SAND2013-8195P

A Toolkit for Visualization at Extreme Scale

http://daxtoolkit.org

Kenneth Moreland
Sandia National Laboratories

Berk Geveci
Kitware, Inc.

Kwan-Liu Ma
University of California at Davis

Robert Maynard
Kitware, Inc.

High-performance computing relies on ever finer threading. Recent advances in processor technology include greater numbers of cores, hyperthreading, and accelerators with integrated blocks of cores, all of which require more software parallelism to achieve peak performance. Current visualization solutions cannot support this extreme level of concurrency. Extreme scale systems require a new programming model and a fundamental change in how we design algorithms. To address these issues, our project builds the Data Analysis at Extreme (Dax) Toolkit.

Simplified Parallel Programming

The Dax Toolkit simplifies the development of parallel visualization algorithms. Below is the Dax code that implements a *threshold* operation. Algorithm implementations are encapsulated in *worklets*, which provide fine-grained parallelism and thread safety. The Dax Toolkit provides schedulers that apply worklets to all elements in a mesh as well as common and versatile communicative operations such as array compaction and point merging. Despite the higher levels of abstraction and generalized programming interface, the speed of Dax algorithms are competitive with other "hand-coded" algorithms.

```
// Run classify algorithm (determine how many cells are passed).
ClassifyResultType classificationArray;
scheduler.Invoke(dax::worklet::ThresholdClassify<dax::Scalar>(0.07, 1.0),
                 grid,
                 inArray,
                 classificationArray);

// Build thresholded topology.
ScheduleGenerateTopologyType resolveTopology(classificationArray);
UnstructuredGridType outGrid;
scheduler.Invoke(resolveTopology, grid, outGrid);

// Compact scalar array to new topology.
ArrayHandleScalar outArray;
resolveTopology.CompactPointField(inArray, outArray);
```

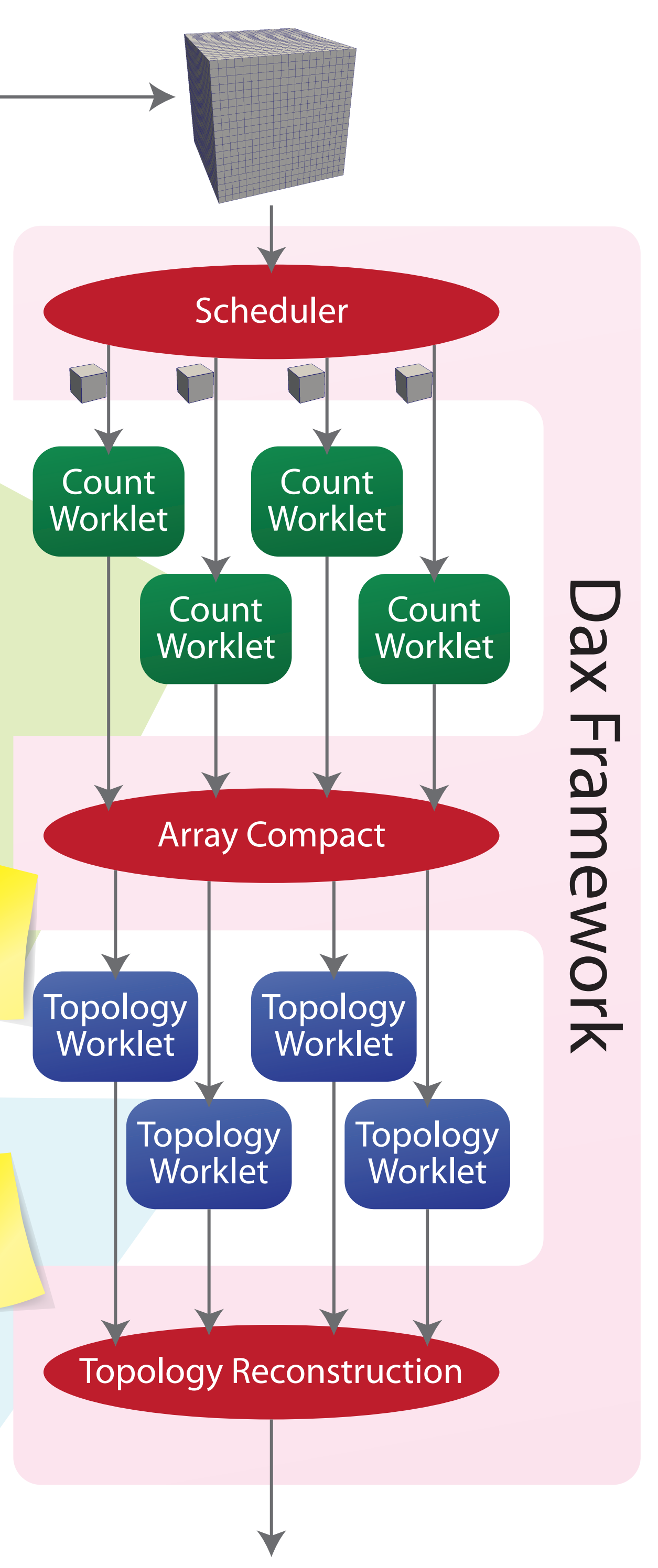
Automatic point merging is optional.

