

MOAB (& friends)

Vijay Mahadevan
Mathematics & Computer Science Division
Argonne National Laboratory

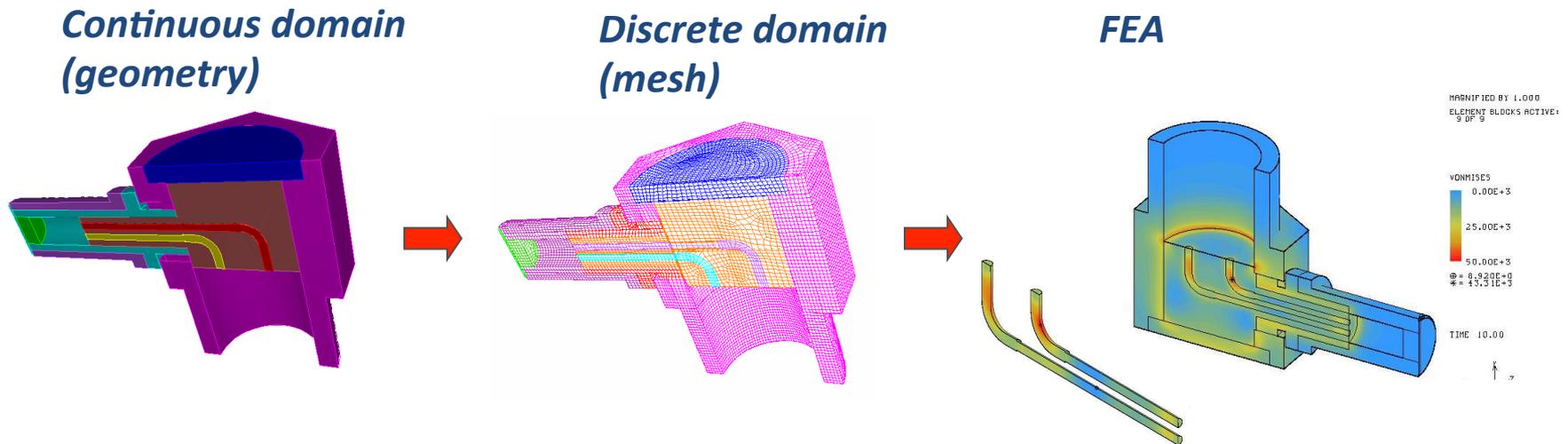
Outline

- Introduction: why geometry & mesh components?
- ITAPS: common interfaces for accessing geometry, mesh components
- Mesh-Oriented datABase (MOAB)
- Lasso
- Conclusions



Geometry, Mesh, Simulation Data

- Spatial domain model the starting point for most PDE-based simulation



- Sometimes geometric details are important, sometimes not
 - MPP-enabled resolution should resolve geometric features (where possible & useful?)
 - The more details you resolve, the harder it is to generate the mesh
- Large-code architecture often organized around handling of the spatial domain (mesh) and fine-grained data on the mesh (fields)



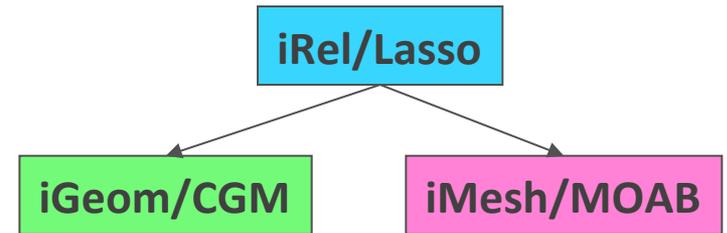
Code Frameworks

- For general-purpose codes, 2 common implementation approaches
 - “Top-down”, “large-F Framework”
 - “Bottom-up”, “small-f framework”
- Large-F Framework
 - Framework defines datastructures for representing mesh, solution data, etc.
 - Physics is re-implemented in that framework
 - + Simpler to arrive at working code given equations to solve
 - Difficult to port large codes into
 - Examples:
 - CHOMBO (LBNL), Sierra (SNL), MOOSE (INL), Trilinos? (SNL)
- Small-f framework
 - Distinct components defined along functional lines
 - Individual components can be used w/o other components
 - Applications composed from many of these components
 - + Get just what you need, no more
 - Assembly from bottom-up is more work
 - Examples:
 - ITAPS (mesh, geometry, relations), PETSc

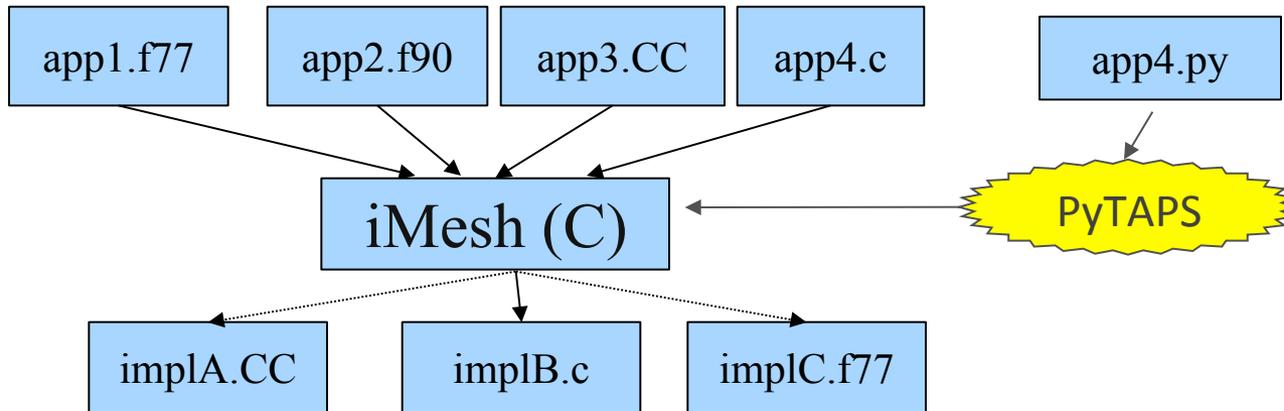


ITAPS Interfaces

- ITAPS promotes interoperability in both directions...
 - *Vertical*: put together components into higher-level applications
 - *Horizontal*: substitute one component for another providing similar capability
- ITAPS defines interfaces (APIs) by function:
 - iGeom: Geometric models (CAD)
 - iMesh: Mesh
 - iRel: Relations
- ANL implementations: CGM, MOAB, Lasso
- *SciDAC3: ITAPS + TOPS + APDEC – (lots of funding) = FASTMath*



ITAPS Interface Design

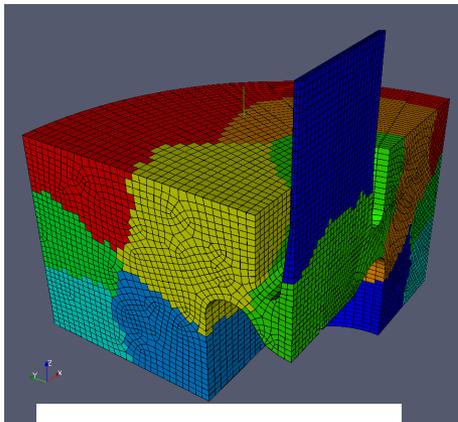


- C-based interface, but designed to be callable directly from Fortran and C++
 - Good portability, performance
 - Maintenance easier
 - iGeom, iRel too
- Quick startup for new users
- Considering language-based wrappers for FASTMath

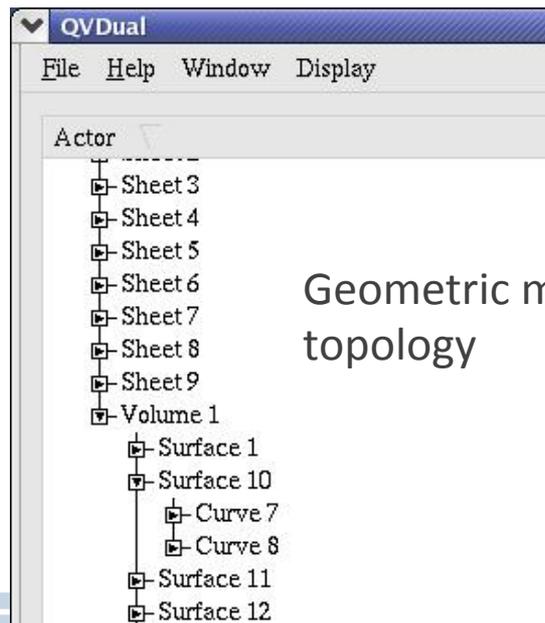


ITAPS Data Model

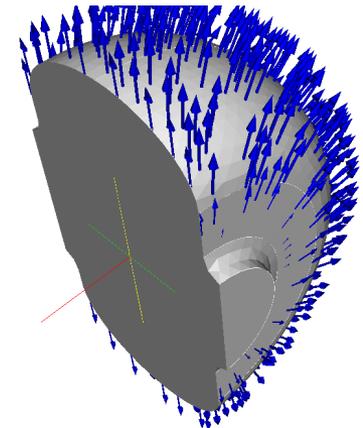
- Entities
 - Vertex, Edge, Tri, Quad, (Pentagon?), (Hexagon?), Polygon, Tet, Pyramid, Prism, Knife, Hex, Polyhedron
- Sets (collections of entities & sets, parent/child links)
 - BC groups, materials, proc partitions, kd-tree nodes, ...
- Tags (annotation of data on other 3)
 - Fine-grained (entities): vertex-based temperature, element-based heat generation rate
 - Coarse-grained (sets, interface): BC type, proc rank, provenance
- Interface (OOP, owns data)



Parallel Partition



Geometric model topology



Vertex-based displacements

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
- Optimized for memory usage first, speed second
- Implemented in C++, but uses array-based storage model
 - Avoids C++ object-based allocation/deallocation
 - Allows access in contiguous arrays of data
- Mostly an ITAPS-like data model
 - Entity, set, tag, interface
- 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, ...)



