70 YEARS OF CREATING TOMORROW

Los Alamos NATIONAL LABORATORY

On-Demand Unstructured Mesh Translation for Reducing Memory Pressure during In Situ Analysis

J. Woodring[1], J. Ahrens[1], T. Tautges[2], T. Peterka[2], V. Vishwanath[2], B. Geveci[3]

UltraVis '13, November 17, 2013

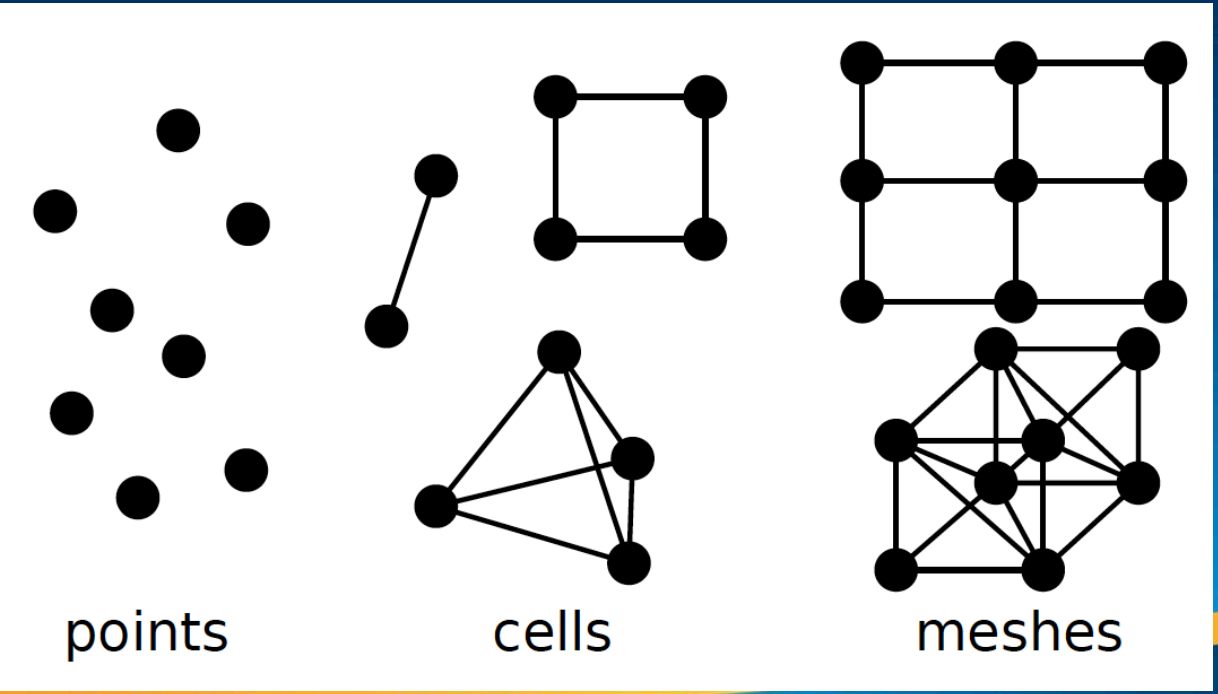
[1]Los Alamos National Laboratory, [2]Argonne National Laboratory, [3]Kitware, Inc.

Memory Pressure in HPC Simulations

- Ratio of available memory to processing elements going down
- Use of in situ analysis and coupled multi-physics codes is going up
- This results in contention on available memory between the coupled codes running in the same address space
- The majority of the memory footprint is the data of the simulation, which is likely a “mesh”

Meshes

- Analysis and simulations code use meshes to represent the data – points and cells with attribute data



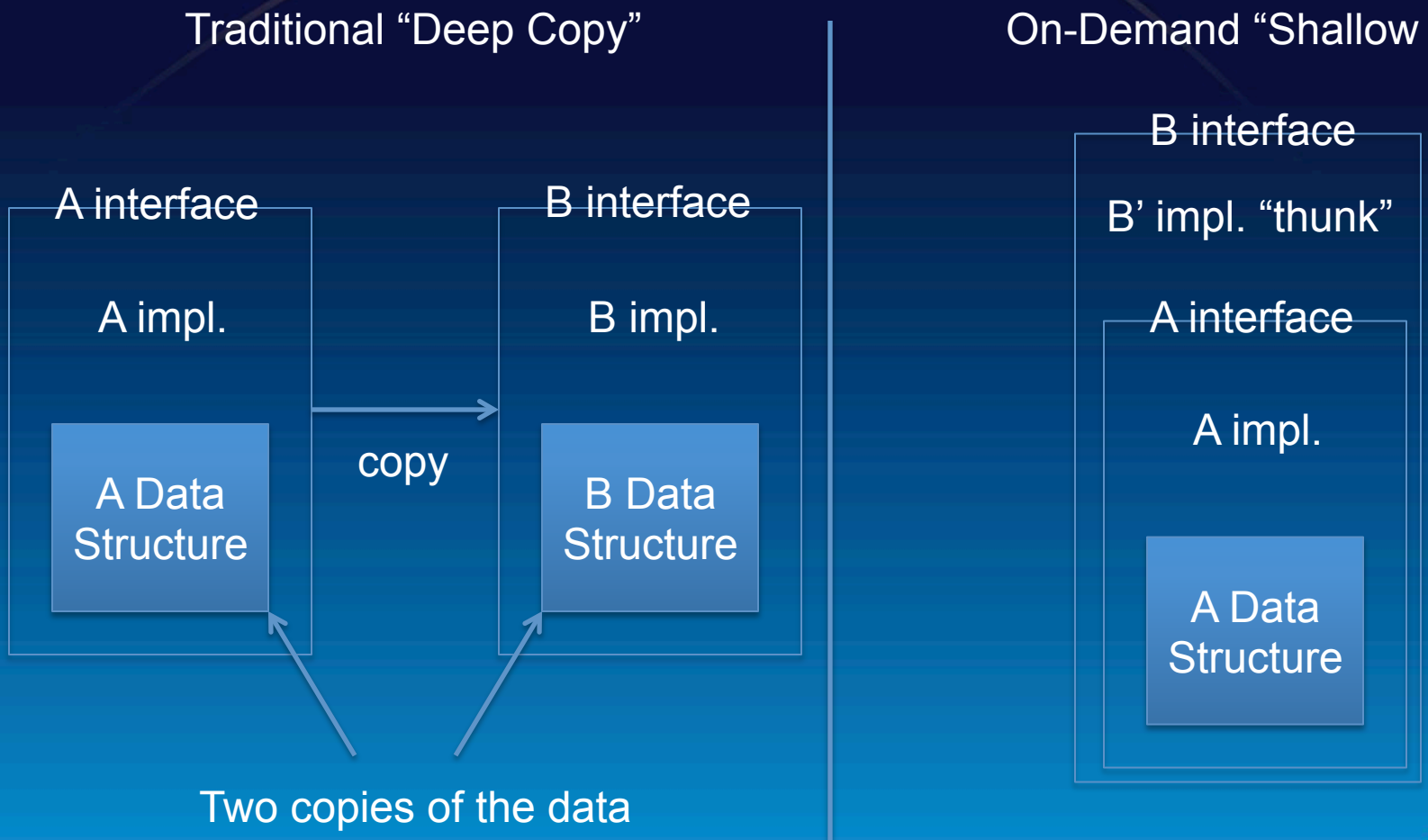
Copying Meshes to Deal with Different Implementations

