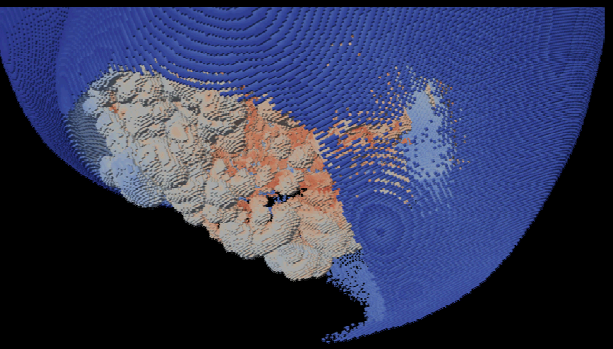
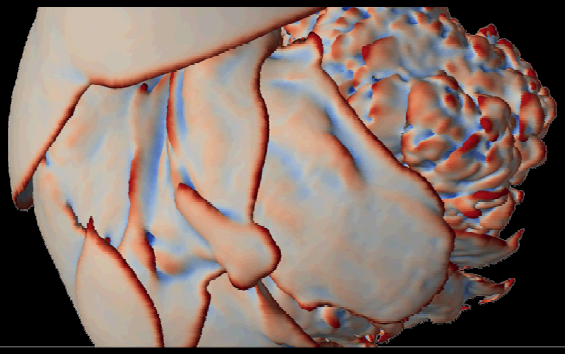
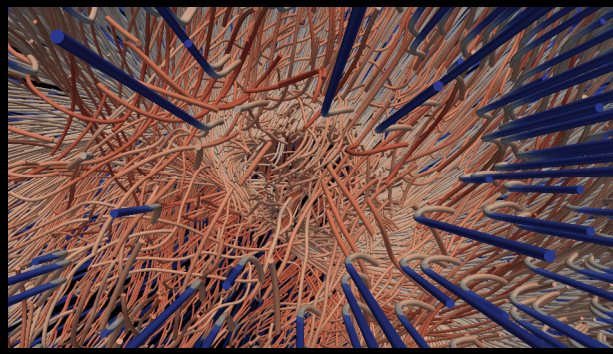
Exceptional service in the national interest

SAND2012-7796P

Sandia National Laboratories







Dax Tutorial

Many-Core Code Sprint

September 19, 2012

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

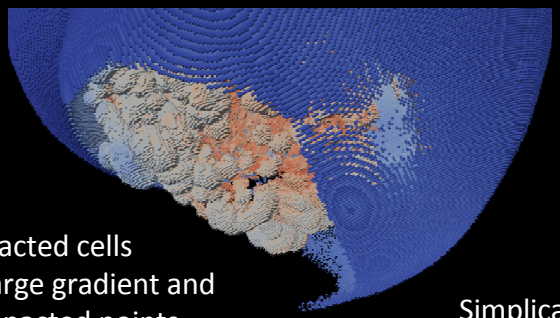
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Dax: A Toolkit for Accelerated Visualization

Sandia National Laboratories

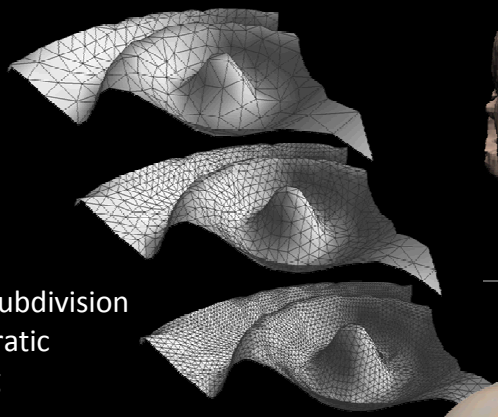
The Dax Toolkit brings together parallel modules to supplement combinable visualization algorithms.

