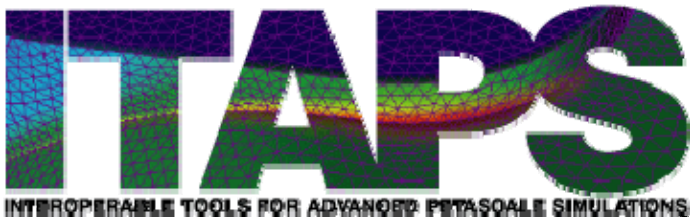
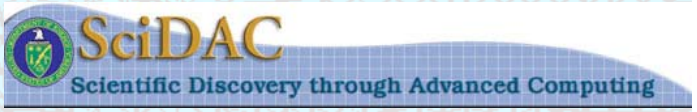
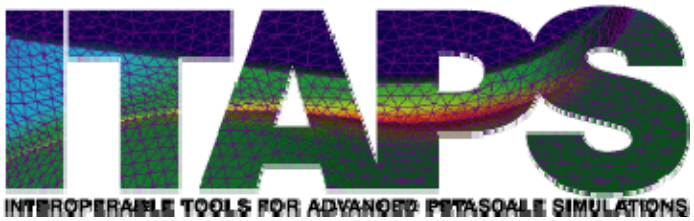


Interoperable Tools for Advanced Petascale Simulations (ITAPS)

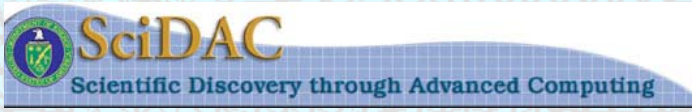
TUTORIAL

June 29, 2007



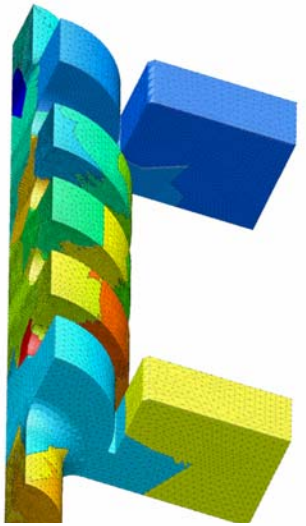


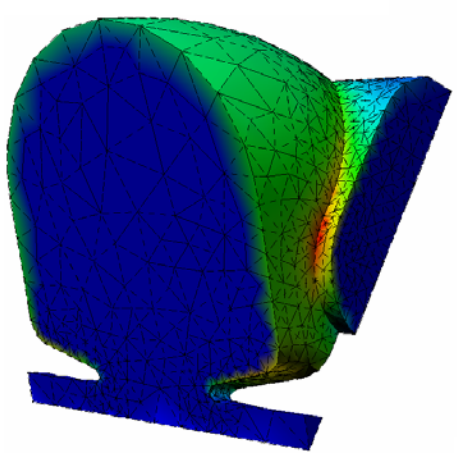
PART 1: Overview and Introduction

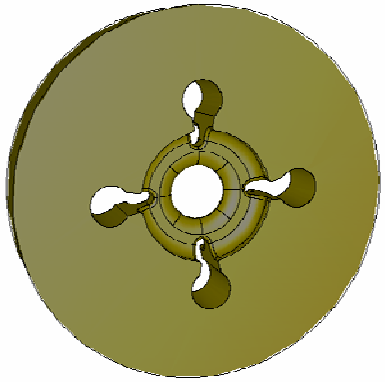



ITAPS focuses on interoperable meshing and geometry services for SciDAC

- ITAPS Goal
  - Improve SciDAC applications' ability to take advantage of state-of-the-art meshing and geometry tools
  - Develop the next generation of meshing and geometry tools for petascale computing
- Technology Focus Areas
  - Complex geometry
  - High quality meshes and adaptivity
  - Coupled phenomenon
  - Dynamic mesh calculations
  - Tera/Petascale computing







The Challenge

- Pre-existing ITAPS tools all meet particular needs, but
  - They do not interoperate to form high level services
  - They cannot be easily interchanged in an application
- In general the technology requires too much software expertise from application scientists
  - Difficult to improve existing codes
  - Difficult to design and implement new codes

The ITAPS center recognizes this gap and is addressing the technical and human barriers preventing use of advanced, adaptive, and/or hybrid methods

The ITAPS team has diverse expertise

Lori Diachin
LLNL

Ed d'Azevedo
ORNL

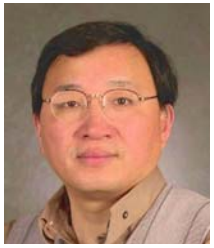
Karen Devine
SNL

Jim Glimm
BNL/SUNY SB

Bill Henshaw
LLNL

Ken Jansen
RPI

Pat Knupp
SNL



Xiaolin Li
SUNY SB

Ahmed Khamayseh
ORNL

Roman Samulyak
BNL

Carl Ollivier-Gooch
UBC

Mark Shephard
RPI

Tim Tautges
ANL

Harold Trease
PNNL

Our senior personnel are experts in complex geometry tools, mesh generation, mesh quality improvement, front tracking, partitioning, mesh refinement, PDE solvers, and working with application scientists

SciDAC
Scientific Discovery through Advanced Computing

NEEDED

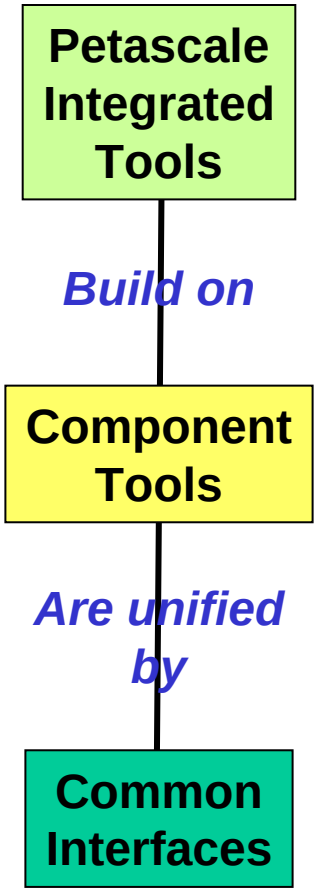
- Need for advanced mesh/geom libraries for parallel simulations

