# Unstructured Mesh Handling for Extreme-Scale Computing

Timothy J. Tautges

Computational Scientist

Mathematics & Computer Science Division

Argonne National Laboratory

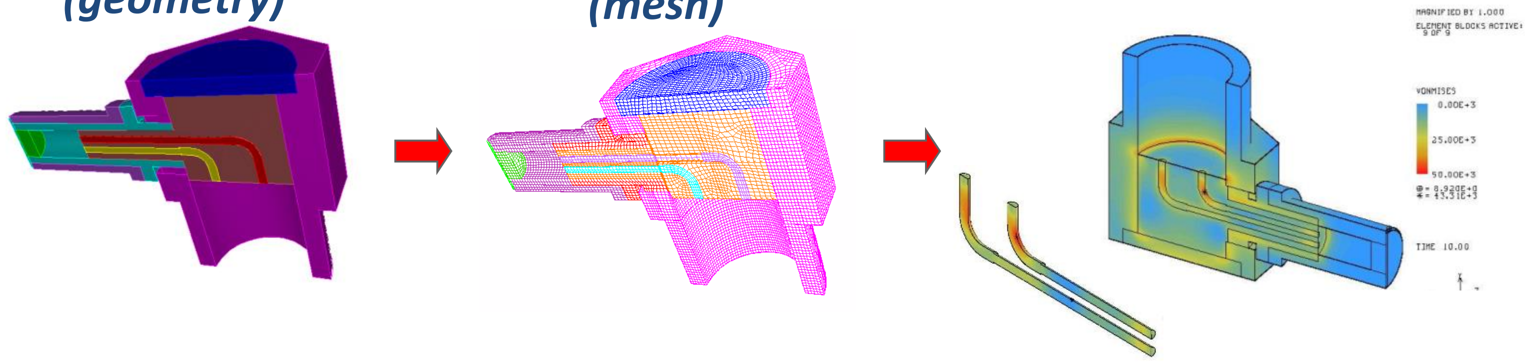
## Outline

- MOAB
  - Overview: mesh & simulation
  - 2-slide overview
  - Data model
  - Basic mesh access
  - Sets & tags
  - Parallel mesh access
  - iMeshP
- CGM/Lasso
  - CGM 1-slide overview
  - Lasso 1-slide overview
- Usage: MOAB-native tools
- Usage: mbpart / Zoltan
- Usage: mbcoupler

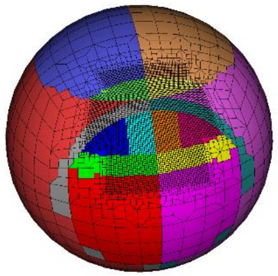
# Introduction

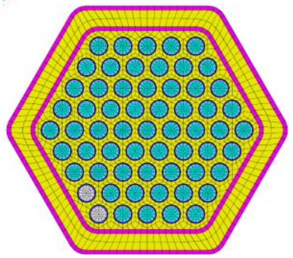
- PDE-based simulation discretizes PDEs over a discrete representation of the spatial and often time domain, and solves for specific discrete model(s)

***Continuous domain (geometry)*** ***Discrete domain (mesh)*** ***FEA***



- Sometimes geometric details of the spatial domain are important, sometimes not
  - MPP-enabled resolution should resolve geometric features (where possible & useful?)
- Depending on the geometric features & resolution requirements, generating the mesh can be either trivial or not



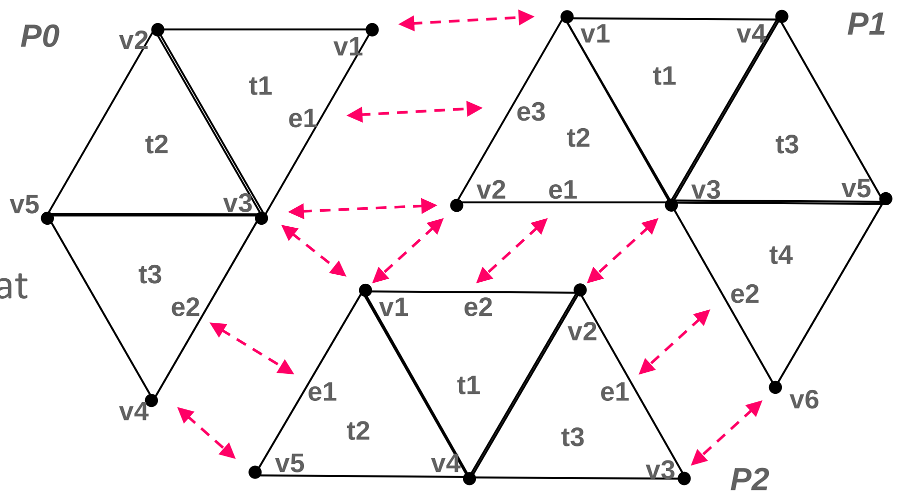


- Mesh, and data on the mesh, are involved in simulation at the front (generation), middle (simulation), and back (viz & data analysis)

# Mesh-Oriented datABase (MOAB)

- Library for representing, manipulating structured, unstructured mesh models
- Supported mesh types:
  - FE zoo (vertices, edges, tri, quad, tet, pyramid, wedge, knife, hex)
  - Polygons/polyhedra
  - Structured mesh
- Implemented in C++, but uses array-based storage model
  - Efficient in both memory and, for set-based access, in time
- Mesh I/O from/to various formats
  - HDF5 (custom), vtk, CCMIO (Star CD/CCM+), Abaqus, CGM, Exodus
- Main parts:
  - Core representation
  - Tool classes (skinner, kdtree, OBBtree, ParallelComm, …)
  - Tools (mbsize, mbconvert, mbzoltan, mbcoupler, …)
- Parallel model supports typical element-based decompositions, with typical mesh-based functions (shared interface, ghost exchange, ownership queries)
- Runs on 32k+ cores

