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

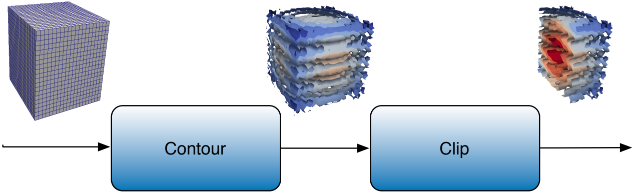
October 24, 2011

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.

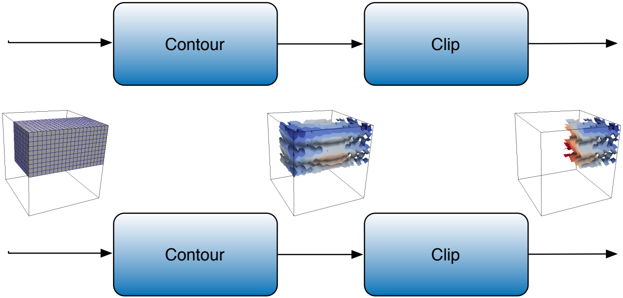
Dax Toolkit

A new visualization framework designed to exhibit the pervasive parallelism necessary for exascale machines.

Visualization Pipeline



Parallel Visualization Pipeline



Petascale To Exascale

	Jaguar – XT5	Exascale	Increase
Cores	224,256	100 million – 1 billion	~1,000×
Threads	224,256 way	1 – 10 billion way	~50,000×
Memory	300 Terabytes	10 – 128 Petabytes	~500×

Estimates consolidated from International Exascale Software Project Roadmap and the DOE Exascale Initiative Roadmap.

MPI-Only Approach?

	Jaguar – XT5	Exascale	Increase
Cores	224,256	100 million – 1 billion	$\sim$1,000$\times$
Threads	224,256 way	1 – 10 billion way	$\sim$50,000$\times$
Memory	300 Terabytes	10 – 128 Petabytes	$\sim$500$\times$

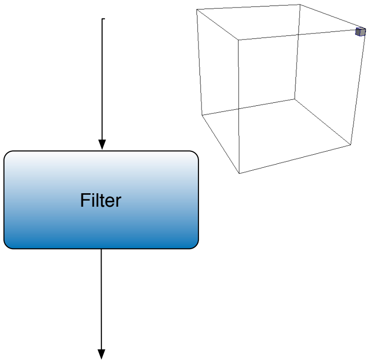
Vis object code + state : 20 MB
On Jaguar : 20 MB x 200,000 processes = 4 TB
On Exascale: 20 MB x 10,000,000,0000 processes = 200 PB!

Visualization Pipeline too heavyweight?

	Jaguar – XT5	Exascale	Increase
Cores	224,256	100 million – 1 billion	$\sim$1,000$\times$
Threads	224,256 way	1 – 10 billion way	$\sim$50,000$\times$
Memory	300 Terabytes	10 – 128 Petabytes	$\sim$500$\times$

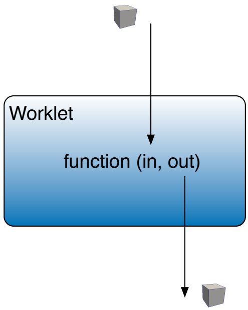
On Jaguar : 1 trillion cells $\rightarrow$ 5 million cells/thread
On Exascale: 500 trillion cells $\rightarrow$ 50K cells/thread

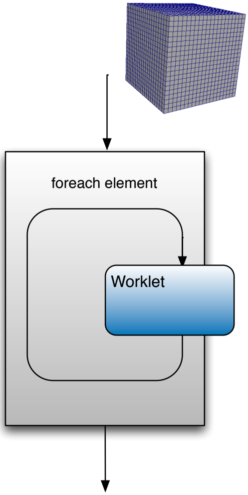
Revisiting the Filter

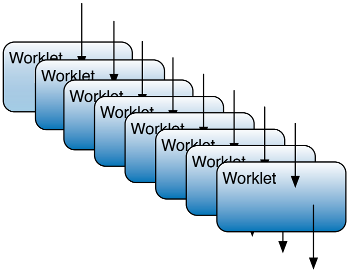


- Lightweight Object
- Serial Execution
- No explicit partitioning
- No access to larger structures
- No state

function $(in,\ out)$


Worklet
function (in, out)


foreach element
Worklet


Worklet

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]

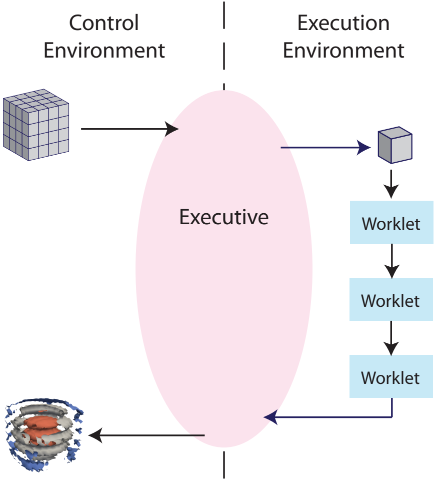
- Pros: Mesh flexibility; low memory overhead; lazy evaluation; straighforward to parallelize.
- Cons: Does not manage massive multi-threading; no mechanism for topology generation.

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 opeartion inefficient because it ignores known neighborhood or domain decompositions.

Dax Toolkit

Dax Programming Environment



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`

Execution Environment

```
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

```
int vtkCellDerivatives::RequestData(...)
{
  ...[allocate output arrays]...
  ...[validate inputs]...
  for (cellId=0; cellId < numCells; cellId++)
    {
    ...[update progress]...
    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);
    }
  ...[cleanup]...
}
```

```
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

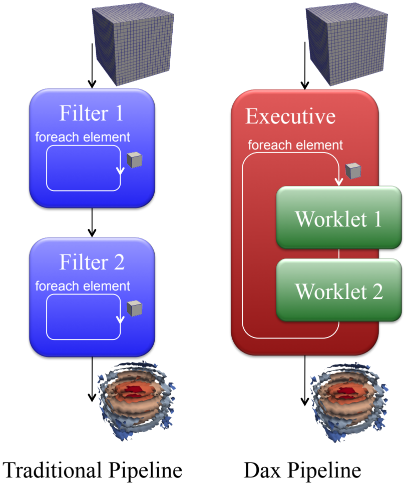
Mesh Size	VTK Time	Dax Time	Speedup
Elevation → Gradient			
144^3	2.75 s	0.013 (0.024) s	210 (114)
256^3	15.52 s	0.074 (0.135) s	210 (115)
512^3	125.75 s	0.589 (1.076) s	213 (117)
Elevation → Sine → Square → Cosine			
144^3	2.32 s	0.002 (0.006) s	1169 (386)
256^3	12.99 s	0.013 (0.034) s	999 (382)
512^3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

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


Filter 1
foreach element
Filter 2
foreach element
Traditional Pipeline
Executive
foreach element
Worklet 1
Worklet 2
Dax Pipeline

Acknowledgements

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.