- The problem is that different codes, in a coupled simulation, will typically use different mesh implementations and interfaces
- This means that for two codes to work together on the same data, the mesh is copied from one implementation to another
- This increases the memory footprint by at least x2, which means then the simulation must run with more processing elements, wasting cycles

How can we share the mesh w/o copying? A few Ideas (not exhaustive)

- Rewrite the coupled codes to use the same mesh data model
  - Thousands of man hours have likely gone into the existing code bases, very non-trivial
- Pass internal data structures by reference
  - Same problem as above, but worse: pushes implementation level details to algorithms
- Write the data to storage and read it back
  - …

Thunking: Native Interfaces, Translating Implementation, One Copy



UNCLASSIFIED

On-demand Translation of Meshes Fine grained, lazy evaluation

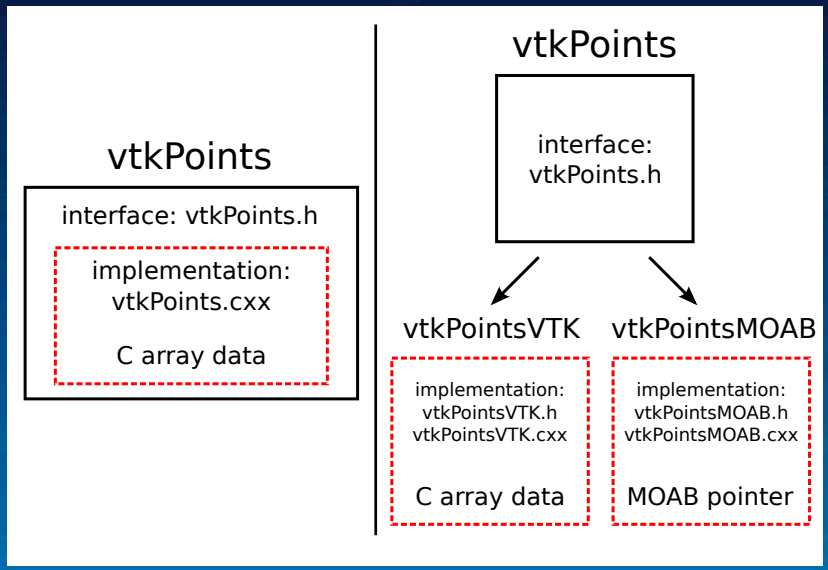
- Benefits
  - Only one copy of the data
  - Don't have to rewrite algorithms
    - Separation of interface and implementation
  - Copying/sharing is fast (deep copy takes time)
  - Automatic updates of a dynamic mesh
- Drawbacks
  - Slows down algorithms due to translation
  - Repeated work

In Situ Coupling, Study on Two Meshes

- MOAB (not the scheduler)
  - Mesh Oriented datABase
  - Implementation of iMesh interface (ITAPS)
  - Simulation mesh
- VTK Unstructured Grid
  - Visualization ToolKit
  - Used in ParaView, VisIt, etc.
  - Analysis mesh
- Goal: Run VTK algorithms on MOAB mesh w/o copy

Create a VTK Unstructured Grid with "a MOAB data structure"

- vtkUnstructuredGrid uses:
  - vtkPoints - points
  - vtkCellArray - cells
  - vtkDataArrays - attributes
  - cell type array
  - cell offset for random access
- Create new implementations of vtkPoints, vtkCellArray, vtkDataArray, & vtkUG that translate from MOAB to VTK