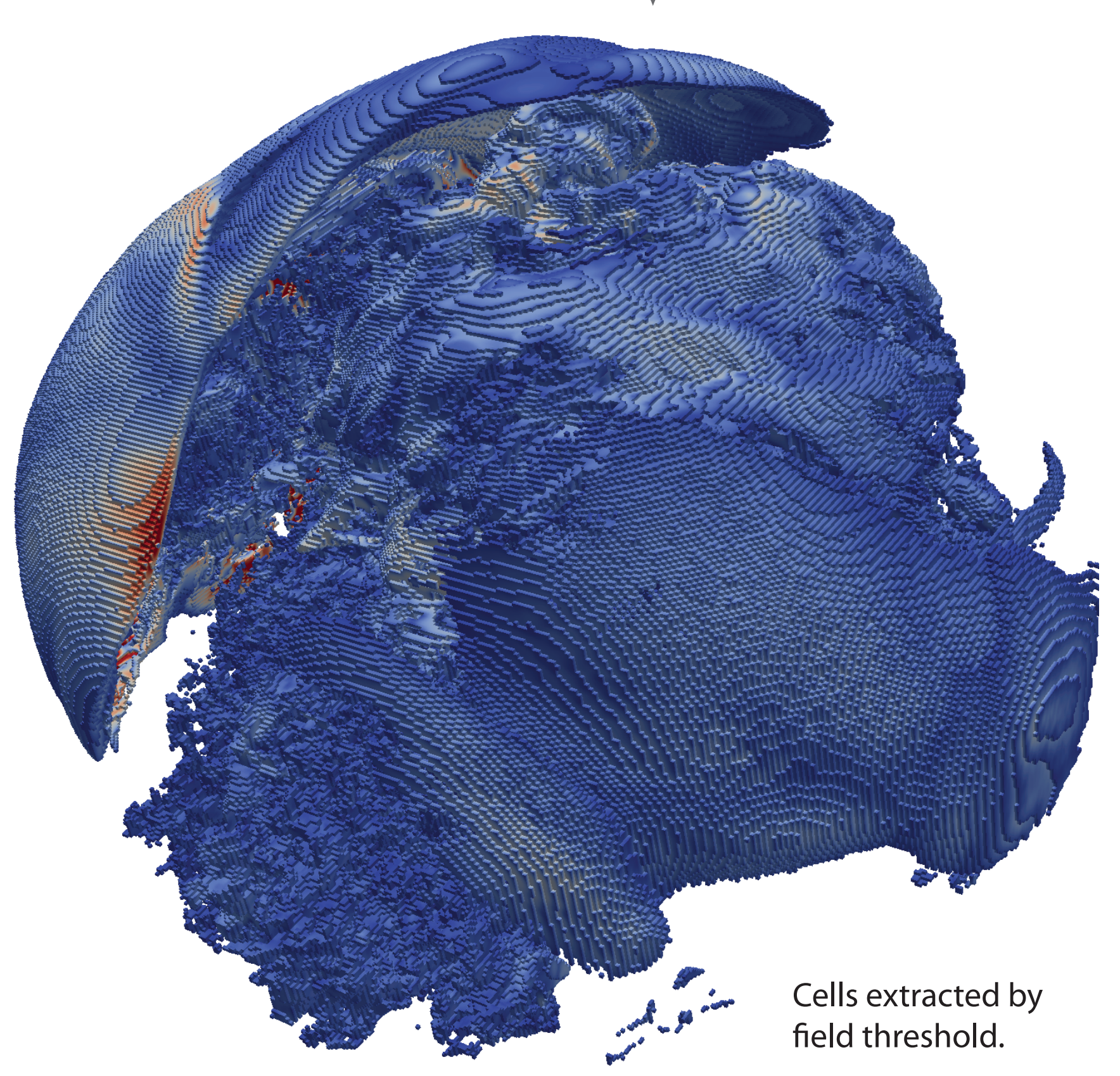
```
template<typename ValueType>
class ThresholdClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology,Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_2);

  DAX_CONT_EXPORT
  ThresholdClassify(ValueType thresholdMin, ValueType thresholdMax)
    : ThresholdMin(thresholdMin), ThresholdMax(thresholdMax) {  }

  template<typename CellTag>  DAX_EXEC_EXPORT  dax::Id operator()(
      const dax::exec::CellField<ValueType,CellTag> &values) const
  {
    ThresholdFunction<ValueType> threshold(this->ThresholdMin,
                                           this->ThresholdMax);
    dax::exec::VectorForEach(values, threshold);
    return threshold.valid;
  }
private:
  ValueType ThresholdMin;
  ValueType ThresholdMax;
};
```

Hazard-free access with lightweight containers.