# Parallel Mesh Model



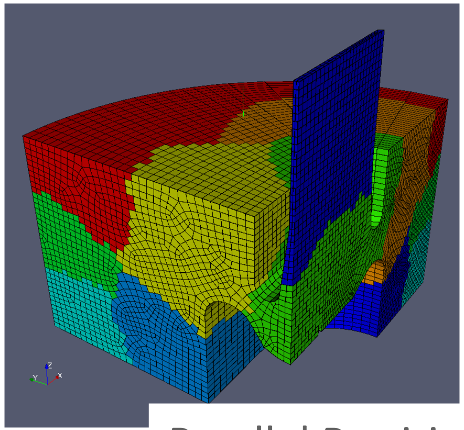
- Definitions:
  - Element-based partition:
    decomposition of mesh over processors such that each element assigned to exactly one proc/part, with vertices shared between parts
  - Shared entity: an entity represented on multiple procs
  - (Part) interface entity: entity shared by multiple parts (vertices, edges, faces in a 3D mesh & element-based partition)
  - Owned entity, owner: each mesh entity owned by exactly one proc
  - Ghost entity: shared non-owned entity (sometimes referred to as “halo”)
- “Degree” of parallel-ness depends on application requirements, and can be adjusted as needed during calculation
  - Duplicated model on every proc
  - Domain-decomposed
  - Shared vertices, non-vertices
  - 1 or more layer of ghost elements

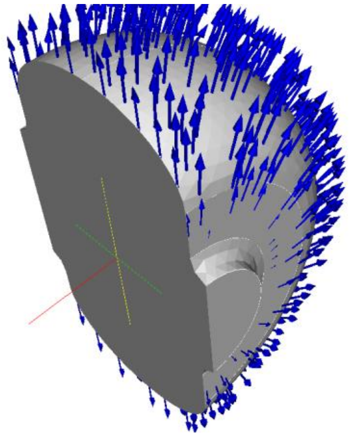
## MOAB Parallel Mesh Model

- Parallel model based on element-based partition
  - Each element assigned to exactly one part, with entities optionally shared between parts/procs
  - Arbitrary number of layers of ghost elements
- Supported parallel mesh constructs:
  - For each shared entity, every sharing proc knows all other sharing procs & handles on those
  - Sharing data stored as either single int/handle (shared with 1 other proc) or mult sharing procs/handles
  - Ghost/owned status also stored
  - Stored in 1-byte ‘pstatus’ bitmask tag
- Parallel model usually initialized by loading from some decomposition in file
  - Can be any subset structure that’s a “covering” (each entity in exactly 1 subset)
  - Material set, geometric volume, or Zoltan-generated partitioning
- Single-file parallel read/write using parallel HDF5
- All parallel functionality usually accessed through ParallelComm class

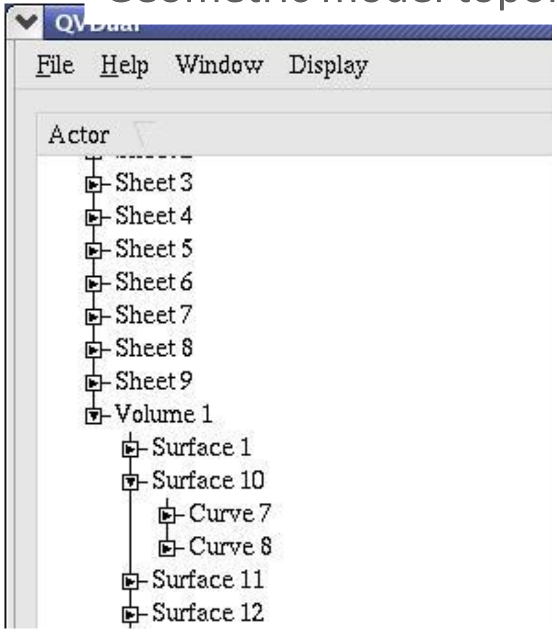
# MOAB Data Model

- 4 basic types of data:
  - Entities (FE zoo, polygons, polyhedra)
  - Sets (collections of entities & sets, parent/child links)
  - Tags (annotation of data on other 3)
  - Interface (OOP, owns data)
- Tags used for both fine-grained and coarse-grained data
  - Fine grained: vertex-based temperature
  - Coarse-grained: provenance of mesh
- Sets + tags used for a variety of mesh groupings


Parallel Partitions

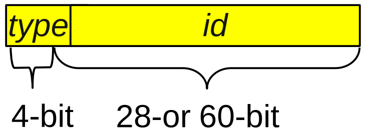

Design velocities

Geometric model topology

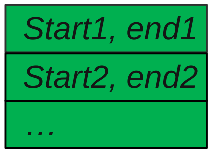


# MOAB Entity Storage

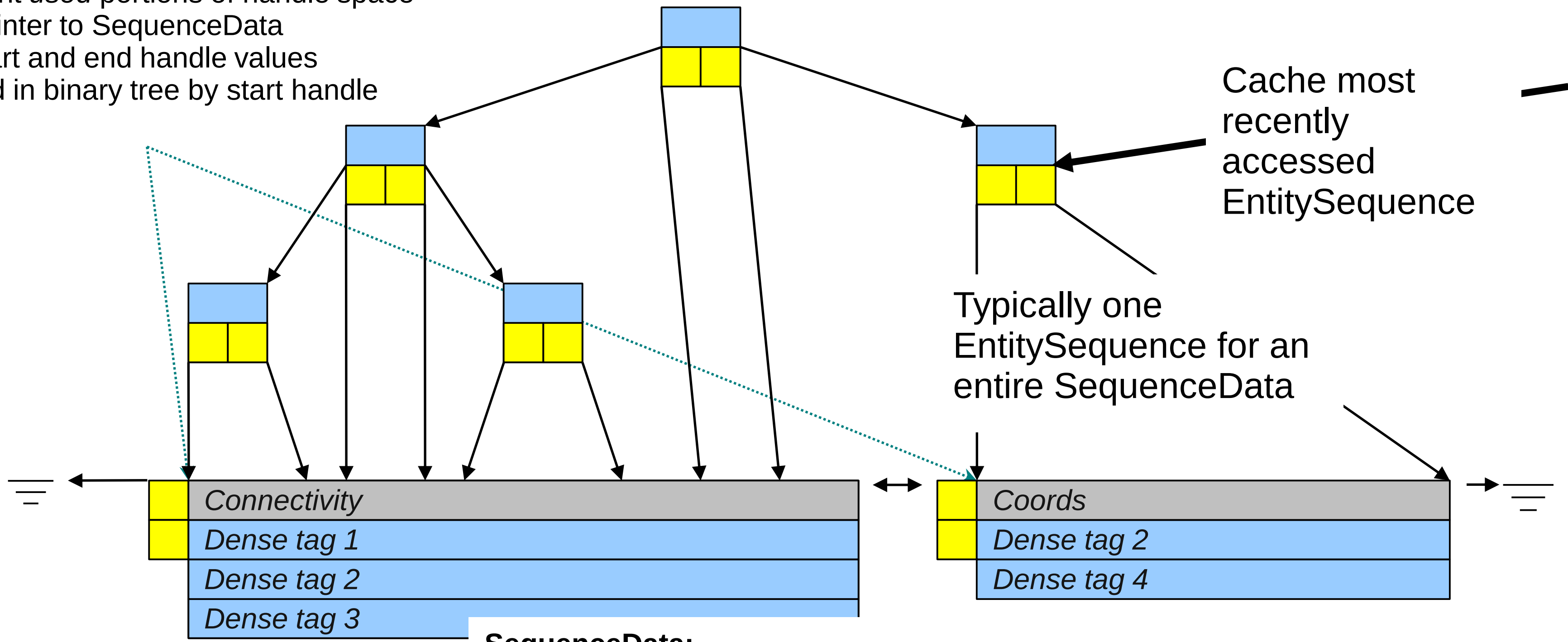
**EntityHandle:**
- Fundamental unit of access in MOAB Bitmask of type, id