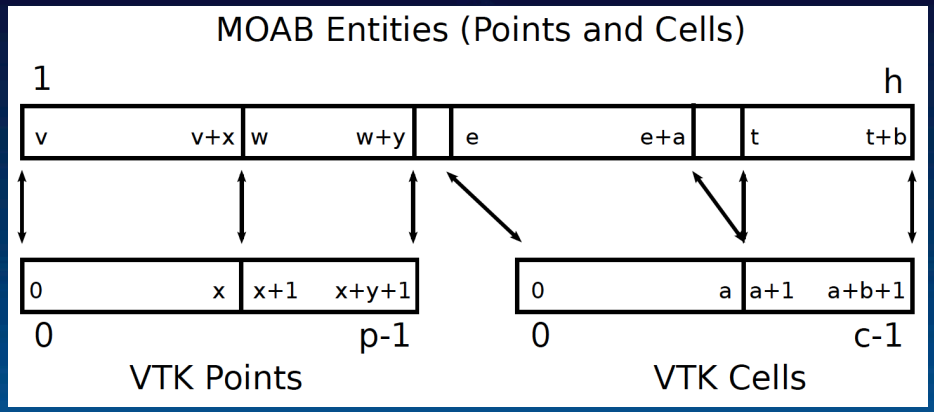


Pseudocode for VTK Mesh Operations (id = point/cell address in mesh)

- Operation called on VTK mesh with VTK id
  - Convert VTK id into MOAB id
  - Call MOAB operation with MOAB id
  - Get MOAB data from MOAB operation
  - Convert MOAB data to VTK data (especially important for cell connectivity arrays, have to translate point ids from MOAB addresses to VTK addresses – other caveats like cell type)
  - Return VTK data

Address (id) Translation

- Translating between MOAB and VTK interfaces requires address translation
- MOAB has a unified address space for points and cells, VTK doesn't
- MOAB addresses can be sparse, VTK addresses are dense
- Done at run-time with a range map and lower bound



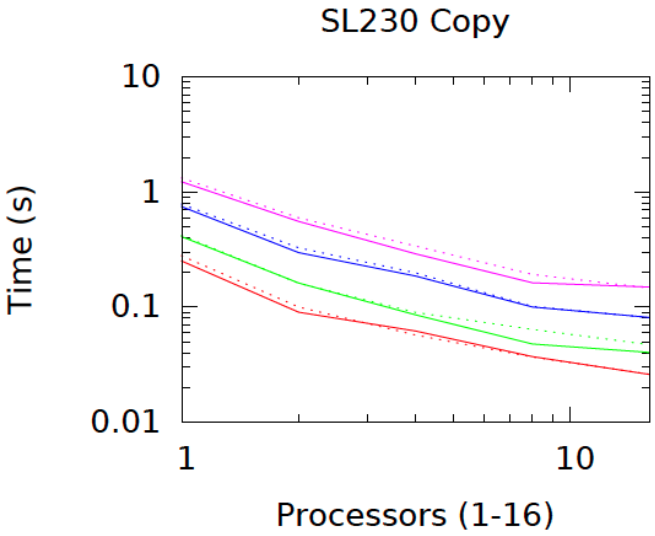
$$m[upper_s] = lower_d - upper_d$$

$$dest = m.lower\_bound(src) + src$$

Performance Tests Compare "Deep Copy" vs. On-Demand

- Memory savings
- Overhead on visualization algorithms
- Two single node tests "SL230" & "DL980" (1-16 processors and 1-64 processors) and "ML" cluster test (16 to 512 processors)
- 1 to 8 million tetrahedral MOAB mesh on single node, 16 to 512 million quadrilateral MOAB mesh on cluster – only 1 attribute in the mesh
- VTK algorithms: Touch (read) all data, slice, clip, isosurface, threshold, surface rendering
- Also, compare unmodified VTK vs. "refactored" VTK

What's the overhead of the virtualized functions? (Comparing 2 deep copies)



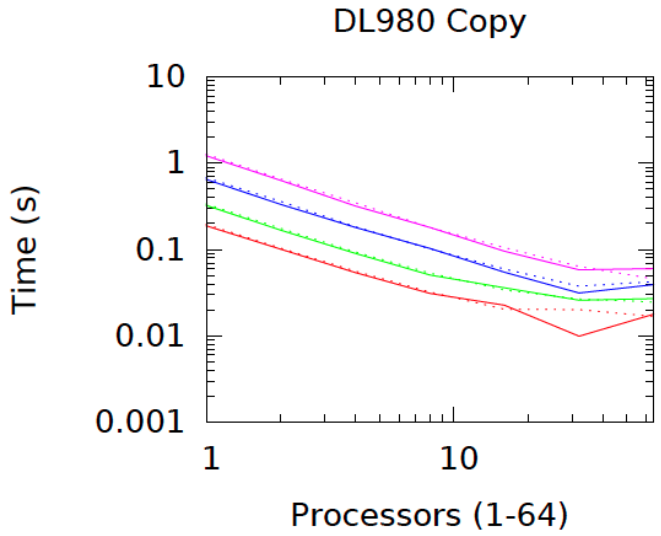


Figure 3: Refactored deep copy compared to default VTK deep copy. SL230 is 1.07 times slower and DL980 is 1.04 times slower.