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

  template<typename InputCellTag, typename OutputCellTag>
  DAX_EXEC_EXPORT
  void operator()(const dax::exec::CellVertices<InputCellTag> &inVertices,
                  dax::exec::CellVertices<OutputCellTag> &outVertices) const
  {
    outVertices.SetFromTuple(inVertices.GetAsTuple());
  }
};
```

Templated cell tags specialize for cell type.

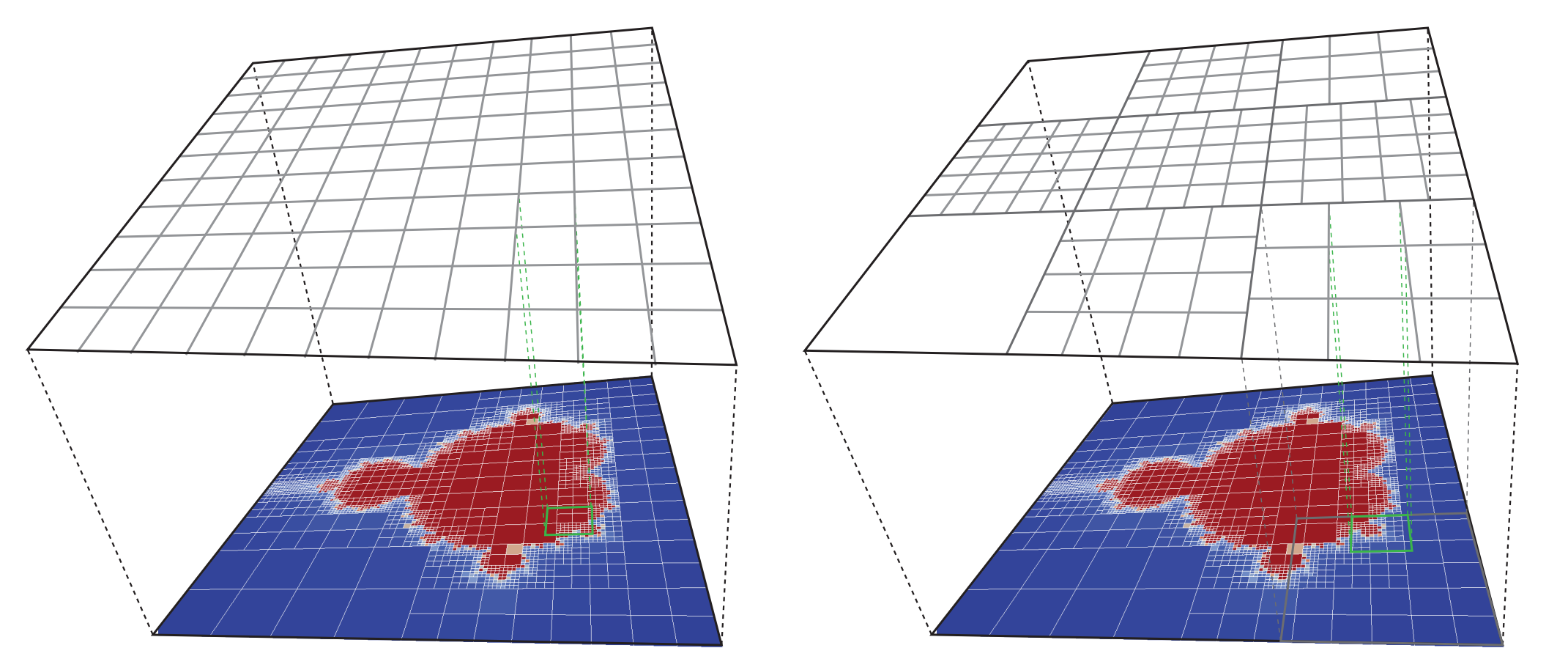



Cells extracted by field threshold.

Topological Connections

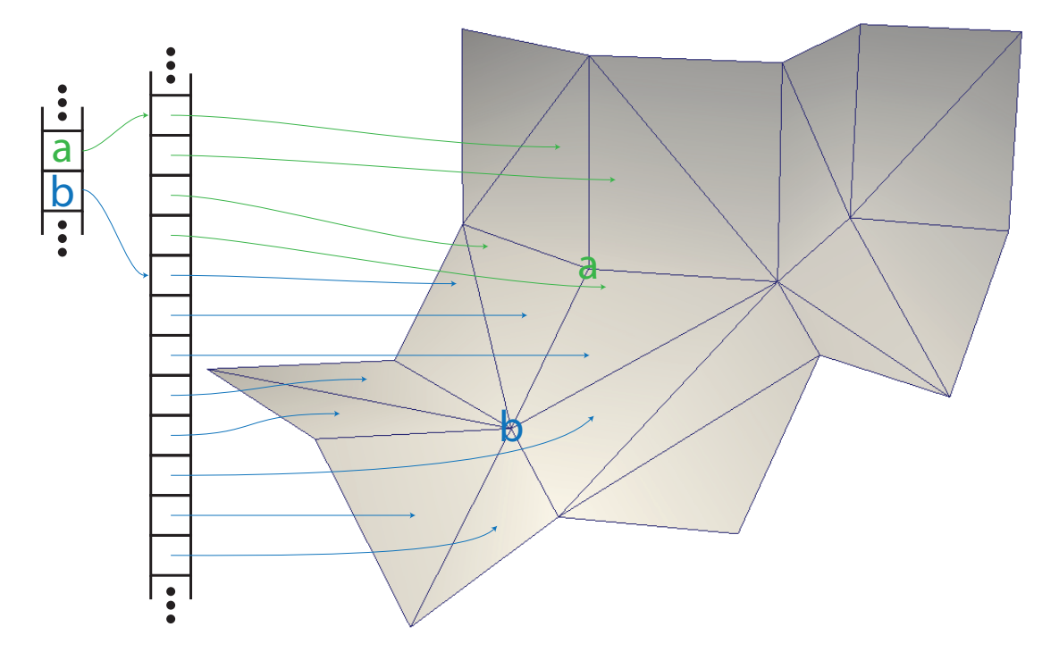
The Dax Toolkit contains algorithms designed to find, preserve, follow, and use topological connections in ways that are often ignored in GPU algorithms. For example, it is straightforward to implement Marching Cubes on millions of threads each independently producing geometry. But reconnecting these pieces to form a manifold surface on massive threads is challenging. Furthermore, data structures are often represented as connection lists from cells to