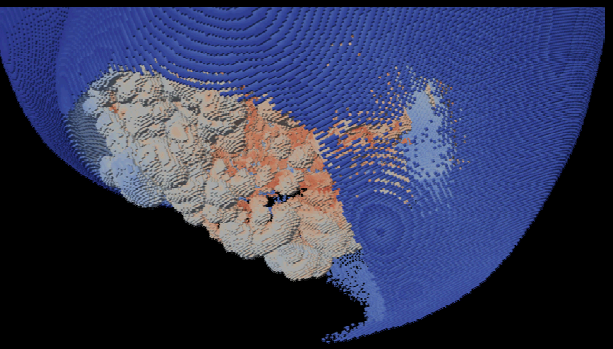
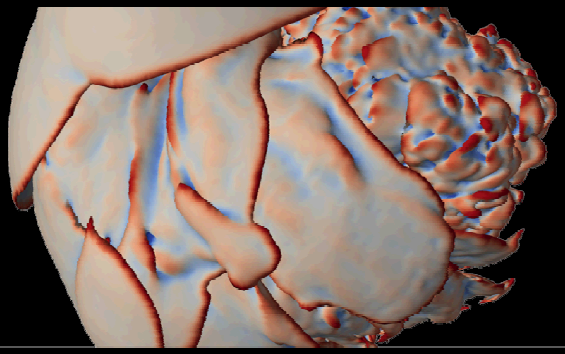
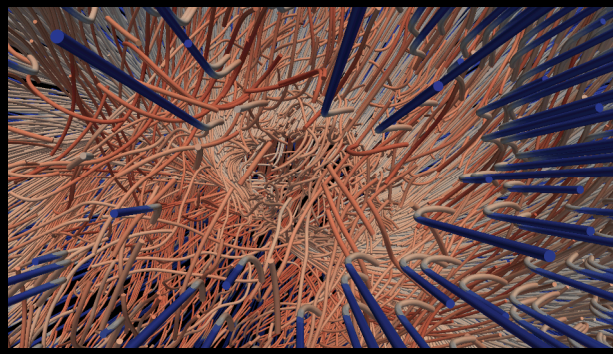
Exceptional service in the national interest

SAND2013-1426P

Sandia National Laboratories







Dax: Data Analysis at Extreme

SDAV All Hands Meeting

Kenneth Moreland Sandia National Laboratories

February 20, 2013

Kitware

UC DAVIS
UNIVERSITY OF CALIFORNIA

SDAV

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. 2013-XXXXP

Slide of Doom

System Parameter	2011	"2018"		Factor Change
System Peak	2 PetaFLOPS	1 ExaFLOP		500
Power	6 MW	≤ 20 MW		3
System Memory	0.3 PB	32 – 64 PB		100 – 200
Total Concurrency	225K	1B × 10	1B × 100	40,000 – 400,000
Node Performance	125 GF	1 TF	10 TF	8 – 80
Node Concurrency	12	1,000	10,000	83 – 830
Network BW	1.5 KB/s	100 GB/s	1000 GB/s	66 – 660
System Size (nodes)	18,700	1,000,000	100,000	50 – 500
I/O Capacity	15 PB	300 – 1000 PB		20 – 67
I/O BW	0.2 TB/s	20 – 60 TB/s		10 – 30

Slide of Doom

System Parameter	2011	"2018"		Factor Change
System Peak	2 PetaFLOPS	1 ExaFLOP		500
Power	6 MW	≤ 20 MW		3
System Memory	0.3 PB	32 – 64 PB		
Total Concurrency	225K	1B × 10	1B × 100	
Node Performance	125 GF	1 TF	10 TF	8 – 80
Node Concurrency	12	1,000	10,000	83 – 830
Network BW	1.5 KB/s	100 GB/s	1000 GB/s	66 – 660
System Size (nodes)	18,700	1,000,000	100,000	50 – 500
I/O Capacity	15 PB	300 – 1000 PB		20 – 67
I/O BW	0.2 TB/s	20 – 60 TB/s		10 – 30

Massive Concurrency

	Jaguar – XT5	Exascale*	Increase
Memory	300 Terabytes	32 – 64 Petabytes	100 – 200×
Concurrency	224,256 way	10 – 100 billion way	Up to 400,000×

Overhead of ghost/halo cells?