MOAB Parallel Model

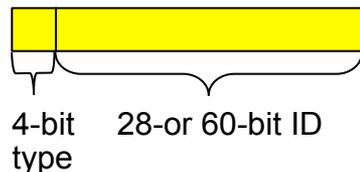
- Each proc has one MOAB instance, which appears locally as a serial mesh
 - All serial MOAB function calls available locally
- Parallel model based on element-based partition
 - Each element assigned to exactly one part, with vertices shared between parts
 - Arbitrary number of layers of ghost elements
- 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 Entity Storage

Entity Handle:

- Unsigned long type
- Bitmask
- Sorts by dimension, type



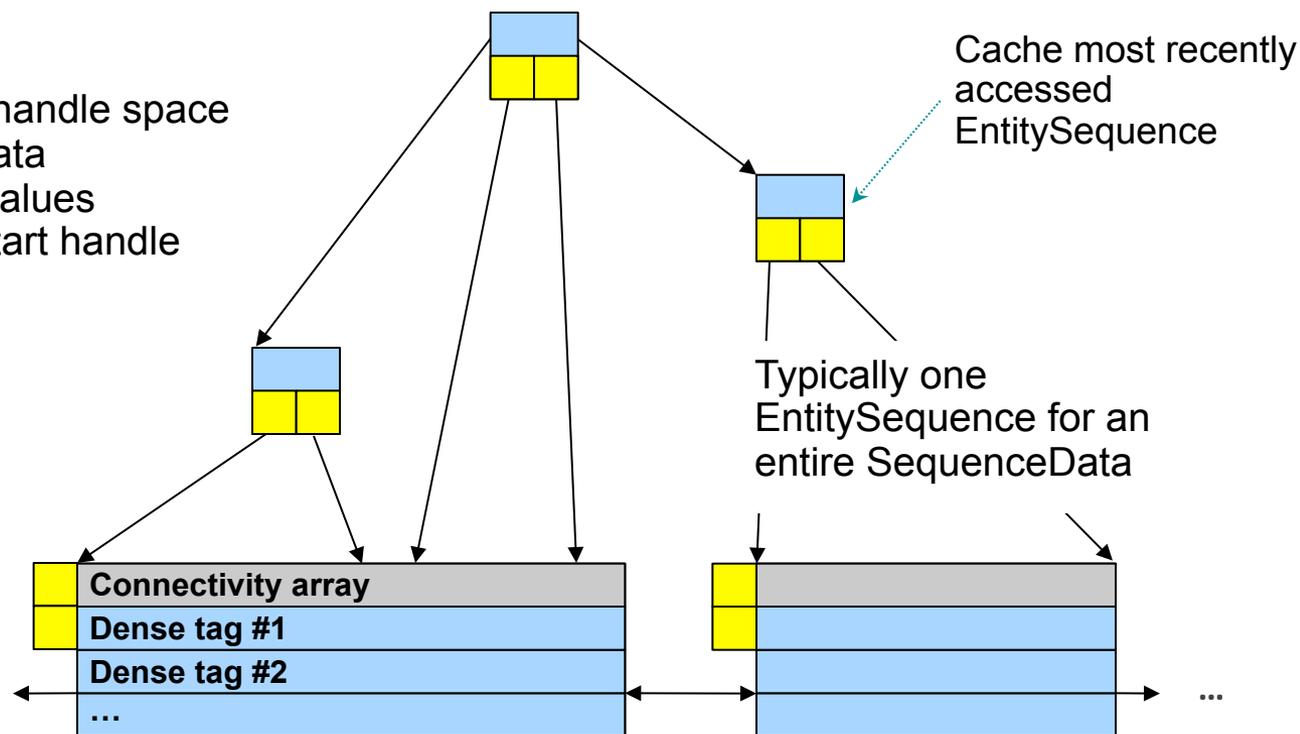
Range:

- Container of handles
- Constant-size if contiguous handles



EntitySequences:

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



SequenceData:

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



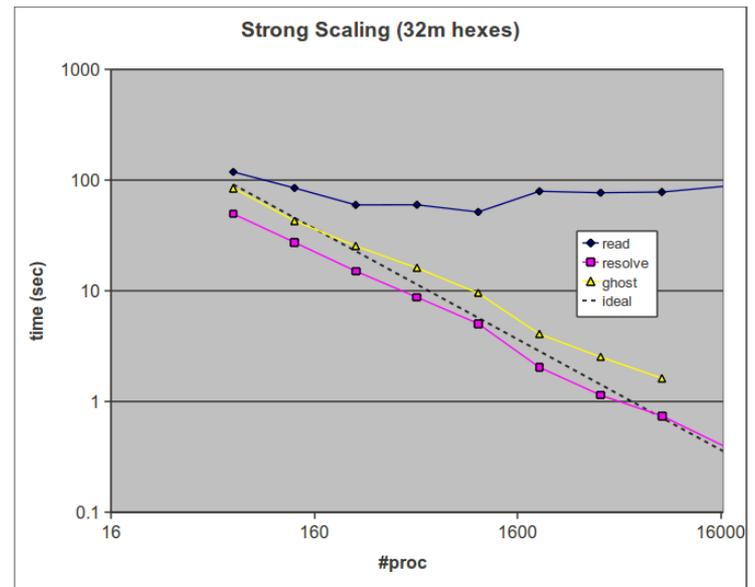
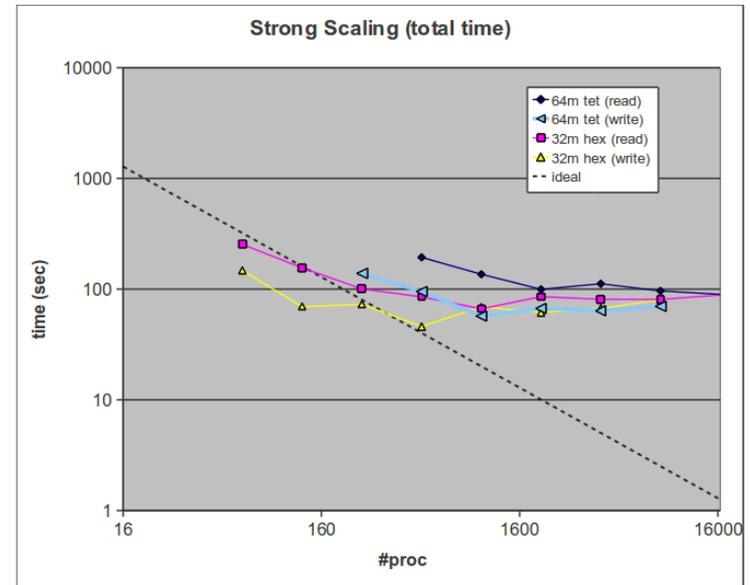
MOAB Parallel I/O

- MOAB supports various mesh formats used for FE data
 - ExodusII, vtk, Abaqus, Star-CCM+
- Native storage format is HDF5-based
 - Need a format that can store the full data model representable in MOAB (sets, tags)
- Need to support parallel I/O on large #s of processors
- Most other efforts implement parallel I/O by splitting up the mesh into several or many files and reading each on a subset of processors
 - In our approach, read/write from/to single file
- Read:
 - Read partition sets, then elements, then vertices, then other sets
 - Resolve shared vertices on inter-processor boundaries
 - If desired, exchange layer(s) of ghost cells with neighbor processors
- Write:
 - Negotiate file id space, shared sets (based on tags)
 - Concurrent write of vertices, elements, sets

