [24 cores]
Virtual Methods

VTK-m has identified a need to have certain execution objects leverage virtual methods. Things such as color space, implicit functions and coordinate systems now use virtuals.

The VTK-m DeviceAdapter offers a per-device way to move objects with virtuals to execution space efficiently.
Virtual Methods

Implicit Function over 4M values

- Function Pointer
- Virtuals

```
template <typename DerivedType, typename DeviceTag>
void move_to_device(DerivedType* host_side, DerivedType* & exec_side, DeviceTag)
{
    // Allocate a buffer on the device
    DerivedType* state; cudaMalloc(&state, sizeof(DerivedType));
    // Copy host side members state
    cudaMemcpy(state, host_side, sizeof(DerivedType), cudaMemcpyHostToDevice);
    // Allocate memory for the object
    cudaMalloc(&exec_side, sizeof(DerivedType));
    // Initialize the device object using placement new
detail::PlacementNewKernel<<<1, 1>>>(exec_side, state);
    // Clean up intermediate copy
    cudaFree(state);
```

CUDA: NVIDIA GP100 TBB: 2x Intel Xeon CPU E5-2620 v3 [24 cores]
VTK-m just gained the concept of a MultiBlock container. VTK-m MultiBlock != VTK MultiBlock

- VTK-m MultiBlock entries can only be DataSets, no support for nested MultiBlocks
- In VTK-m a MultiBlock can span multiple nodes (MPI/DIY), but a block must be fully contained on a single node
CUDA Streams

When VTK-m executes using the CUDA device adapter all kernels and memory transfers now use per-thread default streams explicitly.

This work was designed not only for better in-situ integration, but to allow VTK-m the option of doing coarse grained block level parallelism with MultiBlock.
VTK-m ArrayHandle now properly handles users passing CUDA allocated pointers for input data.

- No extra data transfers or copies
- If UVM allocated can also be used with other devices

When VTK-m executes on Pascal+ hardware all device memory will be allocated using UVM.

- Includes hints to the UVM system if the memory is read, write, or r+w
- If the ArrayHandle doesn’t have host data, will use the UVM memory
Thank You!

Robert Maynard

robert.maynard@kitware.com

@robertjmaynard

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

Checkout out VTK-m @ gitlab.kitware.com/vtk/vtk-m and Kitware @ www.kitware.com

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!