- On Jaguar: 1 trillion cells → 5 million cells/thread
  - Partition into ~171^3 blocks
  - $6 \times 171^2 \approx 175$K ghost/block → 35 billion ghost total
  - Ghost cells **~3.5% size of original data**
- On Exascale: 100 trillion cells → 1000 cells/thread
  - Partition into 10^3 blocks
  - $6 \times 10^2 \approx 600$ ghost/block → 60 trillion ghost total
  - Ghost cells **60% size of original data**

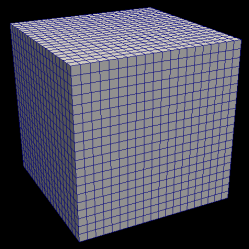
*Source: Scientific Discovery at the Exascale, Ahern, Shoshani, Ma, et al.

Dax Project

- Reduce the challenges of writing highly concurrent algorithms.

"Everybody who learns concurrency thinks they understand it, ends up finding mysterious races they thought weren't possible, and discovers that they didn't actually understand it yet after all." Herb Sutter

Dax Framework

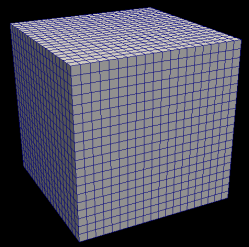


Control Environment

Grid Topology
Array Handle
Invoke

Execution Environment

Dax Framework



Control Environment

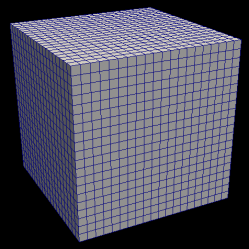
Grid Topology
Array Handle
Invoke

Execution Environment

Cell Operations
Field Operations
Basic Math
Make Cells

Worklet

Dax Framework



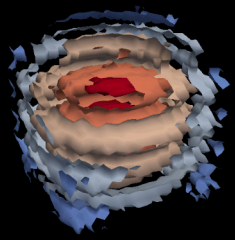
Control Environment

Grid Topology
Array Handle
Invoke

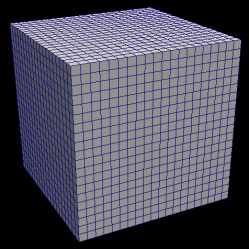
Execution Environment

Cell Operations
Field Operations
Basic Math
Make Cells

Worklet



Dax Framework



Control Environment

Grid Topology
Array Handle
Invoke

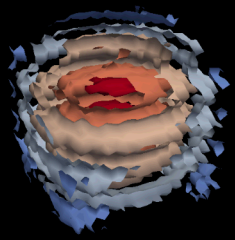
Device Adapter

Allocate
Transfer
Schedule
Sort
…

Execution Environment

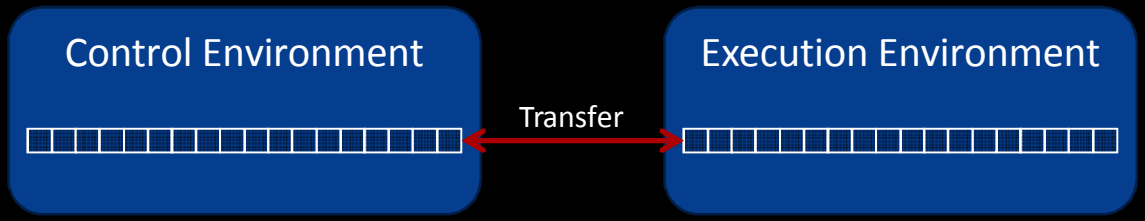
Cell Operations
Field Operations
Basic Math
Make Cells