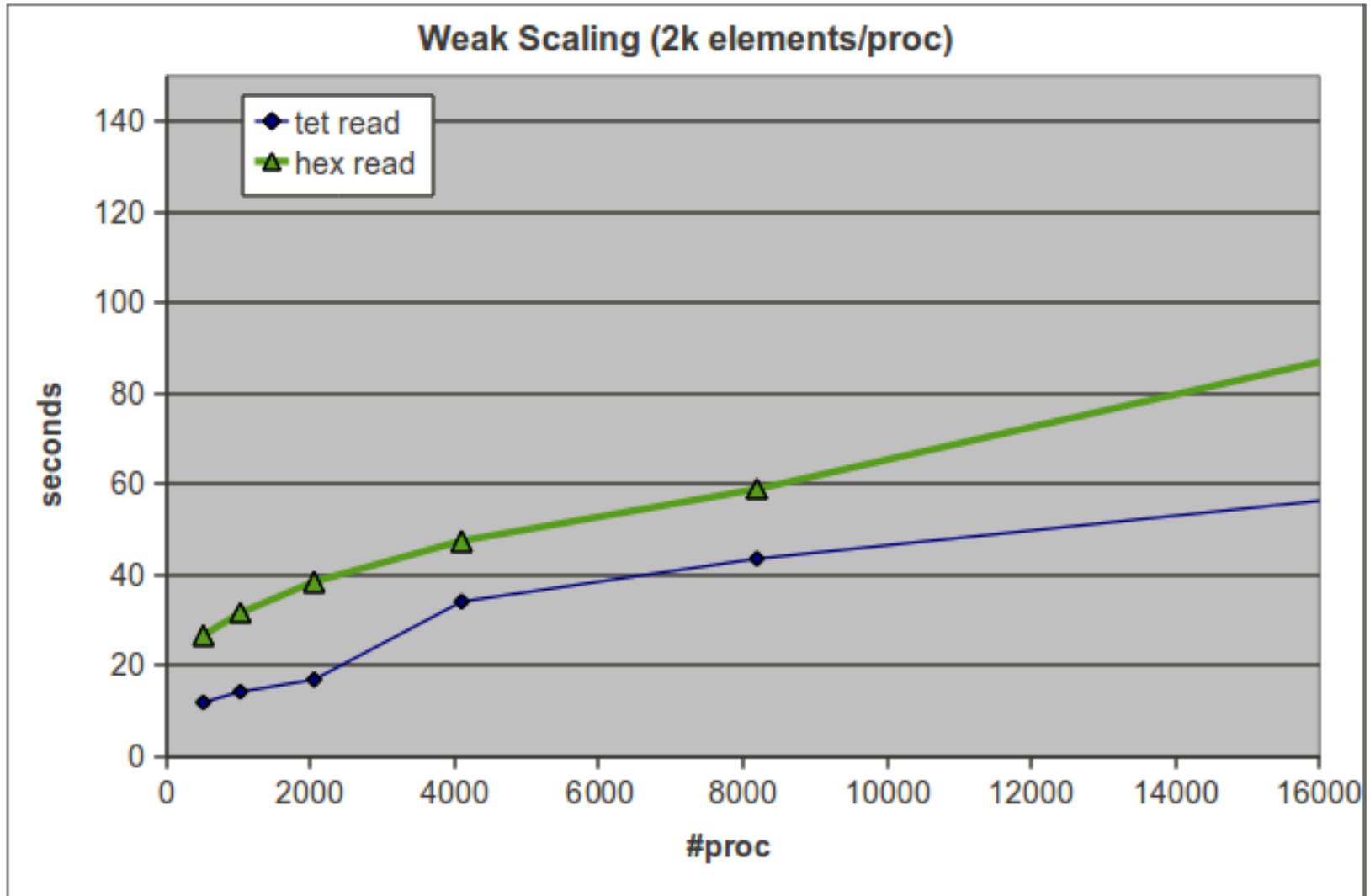


MOAB Parallel I/O

- 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

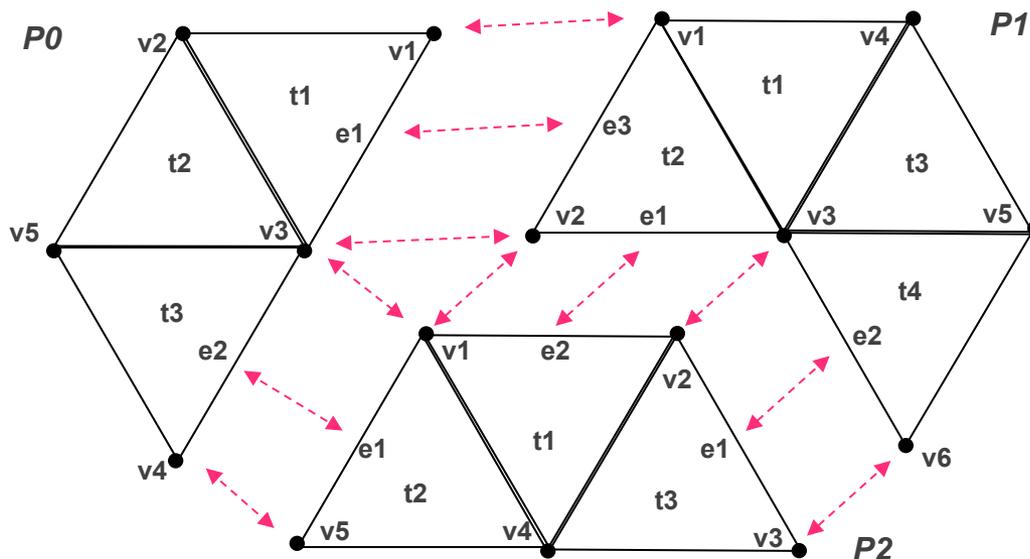


MOAB Parallel I/O: Weak Scaling



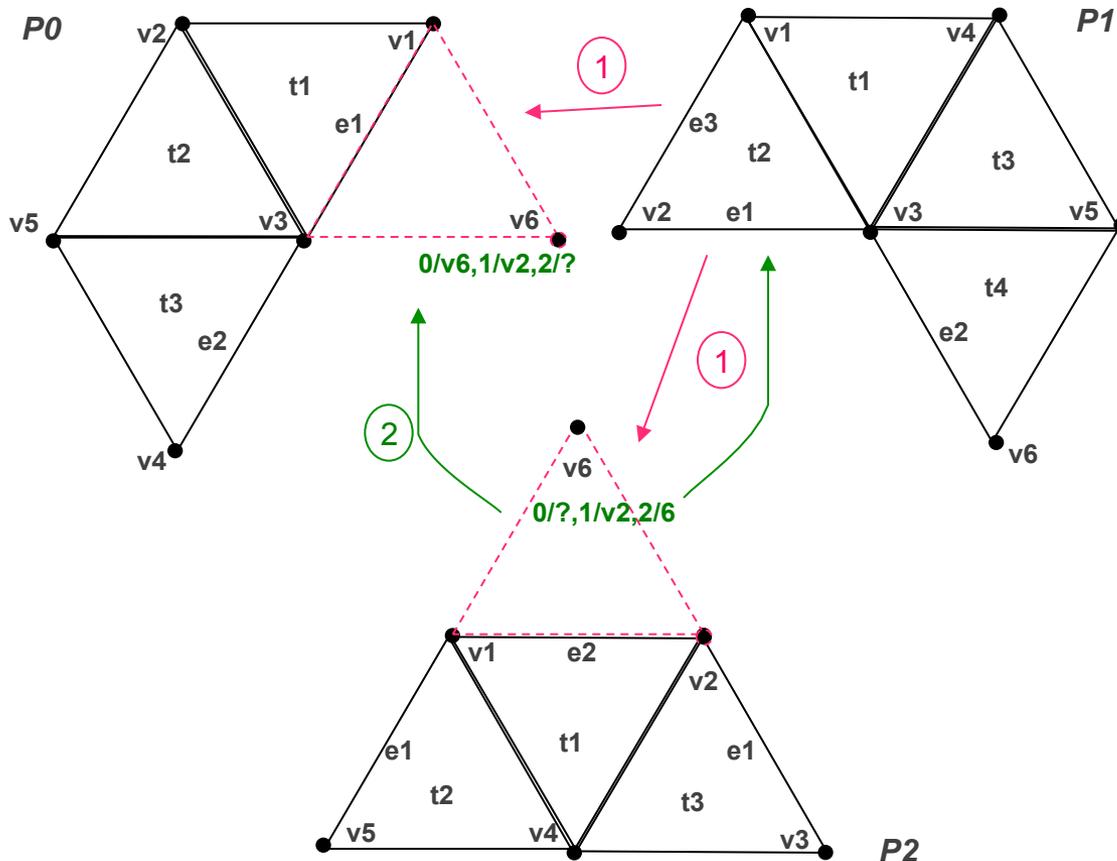
Resolving Interface Mesh

- Start: local mesh, with global ids on vertices
- Find: vertices & other entities shared by other procs; for each shared entity, need remote processor(s), remote entity handle(s)
- Solve in 2 stages:
 - Shared vertices (using TL.gs_init on global ids)
 - Shared edges/faces (based on connectivity, after remote vertices known)



Exchanging Ghost Entities

- Identify ghosts & destinations based on interface shared entities
- After sending ghosts, receive return message with remote handles



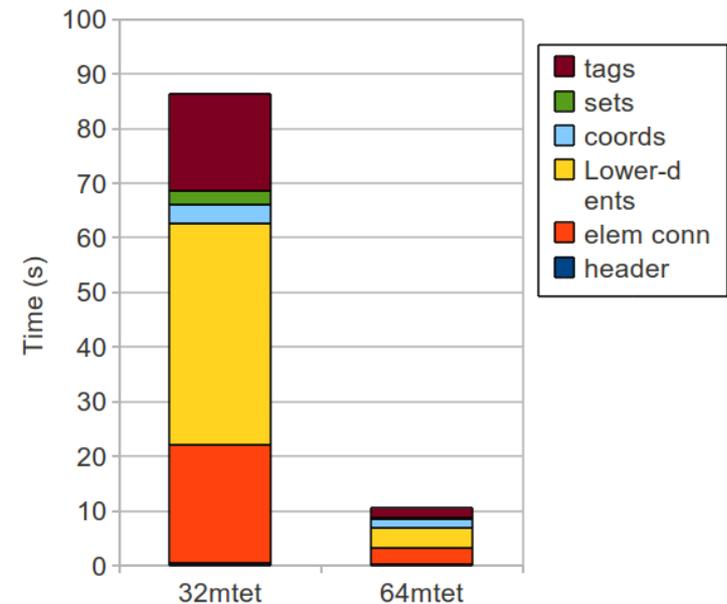
- Send entities, sharing lists, *extra processors*
- Return sharing info, including to extra processors

6 Gotchas:

- Sending new vertices *and* new entities that use them in same message (reference as negative handle = index in message's new entity list)
- Don't know destination handle of all (eventual) sharing procs (store as 0 until known)
- Ghost exchange might introduce new sharing procs (send extra sharing procs in 1st message)
- Might get same new entity from multiple other procs (store list on receiving proc, indexed by owner handle)
- Need to fill in missing remote handle data
- Sending proc doesn't know how to reference new entity for remote handle update (mark using negative proc handle)

Read Performance

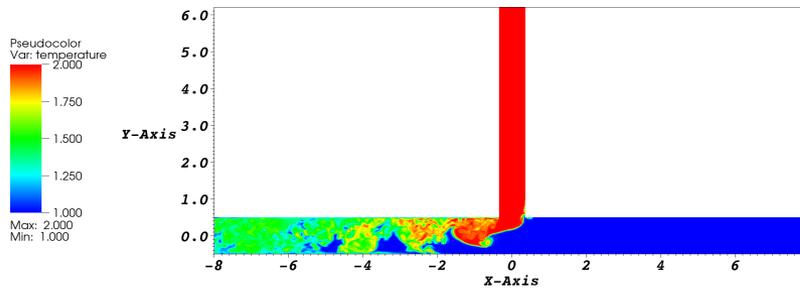
- Read consists of several steps
 - Read header
 - Read partition sets
 - Read element connectivity
 - Read vertex coords
 - Read/process lower-dimensional entities
 - Read/process other sets
 - Read tags
- Tried removing geometric topology sets, lower dimensional entities from dataset
 - On 64 procs (fusion), reduced read time by almost 90%!
- Need to try 2 options:
 - Remove that data, benchmark on larger proc counts
 - Include complete information in partition sets, benchmark



Connecting Physics Modules

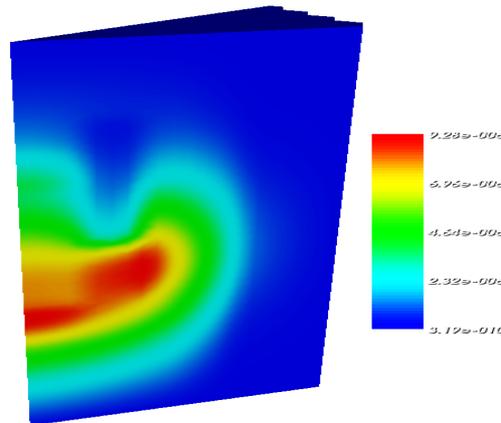
- Nek: parallel, input

- OECD Vettenhall T-Junction benchmark



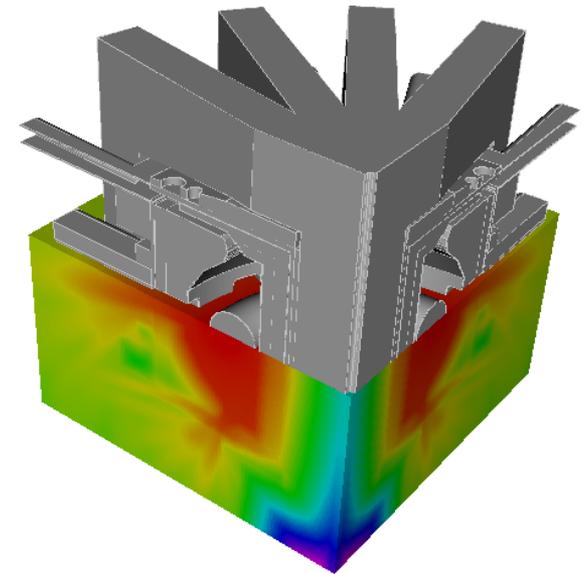
- UNIC: serial, input

- Takeda4 benchmark problem



- Denovo: serial, input

-



Simple Example: HELLO iMesh (C++)

- Simple, typical application which 1) Instantiates iMesh interface, 2) Reads mesh from disk, 3) Reports # entities of each dimension

```
#include <iostream>
#include "iMesh.h"

int main( int argc, char *argv[] )
{
    // create the Mesh instance
    char *options = NULL;
    iMesh_Instance mesh;
    int ierr, options_len = 0;
    iMesh_newMesh(options, &mesh, &ierr,
                 options_len);

    // load the mesh
    iMesh_load(mesh, argv[1], options, &ierr,
              strlen(argv[1]), options_len);

    // report the number of elements of each dimension
    for (int dim = iBase_VERTEX; dim <= iBase_REGION; dim++) {
        int numd;
        iMesh_getNumOfType(mesh, 0, dim, &numd, &ierr);
        std::cout << "Number of " << dim << "d elements = "
                  << numd << std::endl;
    }
    return true;}

1
2
3
```

• Makefile:

```
include ../../iMesh-Defs.inc

HELLOiMesh: HELLOiMesh.o  ${IMESH_FILES}
             $(CXX)  $(CXXFLAGS) -o $@ HELLOiMesh.o \
             ${IMESH_LIBS}

.cpp.o:
        ${CXX} -c ${CXXFLAGS} ${IMESH_INCLUDES} $<
```