**Range:**
- Class for storing lists of handles
- Near constant-size for near-contiguous subranges of handles
- Methods for efficient Booleans on lists



**EntitySequence:**
- Represent *used* portions of handle space
- Have pointer to SequenceData
- Have start and end handle values
- Arranged in binary tree by start handle


Cache most recently accessed EntitySequence
Typically one EntitySequence for an entire SequenceData
Connectivity
Dense tag 1
Dense tag 2
Dense tag 3
Coords
Dense tag 2
Dense tag 4

*Linked list of all SequenceDatas for a single entity type*

**SequenceData:**
- Represent *allocated* portions of handle space
- Have start and end handle
- Coordinates or Connectivity, + Dense Tag Data

# MOAB Mechanics (I)

- MOAB implemented in C++, but internally uses array-based storage
  - More memory efficient for simulation, with functionality appropriate for all uses
- Data accessed through a MOAB instance
  - Multiple instances can co-exist, but single instance is not thread-safe
  - Parallel instances independent except through parallel mesh constructs mentioned earlier
- MOAB supports a variety of platforms
  - Linux, MacOS, IBM BG/x, Cray
  - Windows maybe coming soon
- MOAB configure/build process using autoconf OR cmake
  - Makefile "snippets" built to simplify using it in application makefiles (examples later)
  - Can build with no dependencies, but you should probably build with NetCDF and HDF5
  - For the purposes of this training course, MOAB already built on Vesta
  - You can also build a local copy on your machine, and in many cases it's easier to learn that way

    http://trac.mcs.anl.gov/projects/ITAPS/MOAB/wiki

## HelloMOAB: Basic Mesh Access

http://www.mcs.anl.gov/~fathom/moab-docs/html/HelloMOAB_8cpp-example.html

- Interface instantiation using moab::Core constructor
  - Normally, all MOAB access should be through moab:: namespace, not used here for brevity
- Mesh can be loaded from file (Interface::load_file) or created in-place (Interface::create_vertex, Interface::create_element)
  - MOAB source comes with various mesh files, in MeshFiles/unittest/…
- Lists usually handed through interface as either Range or std::vector<EntityHandle>

## GetEntities: Basic Mesh Access

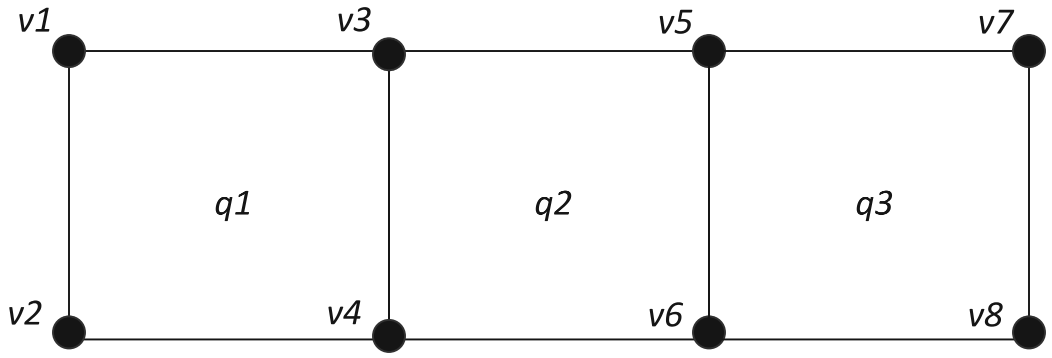
http://www.mcs.anl.gov/~fathom/moab-docs/html/GetEntities_8cpp-example.html

- MOAB provides functions for getting handle type, id (type_from_handle, id_from_handle)
  - These are bitmask functions, you could implement your own in C/C++
  - EntityType enumeration: MBVERTEX, MBEDGE, … (use Doxygen to find definition)
  - Ids usually assigned in sequence, starting with 1 (note, 0 is never a valid id, except for handle 0, which refers to the "root set" or instance)
- Range provides API very similar to std::vector
  - begin(), end(), rbegin(), etc.
  - Range::range_inserter type for handing to std::copy
- moab::CN class for Canonical Numbering
  - Tells how vertices, edges, faces are numbered in local element
  - Provides functions for e.g. getting string name, getting # edges in an element, etc.