Worklet

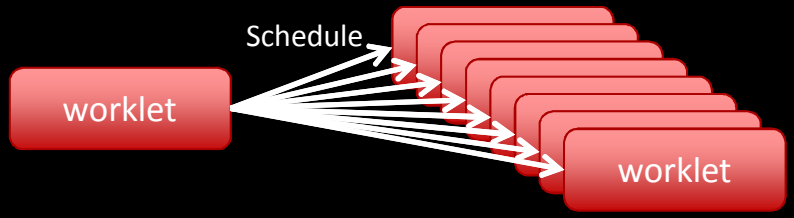


Device Adapter Encapsulates Architecture Ambiguity

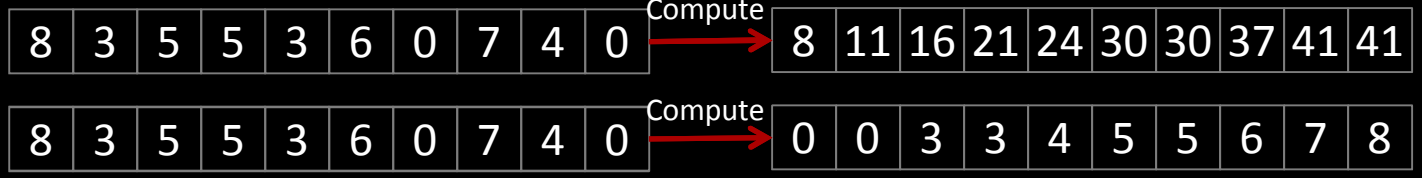
- Execution Array Manager



- Schedule

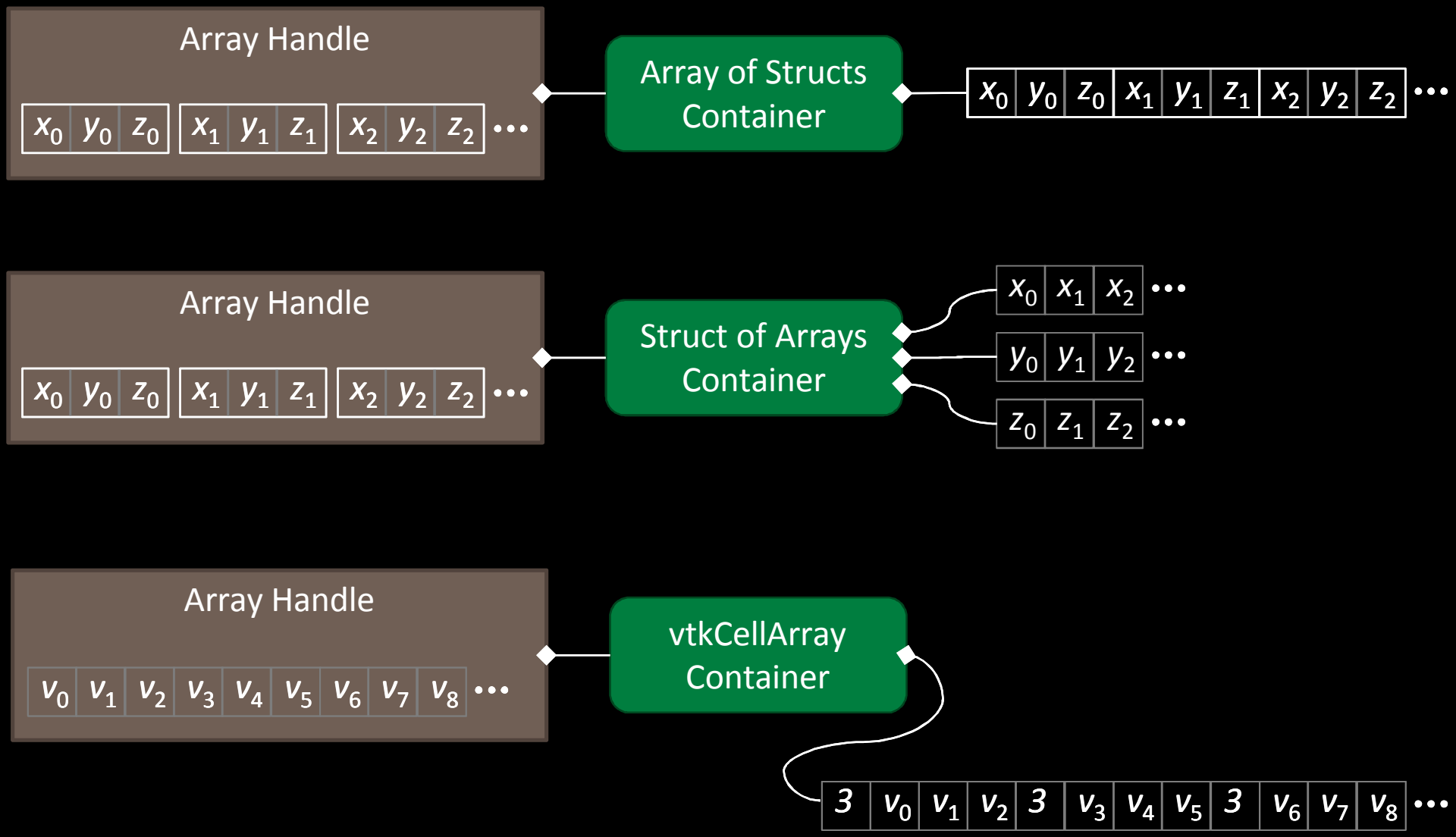


- Scan
- Sort



- Other Support algorithms
  - Stream compact, copy, parallel find, unique

Array Handle Container Minimizes Memory Overhead



Worklet

```
struct Classify: dax::exec::WorkletMapCell {
  typedef void ControlSignature(Topology(In),Field(Point,In)
                                Field(Out));
  typedef _3 ExecutionSignature(_2);

  template<class InCellTag>
  DAX_EXEC_EXPORT
  dax::Id operator()(const CellVertices<InCellTag>& v) const
    ...
```

Execution Environment

Control Environment

```
dax::cont::UniformGrid grid;
dax::cont::ArrayHandle<dax::Scalar> inputHandle =
    dax::cont::make_ArrayHandle(input);
dax::cont::ArrayHandle<dax::Scalar> result;

dax::cont::Scheduler<> scheduler;
scheduler.Invoke(Classify(), grid, inputHandle, result);
```

Benefits of this system

```
struct CopyCell: dax::exec::WorkletGenerateTopology {
  typedef void ControlSignature(Topology(In), Topology(Out));
  typedef void ExecutionSignature(Vertices(_1), Vertices(_2));

  template<class InCellTag, class OutCellTag>
  DAX_EXEC_EXPORT void operator()(
    const CellVertices<InCellTag>&inVertices,
    CellVertices<OutCellTag> &outVertices) const
    {
    return outVertices.SetFromTuple(inVertices.GetAsTuple());
    }
};
```

- Word Aligned Types
- Reduce False Sharing
- Let somebody else deal with compiler quirks

Basic Threshold Algorithm

Generic Parallel Primitives

- For each cell, set valid (0 or 1)
- Inclusive scan on valid flag
  - Makes input -> output index map
- Parallel upper bound of count in input->output map
  - Makes output -> input index map
- For each output cell, copy corresponding input
- For each output cell, mark used points
- Parallel copy point fields if point marked

Dax Parallel Primitives

- For each cell, set valid (0 or 1)
- For each output cell, copy corresponding input

```
template<typename ValueType>
class ThresholdClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology, Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_1,_2);

  template<class InputCellType>
  DAX_EXEC_EXPORT dax::Id operator()(
      InputCellType,