NEEDED

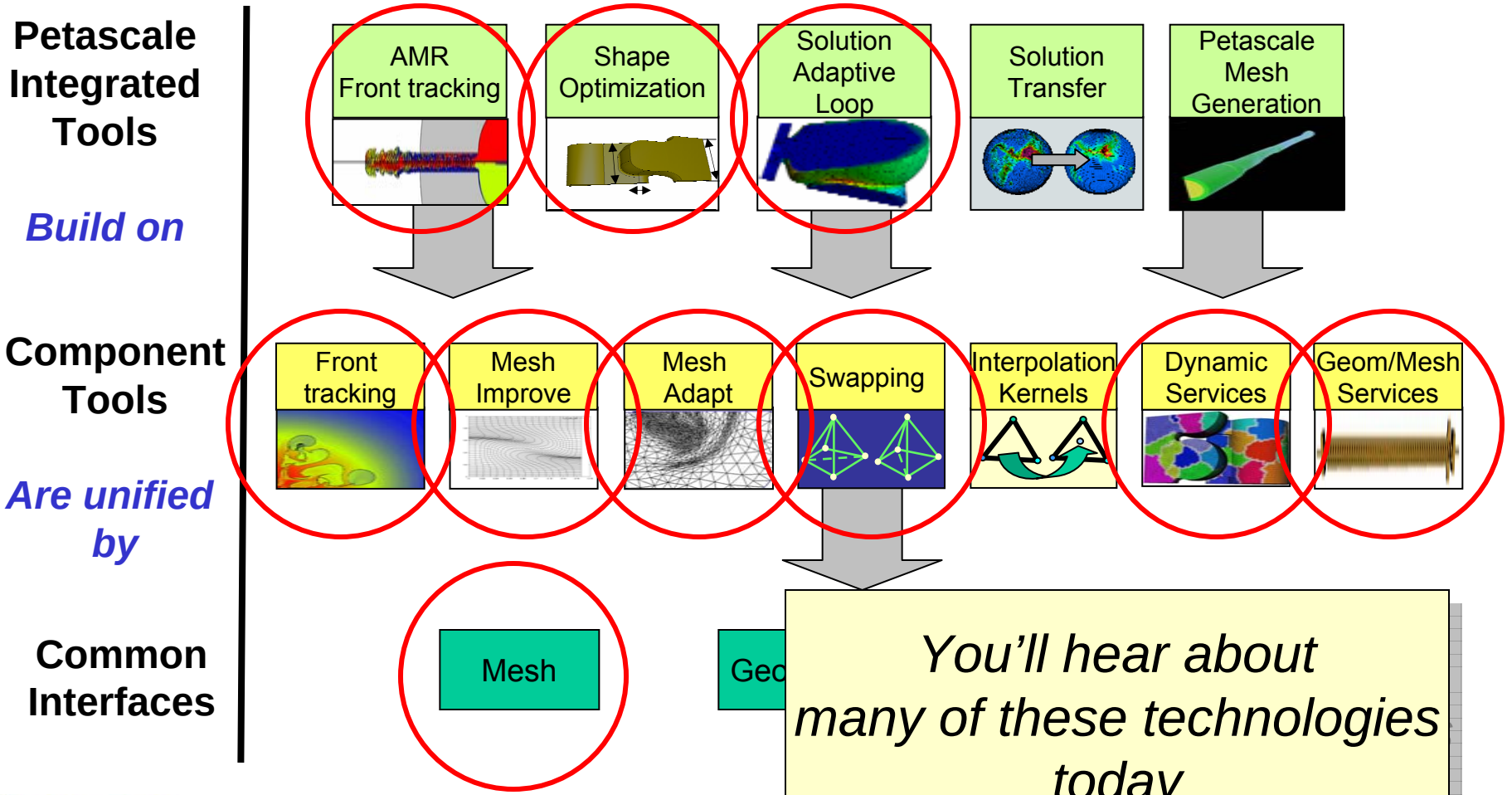
- Motivating examples

The ITAPS long term technology goal uses a hierarchical approach

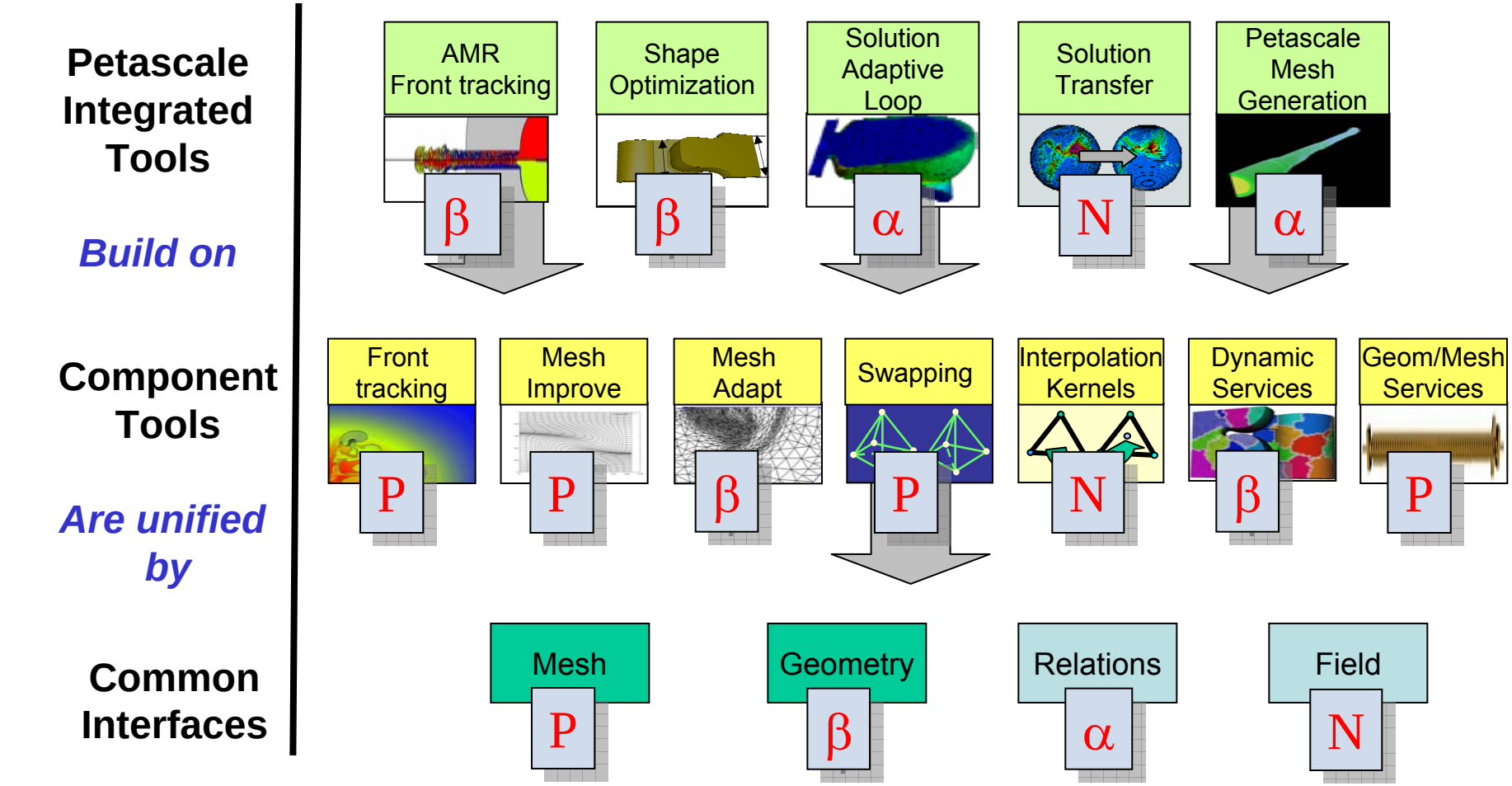
- Build on successes with SciDAC-1 applications and explore new opportunities with SciDAC-2 application teams
- Develop and deploy key mesh, geometry and field manipulation *component services* needed for petascale computing applications
- Develop advanced functionality *integrated services* to support SciDAC application needs
  - Combine component services together
  - Unify tools with *common interfaces* to enable interoperability



ITAPS will expand and build new DOE meshing technologies in SciDAC-2



Status of the ITAPS services and their use of the TSTT interfaces



NEEDED

- Comparison with other development approaches

ITAPS
INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

PART 2: ITAPS Data Model

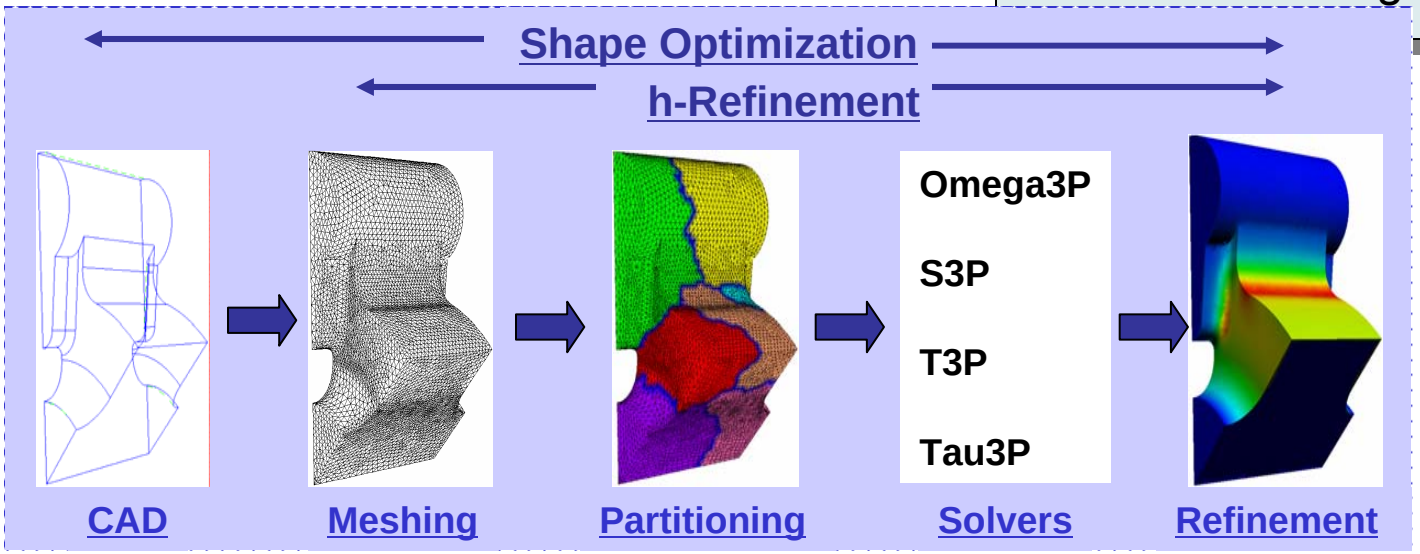
SciDAC
Scientific Discovery through Advanced Computing