Mapping on Fields

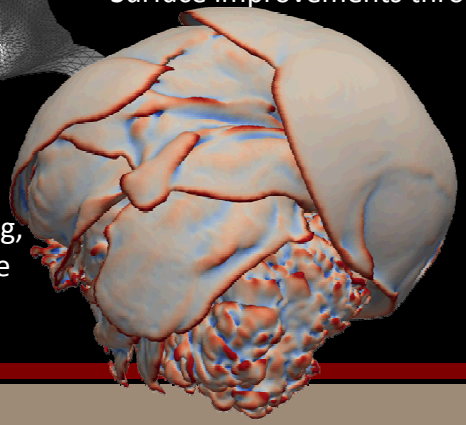


Extracted cells of large gradient and compacted points

Topology Generation



Simplical subdivision with quadratic smoothing



Contour with subsequent vertex welding, coarsening, subdivision, and curvature estimation

Connectivity Reconstruction

Surface improvements through connection construction.

Sandia National Laboratories

GETTING STARTED

Prerequisites

Sandia National Laboratories

- git
- CMake (2.8.8 or newer recommended)
- Boost 1.48.0
- CUDA Toolkit 4+ (for CUDA backend)
- Thrust 1.4 or 1.5 (comes with CUDA)

Getting Dax

Sandia National Laboratories

- Clone from one of the git repositories
  - http://public.kitware.com/daxtoolkit.git
  - https://github.com/Kitware/DaxToolkit.git

```
git clone http://public.kitware.com/daxtoolkit.git
```

Configuring Dax

Sandia National Laboratories

- Create a build directory
- Run ccmake (or cmake-gui) pointing back to source directory

```
git clone http://public.kitware.com/daxtoolkit.git
mkdir daxBuild
cd daxBuild
ccmake ../daxtoolkit
```

Important Configuration Parameters

Sandia National Laboratories

Variable	Description
DAX_ENABLE_CUDA	Turn this ON to enable CUDA backend. Requires CUDA Toolkit and Thrust.
DAX_ENABLE_OPENMP	Turn this ON to enable OpenMP backend. Requires Thrust and OpenMP compiler support (not Clang).
DAX_ENABLE_TESTING	Turn on header, unit, and benchmark tests.
DAX_USE_64BIT_IDS	Enable 64bit index support. Using this with CUDA backend generally requires a Tesla card.
DAX_USE_DOUBLE_PRECISION	Enable using double as the representation type for dax::Scalar. Using this with CUDA backend generally requires a Tesla card.
CMAKE_BUILD_TYPE	Debug, RelWithDebInfo, or Release
CMAKE_INSTALL_PREFIX	Location to install headers

Building Dax

Sandia National Laboratories

- (Technically optional but a good check for your system.)
- Run make (or use your favorite IDE)
- Run tests (“make test” or “ctest”)
- Parallel builds (-j flag) work, too.

```
git clone http://public.kitware.com/daxtoolkit.git
mkdir daxBuild
cd daxBuild
ccmake ../daxtoolkit
make
ctest
```

Installing Dax

- (Optional but good if your project doesn't use CMake)
- make install target

```
git clone http://public.kitware.com/daxtoolkit.git
mkdir daxBuild
cd daxBuild
ccmake ../daxtoolkit
make
ctest
make install
```

More Information

Sandia National Laboratories

- We know, documentation is sparse
- http://daxtoolkit.org
  - Click down to “Using Dax”
- Doxygen: http://daxtoolkit.org/Doc/classes.html

Sandia National Laboratories

BASIC COMMON SUPPORT TYPES AND FUNCTIONS

Basic Types

Sandia National Laboratories

- `#include <dax/Types.h>`
- `dax::Id`
  - Integer/index type
- `dax::Scalar`
  - Floating point
- Use of core C types (int, float, double, etc.) discouraged.

