Optimizing Threshold for Extreme Scale Analysis

Robert Maynard
Kitware, Inc.

Kenneth Moreland
Sandia National Laboratories

Utkarsh Ayachit
Kitware, Inc.

Berk Geveci
Kitware, Inc.

Kwan-Liu Ma
University of California at Davis

As the HPC community starts focusing its efforts towards exascale, it becomes clear that we are looking at machines with a billion way concurrency. Although parallel computing has been at the core of the performance gains achieved until now, scaling over 1,000 times the current concurrency can be challenging. As discussed in this paper, even the smallest memory access and synchronization overheads can cause major bottlenecks at this scale. As we develop new software and adapt existing algorithms for exascale, we need to be cognizant of such pitfalls. In this paper, we document our experience with optimizing a fairly common and parallelizable visualization algorithm, threshold of cells based on scalar values, for such highly concurrent architectures. Our experiments help us identify design patterns that can be generalized for other visualization algorithms as well. We discuss our implementation within the Dax toolkit, which is a framework for data analysis and visualization at extreme scale. The Dax toolkit employs the patterns discussed here within the framework's scaffolding to make it easier for algorithm developers to write algorithms without having to worry about such scaling issues.

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.

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 2013-0401P

http://daxtoolkit.org

```
// 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);
```

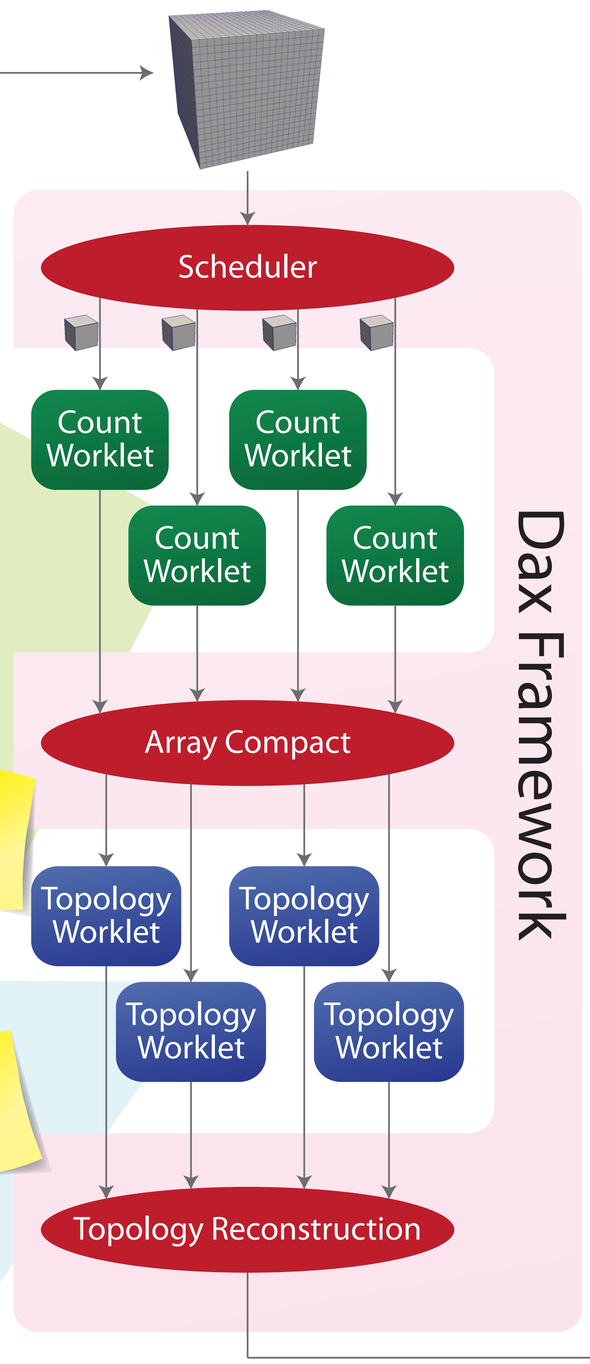
Automatic point merging is optional.

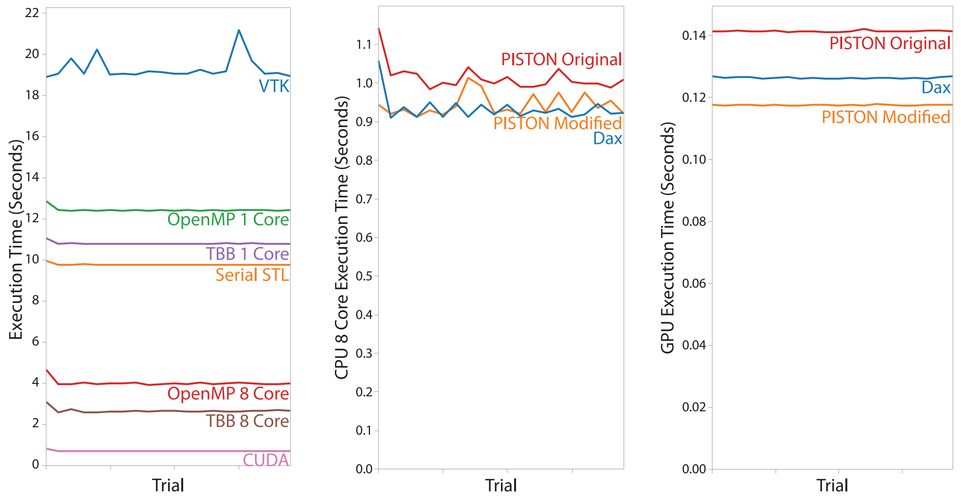
```
template<typename ValueType>
class ThresholdClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology,Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_2);

  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.





Our end performance tests show both that our optimizations to the threshold algorithm are effective in providing efficient parallel performance and that these optimizations 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. (The modified PISTON is changed to make its output compatible with Dax.)