Dashed – refactored VTK, Solid – default VTK, red – 1 million tets, green – 2 million tets, blue – 4 million tets, purple – 8 million tets

What's the overhead of the virtualized functions? (Comparing 2 deep copies)

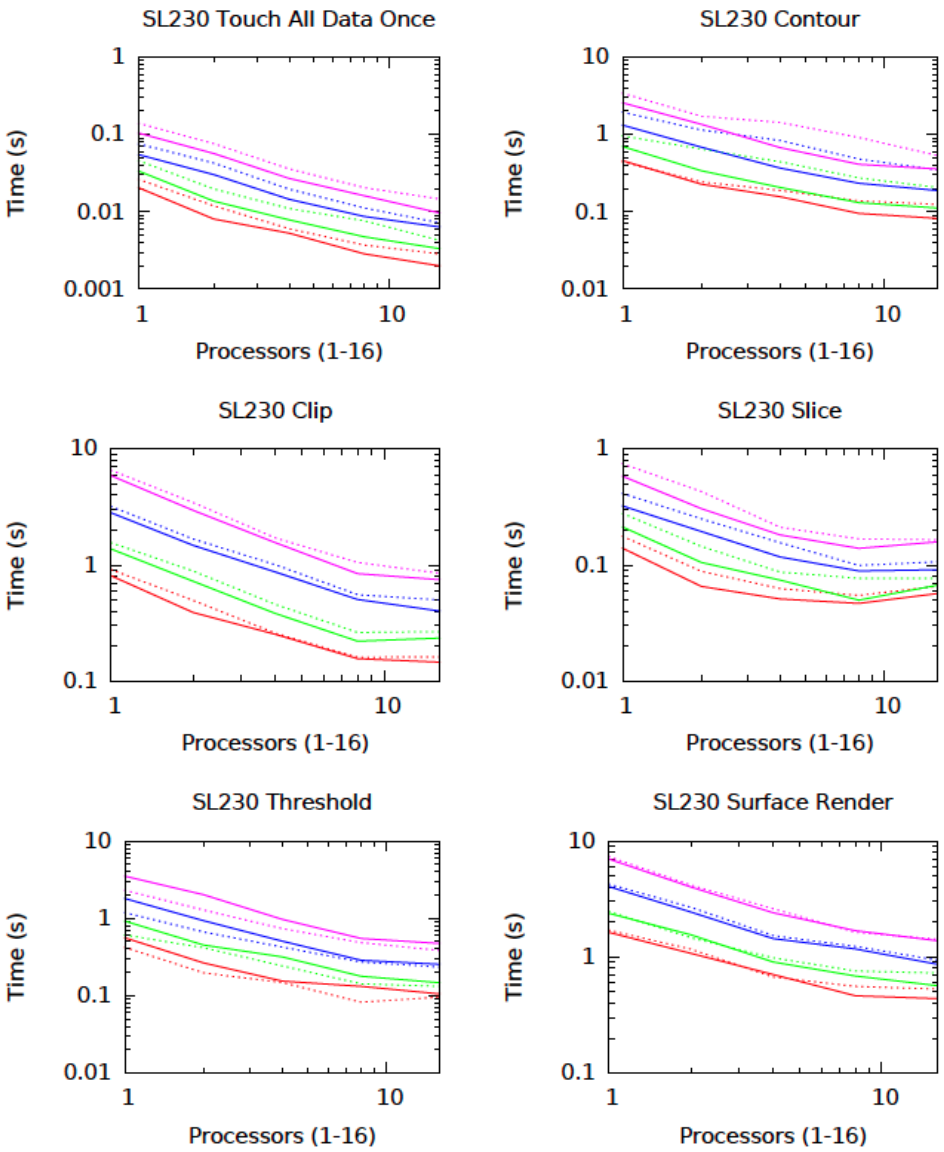


Figure 4: SL230 refactored compared to default VTK filtering. Touching is 1.35 times slower, isocontouring is 1.53 times slower, clipping is 1.14 times slower, slicing is 1.26 times slower, thresholding is 1.39 times faster, and rendering is 1.05 times slower.

Dashed – refactored VTK, Solid – default VTK, red – 1 million tets, green – 2 million tets, blue – 4 million tets, purple – 8 million tets

What's the overhead of the virtualized functions? (Comparing 2 deep copies)