Basic Tuples

Sandia National Laboratories

- `#include <dax/Types.h>`
- `dax::Id3`, `dax::Vector2`, `dax::Vector3`, `dax::Vector4`
- Operator [] overloaded for component access
- Operators +, -, *, / overloaded for component-wise arithmetic
- `make_Id3()`, `make_Vector2()`, `make_Vector3()`, etc.
- `dax::dot()`
- Generic `dax::Tuple<int>`

Vector Traits

- `#include<dax/VectorTraits.h>`
- `template<class VectorType> struct VectorTraits`
  - `NUM_COMPONENTS`
  - `GetComponent()`
  - `SetComponent()`

Basic Math

Sandia National Laboratories

- `#include <dax/math/Compare.h>`
  - `dax::math::Min(x,y), dax::math::Max(x,y)`
- `#include <dax/math/Precision.h>`
  - `dax::math::Nan(), dax::math::Infinity(), dax::math::NegativeInfinity(), dax::math::Epsilon()`
  - `dax::math::IsNan()`
  - `dax::math::IsInf(), dax::math::IsFinite()`
  - `dax::math::FMod(), dax::math::Remainder(), dax::math::RemainderQuotient(), dax::math::ModF()`
  - `dax::math::Ceil(), dax::math::Floor(), dax::math::Round()`
- #include <dax/math/Sign.h>
  - `dax::math::Abs(), dax::math::IsNegative(), dax::math::CopySign()`

Basic Math

Sandia National Laboratories

- `#include <dax/math/Exp.h>`
  - `dax::math::Pow(), dax::math::Sqrt(), dax::math::RSqrt(), dax::math::Cbrt(), dax::math::RCbrt()`
  - `dax::math::Exp(), dax::math::Exp2(), dax::math::ExpM1(), dax::math::Exp10()`
  - `dax::math::Log(), dax::math::Log2(), dax::math::Log10(), dax::math::Log1P()`
- `#include <dax/math/Trig.h>`
  - `dax::math::Pi()`
  - `dax::math::Sin(), dax::math::Cos(), dax::math::Tan()`
  - `dax::math::ASin(), dax::math::ACos(), dax::math::ATan(), dax::math::ATan2()`
  - `dax::math::SinH(), dax::math::CosH(), dax::math::TanH(), dax::math::ASinH(), dax::math::ACosH(), dax::math::ATanH()`

Basic Vector Operations

Sandia National Laboratories

- `#include <dax/math/VectorAnalysis.h>`
  - `dax::math::MagnitudeSquared()`,`dax::math::Magnitude()`, `dax::math::RMagnitude()`
  - `dax::math::Normal()`,`dax::math::Normalize()`
  - `dax::math::Cross()`
  - `dax::math::TriangleNormal()`

Basic Small Matrix Operations

Sandia National Laboratories

- `#include <dax::math::Matrix.h>`
  - class `dax::math::Matrix<`*`Type, numRow, numCol`*`>`
    - Access components by `matrix(row,col)` or `matrix[row][col]`
    - Specialty types `dax::math::Matrix2x2`, `dax::math::Matrix3x3`, `dax::math::Matrix4x4`
  - `dax::math::MatrixRow(matrix, rowIndex)`, `dax::math::MatrixColumn(matrix, colIndex)`
  - `dax::math::MatrixSetRow(matrix,rowIndex,rowValues)`, `dax::math::MatrixSetColumn(matrix,colIndex,colValues)`
  - `dax::math::MatrixMultiply(leftFactor, rightFactor)`
  - `dax::math::MatrixIdentity()`
  - `dax::math::MatrixTranspose()`
  - `dax::math::MatrixInverse()`
  - `dax::math::MatrixDeterminant()`