their vertices. Finding the reverse connection from points to incident cells, important for many visualization algorithms, requires building complimentary data structures.

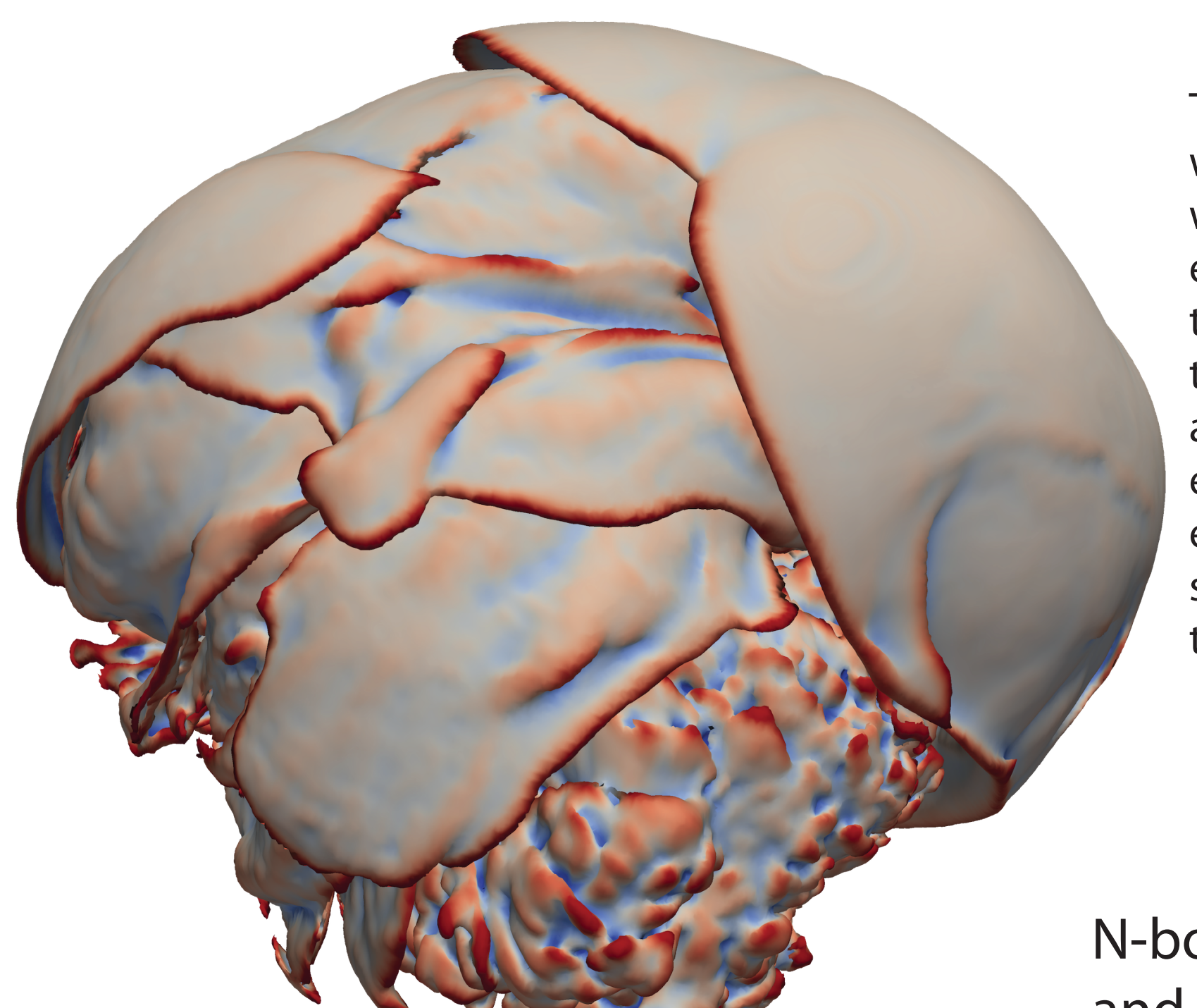
By providing these topology features, the Dax Toolkit can support a number of algorithms otherwise difficult in a massively threaded environment. The current growing list of such algorithms includes vertex welding, C^0 continuous gradient estimation, normal generation for smooth surface rendering, curvature, vertex welding, and cell-finding search structures.



Finding a containing cell in an unstructured grid efficiently requires a supporting search structure. Our initial experiments suggest that regular structures like uniform grids and two level grids are faster on GPUs than structures of deeper hierarchies (like k-d trees).

Meshes that do not have an implicit topology are typically represented using lists of identifiers that for each cell defines its connecting vertices. Thus, points