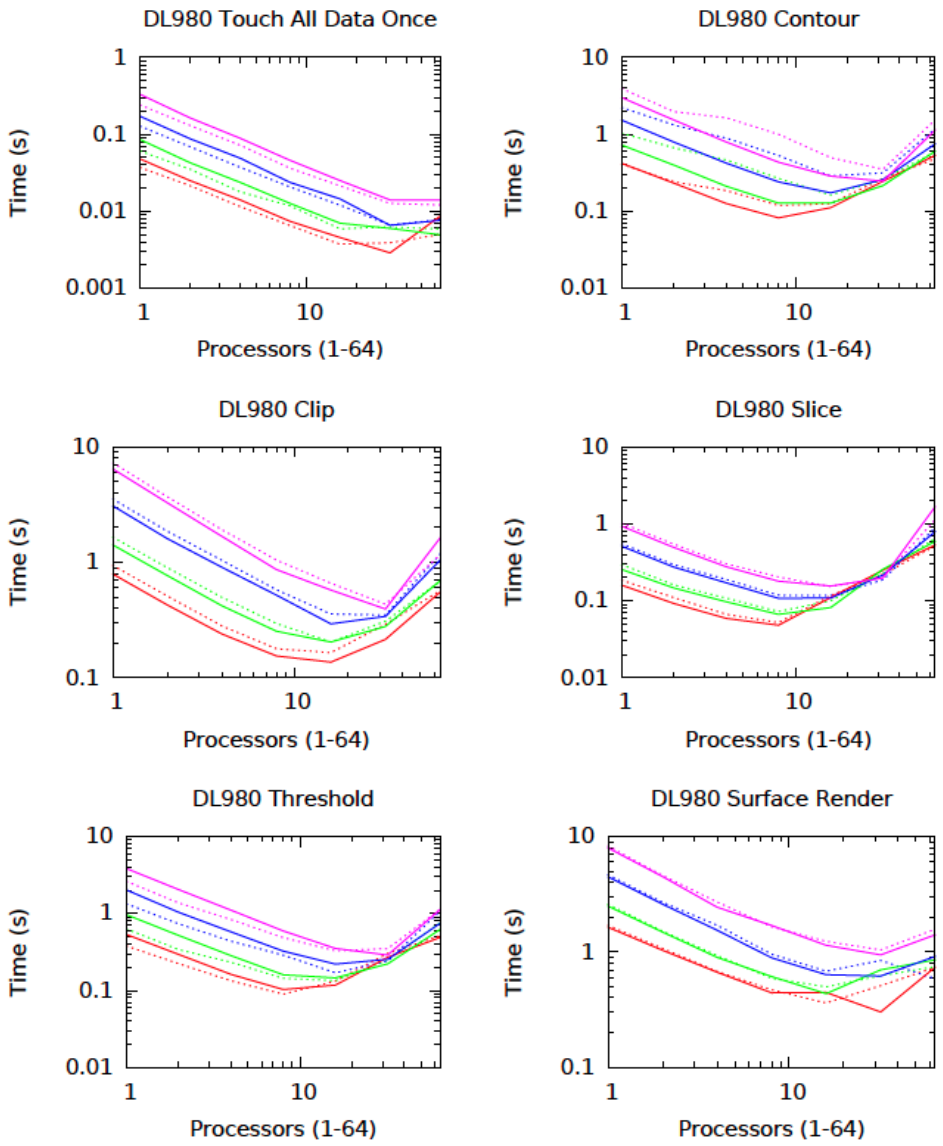


Figure 5: DL980 refactored compared to default VTK filtering. Touching is 1.28 times faster, isocontouring is 1.46 times slower, clipping is 1.12 times slower, slicing is equal in speed, thresholding is 1.24 times faster, and rendering is 1.04 times slower.

Dashed – refactored VTK, Solid – default VTK, red – 1 million tets, green – 2 million tets, blue – 4 million tets, purple – 8 million tets

How much faster is the "copy"? (on-demand vs. deep copy) – also note, the on-demand version only has to be done once

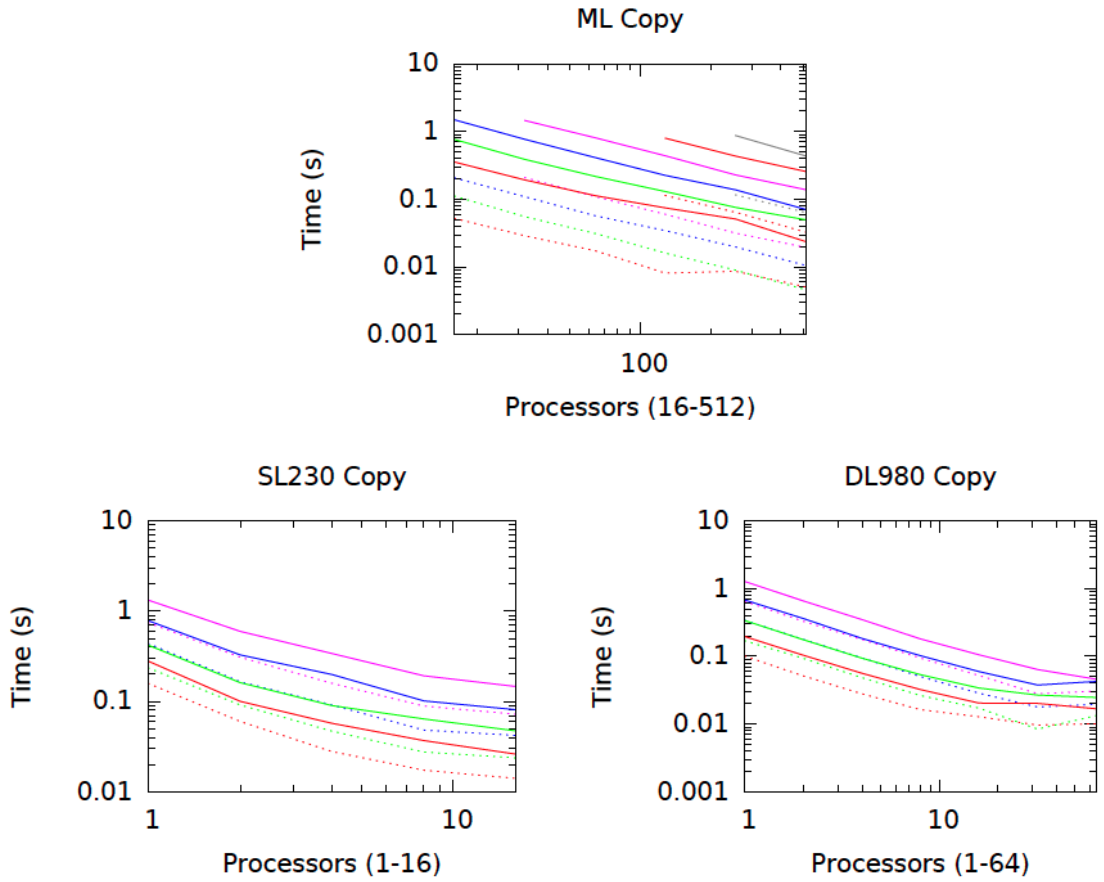


Figure 6: Moonlight shallow copy compared to deep copy is 7.11 times faster, SL230 is 1.88 times faster, and DL980 is 1.98 times faster.