***Note: no error checking here for brevity,
but there should be in your code!!!***

Slightly More Complicated Example: FindConnect (C)

```
#include <iostream>
#include "iMesh.h"

typedef void* EntityHandle;

int main( int argc, char *argv[] )
{
    // create the Mesh instance
    iMesh_Instance mesh;
    int ierr;
    iMesh_newMesh("", &mesh, &ierr, 0);

    // load the mesh
    iMesh_load(mesh, 0, "125hex.vtk", "",
        &ierr, 10, 0);

    // get all 3d elements
    iMesh_EntityHandle *ents;
    int ents_alloc = 0, ents_size;
    iMesh_getEntities(mesh, 0, iBase_REGION,
        iMesh_ALL_TOPOLOGIES,
        &ents, &ents_alloc,
        &ents_size, &ierr);

    int vert_uses = 0;

    // iterate through them
    for (int i = 0; i < ents_size; i++) {
        // get connectivity
        iBase_EntityHandle *verts;
        int verts_alloc = 0, verts_size;

        iMesh_getEntAdj(mesh, ents[i], iBase_VERTEX,
            &verts, &verts_alloc, &verts_size,
            &ierr);

        // sum number of vertex uses
        vert_uses += verts_size;
        free(verts);

        // now get adjacencies in one big block
        iBase_EntityHandle *allv;
        int *offsets;
        int allv_alloc = 0, allv_size,
            offsets_alloc = 0, offsets_size;
        iMesh_getEntArrAdj(mesh, ents, ents_size,
            iBase_VERTEX,
            &allv, &allv_alloc, &allv_size,
            &offsets, &offsets_alloc, &offsets_size,
            &ierr);

        // compare results of two calling methods
        if (allv_size != vert_uses)
            std::cout << "Sizes didn't agree" << std::endl;
        else
            std::cout << "Sizes did agree" << std::endl;

        return true;
    }
}
```

```
    // iterate through them
    for (int i = 0; i < ents_size; i++) {
        // get connectivity
        iBase_EntityHandle *verts;
        int verts_alloc = 0, verts_size;

        iMesh_getEntAdj(mesh, ents[i], iBase_VERTEX,
            &verts, &verts_alloc, &verts_size,
            &ierr);

        // sum number of vertex uses
        vert_uses += verts_size;
        free(verts);

        // now get adjacencies in one big block
        iBase_EntityHandle *allv;
        int *offsets;
        int allv_alloc = 0, allv_size,
            offsets_alloc = 0, offsets_size;
        iMesh_getEntArrAdj(mesh, ents, ents_size,
            iBase_VERTEX,
            &allv, &allv_alloc, &allv_size,
            &offsets, &offsets_alloc, &offsets_size,
            &ierr);

        // compare results of two calling methods
        if (allv_size != vert_uses)
            std::cout << "Sizes didn't agree" << std::endl;
        else
            std::cout << "Sizes did agree" << std::endl;

        return true;
    }
}
```



Slightly More Complicated Example: FindConnect (Fortran)

```
program findconnect
#include "iMesh_f.h"

c declarations
iMesh_Instance mesh
integer*8 ents
pointer (rpents, ents(0:*))]
integer*8 rpverts, rpallverts, ipoffsets
1 pointer (rpverts, verts(0:*))]
1 pointer (rpallverts, allverts(0:*))]
1 pointer (ipoffsets, ioffsets(0,*))]
integer ierr, ents_alloc, ents_size
integer verts_alloc, verts_size
integer allverts_alloc, allverts_size
integer offsets_alloc, offsets_size

c create the Mesh instance
call iMesh_newMesh("MOAB", mesh, ierr)]

c load the mesh
call iMesh_load(%VAL(mesh), %VAL(0),
1 "125hex.vtk", "", ierr)]

c get all 3d elements
ents_alloc = 0
call iMesh_getEntities(%VAL(mesh),
1 %VAL(0), %VAL(iBase_REGION),
2 1 %VAL(iMesh_ALL_TOPOLOGIES),
1 rpents, ents_alloc, ents_size,
1 ierr)]
```

```
ivert_uses = 0

c iterate through them;
do i = 0, ents_size-1
c get connectivity
verts_alloc = 0
3 call iMesh_getEntAdj(%VAL(mesh),
1 %VAL(ents(i)), %VAL(iBase_VERTEX),
1 rpverts, verts_alloc, verts_size, ierr)]
c sum number of vertex uses
vert_uses = vert_uses + verts_size
call free(rpverts)]
4 end do

c now get adjacencies in one big block
allverts_alloc = 0
offsets_alloc = 0
call iMesh_getEntArrAdj(%VAL(mesh),
1 %VAL(rpents), %VAL(ents_size),
1 %VAL(iBase_VERTEX), rpallverts,
1 allverts_alloc, allverts_size, ipoffsets,
1 offsets_alloc, offsets_size, ierr)]

c compare results of two calling methods
if (allverts_size .ne. vert_uses) then
write(*, '("Sizes didn't agree!")')]
else
write(*, '("Sizes did agree!")')]
endif

end
```



Nek-MOAB Solution Memory Sharing

- Recent enhancement in MOAB allows applications to get pointer to tag memory
- Allows application to share solution variable memory with MOAB
- MOAB-based output, coupling doesn't require any memory duplication
- Changes required in application almost trivial, even for Fortran

c use “cray pointers” to associate array with ptr

pointer (rpxm1, xm1(lx1, ly1, lz1))

c get iterator over regions (hexes)

**call iMesh_initEntArrIter(%VAL(imeshh), %VAL(rootset), %VAL(iBase_REGION),
%VAL(iMesh_HEXAHEDRON),%VAL(num_hexes), iter, ierr)**

c create dense tag

**call iMesh_createTagWithOptions(%VAL(imeshh), "XM1", "moab:TAG_STORAGE_TYPE=DENSE
moab:TAG_DEFAULT_VALUE=0.0", %VAL(lx1*ly1*lz1), %VAL(iBase_DOUBLE), tagh, ierr)**

c get ptr to tag memory

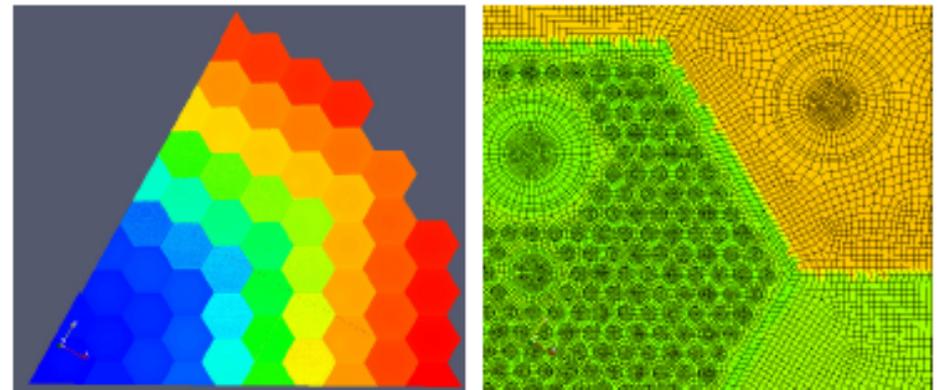
call iMesh_tagIterate(%VAL(imeshh),%VAL(tagh),%VAL(iter),rpxm1,count,ierr)



Parallel Merge Mesh

- Current resolution of shared vertices matches vertices based on global id
- If you match based on geometric proximity instead, you get a mesh merging tool, which is also quite useful
- RGG: Reactor Geometry (& mesh) Generator
 - Generates nuclear reactor core meshes using 2-level lattice (assembly = lattice of fuel rods, core = lattice of assemblies)
- Developed parallel RGG
 - Copy/move assemblies in parallel
 - Parallel mesh merge
- Super-linear speedup results on up to 56 procs (due to thrashing)

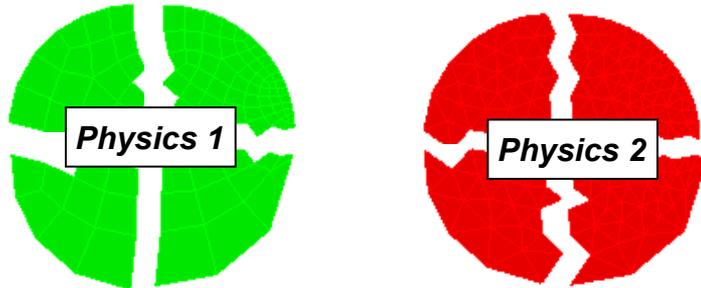
	#procs	11M hex VHTR	58M hex VHTR
Copy/move	1	10.4	141.8
	8	5.8	59.6
	16	0.27	1.4
	32	0.036	0.12
	56	0.004	0.001
Join	1	3.81	70.48
	8	7.54	28.29
	16	6.7	17.62
	32	0.92	2.34
	56	0.25	0.8



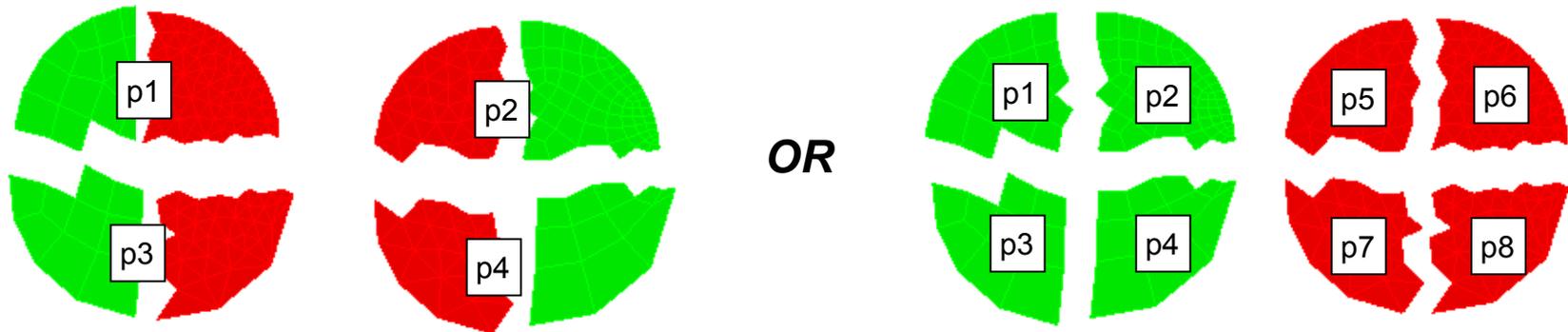
1/6 VHTR core (58M hex elements)

MOAB-Based Solution Transfer

- **Meshes:** Each physics type is solved on an *independent mesh* whose characteristics (element type, density, etc.) is most appropriate for the physics