Sandia National Laboratories
Rensselaer
STONY BROOK
THE UNIVERSITY OF BRITISH COLUMBIA
BROOKHAVEN NATIONAL LABORATORY
OAK RIDGE NATIONAL LABORATORY
Pacific Northwest National Laboratory

The ITAPS interoperability goal requires abstracting the data model and information flow

- Information flows from geometrical representation of the domain to the mesh to the solvers and post-processing tools
- Adaptive loops and design optimization requires a loop

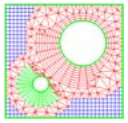
- The data model must encompass a broad spectrum of mesh types and usage scenarios
- A set of common interfaces
  - Implementation and data structure neutral
  - Small enough to encourage adoption
  - Flexible enough to support a broad range of functionality



The ITAPS data model abstracts PDE-simulation data hierarchy

- Core Data Types

  - *Geometric Data*: provides a high leve
    boundaries of the computational dom
    mesh data



  - *Mesh Data*: provides the geometric a
    associated with the discrete represer

  - *Field Data*: provides access to the tir
    variables associated with application solution. These can be scalars, vectors, tensors, and associated with any mesh entity.

Each core data type has an ITAPS interface
- Mesh: iMesh
- Geometry: iGeom
- Fields: iField
- Relations: iRel

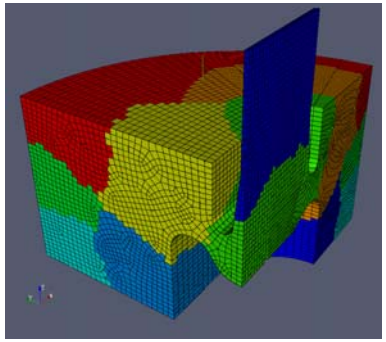
- Data Relation Managers
  - Provides control of the relationships among the core data types
  - Resolves cross references between entities in different groups
  - Provides functionality that depends on multiple core data types

ITAPS Data Model – Tim's slides

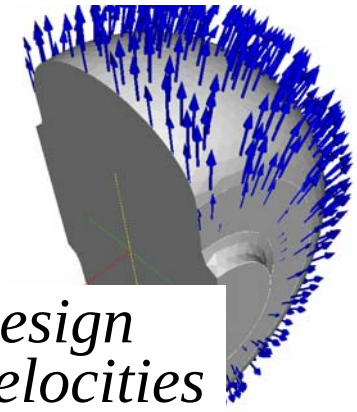
- 4 fundamental "types":
  - *Entity:* fine-grained entities in interface (vertex, tri, hex)
  - *Entity Set:* arbitrary set of entities & other sets
    - Parent/child relations, for embedded graphs between sets
  - *Interface:* object on which interface functions are called and through which other data are obtained
  - *Tag:* named datum annotated to Entitys, Entity Sets, Interface
- Instances accessed using opaque (type-less) "handles"

ITAPS Data Model Usage – Tim's slides

Mesh Partition

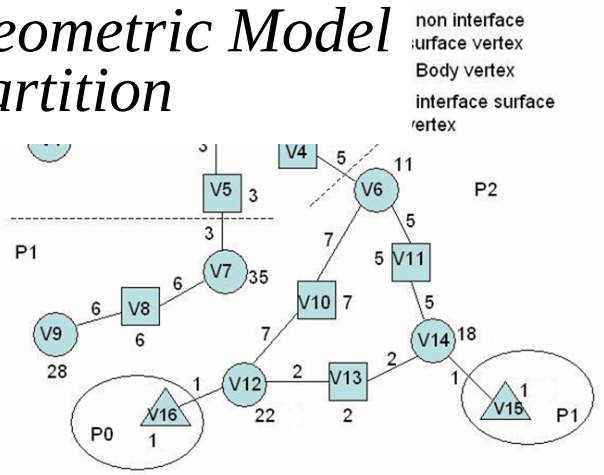


Klystron mesh, SLAC/SNL

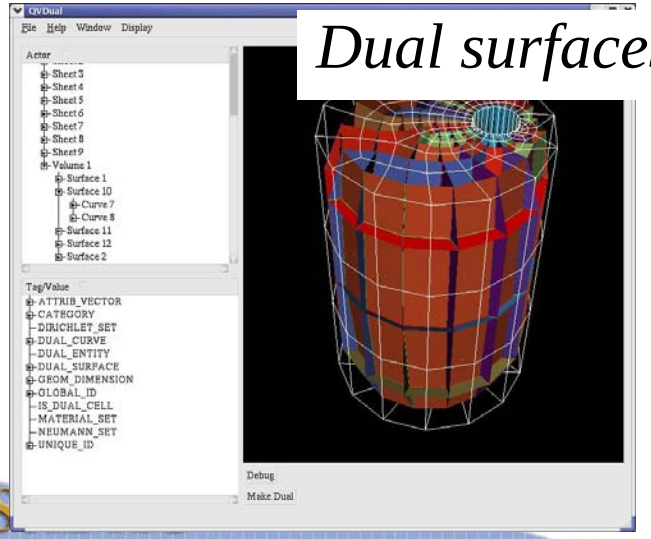


Design Velocities

Geometric Model Partition

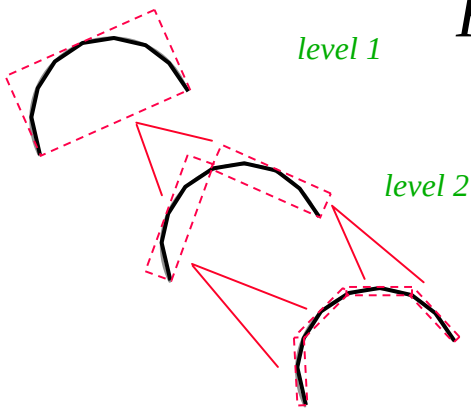


Dual surfaces



Hierarchical OBB Tree

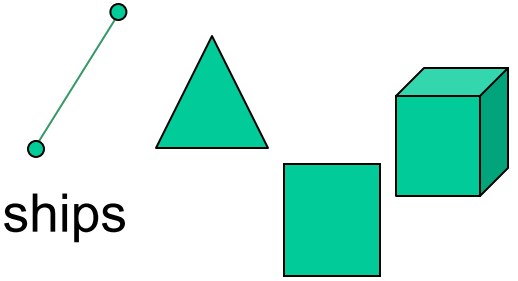
OBB Tree



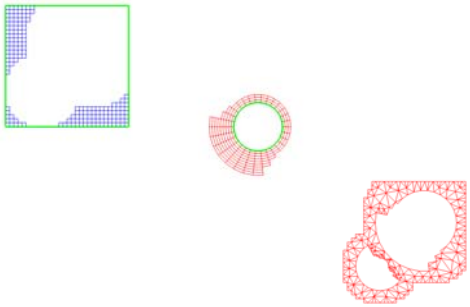
ITAPS Data Model (cont.) - Tim Slides'

- Important enumerated types:
  - EntityType (iBase_VERTEX, EDGE, FACE, REGION)
  - EntityTopology (iMesh_POINT, LINE, TRI, QUAD, ...)
  - StorageOrder (iBase_BLOCKED, INTERLEAVED)
  - TagDataType (iBase_INTEGER, DOUBLE, ENTITY_HANDLE)
  - ErrorType (iBase_SUCCESS, FAILURE, ...)
- Enumerated type & function names both have iBase, iMesh, iGeom, other names prepended