ML: Dashed – on-demand, Solid – deep copy, red – 16 million quads, green – 32 million quads, blue – 64 million quads, purple – 128 million quads, orange, 256 million quads, grey – 512 million quads

SL230 & DL980: Dashed – on-demand, Solid – deep copy, red – 1 million tets, green – 2 million tets, blue – 4 million tets, purple – 8 million tets

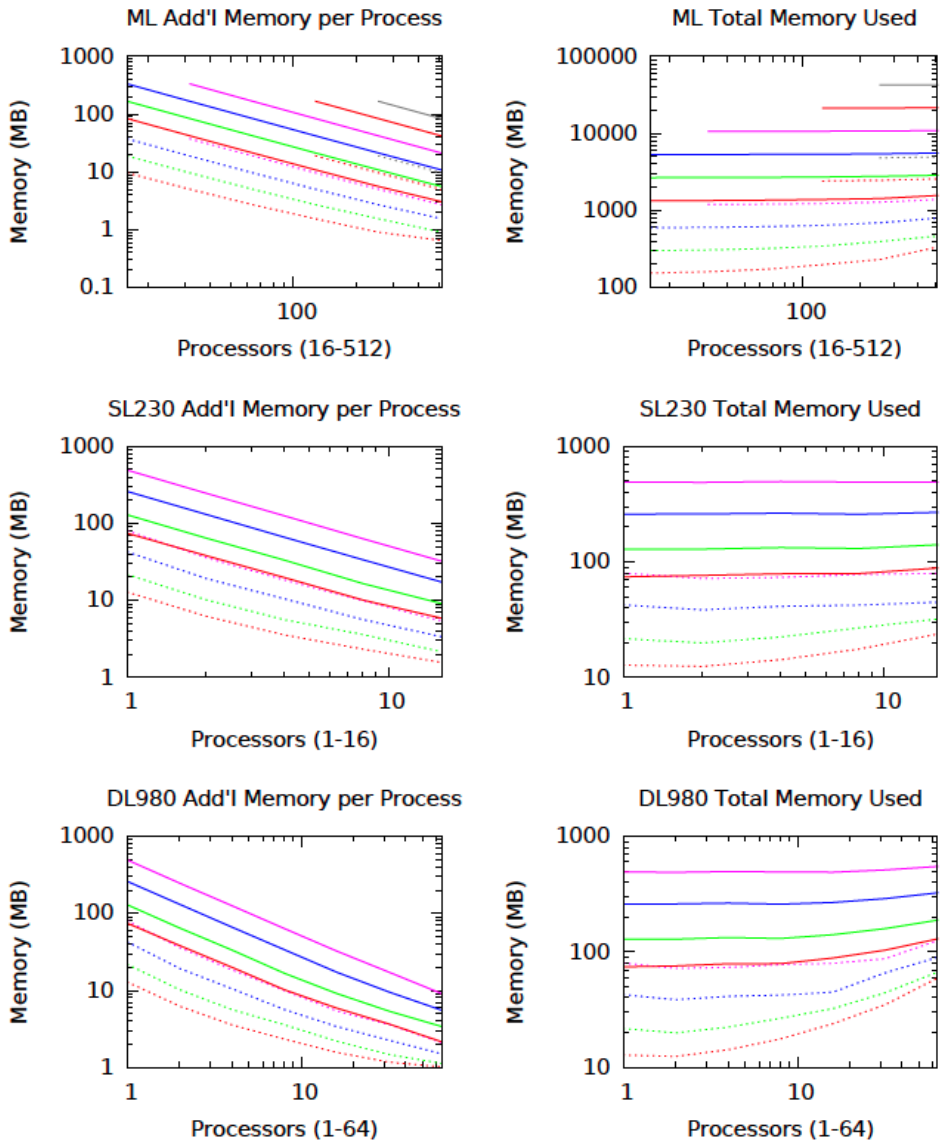


Figure 7: Moonlight shallow copy compared to deep uses 8.45 times less memory, for a quadrilateral mesh. SL230 uses 6.07 times less memory and DL980 uses 5.17 times less memory, for a tetrahedral mesh.

How much memory do we save? (on-demand vs. deep copy)

ML: Dashed – on-demand, Solid – deep copy, red – 16 million quads, green – 32 million quads, blue – 64 million quads, purple – 128 million quads, orange, 256 million quads, grey – 512 million quads

SL230 & DL980: Dashed – on-demand, Solid – deep copy, red – 1 million tets, green – 2 million tets, blue – 4 million tets, purple – 8 million tets