incident on each cell are explicitly defined, but cells incident on each point are not directly accessible. We have recently added a parallel algorithm to build links from the points to the cells. These structures allow us to complete algorithms that , for example, use interpolation on cells to derive a field and then write that field back to points so that the generated field may also be interpolated.

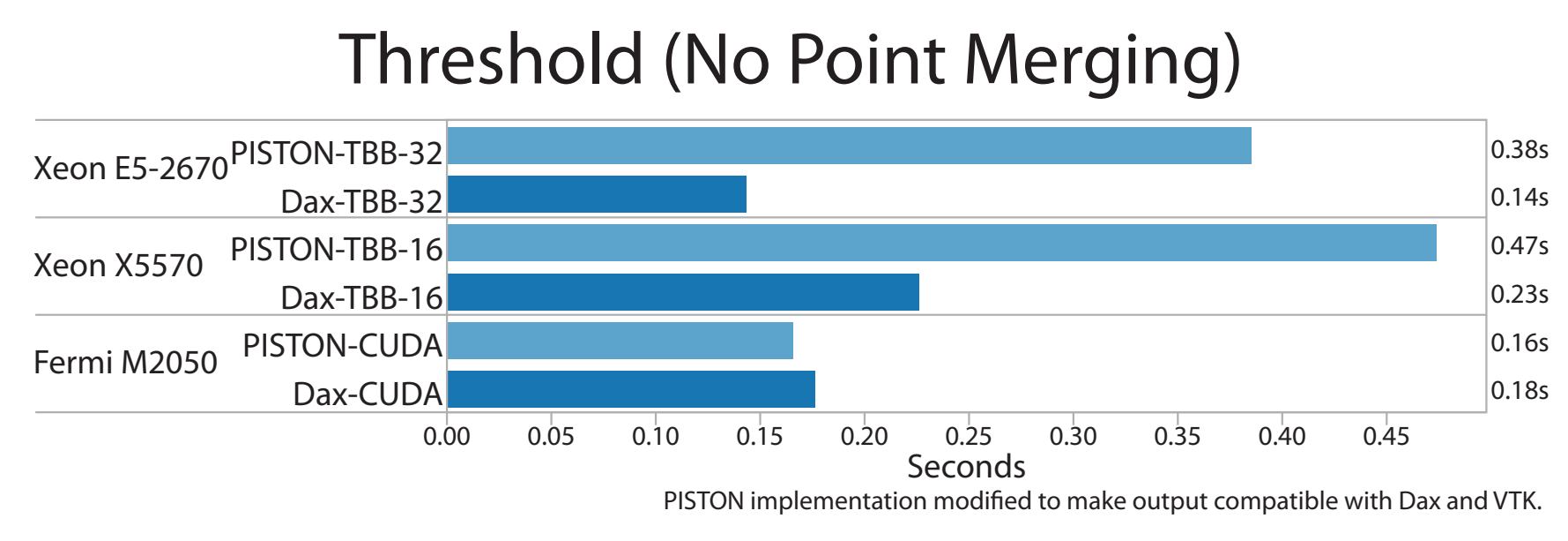


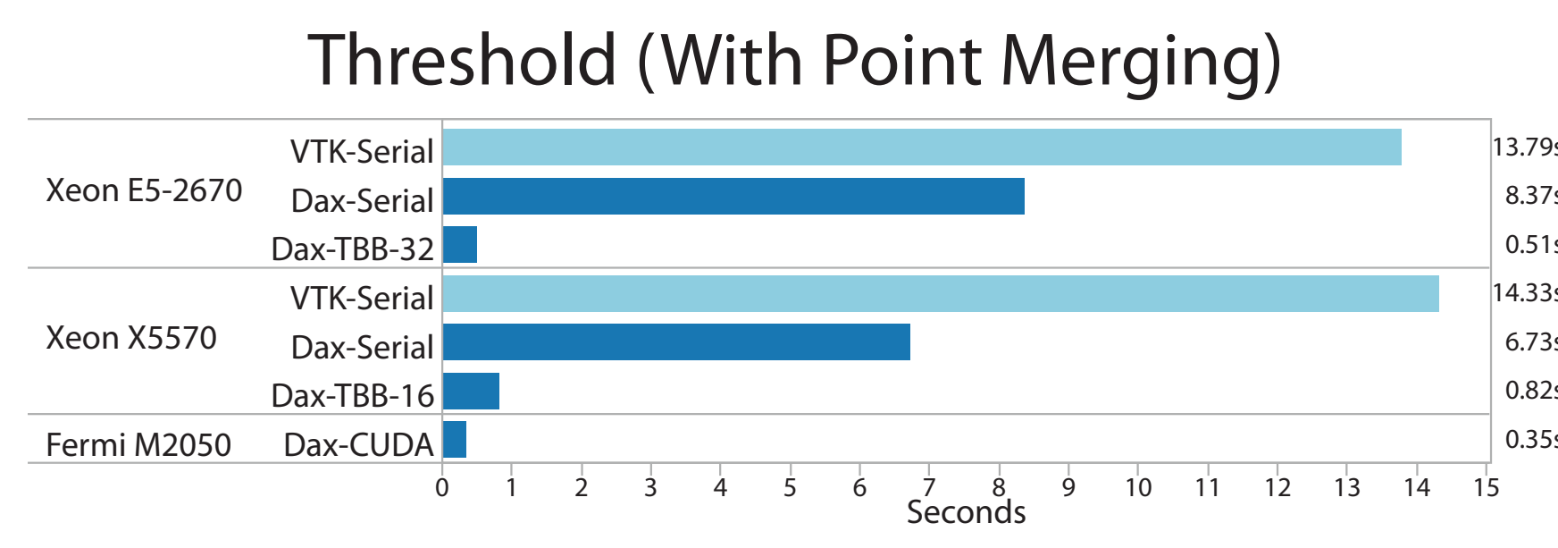


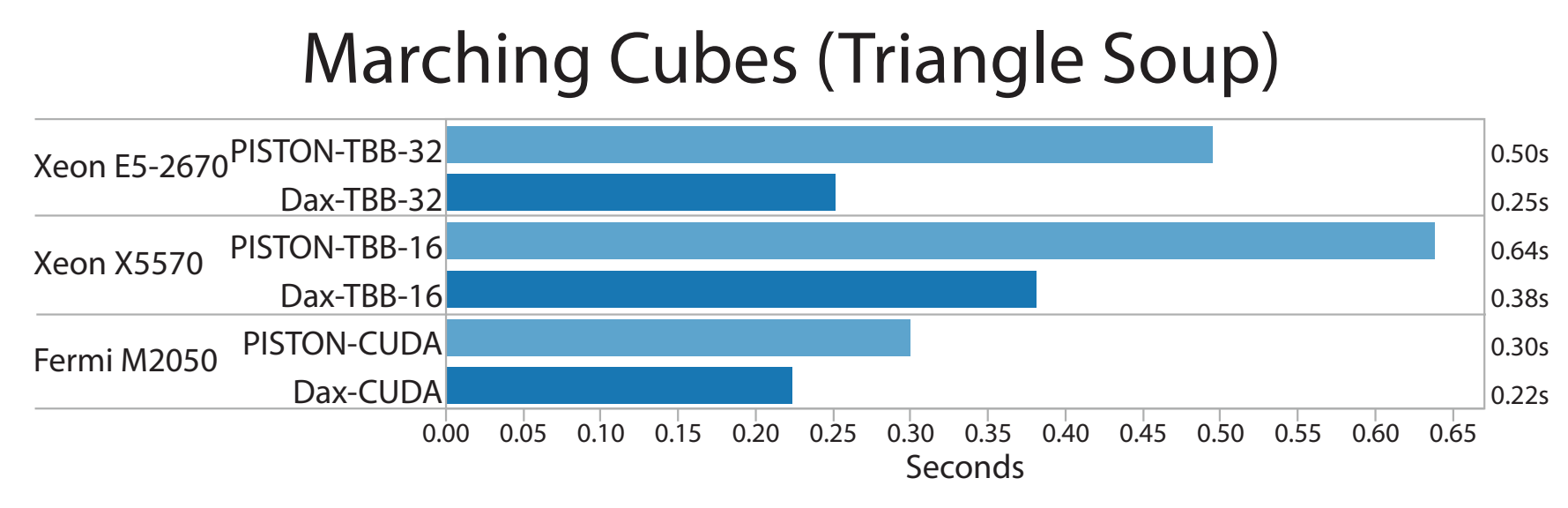
Topological connections are often ignored when results are fed directly to a renderer where elements are visually connected. However, users of current production tools expect the ability to perform a sequence of operations for more versatile visualizations. For example, the data shown here is a contour operation combined with vertex welding, coarsening, subdivision, curvature estimation, and smoothed surface normals. The Dax Toolkit's topological capabilities make these possible.