      const dax::Tuple<ValueType,InputCellType::NUM_POINTS> &values) const
  {
    ThresholdFunction<ValueType> threshold(this->ThresholdMin, this->ThresholdMax);
    dax::exec::VectorForEach(values, threshold);
    return threshold.valid;
  }
};


class ThresholdTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(Topology::PointIds(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    out = in;
  }
};
```

```
template<typename ValueType>
class ThresholdClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology, Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_1,_2);

  template<class InputCellType>
  DAX_EXEC_EXPORT dax::Id operator()(
      InputCellType,
      const dax::Tuple<ValueType,InputCellType::NUM_POINTS> &values) const
  {
    ThresholdFunction<ValueType> threshold(this->ThresholdMin, this->ThresholdMax);
    dax::exec::VectorForEach(values, threshold);
    return threshold.valid;
  }
};
```

Report cells to be generated.

```
class ThresholdTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(Topology::PointIds(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    out = in;
  }
};
```

Define cells.

```
class HalfSpaceClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology, Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_1,_2);

  template<class InputCellType>
  DAX_EXEC_EXPORT dax::Id operator()(
      InputCellType,
      const dax::Tuple<dax::Vector3,InputCellType::NUM_POINTS> &values) const
  {
    for (int index = 0; index < InputCellType::NUM_POINTS; index++)
      {
      if (dax::dot(values[index],this->normal) + offset > 0) { return 1; }
      }
    return 0;
  }
};
```

Report cells to be generated.

```
class ThresholdTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(Topology::PointIds(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    out = in;
  }
};
```

Define cells.

```
class SphereClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology, Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_1,_2);

  template<class InputCellType>
  DAX_EXEC_EXPORT dax::Id operator()(
      InputCellType,
      const dax::Tuple<dax::Vector3,InputCellType::NUM_POINTS> &values) const
  {
    for (int index = 0; index < InputCellType::NUM_POINTS; index++)
      {
      if (Magnitude(values[index]-this->Center) < this->Radius) { return 1; }
      }
    return 0;
  }
};
```