Other Linear and Numeric Algorithms

Sandia National Laboratories

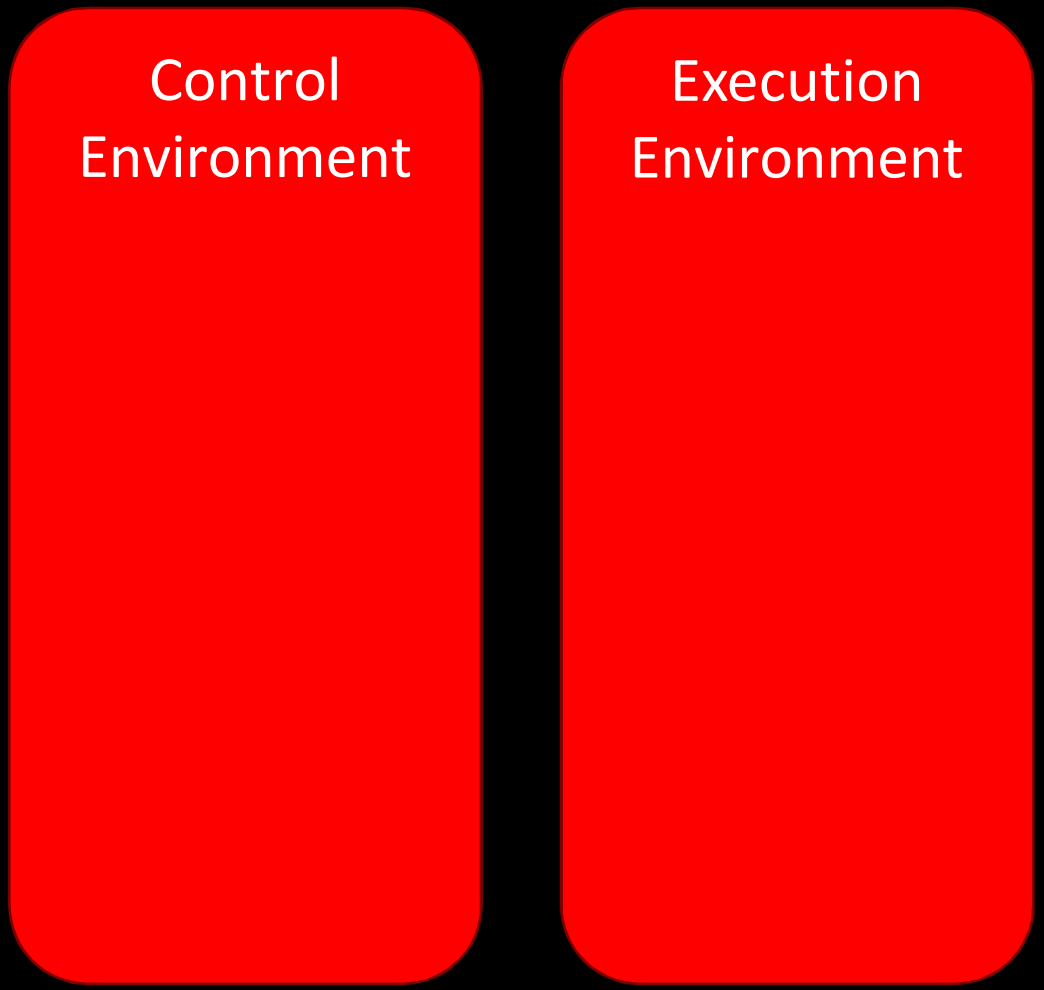
- `#include <dax/math/Matrix.h>`
  - `dax::math::SolveLinearSystem()`
- `#include <dax/math/Numerical.h>`
  - `dax::math::NewtonsMethod()`

Sandia National Laboratories

SYSTEM OVERVIEW

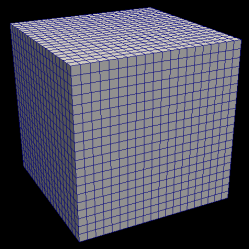
Dax Framework

Sandia National Laboratories



Dax Framework

Sandia National Laboratories



Control Environment

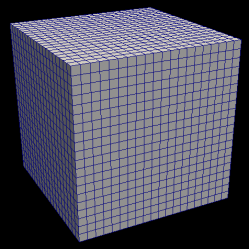
Grid Topology
Array Handle
Invoke

Execution Environment

dax::cont

dax::exec

Dax Framework



Control Environment

Grid Topology
Array Handle
Invoke

Execution Environment

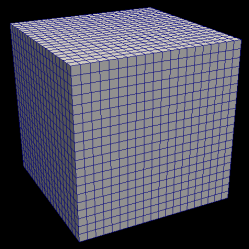
Cell Operations
Field Operations
Basic Math
Make Cells