## Intermediate-Dimension Entities ("aentities")

- Explicit representation of edges and edges/faces in 2D, 3D meshes is optional
  - Sometimes useful (e.g. adaptive refinement), sometimes not
  - For tetrahedral meshes, can increase memory cost by ~4-6x, hex meshes slightly lower
- "Real" meshes usually come with aentities necessary for defining boundary condition groupings, but no other ones
- In MOAB:
  - You can request creation of aentities by requesting them from adjacency calls with "*create_if_missing*" argument = *true*
  - Calling get_entities_by_xxx will return only those explicitly represented
- To force creation of interior edges/faces for whole mesh:
  - Get all vertices using *get_entities_by_dimension* with *dim* = 0 (use *Range* version)
  - Call *get_adjacencies* with *to_dimension* = *1* or *2* and *create_if_missing* = *true*

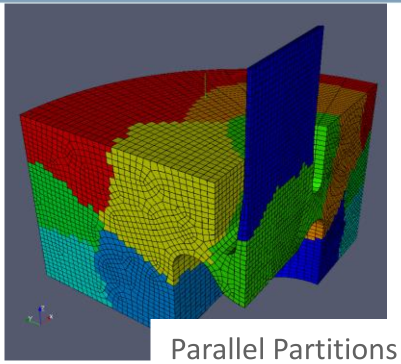
# MOAB *get_adjacencies* Allows Booleans on Results List



- *get_adjacencies(from=v1-v4; to_dim=2; op=INTERSECT;to_list=<empty>) = q1*
- *get_adjacencies(from=v3; to_dim=2; op=UNION;to_list=<empty>) = q1, q2*
- *get_adjacencies(from=v3-v4; to_dim=2; op=INTERSECT; to_list=<q2,q3>) = q2*

- Useful for reducing lines of code for mesh query & list manipulation
- Interface::Range also defines Boolean operations, for both code reduction and time efficiency

# Sets & Tags


Parallel Partitions

- The combination of sets and tags is one of the most powerful abstractions in MOAB
  - I have yet to see a construct useful in mesh-based simulation that cannot be efficiently represented using sets and tags
- Tags are useful as both fine-grained (dense) and coarse-grained (sparse) data
  - Sparse tags in MOAB are stored as (handle, value) tuples
  - Dense tags are allocated/stored as (value1, value2, …) for sequences of entity handles
  - Pointer to tag memory can be retrieved through API, useful for unstructured array-based simulations
- A set can have parent and child sets, and this is different from contains relations
  - Can define general directed graphs of sets
- Some more about sets:
  - The whole mesh is specified through the MOAB API as set handle zero (0)
  - Eliminates a whole set of functions for accessing entities for whole mesh vs. subset
  - MOAB has 2 types of sets:
    - List: order is preserved, entities can appear > 1 time (like std::vector)
    - Set: order not preserved (ordered by EntityHandle), each handle can occur only once (like std::set)
  - By default, MOAB does not make entities as being in sets, so can have “stale” sets
    - Can specify “tracking” flag for set at creation time, treats inclusion as entity-set adjacency
    - Tracking efficient memory-wise, but not necessarily time-wise; better to adjust on whole-set basis

## Sets & Tags (cont)

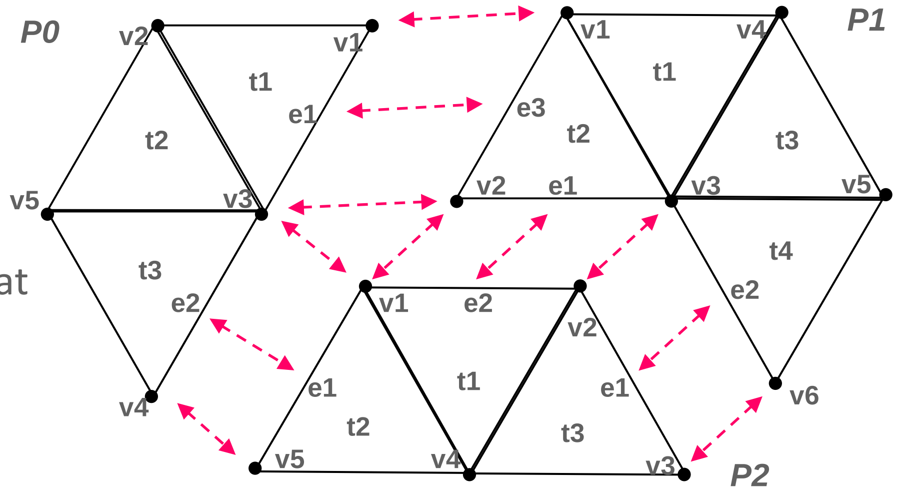
- MOAB *API* does not bind specific set purposes
  - No specific API support for boundary conditions, parallel parts, etc.
- MOAB defines *conventions* for conventional uses of sets
  - MBTagConventions.h, MBParallelConventions.h define various tag names, properties
  - MATERIAL_SET, DIRICHLET_SET, NEUMANN_SET, NAME, PARALLEL_PARTITION, …
- Sets & tags useful for defining “metadata” (data about the data)
  - MOAB documentation includes “I/O and Metadata Storage Conventions” document that describes some common uses
    - http://www.mcs.anl.gov/~fathom/moab-docs/html/md-contents.html
    - This document describes where data from specific file readers gets put in the MOAB data model
- For some meshes (cubit), sets can be used to represent original geometric model topology
  - Not enough time to describe here; check metadata document for details

# SetsNTags: Working with Sets and Tags

http://www.mcs.anl.gov/~fathom/moab-docs/html/SetsNTags_8cpp-example.html

- Interface::tag_get_handle used for both accessing current tags and creating new ones
- 2 types of tag-based access:
  - Get entities by type, tag: most useful for sparse tags
  - Get tag values on specific entity(ies): most useful for dense tags
- For materials and boundary conditions, more memory-efficient to define grouping using sets, and material/BC data using tags on set
- Most modern meshing tools work this way too

# MOAB Parallel Mesh



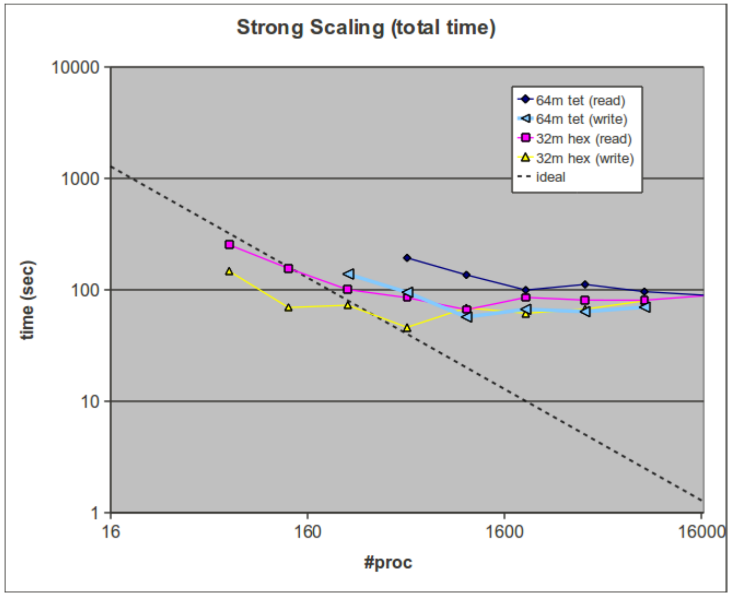
- Recall:
  - Element-based partition: decomposition of mesh over processors such that each element assigned to exactly one proc/part, with vertices shared between parts
  - Shared entity: an entity represented on multiple procs
  - (Part) interface entity: entity shared by multiple parts (vertices, edges, faces in a 3D mesh & element-based partition)
  - Owned entity, owner: each mesh entity owned by exactly one proc
  - Ghost entity: shared non-owned entity (sometimes referred to as "halo")
- For lots of parallel mesh usage, don't need to think about parallel at all
  - Serial mesh API works the same way
- In MOAB, parallel mesh constructs are stored using sets and tags
  - Could access most of the parallel mesh data using same serial API + parallel tag conventions
- MOAB also has a ParallelComm class
  - Provides convenience functions for e.g. getting shared entities, ghost entities
  - Parallel functionality, e.g. resolving shared and ghost entities, exchange/reduce tags