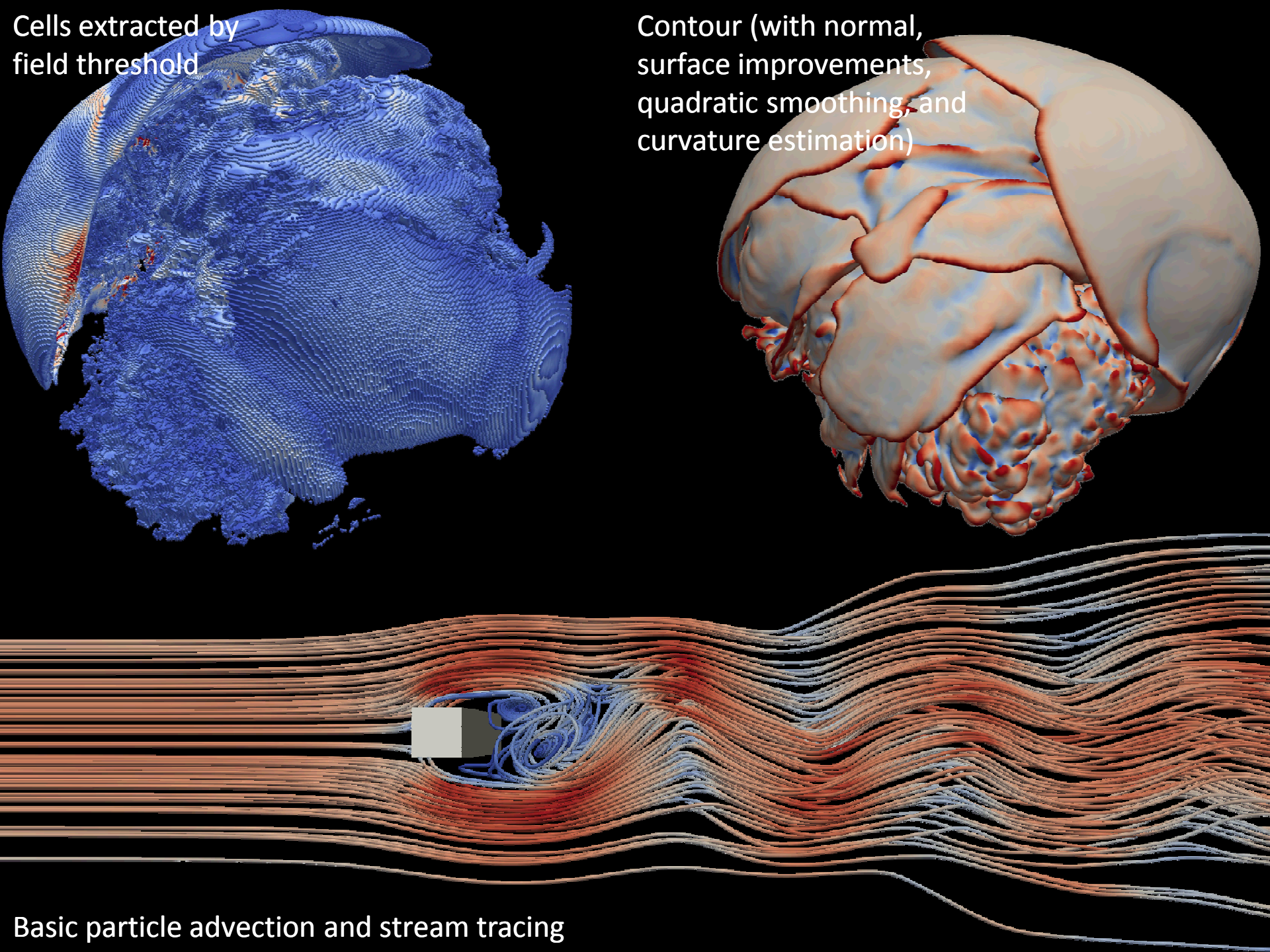
Report cells to be generated.