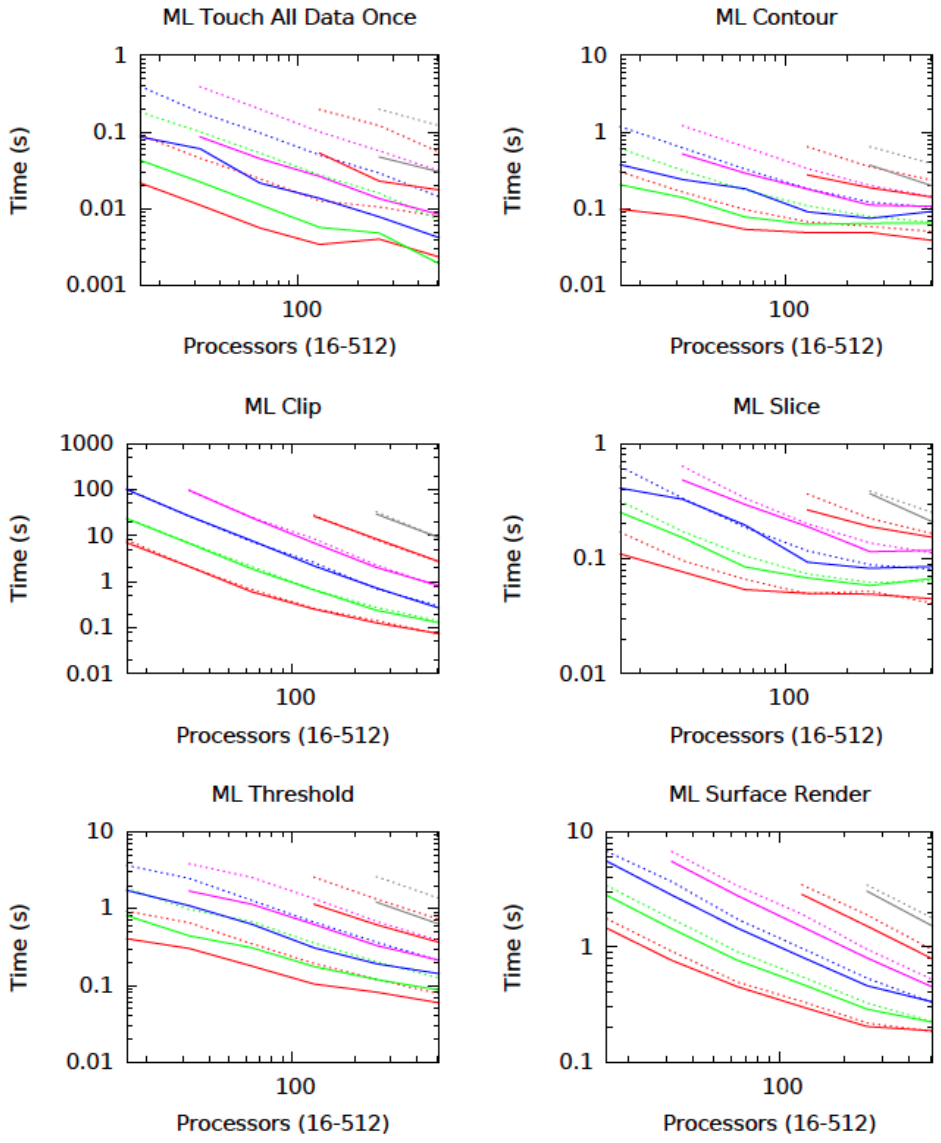


Figure 8: ML shallow copy filter timings compared to deep: touching is 4.12 times slower, isocontouring is 2.13 slower, clipping is 1.02 times slower, slicing is 1.19 times slower, thresholding is 2.16 times slower, and rendering is 1.21 times slower.

How much slower are the algorithms? (on-demand vs. deep copy) ML cluster

ML: Dashed – on-demand, Solid – deep copy, red – 16 million quads, green – 32 million quads, blue – 64 million quads, purple – 128 million quads, orange, 256 million quads, grey – 512 million quads

How much slower are the algorithms? (on-demand vs. deep copy) SL230

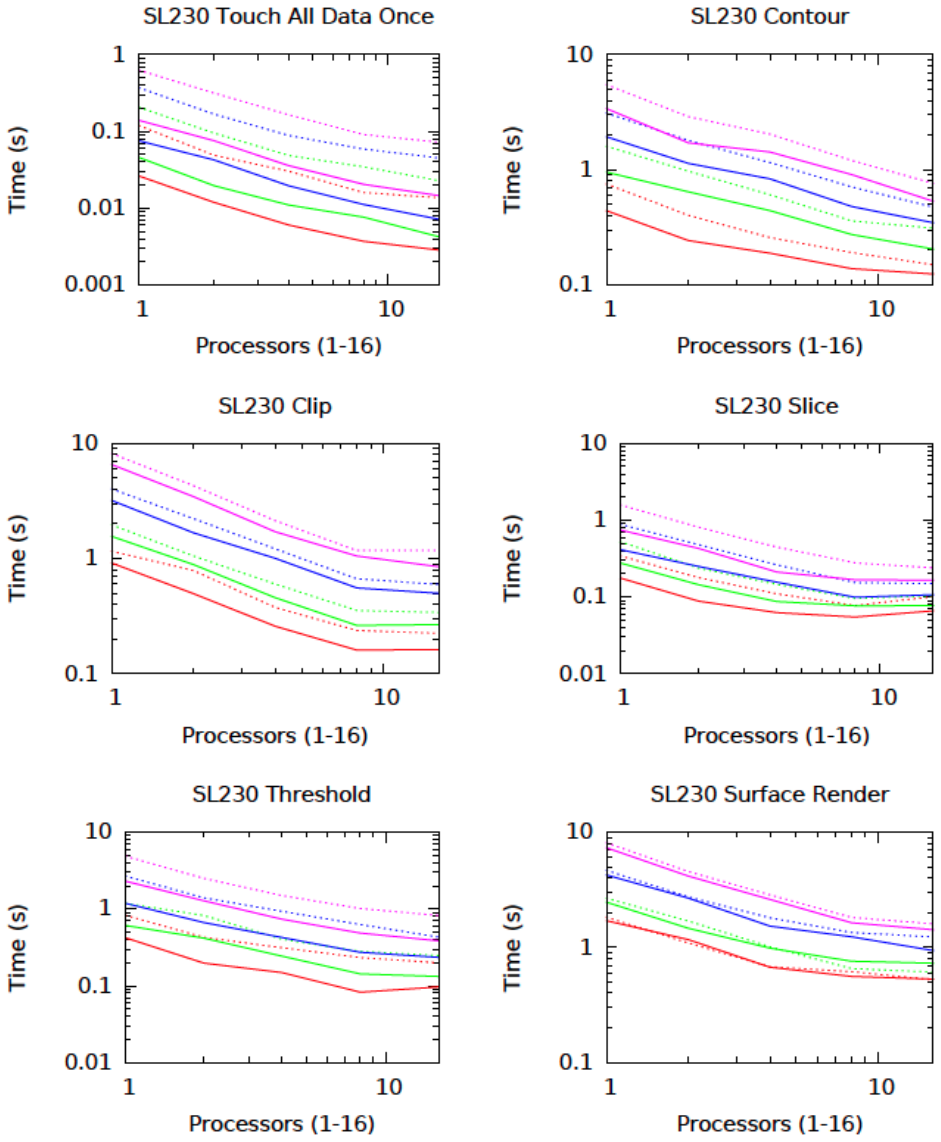


Figure 9: SL230 shallow copy filter timings compared to deep: touching is 4.54 times slower, isocontouring is 1.54 times slower, clipping is 1.26 times slower, slicing is 1.87 times slower, thresholding is 2.08 times slower, and rendering is 1.08 times slower.