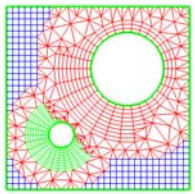
The core data model consists of entities and entity sets



- Entity Definition
  - Unique type and topology
  - Canonical ordering defines adjacency relationships
- Entity Set Definition
  - Arbitrary grouping of ITAPS entities
  - There is a single “Root Set”
  - Relationships among entity sets
    - Contained-in
    - Hierarchical

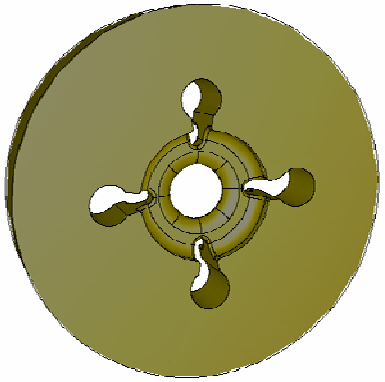


- Tags allow user-defined data association with entities and entity sets
- Blend of abstract concepts and familiar mesh/geometry specifics



The geometry interface provides access to the computational domain

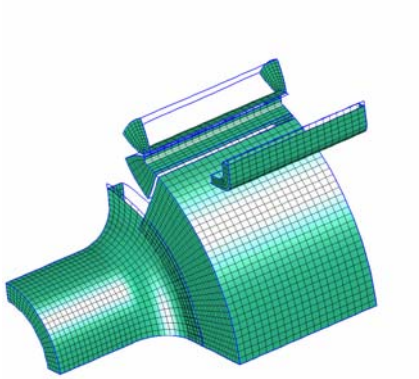
- Must support
  - automatic mesh generation
  - mesh adaptation
  - tracking domain changes
  - relating information between alternative discretizations

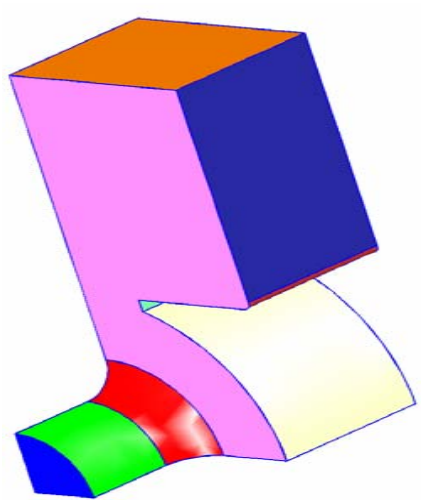


- Builds on boundary representations of geometry
- Used to support various underlying representations
  - Commercial modelers (e.g., Parasolid, ACIS)
  - Modelers that operate from standard files (e.g. IGES, STEP)
  - Models constructed from an input mesh

Basic and advanced functionalities are supported in the geometry interface

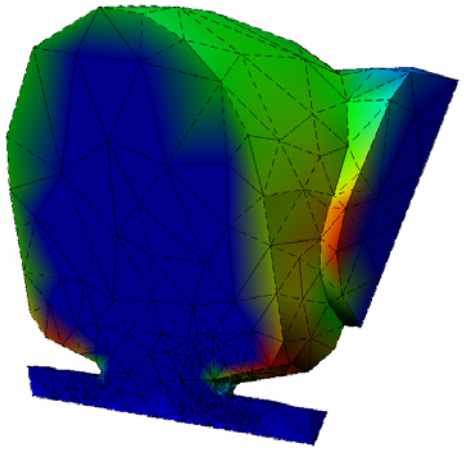
- Model loading and initiation
- Topological queries of entities and adjacencies
- Pointwise geometric shape interrogation
- Parametric coordinate systems
- Model topology modification

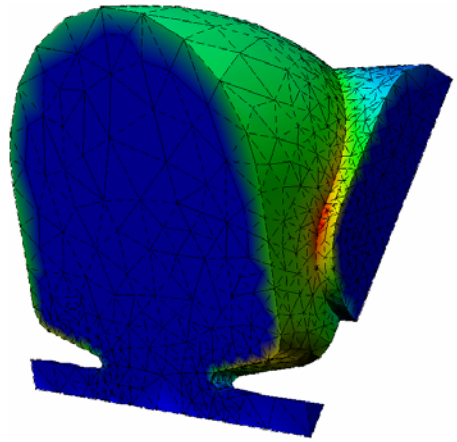




The mesh interface provides access to the discrete representation of the domain

- Must support
  - Access to mesh geometry and topology
  - User-defined mesh manipulation and adaptivity
  - Grouping of related mesh entities together (e.g. for boundary conditions)
  - Distribution across distributed memory machines
  - Relations to the geometry and field data
- Builds on a general data model that is largely suited for unstructured grids
- Implemented using a variety of mesh types, software, and for a number of different usage scenarios



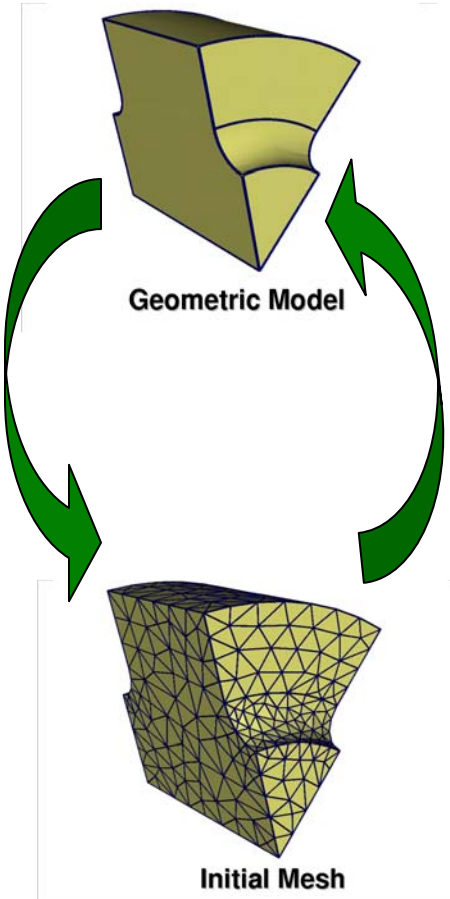


The mesh interface supports both basic and more advanced functionalities

- Provides basic access to vertex coordinates and adjacency information
  - Mesh loading and saving
  - Global information such as the root set, geometric dimension, number of entities of a given type or topology
  - Primitive array access for entity sets
  - Global entity arrays for entity sets
  - Coarse grain ‘block’ iterators to preserve performance
- Mesh modification
  - Adding / Deleting entities
  - Vertex relocation
  - No validity checks

Relating mesh and geometry data is critical for advanced ITAPS services

- Required for e.g., adaptive loops, mesh quality improvement
- Mesh/Geometry Classification Interface
  - Manages the relationship between the high level geometric description and the mesh
  - Called by an application that knows about both
- Capabilities
  - For a given mesh entity, get the geometric entity against which it is classified
  - Establish a classification relationship between a mesh entity and a geometric entity



Core functions must be implemented to be TSTT compliant

- The smallest set of functions required to be considered TSTT compliant
- Provides basic access to vertex coordinates and adjacency information
  - Mesh loading and saving
  - Accessing global information such as the root set, geometric dimension, number of entities of a given type or topology
  - Primitive array access for entity sets
  - Global entity arrays for entity sets
- Reference implementations can be used to provide advanced capabilities needed for TSTT services
  - Requires a copy of mesh data
  - Allows incremental implementation of advanced functions

Issues that have arisen

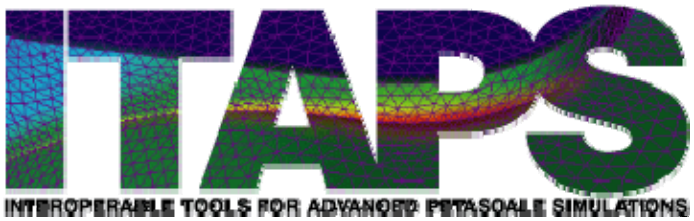
- Nomenclature is harder than we first thought
- Cannot achieve the 100 percent solution, so...
  - What level of functionality should be supported?
    - Minimal interfaces only?
    - Interfaces for convenience and performance?
  - What about support of existing packages?
    - Are there atomic operations that all support?
    - What additional functionalities from existing packages should be required?
  - What about additional functionalities such as locking?
- The devil is in the details
  - Memory management, multi-valued tag behavior, separation of functionality, nomenclature (did I mention that already?)
- Language interoperability and performance