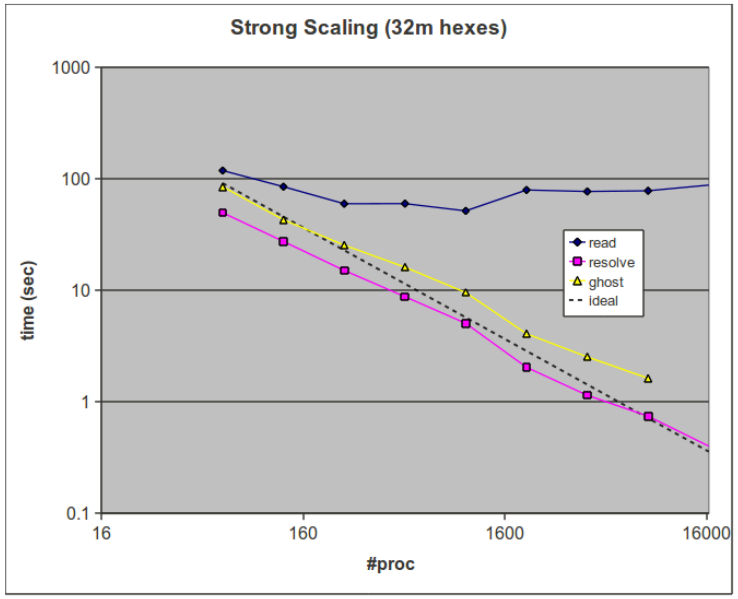
## MOAB Parallel Mesh (cont)

- Most common way of initializing parallel mesh is by reading a file, but details are important
  - Specifying partition type (replicated, by material, geometric volume, partitioning tool)
  - Post-read operations (resolve shared entities, exchange ghost cells)
- Specified in MOAB using file options string
  - See User's Guide, section 5, for list of options and common usages
- MOAB implements parallel I/O using single file approach
  - Different from many other tools, which use 1FPP or other approaches
  - Scalability / workflow simplicity often at odds
- File format also important
  - MOAB uses HDF5 for native format, that file type used to store partitioned file