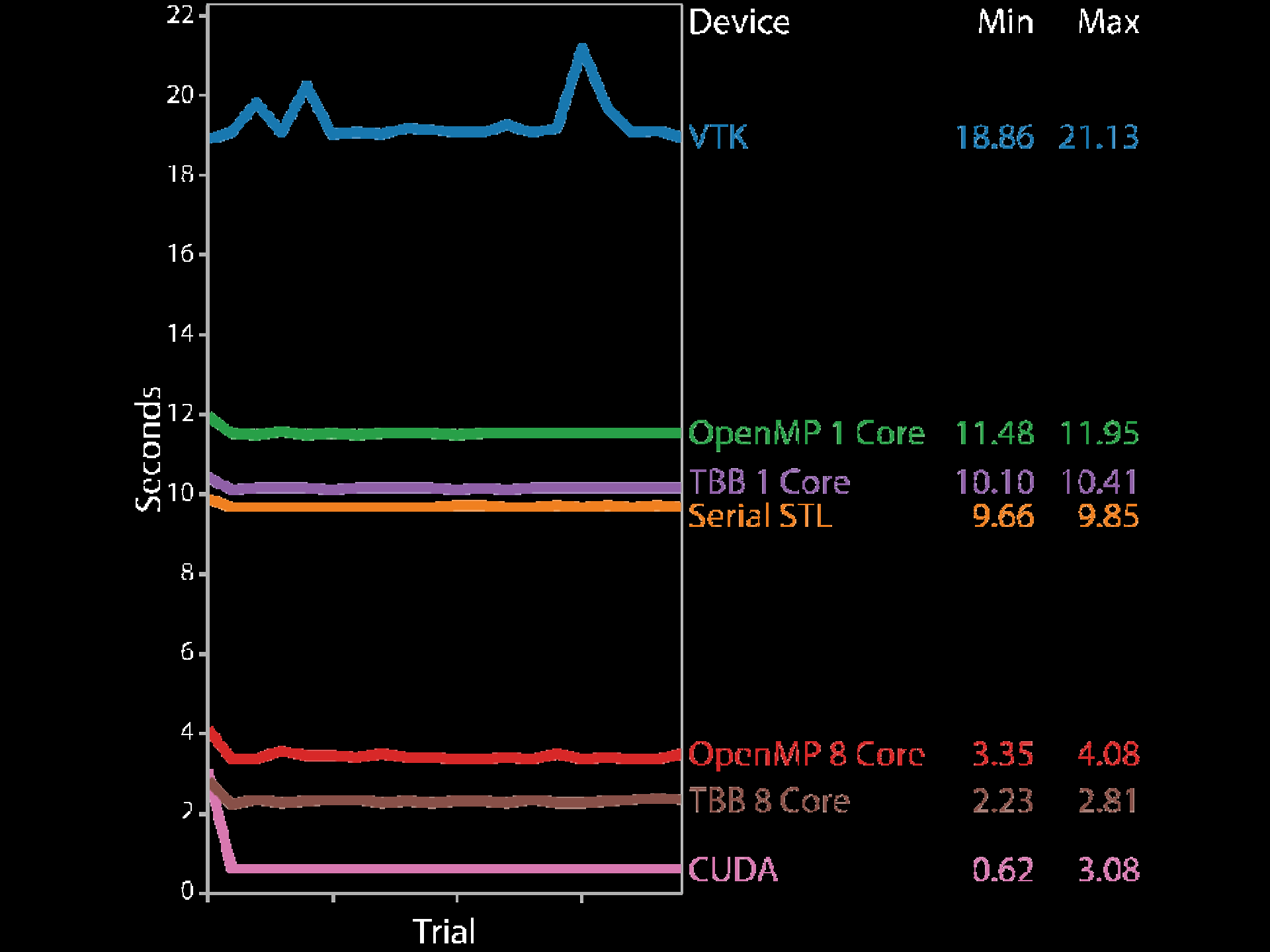
```
class ThresholdTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(Topology::PointIds(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    out = in;
  }
};
```