Interface implementations are well underway

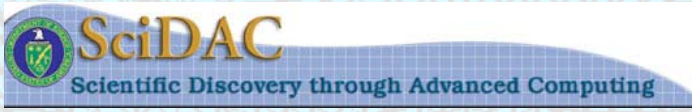
- TSTT Mesh 0.7 Interface complete
- Geometry and relations interfaces well on their way
- Preliminary field data interface
- Implementations
  - Mesh: FMDB, GRUMMP, NWGrid, MOAB, Frontier
  - Geometry: CGM
- C, C++, and Fortran language interoperability through C-bindings and SIDL/Babel (CCA)
  - Analyzing performance ramifications of SIDL/Babel language interoperability tools (joint with the CCA)
- Interfaces stable enough to build services upon them and test interoperability

NEEDED

- Parallelism in the ITAPS data model



PART 3: ITAPS Interfaces



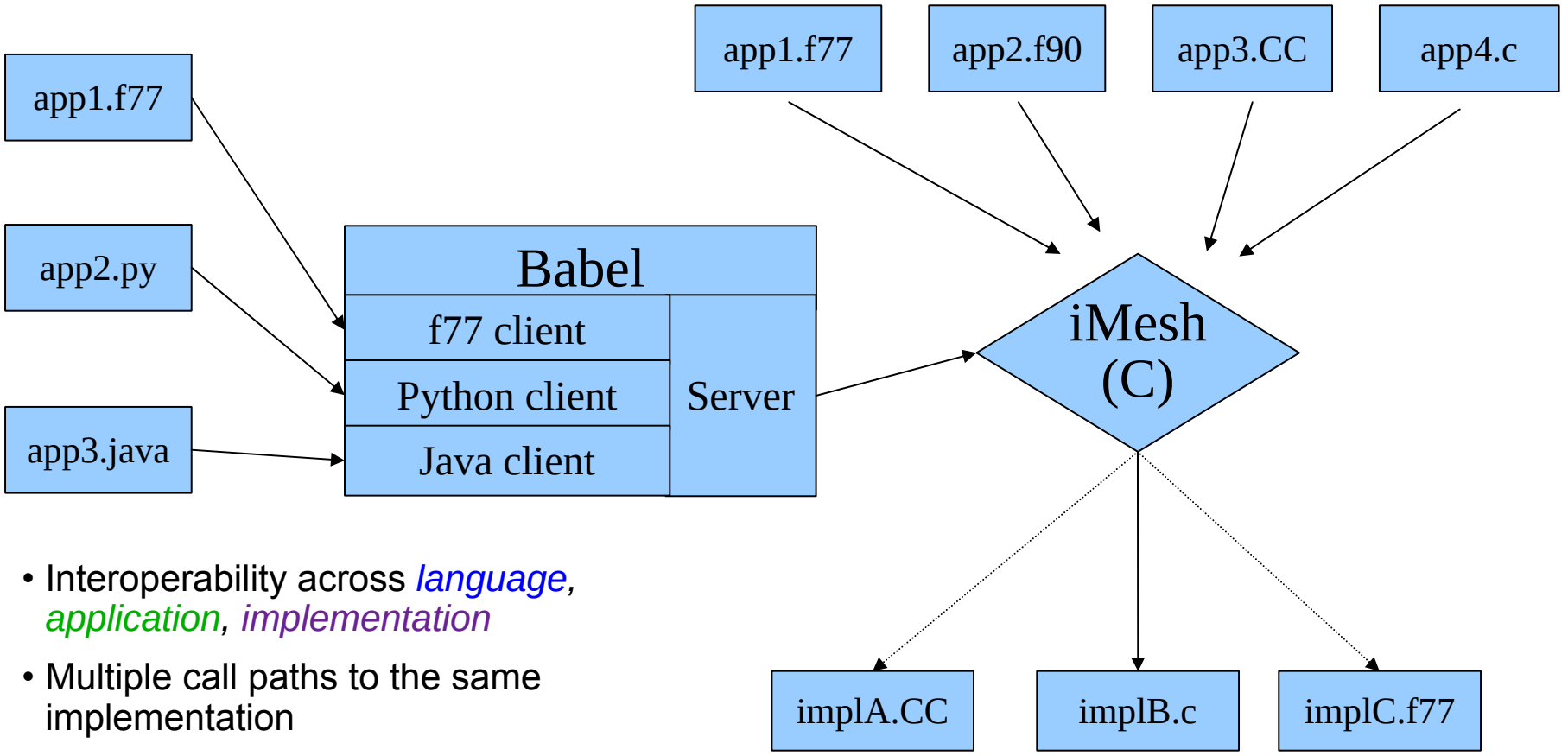

NEEDED

- Design philosophy and basic tenets

Outline

- ITAPS Data Model
- ITAPS Interfaces (w/ examples)
  - iMesh
  - iGeom
  - iRel
- Best Practices (for Performance)
- Language Interoperability
  - C-binding interface
  - SIDL/Babel

ITAPS Interfaces Designed for Interoperability



- Interoperability across *language*, *application*, *implementation*
- Multiple call paths to the same implementation
- Efficiency preserved using direct, C-based interface

SciDAC
Scientific Discovery through Advanced Computing

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_LINK}

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

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

ITAPS API's: Argument Handling Conventions

- ITAPS API's are C-like and can be called directly from C, Fortran, C++
- Arguments pass by value (in) or reference (inout, out)
  - Fortran: use %VAL extension
- Memory allocation for lists done in application *or* implementation
  - If inout list comes in allocated, length must be long enough to store results of call
  - By definition, allocation/deallocation done using C malloc/free; application required to free memory returned by implementation
  - Fortran: Use “cray pointer” extension (equivalences to normal f77 array)
- Handle types typedef'd to size_t (iBase_EntityHandle, iBase_EntitySetHandle, iBase_TagHandle, iMesh_Instance)
- Strings: char*, with length passed by value after all other args
- Enum's: *values (iBase_SUCCESS, etc.) available for comparison operations, but passed as integer arguments*
  - *Fortran: named parameters*

Argument Handling Conventions

Issue	C	FORTRAN	SIDL
Function Names	iXxxx_ prefix	Same as C	Removed iXxxx_ prefix; SIDL interface organization
Interface Handle	Typedef'd to size_t, as type iXxxx_Instance; instance handle is 1st argument to all functions	#define'd as type Integer; handle instance is 1st argument to all functions	Interface type derived from sidl.BaseInterface
Enumerated Variables	All arguments integer-type instead of enum-type; values from enumerated types	Same, with enum values defined as FORTRAN parameters	Int-type arguments; enumerated types defined in iXxxx:: namespace, and values appear as iXxxx::enumName_enumValue
Entity, Set, Tag Handles	Typedef'd as size_t; typedef types iBase_EntityHandle, iBase_EntitySetHandle, iBase_TagHandle	#define'd as type Integer	Handles declared as SIDL opaque type (mapped to void* in C/C++ server)
Lists	· In: X *list, int occupied_size · Inout: X **list, int *allocated_size, int **occupied_size · malloc/free-based memory allocation/deallocation	Same, with Cray pointers used to reference arrays (see FindConnectF example	· In: sidl::array<X> list, int occupied_size · Inout: sidl::array<X> &list, int &occupied_size · sidl::array class memory allocation
String	char*-type, with string length(s) at end of argument list	char[]-type without extra length argument (this length gets added implicitly by FORTRAN compiler)	sidl::string type without extra length argument

iMesh API Summary

- Logically arranged into interfaces, but not explicitly arranged as such in C
  - See iMesh.h or iMesh.sidl
- Basic (Mesh): load, save, getEntities, getNumOfType/Topo, getAllVtxCoordinates, getAdjacencies
- Entity: init/get/reset/endEntIter (iterators), getEntType/Topo, getEntAdj, getVtxCoord
- Arr (Entity arrays): like Entity, but for arrays of entities
- Modify: createVtx/Ent, setVtxCoord, deleteEnt

Imesh API Summary (cont.)