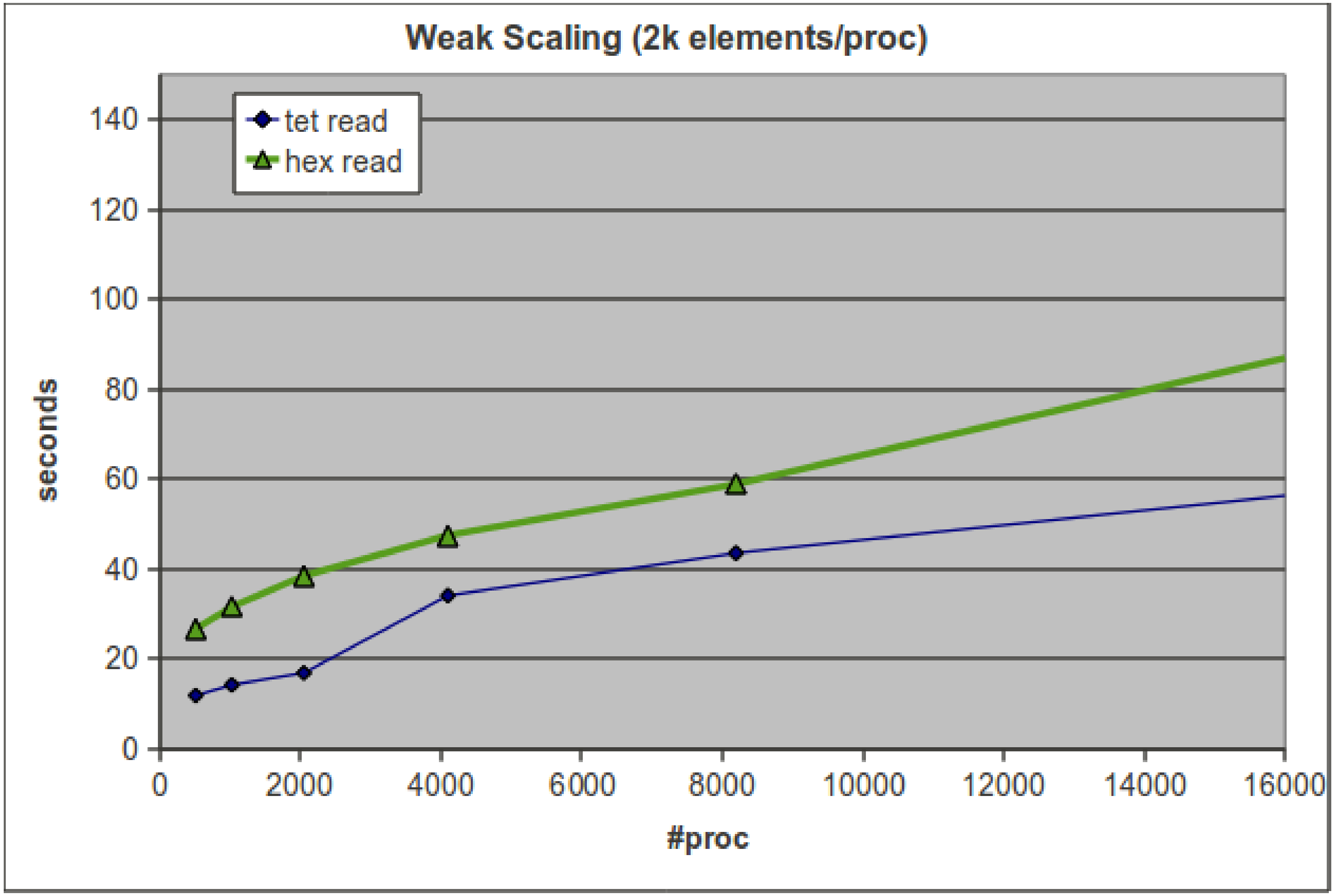
# MOAB Parallel I/O

- Data taken on Intrepid (IBM BG/P)
- Read/write for 32m hex, 64m tet elems
  - Nowhere near ideal I/O bandwidth
  - Absolute time tolerable in most cases
  - Drastic tet time improvement after reordering by partition
    - Fewer small fragments of HDF5 datasets
- Read/resolve/ghost times
  - Read times about constant
  - Resolve, ghost time scaling close to linear





# MOAB Parallel I/O: Weak Scaling



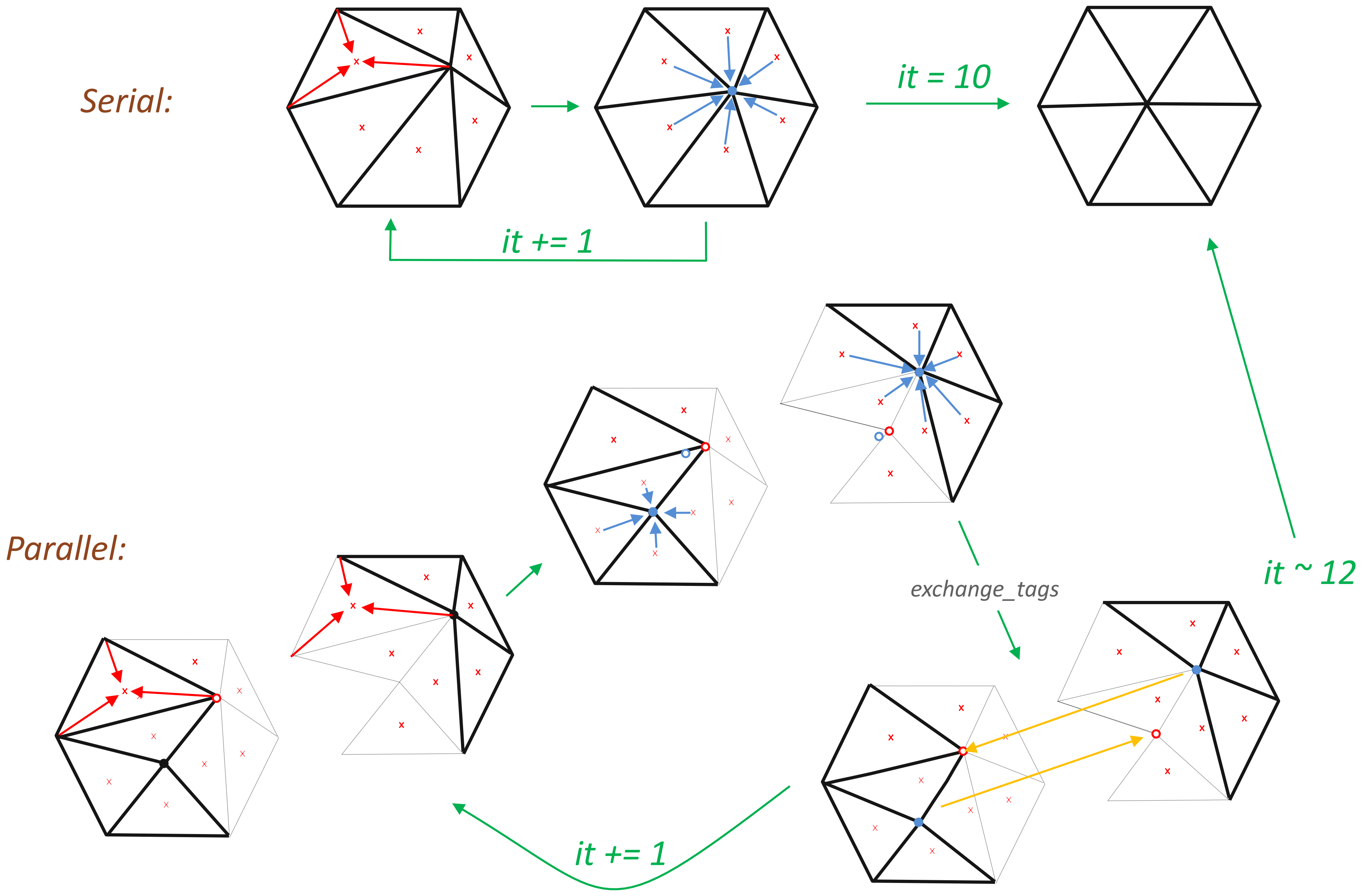
## HelloParMOAB: Parallel Mesh Initilization/Access

http://www.mcs.anl.gov/~fathom/moab-docs/html/HelloParMOAB_8cpp-example.html

- Can initialize MOAB using MPI_COMM_WORLD or other communicator
  - Can also use multiple ParallelComm objects, on different communicators (but sharing may not work right currently…)
- Use ParallelComm::get_shared_entities to get shared entities by dimension and other sharing proc (with defaults for all dimensions/other procs)
- PSTATUS_xxx enumeration/bitmask defines various parallel-relevant states
  - PSTATUS_ SHARED, MULTISHARED, INTERFACE, GHOST
- Use ParallelComm::filter_pstatus to filter range based on status & Boolean (NOT/AND)
- Resolving shared, ghosted entities can be specified at file read time (using option) or as explicit operation through ParallelComm
- MOAB does not restrict or change the way you can use MPI for other things
  - i.e. does not define e.g. MOAB_MPI_Comm datatype or MOAB_MPI_Reduce function