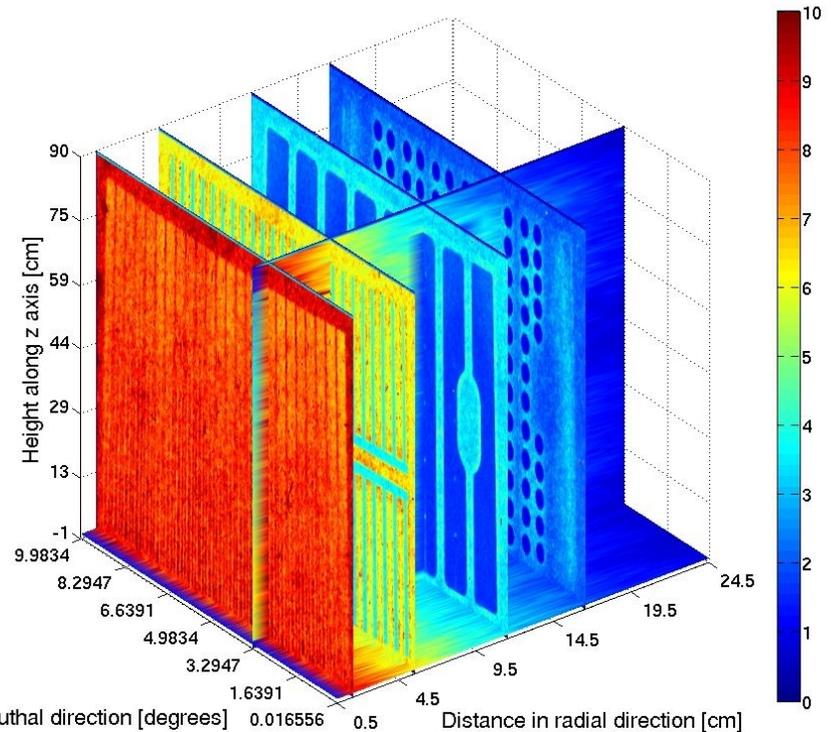
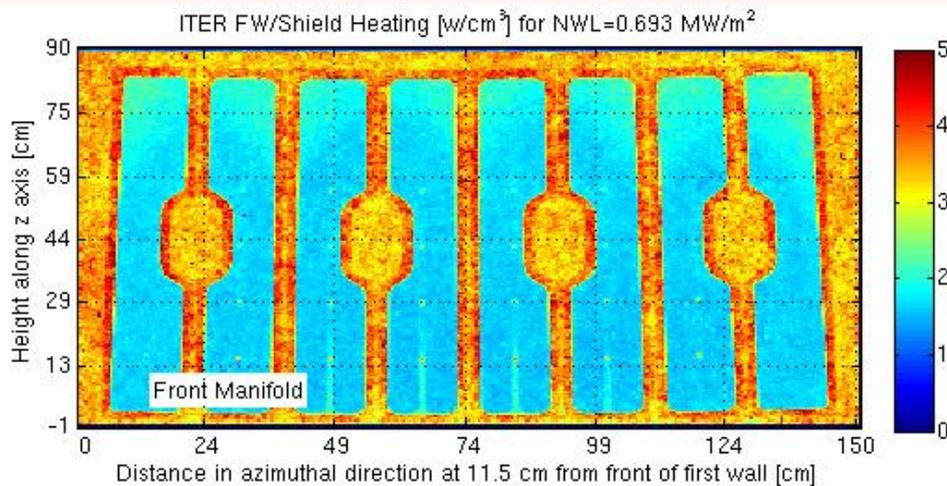
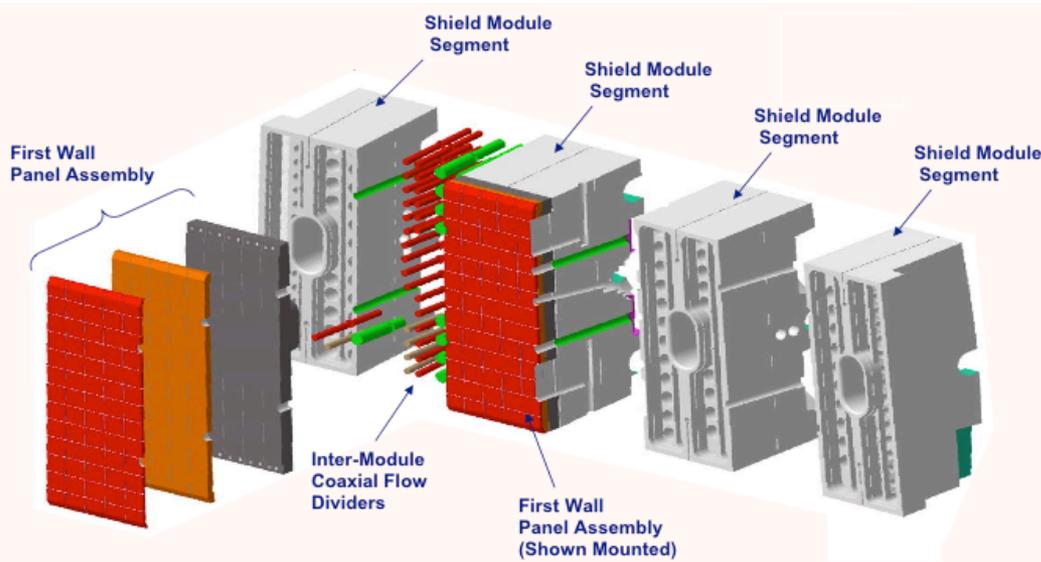


- **Distribution:** Each physics type and mesh is *distributed independently* across a set of processors, defined by an MPI communicator for each mesh

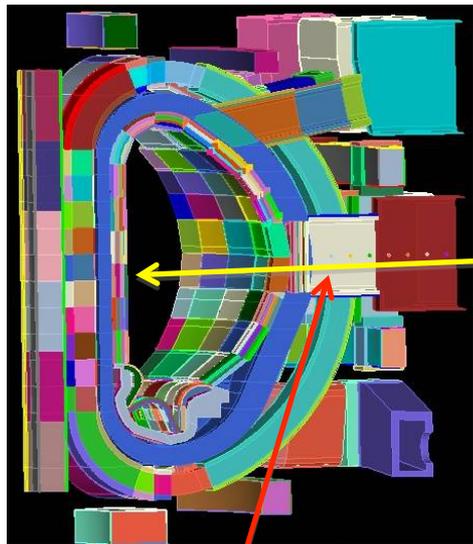


- **Implementation:** On a given processor, all meshes are stored in a *single iMesh instance*, and that instance communicates with all other processors containing pieces of any of those meshes.