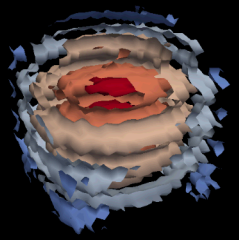
Worklet

dax::cont

dax::exec

Dax Framework





Control Environment

Grid Topology
Array Handle
Invoke

dax::cont

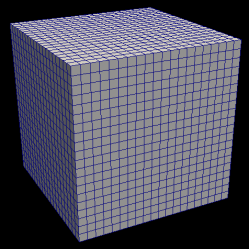
Execution Environment

Cell Operations
Field Operations
Basic Math
Make Cells

dax::exec

Worklet

Dax Framework



Control Environment

Grid Topology
Array Handle
Invoke

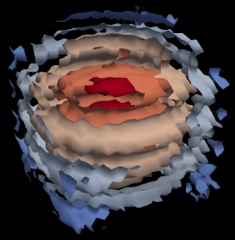
Device Adapter

Allocate
Transfer
Schedule
Sort
…

Execution Environment

Cell Operations
Field Operations
Basic Math
Make Cells

Worklet



dax::cont

dax::exec

Export Macros

Sandia National Laboratories

- DAX_CONT_EXPORT Functions/methods for control environment.
  - Think `__host__` in CUDA.
- DAX_EXEC_EXPORT Functions/methods for execution environment.
  - Think `__device__` in CUDA.
- DAX_EXEC_CONT_EXPORT Either environment.

Sandia National Laboratories

CREATING WORKLETS IN THE EXECUTION ENVIRONMENT

Creating Worklets

Sandia National Laboratories

- Replace this slide with a series of instructions on how to create worklets. Describe the types of worklets, subclassing the worklet classes to make functors, signatures, and whatever else is relevant.

Execution Cell Types

Sandia National Laboratories

- Some worklets can receive a cell as a parameter
- Usually the cell type is templated, but can be specified directly
  - Cell classes defined in dax/exec/Cell*.h
- No shared superclass, but all implement at a minimum
  - Constants `NUM_POINTS` and `TOPOLOGICAL_DIMENSIONS`
  - `GetNumberOfPoints()`
  - `GetPointIndex(vertexIndex)`
  - `GetPointIndices()`
    - Returns a Tuple defined by the typedef *`Cell`*`::PointConnectionsType`
- Currently supported cells: Vertex, Line, Triangle, Quadrilateral, Voxel, Tetrahedron, Wedge, Hexahedron

Cell Operations

Sandia National Laboratories

- #include <dax/exec/ParametricCoordintes.h>
  - dax::exec::ParametricCoordinates<*CellType*>::Center()
  - dax::exec::ParametricCoordinates<*CellType*>::Vertex()
  - dax::exec::ParametricCoordinatesToWorldCoordinates()
  - dax::exec::WorldCoordinatesToParametericCoordinates()
- #include <dax/exec/Interpolate.h>
  - dax::exec::CellInterpolate()
- #include <dax/exec/Derivative.h>
  - dax::exec::CellDerivative()

Errors in the Execution Environment

Sandia National Laboratories

- All worklet classes have a `RaiseError()` method
  - `this->RaiseError("Precondition failed");`
  - Will cause an exception with that message to be thrown in the control environment from whence the worklet was invoked
    - But, you cannot catch the error while still in the execution environment
    - In fact, execution might not immediately stop
- `#include <dax/exec/Assert.h>`
  - `DAX_ASSERT_EXEC(condition, worklet)`
  - Behaves like the POSIX assert except that it throws this error rather than crashing the program to a halt
  - `DAX_ASSERT_EXEC(vertexIndex < 8, this)`