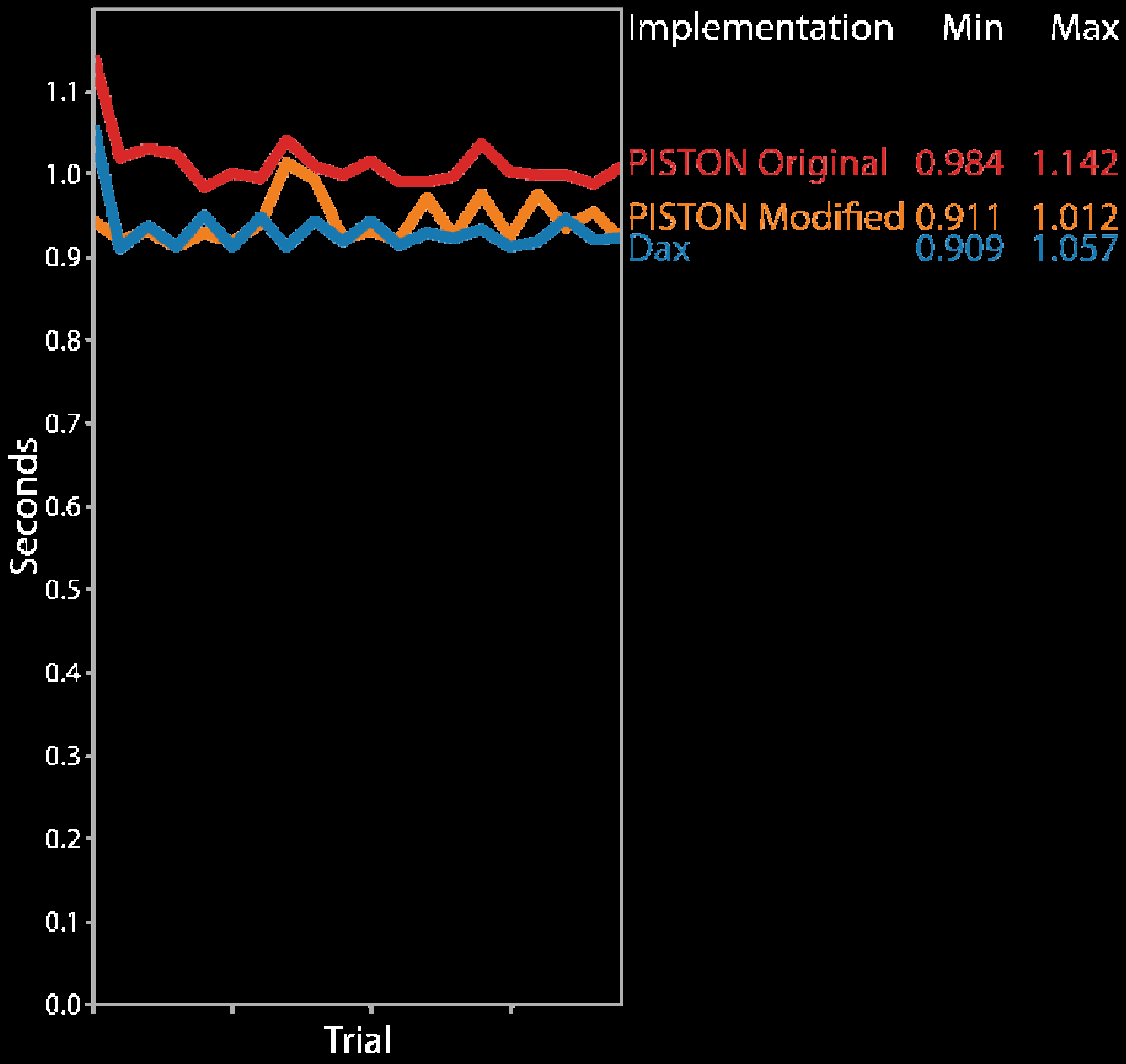
Define cells.