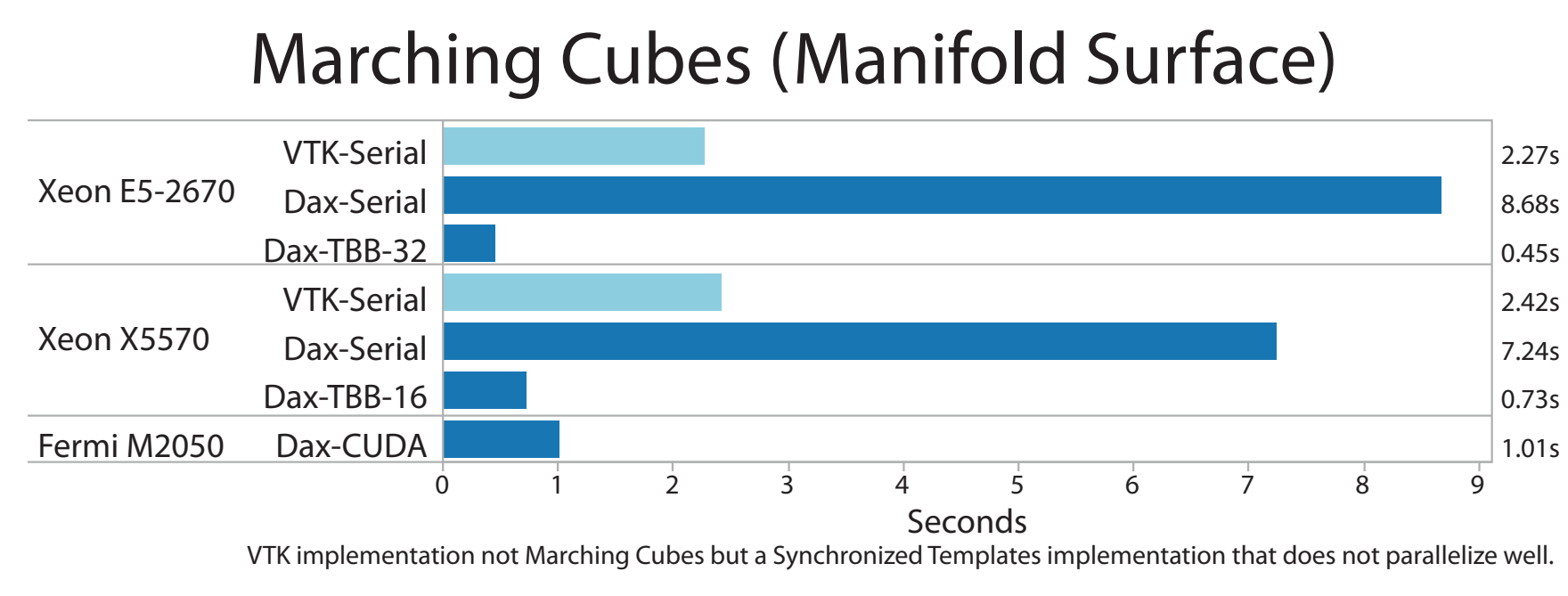
Implementation Efficiency

Our performance tests show both that our implementations of the threshold and marching cubes algorithms are effective in providing efficient parallel performance and that parallel complexity can be hidden beneath a generic templated programming interface. In addition to demonstrating the base performance of our code on many devices, we also compare to VTK and PISTON as good representations of the state of the art.



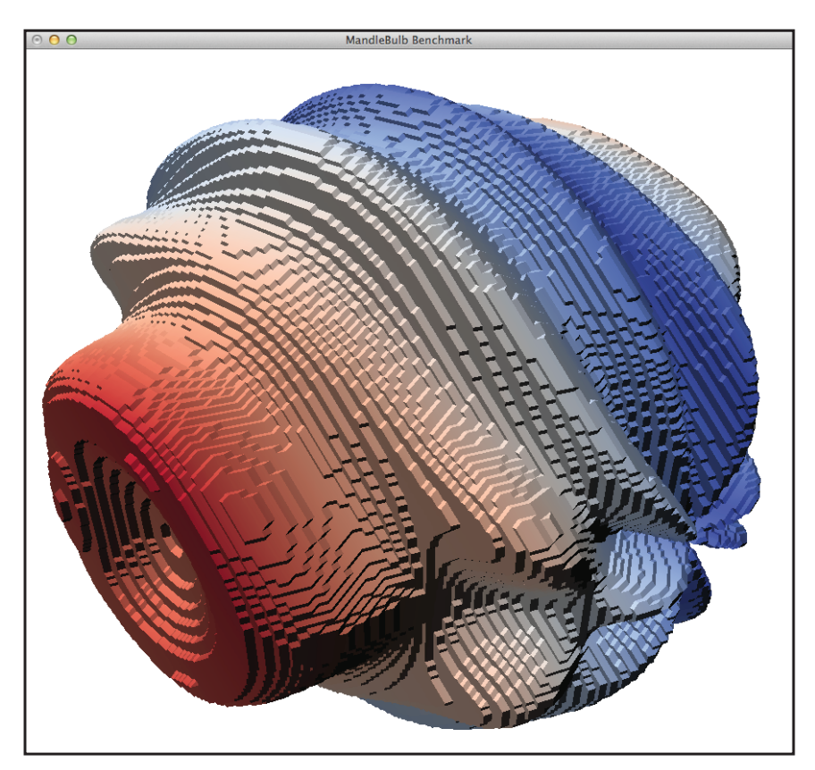






OpenGL Interop

The Dax Toolkit provides simplified integration with OpenGL-based rendering systems with abstracted interop capabilities. Using our TransferToOpenGL service, any Dax array can be bound to an OpenGL object and directly rendered. This interop service works seamlessly with any device on which the array was created. Thus, no customized code is required to manage whether the array exists in CUDA or CPU memory.



Interop with CUDA