# Putting It Together: Parallel Lloyd Relaxation

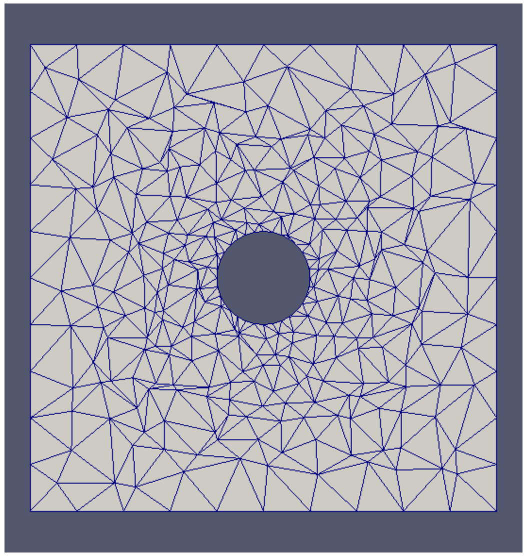
http://www.mcs.anl.gov/~fathom/moab-docs/html/LloydRelaxation_8cpp-example.html

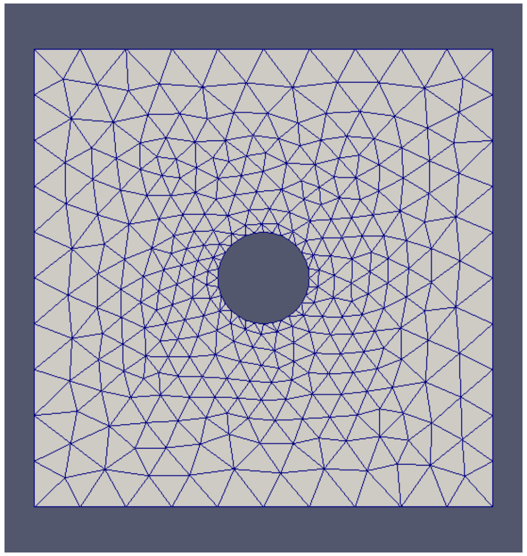


# Putting It Together: Parallel Lloyd Relaxation (cont)

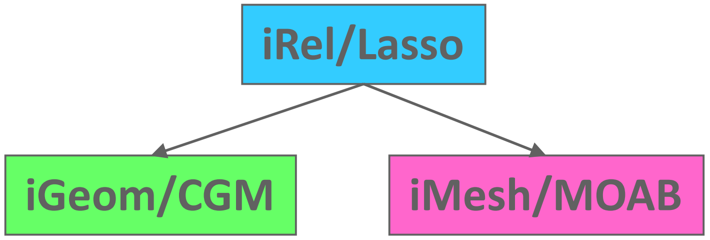
http://www.mcs.anl.gov/~fathom/moab-docs/html/LloydRelaxation_8cpp-example.html

- Initialization of mesh with shared entities resolved, one layer of ghosts exchanged
- Use centroid tag for intermediate storage of new vertex positions
- CN class cpp variables to dimension some lists
- When getting/setting tag values on multiple entities, make use of &stdvec[0] to get pointer to memory
  - STL guarantees this is valid
  - stl::vector dynamically-sized, useful for mesh-based codes
- Judicious choice of default value for tag eliminates need to initialize fixed tag for unfixed vertices
- Results:

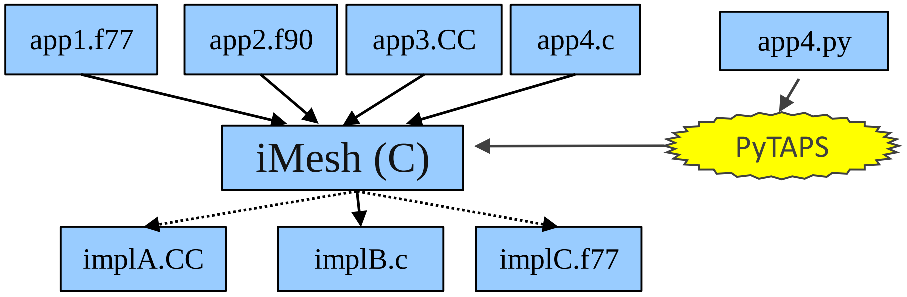




# iMesh, iMeshP Interfaces



- The ITAPS project defined a set of common interfaces (APIs) for mesh, geometry, and relations
- C-based interface, but designed to be callable directly from Fortran and C++
  - Good portability, performance
  - Maintenance easier
  - iMeshP for parallel data, constructs
  - Python also supported, through PyTAPS



- MOAB uses iMesh, iMeshP to support Fortran-based applications
- Primary differences between MOAB, iMesh(P):
  - MOAB parallel model defined entirely through sets+tags; iMeshP uses “Partition”, “Part”
  - In iMeshP, when you have multiple Parts per process, ghosting across parts implies duplicate entities in same iMeshP instance
  - List handling through iMesh/iMeshP somewhat more cumbersome due to lack of Range, std::vector data structures
    - Mitigated a bit using ISO_C_BINDING for F90+
- Not enough time to describe fully here; see MOAB User’s Guide, section 7

# Mesh-Based Tools Packaged With MOAB

- Several tools are packaged with MOAB and built by default
- mbsize
  - Used to read mesh & list numbers of entities of various types

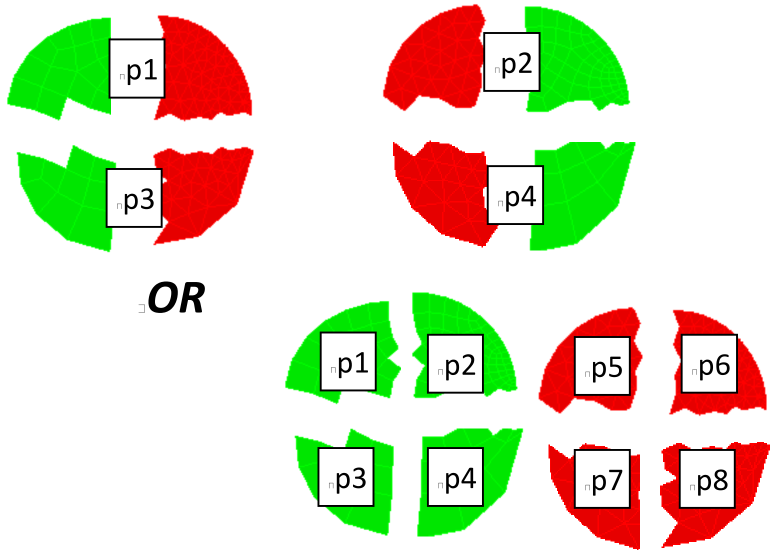
  -ll option (list long): lists everything in mesh; -g and –m list geometry and material/BC sets, resp.