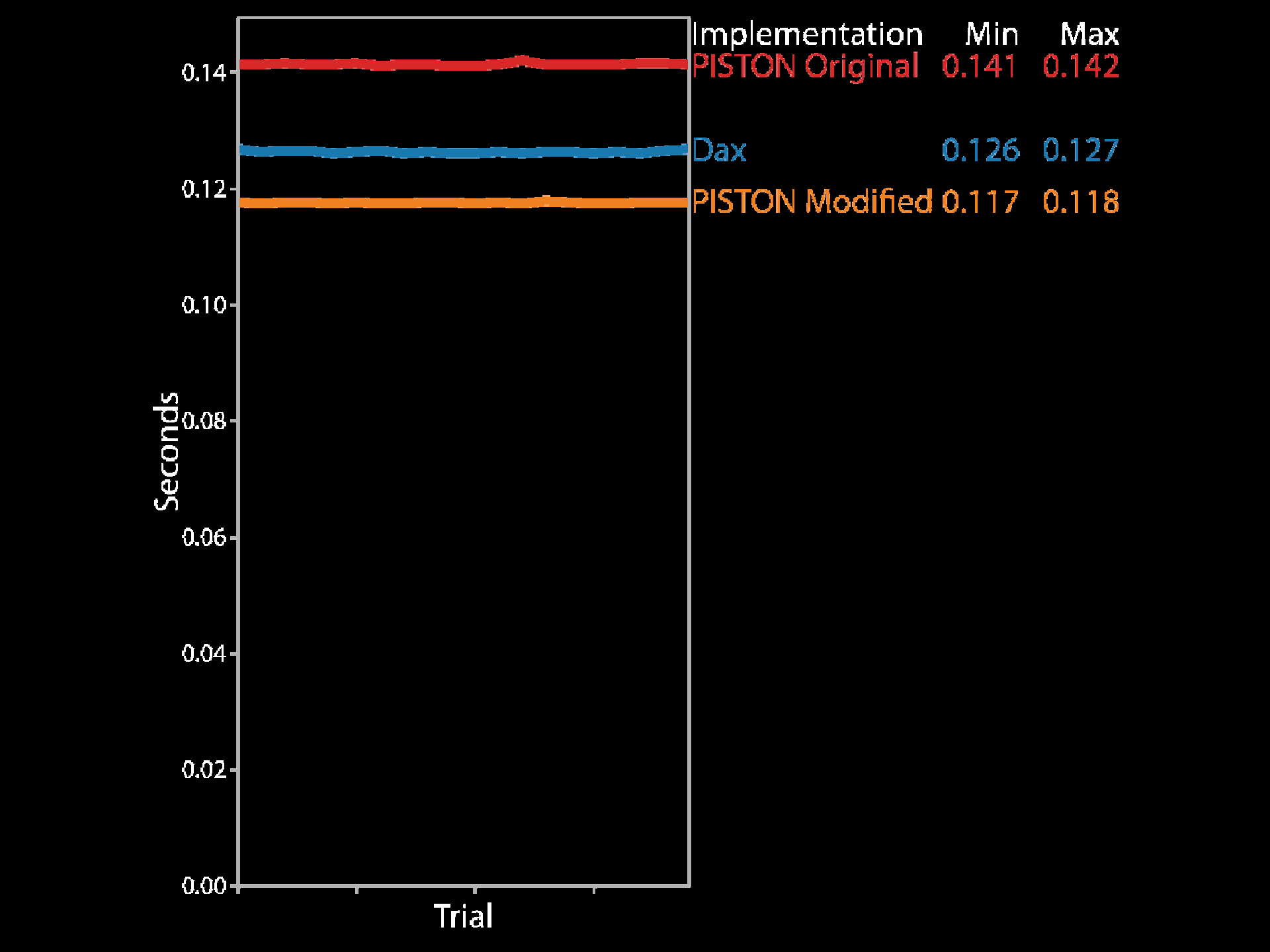
```
class TetrahedraTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(
      Topology::PointIds(_1), Topology::SubIndex(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  int subIndex,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    for (int index = 0; index < NumOutputPoints; index++)
      {
      out[index] = in[TetraVerts[index]];
      }
  }
};
```

Define cells.


Cells extracted by field threshold
Contour (with normal, surface improvements, quadratic smoothing, and curvature estimation)
Basic particle advection and stream tracing


Seconds
22
20
18
16
14
12
10
8
6
4
2
0
Trial
Device
Min
Max
VTK 18.86 21.13
OpenMP 1 Core 11.48 11.95
TBB 1 Core 10.10 10.41
Serial STL 9.66 9.85
OpenMP 8 Core 3.35 4.08
TBB 8 Core 2.23 2.81
CUDA 0.62 3.08


Implementation
Min
Max
PISTON Original 0.984 1.142
PISTON Modified 0.911 1.012
Dax 0.909 1.057
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Seconds
Trial


Implementation Min Max
PISTON Original 0.141 0.142
Dax 0.126 0.127
PISTON Modified 0.117 0.118
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Seconds
Trial

Acknowledgements

- This work was supported by the DOE Office of Science, Advanced Scientific Computing Research, under award number 10-014707, program manager Lucy Nowell.
- Additional support by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. 12-015215, through the Scientific Discovery through Advanced Computing (SciDAC) Institute of Scalable Data Management, Analysis and Visualization.