- From iBase:
  - Tag: create/destroyTag, getTagName/SizeBytes/SizeValues/Handle/Type
  - EntTag: get/setData, get/setInt/Dbl/EHData, getAllTags, rmvTag
  - ArrTag: like EntTag, but for arrays of entities
  - SetTag: like EntTag, but for entity sets
  - EntSet: create/destroyEntSet, add/remove entity/entities/set, isEnt/EntSetContained
  - SetRelation: add/rmvPrntChld, isChildOf, getNumChld/Prnt, getChldn/Prnts
  - SetBoolOps: subtract, intersect, unite
- iBase-inherited function names still start with 'iMesh_' to avoid name collision with other iBase-inherited interfaces (iGeom, iRel, etc.)

NEEDED

- Accessing global information

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;
```

1

```
    // 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;
}
```

2

3

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

1. Inout list declared inside 'for' loop
2. Memory de-allocated inside loop

Slightly More Complicated Example: FindConnect (Fortran)

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

c declarations
      iMesh_Instance mesh
      integer ierr, ents
      pointer (rpents, ents(0:*))
      integer rpverts, rpallverts, ipoffsets
      pointer (rpverts, verts(0:*))
      pointer (rpallverts, allverts(0:*))
      pointer (ipoffsets, ioffsets(0,*))
      integer 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),
     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
        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)
      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), allverts,
     1  allverts_alloc, allverts_size, offsets,
     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
```

1 2 3 4

FindConnect (Fortran) Notes

1. Cray pointer usage
   - "pointer" (rpverts, rpoffsets, etc.) declared as type integer
   - "pointee" (verts, ioffsets, etc.) implicitly typed or declared explicitly
   - pointer statement equivalences pointer to start of pointee array
   - pointee un-allocated until explicitly allocated
2. 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
3. Allocated size set to zero to force re-allocation in every iteration of do loop
4. Use C-based free function to de-allocate memory

Slightly More Complicated Example: FindConnect (SIDL/C++)

```
#include <iostream>
#include "iMesh.hh"
using std;

typedef void* EntityHandle;

int main( int argc, char *argv[] )
{
    // create the Mesh instance
  iMesh::Mesh mesh = iMesh::newMesh("");

    // load the mesh
  mesh.load(0, "125hex.g", "");

    // get all 3d elements
  sidl::array<EntityHandle> ents;
  int ents_size;
  mesh.getEntities(0,
     iBase::EntityType_REGION,

   iMesh::EntityTopology_ALL_TOPOLOGIES,
     ents, ents_size);

  int vert_uses = 0;
```

```
    // iterate through them; first have to get an
    // Entity interface instance
  iMesh::Entity mesh_ent = mesh;
  for (int i = 0; i < ents_size; i++) {
      // get connectivity
    sidl::array<EntityHandle> verts;
    int verts_size;
    mesh_ent.getEntAdj(ents[i],
       iBase::EntityType_VERTEX,
       verts, verts_size);
      // sum number of vertex uses
    vert_uses += verts_size;
  }

    // now get adjacencies in one big block
  sidl::array<EntityHandle> allverts;
  sidl::array<int> offsets;
  int allverts_size, offsets_size;
  iMesh::Arr mesh_arr = mesh;
  mesh_arr.getEntArrAdj(ents, ents_size,
     iBase::EntityType_VERTEX,
     allverts, allverts_size,
     offsets, offsets_size);

    // compare results of two calling methods
  if (allverts_size .ne. vert_uses) then
    cout << "Sizes didn''t agree!" << endl;
  else cout << "Sizes did agree!" << endl;

  return true;
}
```

1 2 3 4 5

FindConnect (SIDL/C++) Notes

1. Static function on iMesh class used to instantiate interface
2. List declaration using SIDL templated array
3. getEntities member function called on iMesh instance
4. Assignment operator-based cast to iMesh::Entity interface
5. Declaration of list inside for loop; re-constructed on every iteration, and de-constructed using use counts

FindConnect Makefile

```
include /home/tautges/MOAB_gcc4.2/lib/iMesh-Defs.inc

FC = gfortran-4.2
CXX = g++-4.2
CC = gcc-4.2

CXXFLAGS = -g
CFLAGS = -g
FFLAGS = -g -fcray-pointer
FLFLAGS = -g  -L/home/tautges/gcc-4.2/lib -L/usr/lib/gcc/i486-linux-gnu/4.2.1 -lgfortranbegin
    -lgfortran -lm

FindConnectS: FindConnectS.o
$(CXX)  $(CXXFLAGS) -o $@ FindConnect.o ${IMESH_SIDL_LIBS_LINK}

FindConnectC: FindConnectC.o
$(CC)  $(CFLAGS) -o $@ FindConnectC.o ${IMESH_LIBS_LINK}

FindConnectF: FindConnectF.o
$(CXX) -o $@ FindConnectF.o $(FLFLAGS) ${IMESH_LIBS_LINK}

.cpp.o:
     ${CXX} -c ${CXXFLAGS} ${IMESH_INCLUDES} $<
.cc.o:
     ${CC} -c ${CFLAGS} ${IMESH_INCLUDES} $<
.F.o:
     ${FC} -c ${FFLAGS} ${IMESH_INCLUDES} $<
```

ListSetsNTags Example

- Read in a mesh
- Get all sets
- For each set:
  - Get tags on the set and names of those tags
  - If tag is integer or double type, also get value
  - Print tag names & values for each set

- Various uses for sets & tags, most interesting ones involve both together
  - Geometric topology
  - Boundary conditions
  - Processor decomposition

ListSetsNTags Example (SIDL/C++)

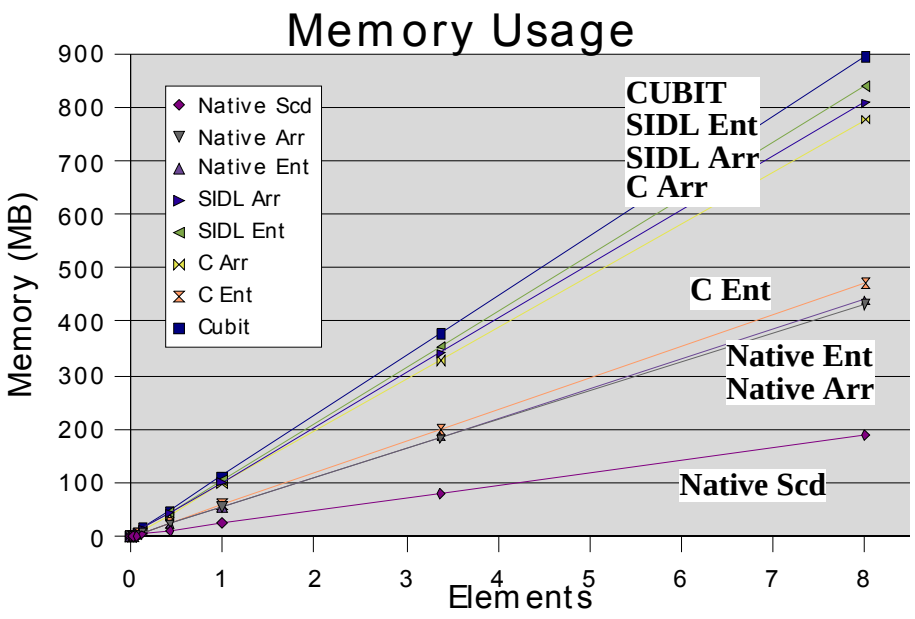
```
#include "iBase.hh"
#include "iMesh_SIDL.hh"

typedef void* iBase_EntityHandle;
typedef void* iBase_EntitySetHandle;
typedef void* iBase_TagHandle;

int main( int argc, char *argv[] )
{
  // Check command line arg
  std::string filename = argv[1];

    // create the Mesh instance
  iMesh::Mesh mesh;
  iMesh_SIDL::MeshSidl::newMesh("", mesh);

    // load the mesh
  string options;
  mesh.load(0, filename, options);

    // get all sets; use EntSet interface
  sidl::array<iBase_EntitySetHandle> sets;
  int sets_size;
  iBase::EntSet mesh_eset = mesh;
  mesh_eset.getEntSets(0, 1,
                       sets, sets_size);

    // iterate through them, checking whether
   they have tags
  iBase::SetTag mesh_stag = mesh;
  for (int i = 0; i < sets_size; i++) {
      // get connectivity
    sidl::array<iBase_TagHandle> tags;
    int tags_size;
    mesh_stag.getAllEntSetTags(sets[i],
                               tags, tags_size);

    if (0 != tags_size) {
      cout << "Set " << sets[i] << ": Tags: ";
        // list tag names on this set
      for (int j = 0; j < tags_size; j++) {
        string tname;
        int int_val;
        double dbl_val;
        mesh_stag.getTagName(tags[j], tname);
        cout << tname;
1       iBase::TagValueType tag_type;
        mesh_stag.getTagType(tags[j], tag_type);
2       if (iBase::TagValueType_INTEGER ==
            tag_type) {
          mesh_stag.getEntSetIntData(sets[i],
                                     tags[j], int_val);
          cout << " (val = " << int_val << "); ";
        }
        else if (iBase::TagValueType_DOUBLE ==
                 tag_type) {
          mesh_stag.getEntSetDblData(sets[i],
                 tags[j], dbl_val);
          cout << " (val = " << dbl_val << "); ";
        }
        else cout << "; ";
      }
    }
    cout << endl;
  }

  return true;
}
```