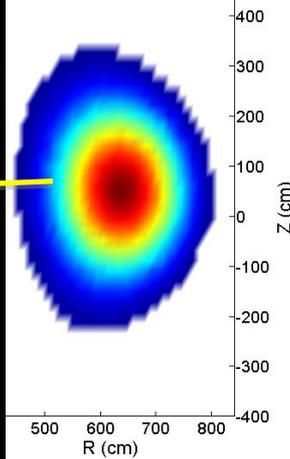
Analysis for an Initial Mod 13 Design



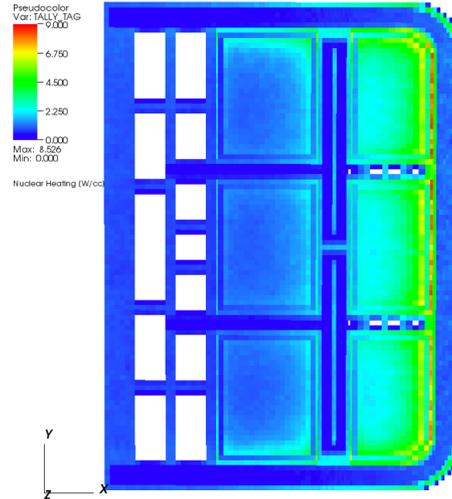
Detailed 3-D Neutronics for DCLL TBM



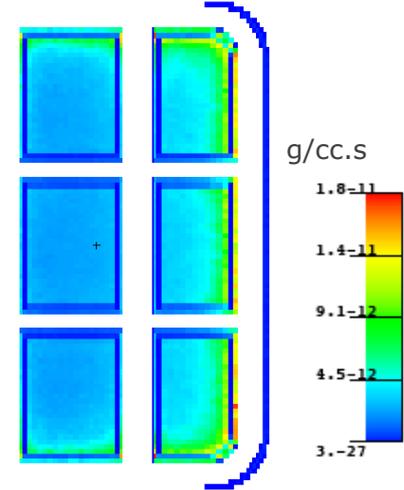
Source Input Table



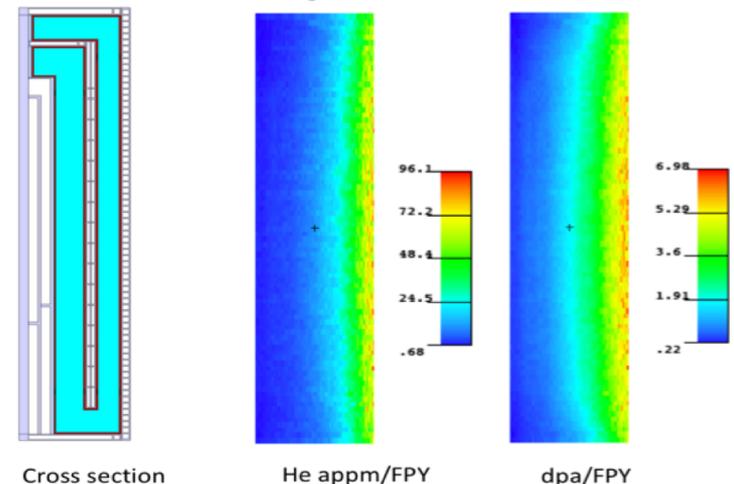
Mid-plane nuclear heating



Mid-plane T production



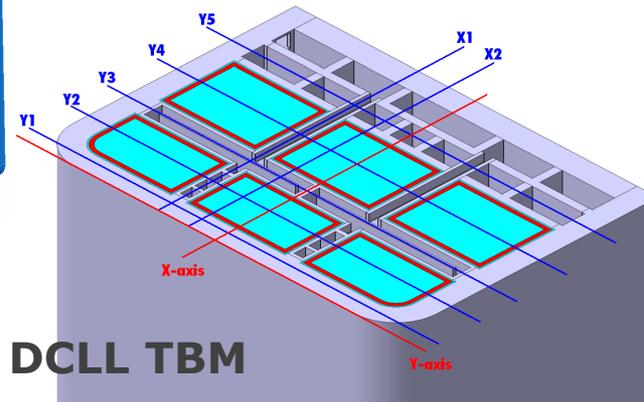
Steel damage at section X2



Cross section

He appm/FPY

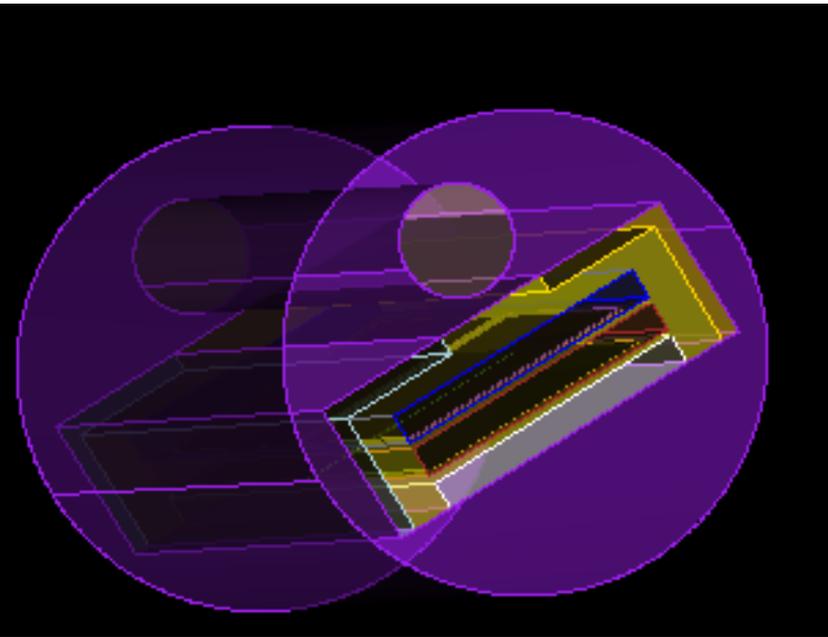
dpa/FPY



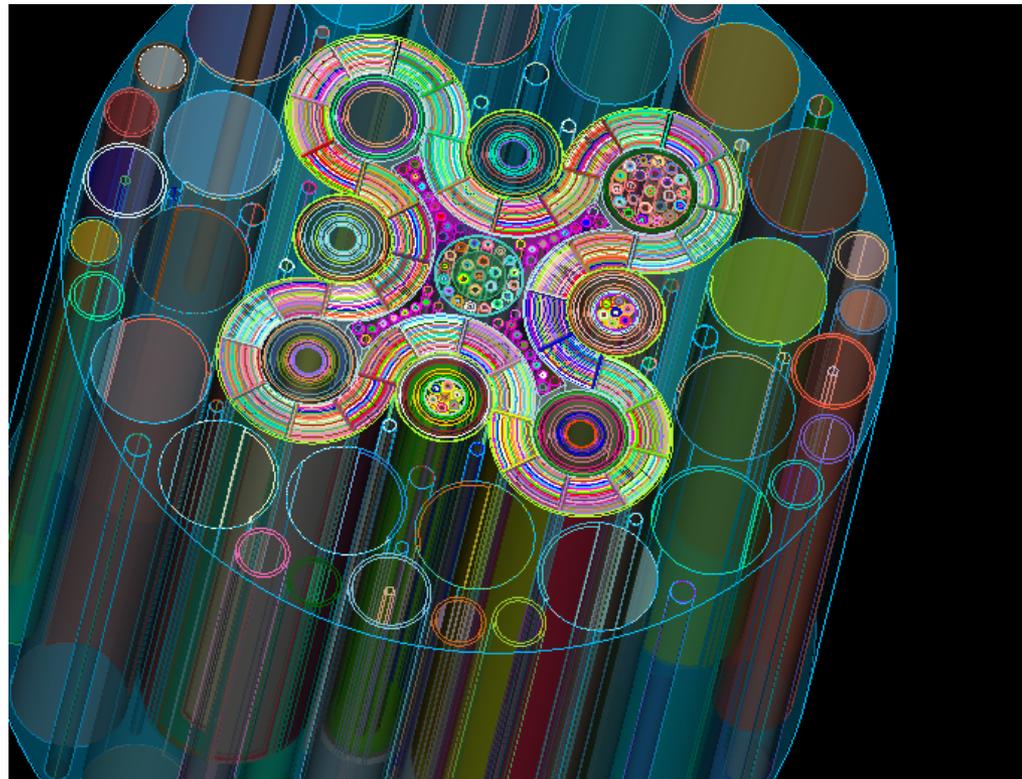
DCLL TBM

Fission Applications

ATR National Science User Facility



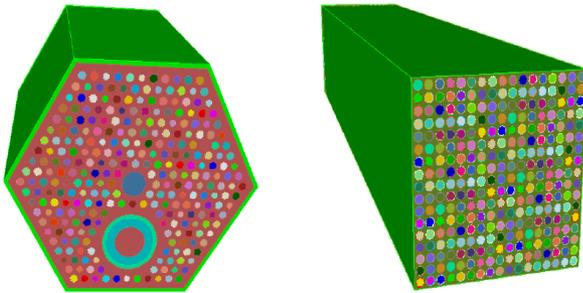
AFIP Experiment In CFT



RGG: Reactor Geometry (&mesh) Generator

Step 1: AssyGen

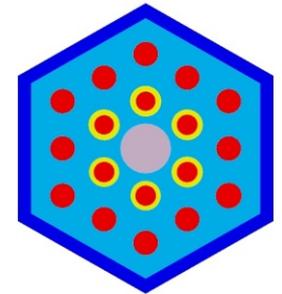
- RGG takes advantage of information about repeated structures in both assembly and core lattices.
- provides a balance between lattice-guided automation and user interaction at key points of the process.
- option to run serially and in parallel.
- supports rectangular and hexagonal lattices.



VHTR (325 pins, 40 seconds) and PWR (578 pins, 90 seconds) assemblies created using AssyGen tool.

```

! KEYWORD BASED ASSYGEN INPUT FILE
GeometryType Hexagonal
Materials 5 CR C Fuel F Dt1 D1 Dt2 D2 Mat M1
Duct 2 0.0 0.0 0.0 79.0 9.0 10.0 D1 D2
Pincells 3 1.87960
ControlRod CR 1
Cylinder 1 0.0 0.0 0.0 79.0 1.0 C
FuelCell_1 F2 1
Cylinder 1 0.0 0.0 0.0 79.0 0.5 F
FuelCell_2 F1 1
Cylinder 2 0.0 0.0 0.0 79.0 0.4 0.6 F M1
Assembly 3
    F2 F2 F2
    F2 F1 F1 F2
    F2 F1 CR F1 F2
    F2 F1 F1 F2
    F2 F2 F2
RadialMeshSize 0.15
Rotate Z -30
Center
END
    
```



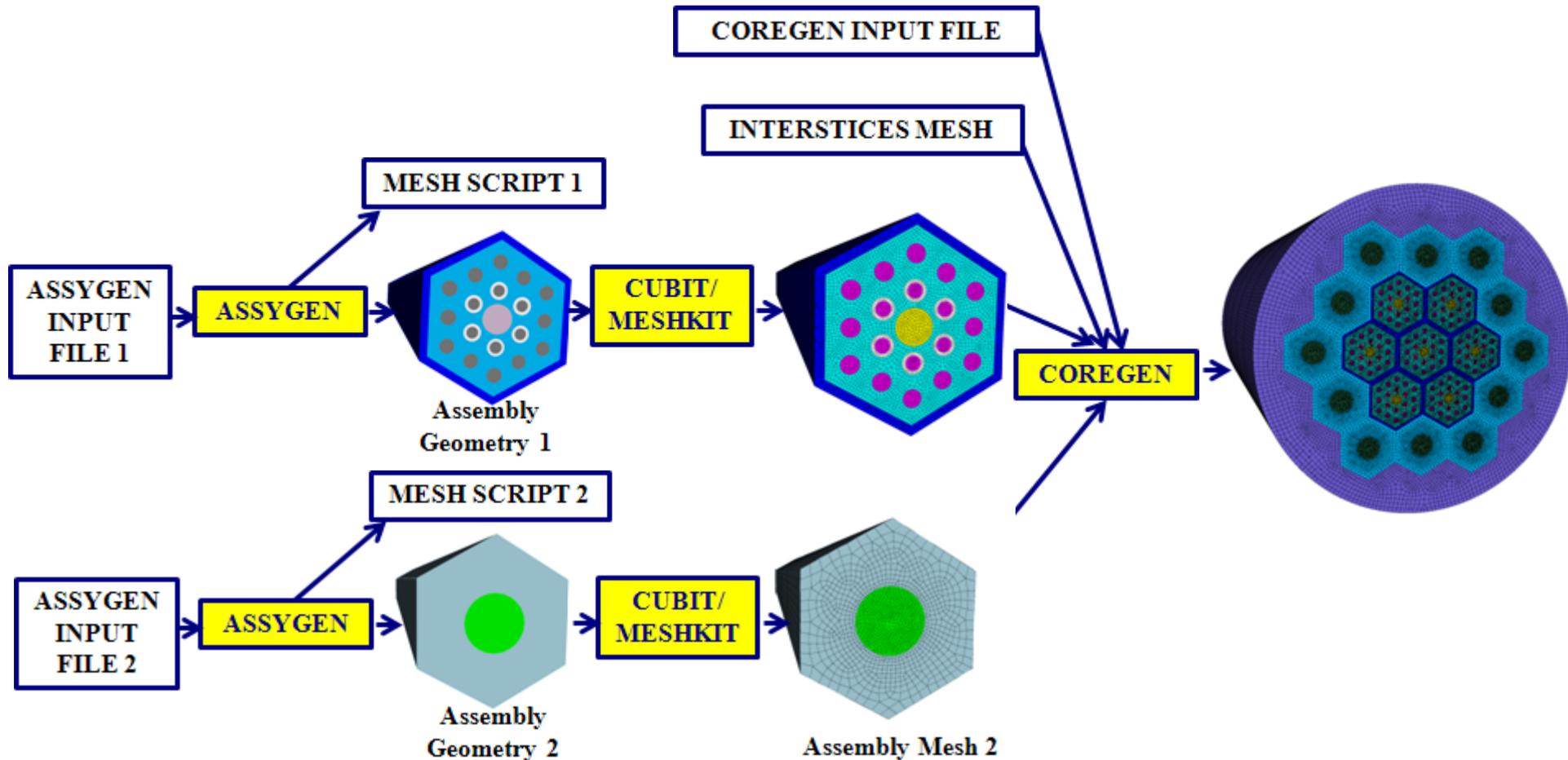
AssyGen input file for creating assembly geometry (right) and CUBIT mesh script.

RGG: Automated 3-Stage Meshing Process

STAGE 1: ASSYGEN

STAGE 2: MESHING

STAGE 3: COREGEN

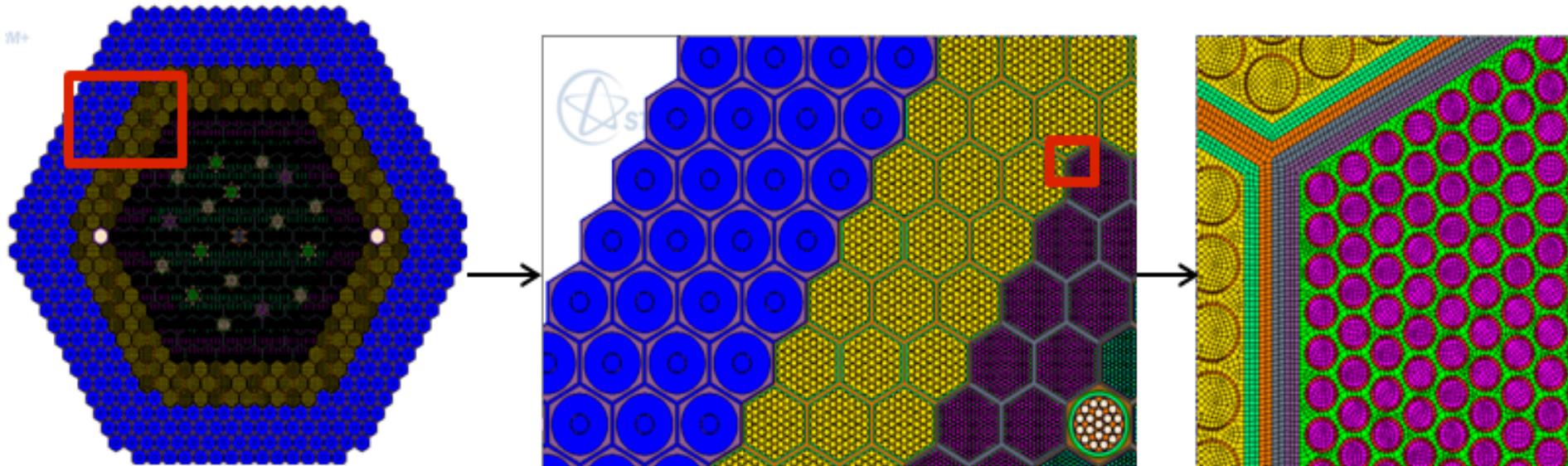


[1] T.J. Tautges and R. Jain (2011) "Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach." Engineering with Computers, 2011.