  -f option: lists formats read and written by MOAB
- mbconvert: use to convert between file formats
  - Multiple –O <read_option> -o <write_option> can be used to test reading/writing in parallel
  - Use to generate vtk files for use by VisIt/ParaView
- mbpart (in mbzoltan subdirectory): partition a mesh for parallel access
  - (requires Zoltan library from Sandia)
  - Implements various partitioners (use –h to list), but Recursive Intertial Bisection seems to be most reliable & relatively fast
- mbtagprop: convert tags between set- and entity-based ata
- mbskin: generate & save the skin of a mesh

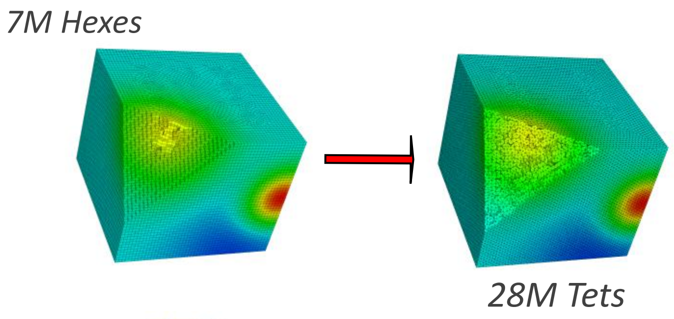
## Advanced Topics

- Direct access to MOAB storage
  - Use to obtain direct pointers to: tags (sparse & dense), connectivity, coordinates, adjacencies
  - Allows near-native speed for array-based applications
  - Uses iterator approach to allow for multiple “chunks” in handle/array space
  - See examples DirectAccessNoHoles, DirectAccessWithHoles, DirectAccessNoHolesF90 for usage
- Mesh searching
  - MOAB implements various tree types that enable local/parallel mesh searching
  - Optionally with finite element shape functions for locating points in elements
  - See AdaptiveKDTree class, tools/mbcoupler in source

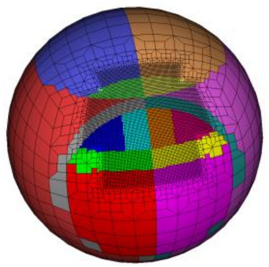
# MOAB-Based Solution Transfer

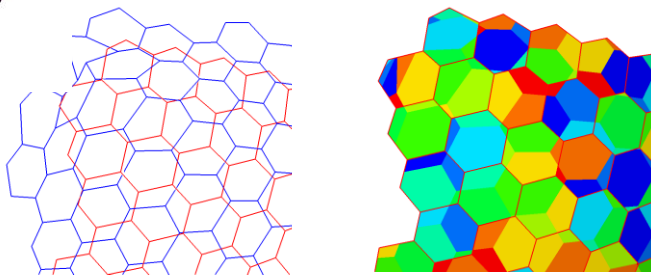


- Each physics type on independent mesh
- Distributed independently
- Both meshes in same MOAB instance



- Works for 2D, 3D in parallel for interpolation-based transfer (+ global or subset conservation)
- Work on tracer transport will provide basis for element-based conservation too





- *MOAB currently being integrated into ESMF to support online weight regeneration too*

# Common Geometry Module (CGM)

- Library for query & modification of BREP CAD-based geometric models
- Supports various modeling engines
  - Open.CASCADE (open-source)
  - ACIS (commercial)
  - CUBIT-ACIS (available for research purposes)
- Designed to represent geometric models as they are represented in CUBIT
- Basic model import & query
  - ACIS .sat, OCC .brep, STEP, IGES
  - Query # vertices/edges/faces/volumes, edge/face closest pt, face normal, etc.
- Model construction
  - 3D/2D primitives, spline fitting, etc.
  - Booleans, transforms, sweeping, lofting, etc.
  - *Not* a parametric modeler like e.g. SolidWorks
- Advanced features
  - Facet-based modeling
  - “Virtual” topology (small feature removal)
  - Decomposition for (hex) meshing

## Common Geometry Module (CGM) Open.CASCADE (OCC) port

- Previously, CGM only supported commercial modeling engines
  - ACIS, SolidWorks, Pro/Engineer, Catia CAA
- Over the past couple years, implemented CGM port to Open.CASCADE, the only general-purpose, open-source geometric modeling engine
- Most CGM functionality supported
  - Geometry construction, booleans, transforms
  - Webcut, imprinting (useful for hex & multi-material meshing)
  - Virtual topology
  - …
- Not supported:
  - Regularize after unite
  - Some history and undo operations recently added to CGM
  - Somewhat slower than ACIS-based CGM

## Lasso Relations Tool

- For some applications, need to relate mesh to geometry
  - Adaptive mesh refinement
  - Smooth surface-based boundary conditions
- Separation of geometry, mesh in independent components means we need a higher-level component that depends on those
- Lasso: tool for recovering, querying, setting geometry-mesh relations
- Prerequisite for querying: restore geometry, mesh with enough information to recover matching
- Lasso matches:
  - iGeom: Entity dimension – iMesh: EntitySet GEOM_DIMENSION tag value, and
  - iGeom: Entity GLOBAL_ID tag value – iMesh: EntitySet GLOBAL_ID tag value
- These matching criteria inherently implementation-dependent, though eventually can hopefully specify generically in terms of ITAPS data model

- Current implementation works with meshes generated by CUBIT, MeshKit

## Summary

- Mesh tends to be involved in front-end, back-end, of simulation process, and everywhere in between
- Mesh-related things tend to be ideal mix between math, CS, computational science (IMO)
- MOAB & friends are easy to use, powerful tools for working with mesh-related data
- For more information:
  - International Meshing Roundtable series of conferences, http://www.imr.sandia.gov
  - S. Cheng, T. Dey, J. Shewchuk, "Delaunay Mesh Generation", Chapman & Hall/CRC Press, 2012
  - H. Edelsbrunner, "Geometry and Topology for Mesh Generation", Cambridge Univ. Press, 2006
  - http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB
  - http://collab.mcs.anl.gov/display/moab
  - tautges@mcs.anl.gov
- MANY jobs and research topics available in this area!