```
dax::opengl::TransferToOpenGL(
  surface.Colors, bufferHandles[1]);
```

Interop with OpenMP

```
dax::opengl::TransferToOpenGL(
  surface.Colors, bufferHandles[1]);
```

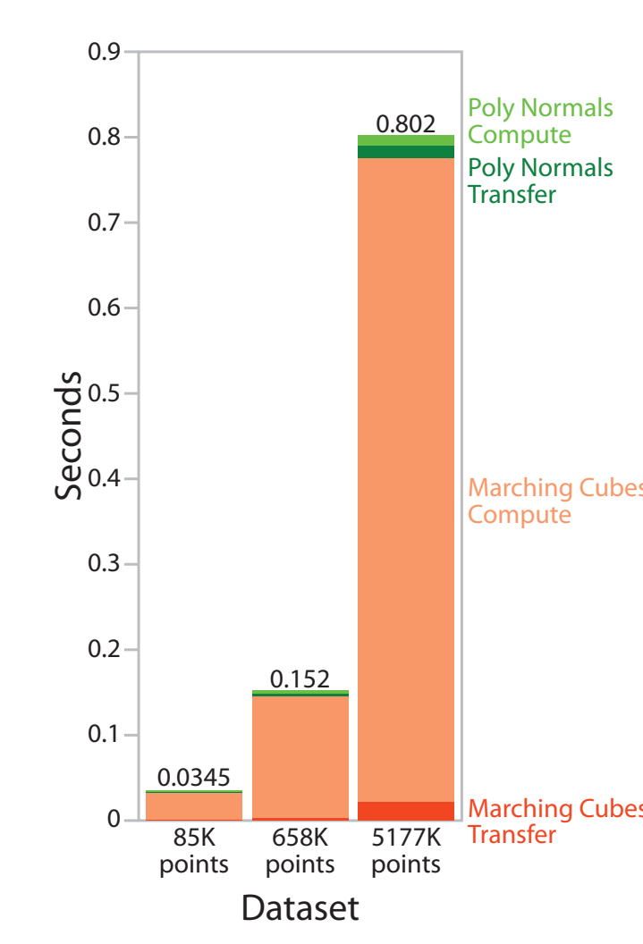
Interop with TBB