[2] RGG-MeshKit README, wiki website (2010)

<http://trac.mcs.anl.gov/projects/fathom/browser/MeshKit/trunk/rgg/README>, <http://trac.mcs.anl.gov/projects/fathom/wiki/rgg>

RGG Application - MONJU Parallel CoreGen



MONJU reactor, full core model: 9.7M hexes, 99k vols takes 4.3GB and 176 mins. 715 assemblies.

Parallel CoreGen Result: 101M hex in **90 seconds** on 712 procs

Conclusions

- Implementing things like geometry, mesh, relations in functionality-based components gives you options on what to include in your application
- CGM, MOAB, Lasso provide functionality for both mesh generation and analysis support
- All the components described here are open-source, most of them under an LGPL-type open source license

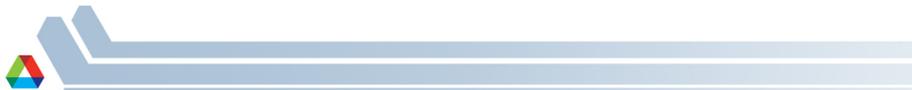
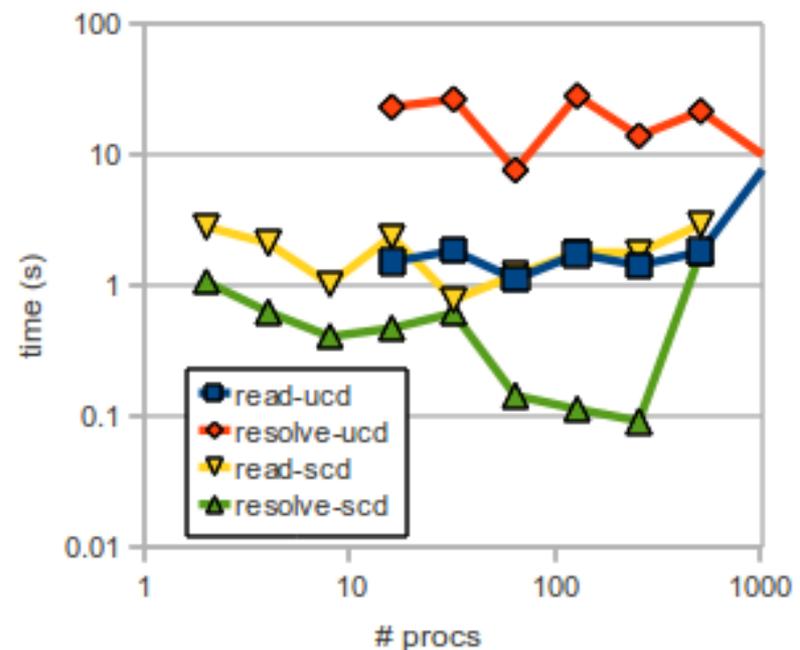
<http://sigma.mcs.anl.gov>

moab-dev@mcs.anl.gov



Resolving Interface Mesh (Structured)

- For structured mesh, can compute shared vertex handles directly, with minimal communication
 - Given: global IJK parametric space, partitioning method
 - Compute neighbors in +/- IJK directions
 - Communicate starting vertex handle to neighbors
 - For vertices on bdy, compute remote handles on all sharing procs using their start handles & 3D to 1D indexing
- Reduces resolve times by more than 1 order of magnitude
 - 150M vertex mesh on fusion



Test Problem Description

- Generate two sets of progressively finer hex, tet meshes from the same geometry
- Put test function (sum of 4 Gaussian distributions) on hex mesh, map to tet mesh
- Measure $f(\vec{x}) = \sum_i f_i \exp\left(-\frac{(\vec{x} - \vec{q}_i)^2}{w_i^2}\right)$

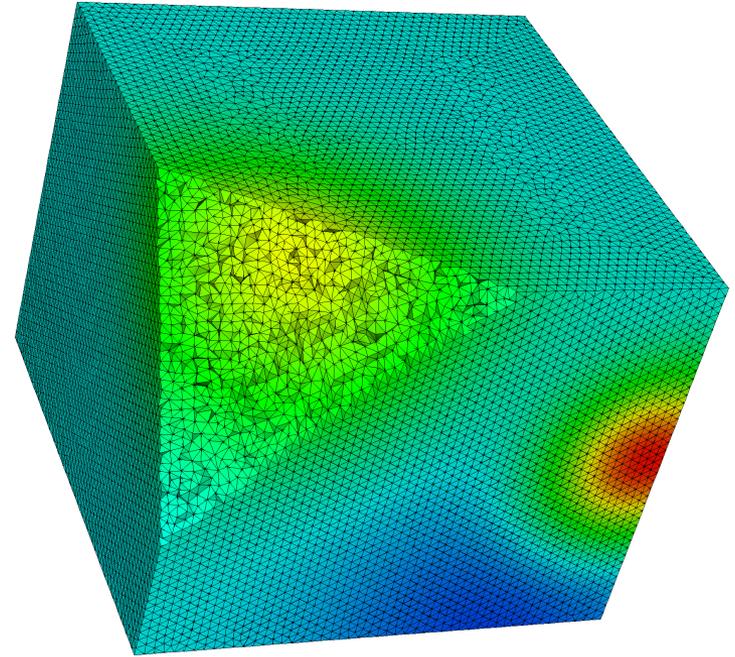
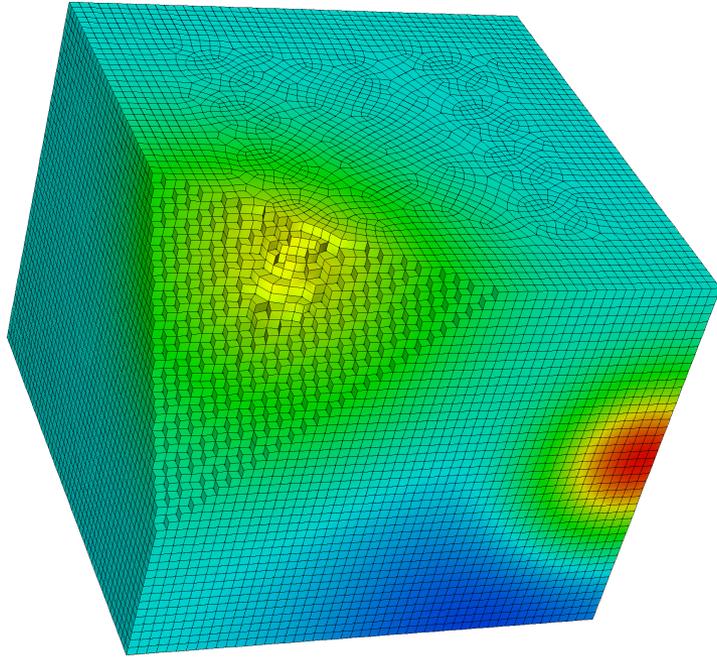
$$w_i = (10 \quad 5 \quad 1 \quad 8)$$

$$f_i = (-1 \quad 3 \quad -5 \quad 2)$$

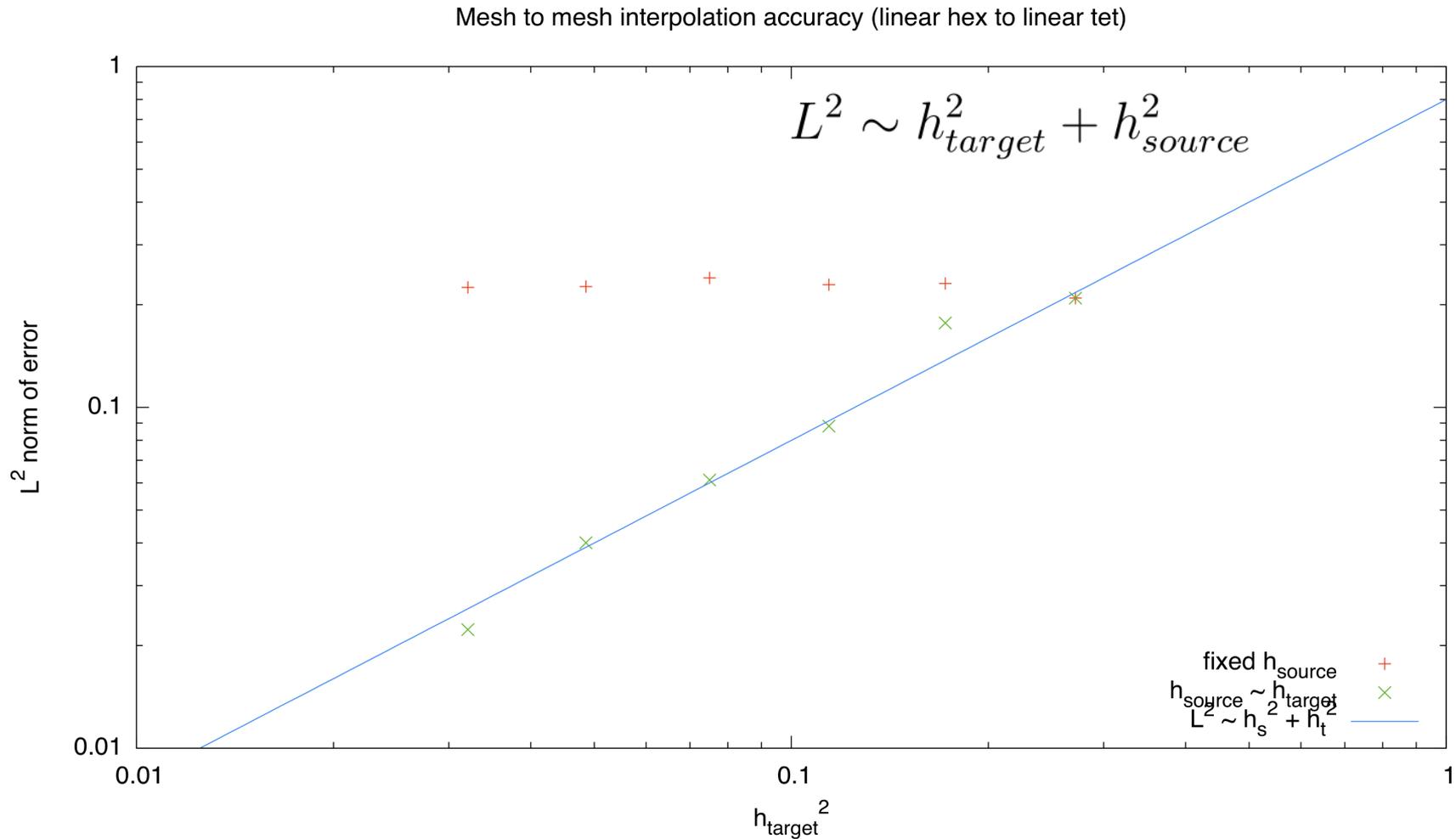
$$\vec{q}_i = \begin{pmatrix} 0 & 0 & 0 \\ -5 & -5 & 10 \\ 12 & 10 & 30 \\ -3 & 14 & 20 \end{pmatrix}$$



