Dax Toolkit:
A Proposed Framework for Data Analysis and Visualization at Extreme Scale

Kenneth Moreland  Sandia National Laboratories

Utkarsh Ayachit  Kitware, Inc.

Berk Geveci  Kitware, Inc.

Kwan-Liu Ma  University of California at Davis

October 24, 2011

This work was supported in full by the DOE Office of Science, Advanced Scientific Computing Research, under award number 10-014707, program manager Lucy Nowell.
Dax Toolkit

A new visualization framework designed to exhibit the pervasive parallelism necessary for exascale machines.
Visualization Pipeline
Parallel Visualization Pipeline
## Petascale To Exascale

<table>
<thead>
<tr>
<th></th>
<th>Jaguar – XT5</th>
<th>Exascale</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>224,256</td>
<td>100 million – 1 billion</td>
<td>$\sim 1,000 \times$</td>
</tr>
<tr>
<td>Threads</td>
<td>224,256 way</td>
<td>1 – 10 billion way</td>
<td>$\sim 50,000 \times$</td>
</tr>
<tr>
<td>Memory</td>
<td>300 Terabytes</td>
<td>10 – 128 Petabytes</td>
<td>$\sim 500 \times$</td>
</tr>
</tbody>
</table>

Estimates consolidated from International Exascale Software Project Roadmap and the DOE Exascale Initiative Roadmap.
**MPI-Only Approach?**

<table>
<thead>
<tr>
<th></th>
<th>Jaguar – XT5</th>
<th>Exascale</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>224,256</td>
<td>100 million – 1 billion</td>
<td>~1,000×</td>
</tr>
<tr>
<td>Threads</td>
<td>224,256 way</td>
<td>1 – 10 billion way</td>
<td>~50,000×</td>
</tr>
<tr>
<td>Memory</td>
<td>300 Terabytes</td>
<td>10 – 128 Petabytes</td>
<td>~500×</td>
</tr>
</tbody>
</table>

Vis object code + state : 20 MB
On Jaguar : 20 MB × 200,000 processes = 4 TB
On Exascale: 20 MB × 10,000,000,000 processes = 200 PB!
<table>
<thead>
<tr>
<th></th>
<th>Jaguar – XT5</th>
<th>Exascale</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>224,256</td>
<td>100 million – 1 billion</td>
<td>~1,000×</td>
</tr>
<tr>
<td>Threads</td>
<td>224,256 way</td>
<td>1 – 10 billion way</td>
<td>~50,000×</td>
</tr>
<tr>
<td>Memory</td>
<td>300 Terabytes</td>
<td>10 – 128 Petabytes</td>
<td>~500×</td>
</tr>
</tbody>
</table>

On Jaguar: 1 trillion cells $\rightarrow$ 5 million cells/thread
On Exascale: 500 trillion cells $\rightarrow$ 50K cells/thread
Revisiting the Filter

- Lightweight Object
- Serial Execution
- No explicit partitioning
- No access to larger structures
- No state
function \((in, out)\)
Worklet

function (in, out)
Existing Approaches

Multicore extensions to VTK pipeline [Vo, et al. 2010]
- Pros: Can be applied to most existing VTK filters.
- Cons: High overhead for each execution thread; VTK algorithms optimized for sizeable chunks.

Functional field definitions (FEL/FM) [Bryson, et al. 1996]
- Pros: Mesh flexibility; low memory overhead; lazy evaluation; straightforward to parallelize.
- Cons: Does not manage massive multi-threading; no mechanism for topology generation.

MapReduce [Dean and Ghemawat 2008] [Vo, et al. 2011]
- Pros: simple programming model for massive parallelism; custom systems specializing in large amounts of data.
- Cons: Difficult to cast visualization algorithms; global shuffling operation inefficient because it ignores known neighborhood or domain decompositions.
Dax Toolkit
Data Model

- dax::exec::Work*

Corresponds to work performed by each Worklet.

- dax::exec::WorkMapField
- dax::exec::WorkMapCell

- dax::exec::Field

Provides access to data arrays.

- dax::exec::FieldCell
- dax::exec::FieldPoint
- dax::exec::FieldCoordinates
DAX_WORKLET void FieldWorklet(
    DAX_IN dax::exec::WorkMapField & work,
    DAX_IN dax::exec::Field & in_field,
    DAX_OUT dax::exec::Field & out_field)
{
    dax::Scalar in_value = in_field.GetScalar(work);
    dax::Scalar out_value = ...;
    out_field.Set(work, out_value);
}
### Code Comparison

**VTK Cell Derivatives**
```cpp
int vtkCellDerivatives::RequestData(...) {
    ...
    for (cellId=0; cellId < numCells; cellId++) {
        ...
        input->GetCell(cellId, cell);
        subId = cell->GetParametricCenter(pcoords);
        inScalars->GetTuples(
            cell->PointIds, cellScalars);
        scalars = cellScalars->GetPointer(0);
        cell->Derivatives(
            subId, pcoords, scalars, 1, derivs);
        outGradients->SetTuple(cellId, derivs);
    }
    ...
}
```

**DAX Worklet**
```cpp
DAX_WORKLET void CellGradient(...) {
    
    dax::exec::Cell cell(work);
    dax::Vector3 parametric_cell_center = dax::make_Vector3(0.5, 0.5, 0.5);
    dax::Vector3 value = cell.Derivative(
        parametric_cell_center, points, point_attribute, 0);
    cell_attribute.Set(work, value);
}
```
Results
Implementation Assumptions

- GPU $\approx$ Exascale Node

- CUDA $\approx$ Development Environment on Exascale Node
### Performance Comparison

<table>
<thead>
<tr>
<th>Mesh Size</th>
<th>VTK Time</th>
<th>Dax Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elevation → Gradient</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$144^3$</td>
<td>2.75 s</td>
<td>0.013 (0.024) s</td>
<td>210 (114)</td>
</tr>
<tr>
<td>$256^3$</td>
<td>15.52 s</td>
<td>0.074 (0.135) s</td>
<td>210 (115)</td>
</tr>
<tr>
<td>$512^3$</td>
<td>125.75 s</td>
<td>0.589 (1.076) s</td>
<td>213 (117)</td>
</tr>
<tr>
<td>Elevation → Sine → Square → Cosine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$144^3$</td>
<td>2.32 s</td>
<td>0.002 (0.006) s</td>
<td>1169 (386)</td>
</tr>
<tr>
<td>$256^3$</td>
<td>12.99 s</td>
<td>0.013 (0.034) s</td>
<td>999 (382)</td>
</tr>
<tr>
<td>$512^3$</td>
<td>103.88 s</td>
<td>0.110 (0.276) s</td>
<td>944 (376)</td>
</tr>
</tbody>
</table>

Performance comparison between Dax toolkit and VTK. Values in parentheses show the corresponding values with data transfer times included.
Challenges and Ongoing Work

- Topology modifying Worklets e.g. Marching Cubes/Streamlines

- I/O and Rendering
Conclusion

Traditional Pipeline

Dax Pipeline

Executive

Worklet 1

Worklet 2

Filter 1

foreach element

Filter 2

foreach element

Dax Pipeline

Filter 1

foreach element

Filter 2

foreach element

Executive

foreach element

Worklet 1

Worklet 2
This work was supported in full by the DOE Office of Science, Advanced Scientific Computing Research, under award number 10-014707, program manager Lucy Nowell.