Sandia National Laboratories

INVOKING ALGORITHMS ON DATA IN THE CONTROL ENVIRONMENT

Catching Errors

Sandia National Laboratories

- Dax throws errors in anomalous conditions
  - Thrown class always a subclass of dax::cont::Error
  - If the error originated in control environment, then a subclass of dax::cont::ErrorControl
  - If the error came from a worklet in the execution environment, then it will be dax::cont::ErrorExecution

```
try
  {
  ...
  }
catch (dax::cont::Error error)
  {
  std::cout << "Dax encountered an error: "
            << error.GetMessage() << std::endl;
  }
```

DeviceAdapter

Sandia National Laboratories

- The DeviceAdapter is an interface between control and execution environments. Adapts to different devices/compilers.
- Easiest way to select is to define DAX_DEVICE_ADAPTER
- Use one of the following (before including any Dax header)
  - `#define DAX_DEVICE_ADAPTER DAX_DEVICE_ADAPTER_SERIAL`
  - `#define DAX_DEVICE_ADAPTER DAX_DEVICE_ADAPTER_OPENMP`
  - `#define DAX_DEVICE_ADAPTER DAX_DEVICE_ADAPTER_CUDA`
- If you select none, a reasonable default will be selected for you
- There are other ways to select (and create) a DeviceAdapter
  - Won’t be talking about them today

ArrayHandle

- `dax::cont::ArrayHandle<type>` manages an "array" of data
  - Acts like a reference-counted smart pointer to an array
  - Manages transfer of data between control and execution
  - Can allocate data for output
- Relevant methods
  - `GetNumberOfValues()`
  - `CopyInto()`
  - `ReleaseResources()`, `ReleaseResourcesExecution()`
- Functions to create an ArrayHandle
  - `dax::cont::make_ArrayHandle(const T *array, dax::Id size)`
  - `dax::cont::make_ArrayHandle(const std::vector<T> vector)`
  - Both of these do a shallow (reference) copy.
    - Do not let the original array be deleted or vector to go out of scope!

Other Important ArrayHandle Features We're Skipping

Sandia National Laboratories

- ArrayContainerControl template parameter
  - Selects array layout for zero-copy semantics
  - Supports implicit arrays
  - Yes, we have a container for vtkDataArray
- Generic array interface through an ArrayPortal
  - In principle like an STL iterator, but simpler

UniformGrid

Sandia National Laboratories

- `#include <dax/cont/UniformGrid.h>`
- `dax::cont::UniformGrid<>` (template parameters default)
  - `Get/SetExtent(),Get/SetOrigin(),Get/SetSpacing()`
  - `GetNumberOfPoints(),GetNumberOfCells()`
  - `ComputePointIndex(dax::Id3),ComputeCellIndex(dax::Id3)`
  - `ComputePointLocation(dax::Id)`
  - `ComputeCellLocation(dax::Id)`
  - `ComputePointCoordinates(dax::Id` *or* `dax::Id3)`
  - `GetPointCoordinates()`

UnstructuredGrid

Sandia National Laboratories

- `#include <dax/cont/UnstructuredGrid.h>`
- `dax::cont::UnstructuredGrid<`*CellType*`>`
  - `Get/SetCellConnections()`
    - Stored in `ArrayHandle<dax::Id>`, one entry per cell vertex
  - `Get/SetPointCoordinates()`
    - Stored in `ArrayHandle<dax::Vector3>`, one entry per point
    - Note that `GetPointCoordinates()` is match in `UniformGrid`
  - `GetNumberOfPoints()`, `GetNumberOfCells()`

Invoking Worklets

Sandia National Laboratories

- At this point, give instructions on using the schedule function to run worklets on data.

Sandia National Laboratories

PUTTING IT ALL TOGETHER

- This would be a good point for a complete example starting from creating a worklet to making a program that runs it on some data.