Hex, Tet Mesh Examples



Solution Transfer: Accuracy Scaling



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)

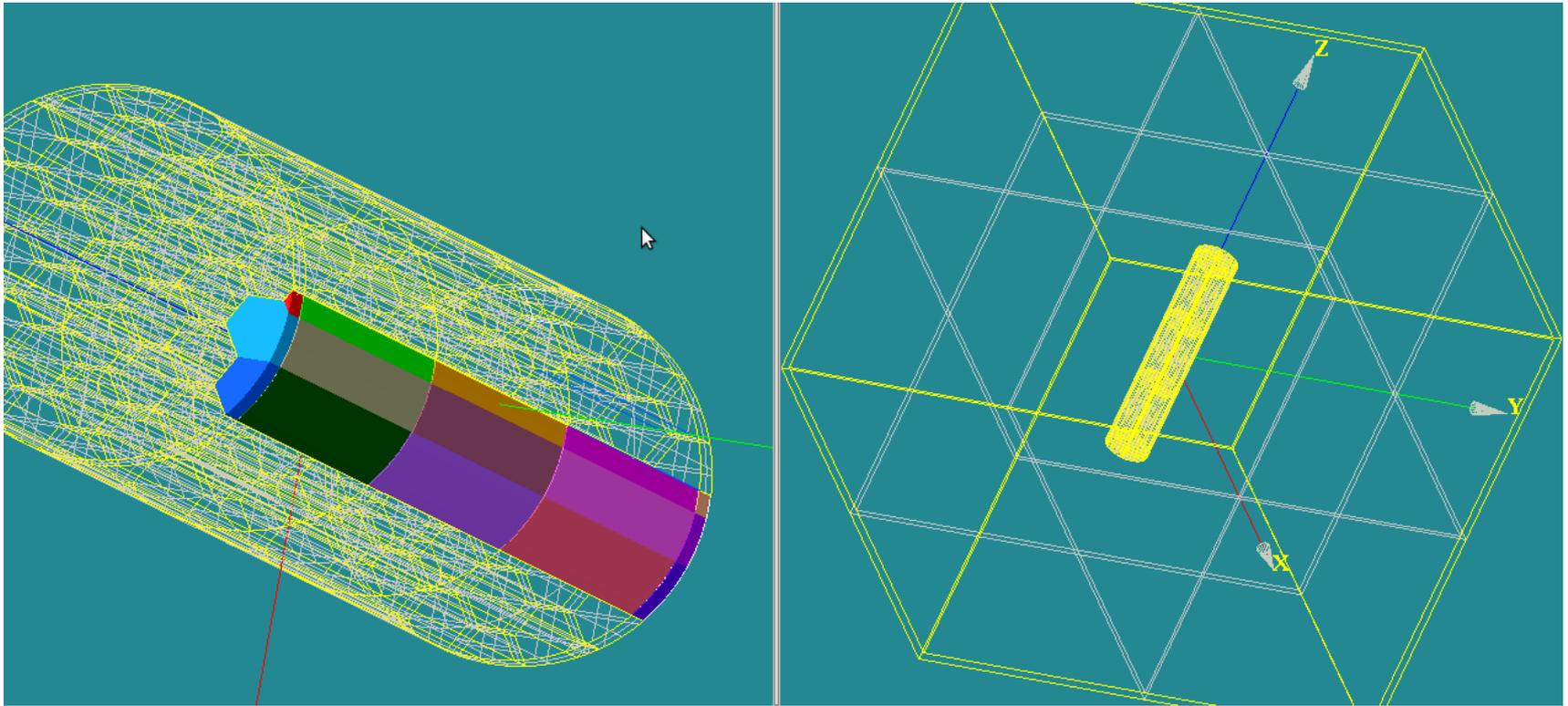
Open.CASCADE (OCC) port

- 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



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

- Mcnp2cad, an MCNP-CAD converter
 - ~10 minutes generation time, 216 volumes



Lasso Relations Tool

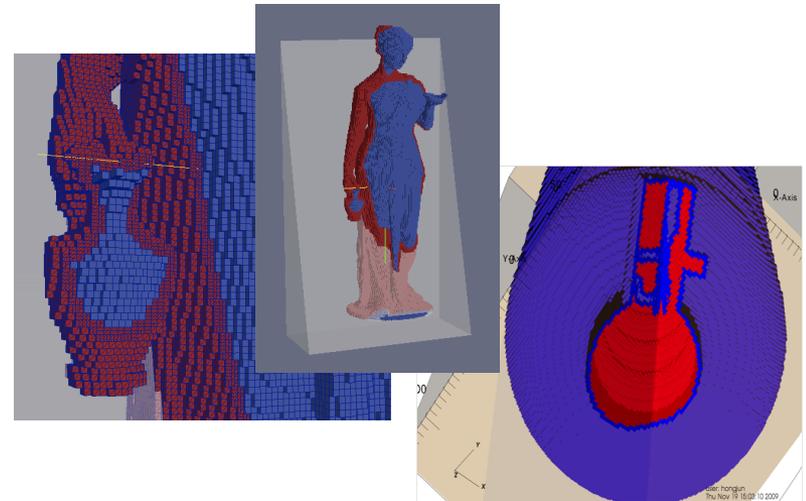
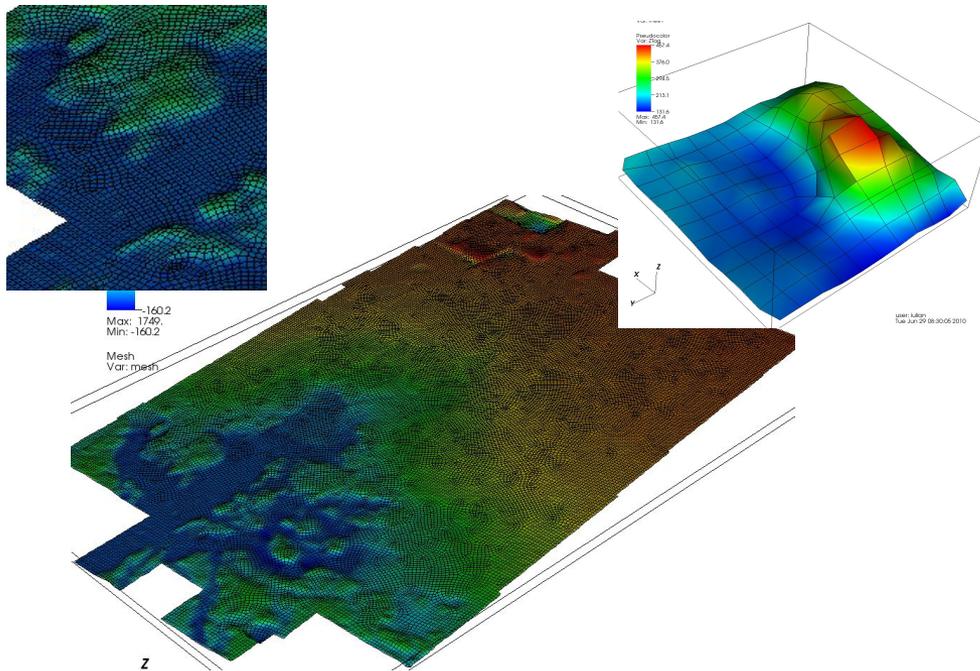
- For some applications, need to relate mesh to geometry
 - Adaptive mesh refinement
 - Smooth surface-based boundary conditions
- Separation of geometry, mesh in independent components means we need a higher-level component that depends on those
- Lasso: tool for recovering, querying, setting geometry-mesh 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



Using Components for Geometry, Mesh Other Examples

Facet-based geometry/API (iGeom/MOAB) +
Decimation (MeshKit) + Paving (MeshKit/CUBIT)
True ice bed geometry & mesh, tangent field for BC evaluation



Facets from CAD (CGM/MOAB) +
Fast ray-tracing (MOAB) =
Fast embedded boundary meshing

Conclusions

- MOAB is a general library for representing structured, unstructured mesh & associated data
- Efficient in memory and time
- Generic data model handles most types of simulation data & metadata
- Parallel support up to 64k procs so far (on BG/P)



FindConnect (C) Notes

- Typical inout list usage
 - X *list, int list_alloc = 0, int list_size
 - Setting list_alloc to zero *OR* list = NULL indicates list is unallocated, so it will be allocated inside *iMesh_getEntities*
 - Addresses of these parameters passed into iMesh_getEntities
- Inout list declared inside 'for' loop
- Memory de-allocated inside loop



FindConnect (Fortran) Notes

- Cray pointer usage
 - “pointer” (rpverts, rpooffsets, etc.) declared as type integer
 - Careful – integer*8 or integer*4, 64- or 32-bit
 - “pointee” (verts, ioffsets, etc.) implicitly typed or declared explicitly
 - pointer statement equivalences pointer to start of pointee array
 - pointee un-allocated until explicitly allocated
- Set allocated size (ents_alloc) to zero to force allocation in iMesh_getEntities; arguments passed by reference by default, use %VAL extension to pass by value; pointers passed by reference by default, like arrays
- Allocated size set to zero to force re-allocation in every iteration of do loop
- Use C-based free function to de-allocate memory



FindConnect Makefile

```
include ${IMESH_DIR}/iMesh-Defs.inc
```

```
FindConnectC: FindConnectC.o
```

```
$(CC) $(CFLAGS) -o $@ FindConnectC.o ${IMESH_LIBS}
```

```
FindConnectF: FindConnectF.o
```

```
$(FC) -o $@ FindConnectF.o ${IMESH_LIBS}
```

```
.cpp.o:
```

```
$(CXX) -c $(CXXFLAGS) ${IMESH_INCLUDES} $<
```

```
.cc.o:
```

```
$(CC) -c $(CFLAGS) ${IMESH_INCLUDES} $<
```

```
.F.o:
```

```
$(FC) -c $(FFLAGS) ${IMESH_INCLUDES} $<
```