```
dax::opengl::TransferToOpenGL(
  surface.Colors, bufferHandles[1]);
```

(Yes, they are all the same.)

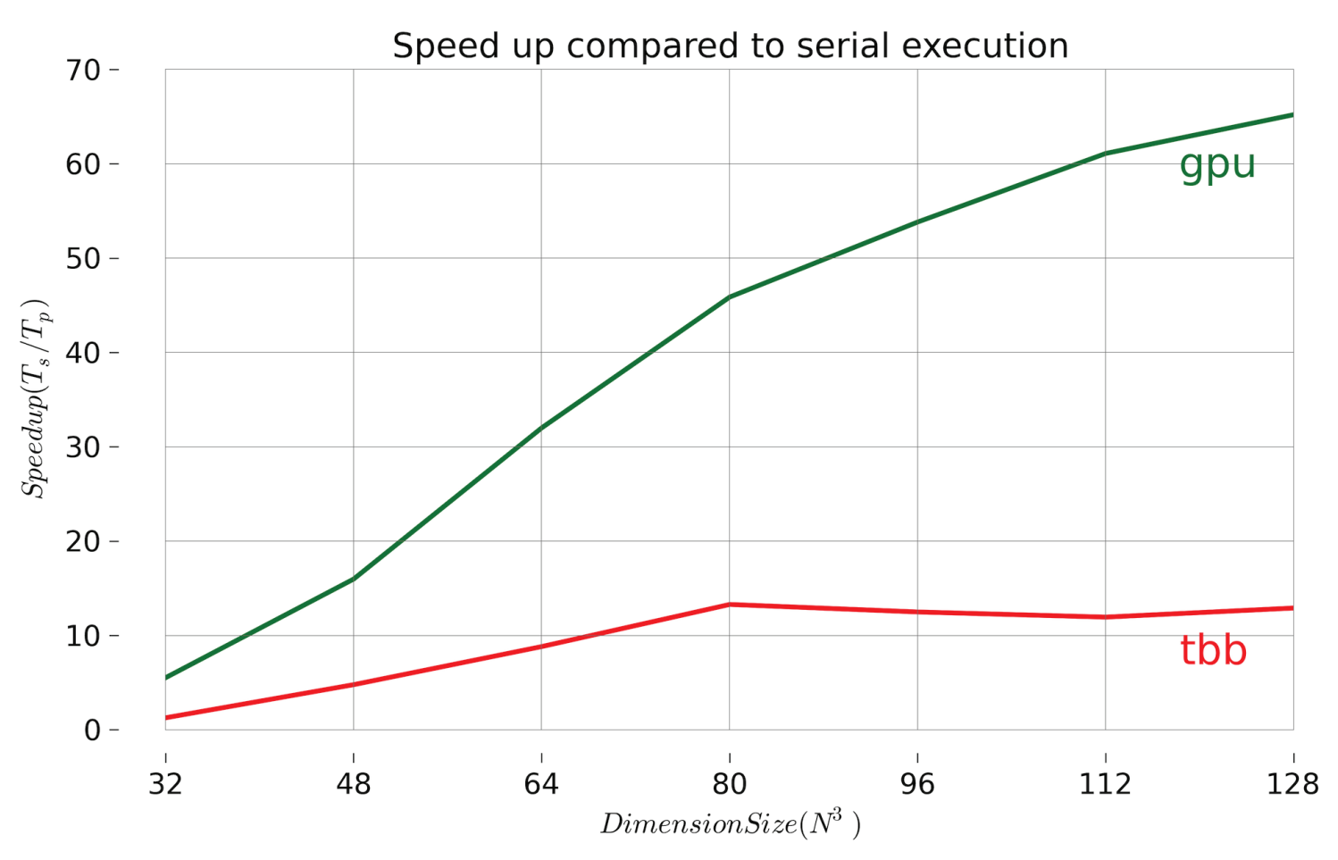
GPU Transfer Time

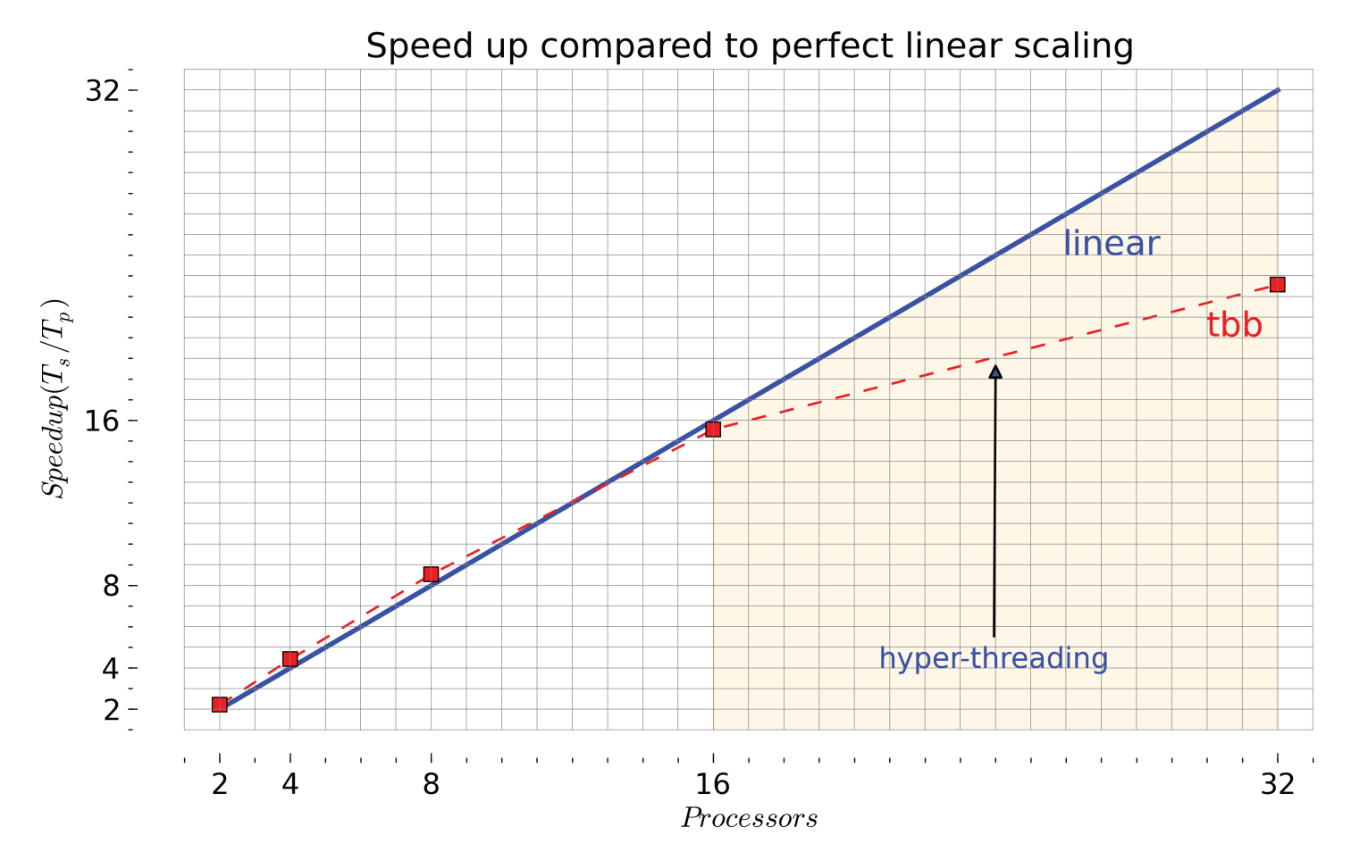
It is "common knowledge" that when leveraging a GPU accelerator, transferring data between host and device is a major bottleneck and should be avoided or hidden whenever possible. However, our informal study shows that load time is inconsequential for larger operations. The chart at right compares the time spent in data transfer and computation for a sequence of common operations. The overall compute time overshadows the time spent transferring data. These results are important when integrating GPU algorithms in object-oriented systems like VTK, ParaView, and VisIt that combine algorithms as independent units. We can simplify the management of GPU memory by using CPU memory as the transport between units without incurring unreasonable overhead.



Application for N-Body Cosmology Simulation

N-body cosmology simulations produce highly irregular and overlapping grids. Analysis of these simulation results requires identifying void, pancake, filament, and clump features. A primary operation of this feature identification is the finding of cells in this irregular mesh, an expensive operation repeated regularly throughout space. A recent project demonstrates that on a small synthetic dataset, using Dax to find cells yields speedups of up to 22× with multiple cores and 65× using a GPU.





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.

This work was supported 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.

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 2013-0930P