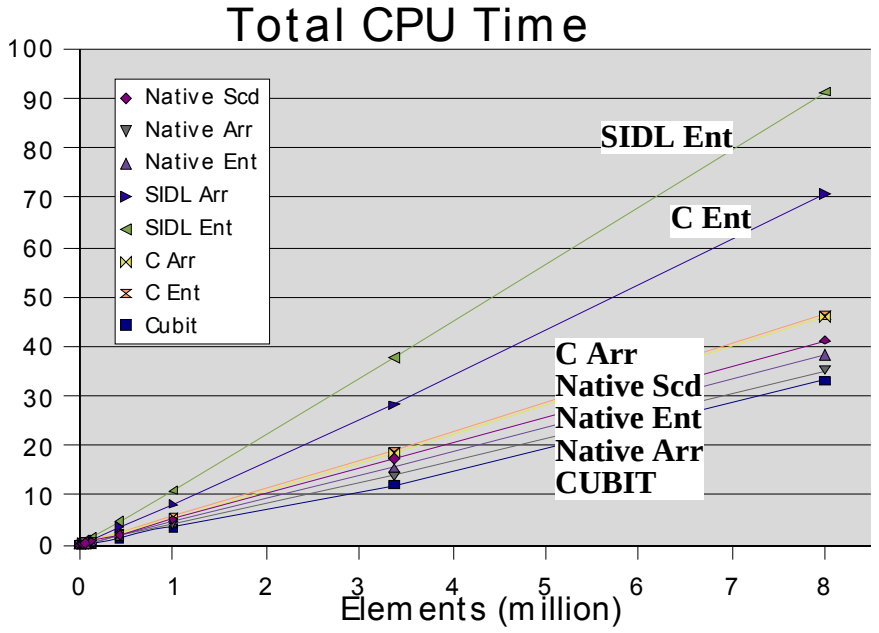
ListSetsNTags Example Notes

- Enumerated variables declared in SIDL-based code as Iface::enumNAME, e.g. iBase::EntityType or iBase::TagType
- Enumerated variable values appear as Iface::enumNAME_enumVALUE, e.g. iMesh::EntityTopology_TETRAHEDRON or iBase::TagType_INTEGER

Performance

- Large applications balance memory and cpu time performance
- Implementations of iMesh vary on speed vs. memory performance
  - Create, v-E, E-v query, square all-hex mesh
  - Entity- vs. Array-based access
- Compare iMesh (C, SIDL), Native (MOAB), Native Scd (MOAB), CUBIT
  - Ent-, Arr-based access
  - All-hexahedral square structured mesh



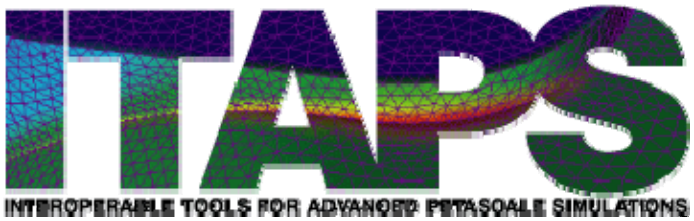


iMesh Review

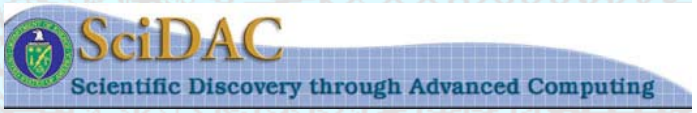
- Data model consists of 4 basic types: Interface, Entity, Entity Set, Tag
- Applications reference instances of these using opaque handles
- ITAPS interfaces use C-based APIs, for efficiency and interoperability
  - SIDL-based implementation also available, which work through C-based API
- Not covered here:
  - Iterators (intermediate-level, “chunked” access to mesh)
  - Modify (relatively coarse-grained, basically create and delete whole entities)
  - Set parent-child links

ITAPS Interfaces Best Practices

- Use C-based interface where possible, for efficiency
- Pre-allocate memory in application or re-use memory allocated by implementation
  - E.g. getting vertices adjacent to element – can use static array, or application-native storage
- Take advantage of implementation-provided capabilities to avoid re-inventing
  - Partitioning, IO, parallel communication, (parallel) file readers
- If one implementation of iMesh doesn't work for you, try another
  - Interoperability is an explicit goal of ITAPS
- Implement iMesh on top of your data structure
  - Take advantage of tools which work on iMesh API
- Let us help you
  - Not all the tricks can be easily described and may not be self-evident



Parallel Mesh Interfaces




Parallel Mesh Interface: iMeshP

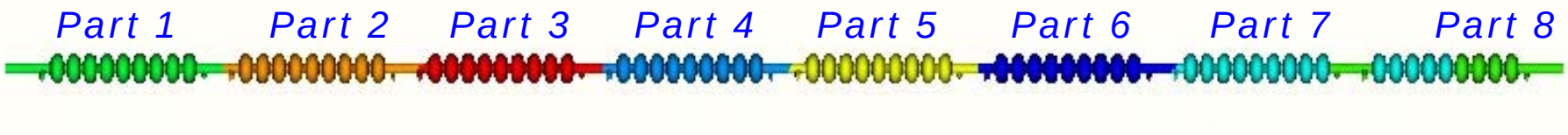
- Primarily support distributed memory.
  - For example: use MPI communicators from application.
  - But allow use of global address space and shared memory paradigms.

- Maintain backward compatibility of serial iMesh.
  - Serial iMesh works as expected within a process.
  - Serial iMesh works as expected for global address space and shared memory programs.

iMeshP Partition Model

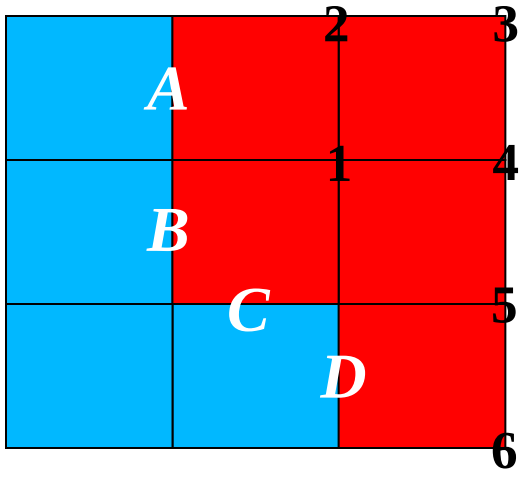
- *Process*: a program executing; MPI process.
  - Number of processes == MPI_Comm_size
  - Process number == MPI_Comm_rank
- *iMesh instance*: mesh database provided by an implementation.
  - Each process has one or more mesh instances.
- *Partition*: describes a parallel mesh.
  - Maps entities to subsets called *parts*.
  - Maps parts to processes.
  - Has a communicator associated with it.

iMesh instance: SLAC linear accelerator
Partition: 8 parts on 8 processes
Communicator = MPI_COMM_WORLD

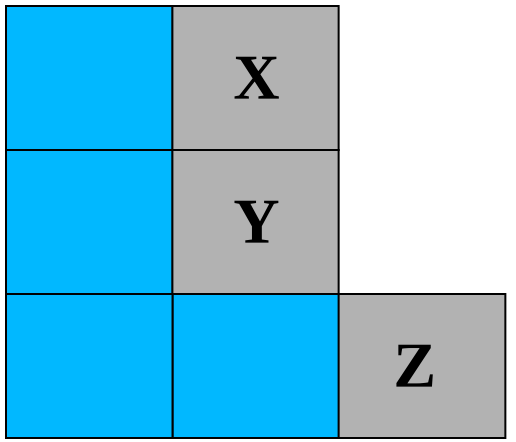


iMeshP Partition Model

- *Ownership*: having the right to modify.
- *Internal entity*: Owned entity not on an interpart boundary.
  - E.g., Vertices 1-6 are internal to the red part.
- *Part-Boundary entity*: Entity on an interpart boundary.
  - E.g., Edges A, B, C & D are part-boundary edges.
  - Typically *shared* between parts (one part is owner; other parts have copies).