SL230 & DL980: Dashed – on-demand, Solid – deep copy, red – 1 million tets, green – 2 million tets, blue – 4 million tets, purple – 8 million tets

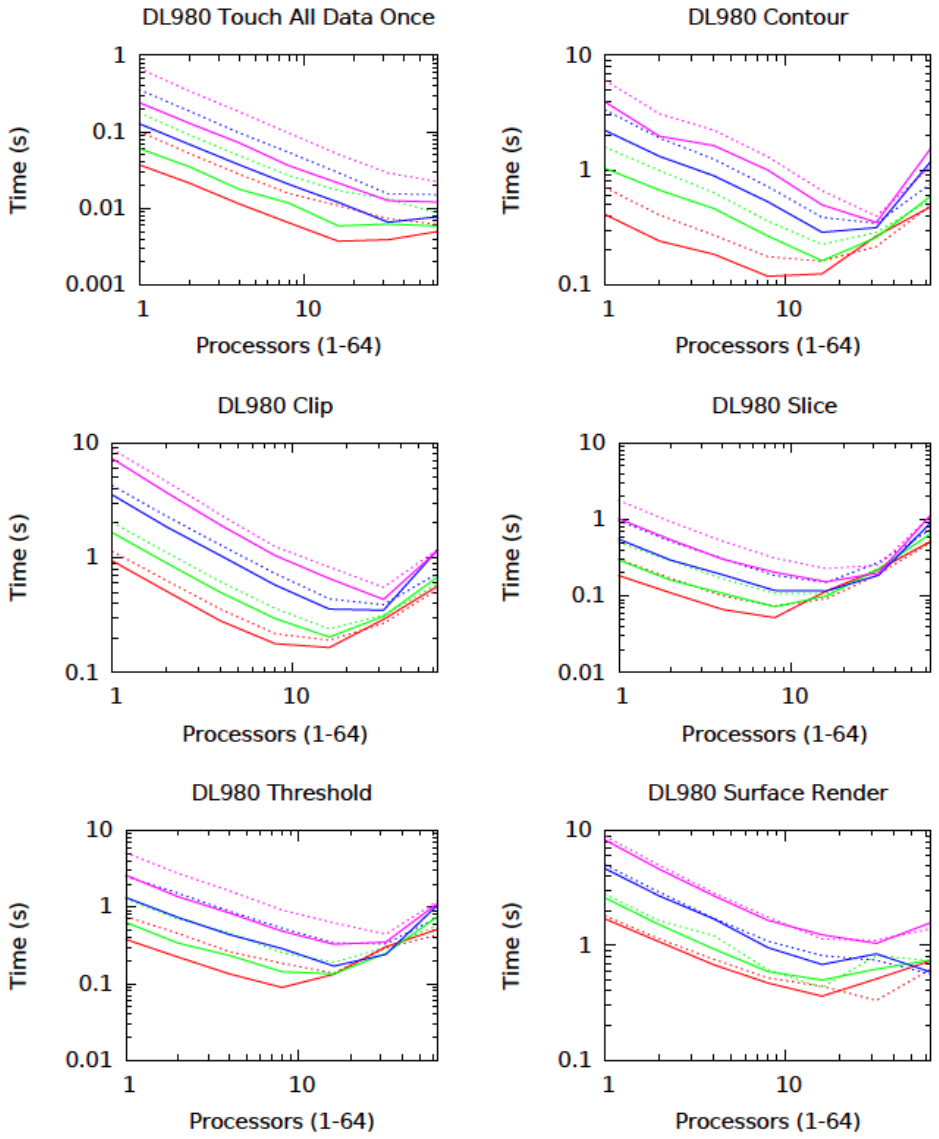


Figure 10: DL980 shallow copy filter timings compared to deep: touching is 2.65 times slower, isocontouring is 1.33 times slower, clipping is 1.17 times slower, slicing is 1.32 times slower, thresholding is 1.65 times slower, and rendering is 1.06 times slower.

How much slower are the algorithms? (on-demand vs. deep copy) DL980

SL230 & DL980: Dashed – on-demand, Solid – deep copy, red – 1 million tets, green – 2 million tets, blue – 4 million tets, purple – 8 million tets

On-Demand vs. Deep Copy Summary
We trade memory savings for speed loss

- Save on average 5 to 9 times the memory footprint – with only one attribute! This is worst case savings, it gets better with more attributes
- Algorithms (ignoring the read test) are only:
  - 1.02 to 2.16 times slower on ML
  - 1.08 to 2.08 times slower on SL230
  - 1.06 to 1.65 times slower on DL980
- Operations are in the seconds… so this is splitting hairs worrying about the speed in this case with such a large memory savings

Future Work

- Kitware has an different implementation that is checked into VTK master
  - Need to update the proxy application to test against Kitware's implementation, also, and release application to public/open source
- Optimize to test against multi-physics coupling or any algorithms that make multiple passes
- Possibly optimize by using compiler tricks to overlap translation with computation

Questions?

- Acknowledgments
  - Department of Energy ASCR
  - CESAR (Center for Exascale Simulation of Advanced Reactors) Office of Science Co-Design Center