- *Ghost entity*: Non-owned, non-part-boundary entity in a part.
  - E.g., Regions X, Y, and Z are ghost regions for the blue part.
- *Copies*: ghost entities + non-owned part-boundary entities.



Partition Characteristics

- Maps entities to parts.
  - Part assignments computed with respect to a set of entities.
  - Computed assignments induces part assignments for adjacent entities.
- Maps parts to processes.
  - Each process may have one or more parts.
  - Each part is wholly contained within a process.
- Has a communicator associated with it.
  - “Global” operations performed with respect to data in all parts in a partition’s communicator.
  - “Local” operations performed with respect to either a part’s or process’ data.

iMeshP Partition API Functions

- Create partition.
  - Accepts pointer to MPI Communicator or NULL.
- Destroy partition.
- Sync partition.
  - After parts are added/updated, compute and store global information about the partition.
- Return number of parts in partition.
- Return global number of entities of

Part characteristics

- Think in terms of parts, not processes.
  - Number of parts may be less than, equal to, or greater than number of processes.
- Part contains entities it owns + copies of entities needed for computation within the part.
- Wholly stored in a single process.
- Accessed and identified via part IDs.
  - Unique global identifiers for parts.
- Local (on-process) parts also can be

iMeshP Part API Functions

- Create part and add to partition.
- Remove part and destroy it.
- Iterate over part-boundary entities.
- Return part neighbors.
  - Parts A and B are neighbors if Part A has copies of entities owned by Part B or vice versa.
- Return entity data within a part.
  - Get number of local entities in a part.
  - Get entities in a part.
- Add/remove entity to/from a local part.

Entity Characteristics

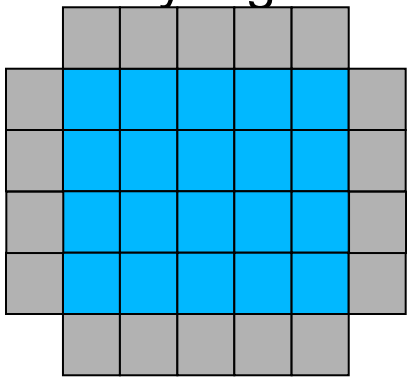
- Each entity is owned by only one part per partition.
  - Ownership grants right to modify.
- Entities may be copied on other parts.
- Duplicated information for copies:
  - Shared boundary entities, ghost entities.
  - Owner of an entity knows remote part and remote entity handle of all its copies.
  - All copies of entity know the entity owner's part and the entity handle on the owner.
  - All boundary entities know all remote parts and remote entity handles of all copies.
- Remote parts and entities are computed by a collective function called after mesh modification.
  - Queries for remote data do not require communication.

iMeshP Entity API Functions

- Determine ownership of entity.
- Determine whether entity is internal, boundary, ghost.
- Return remote part and remote entity handles for copies of entity.
- Return owner part and owner entity handle for an entity.

Ghost Entities

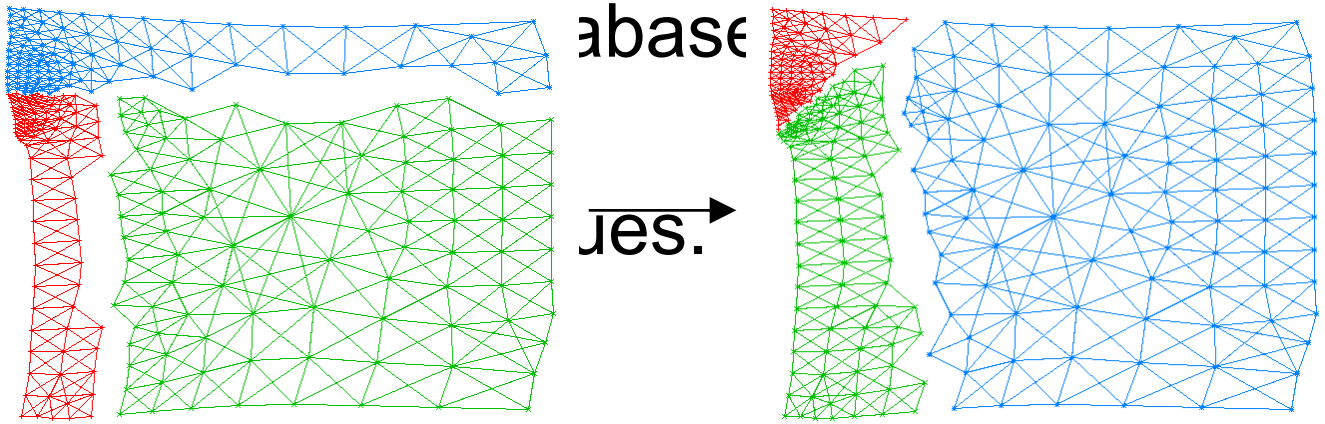
- iMeshP_createGhostEnts
  - Create ghost entities for each part.
  - Specify ghost-entity dimension, bridge dimension, and number of layers.
    - E.g., one layer of ghost regions for all boundary regions sharing faces:
      - Ghost-entity dimension = 2
      - Bridge-entity dimension = 1
      - Number of layers = 1



  - Cumulative over multiple calls.
- iMeshP_deleteGhostEnts
  - Delete all ghost entities.

Inter-part Mesh Operations

- iMeshP provides functions for inter-part operations on mesh entities.
  - Migrate large numbers of entities for, say, load balancing.
  - Migrate small numbers of entities for, say, mesh modification.
  - abase
  - ues.



Inter-part Mesh Operation Requests

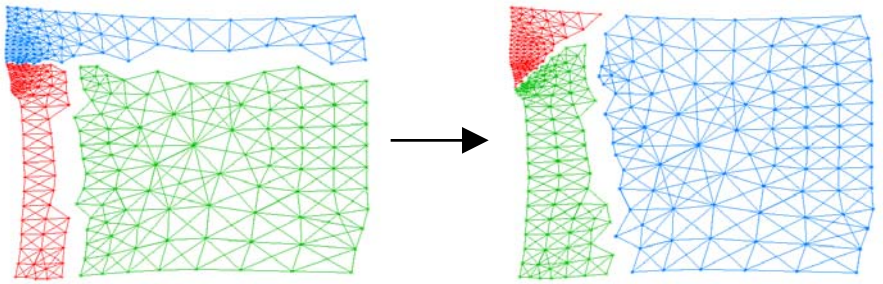
- Inter-part mesh operations are coordinated via iMeshP_Requests.
  - More than an MPI_Request!
  - Indicate status of a given iMeshP mesh operation.
    - Migrate entity.
    - Update vertex coordinates.
    - Update part-boundary entities.
    - Exchange tag data.
  - iMeshP encodes type of request and operations to be performed in

Inter-part Mesh Operations can be blocking or non-blocking.

- Blocking operations do not return from iMeshP until request is complete.
- Non-blocking operations return from iMeshP after request is made. Application later waits until request is fulfilled.
  - iMeshP API contains functions to …
    - Wait for request completion,
    - Test for request completion, and
    - Poll for and carry out requests received.
  - Allows overlapping communication/computation.
  - Allows asynchronous communication.

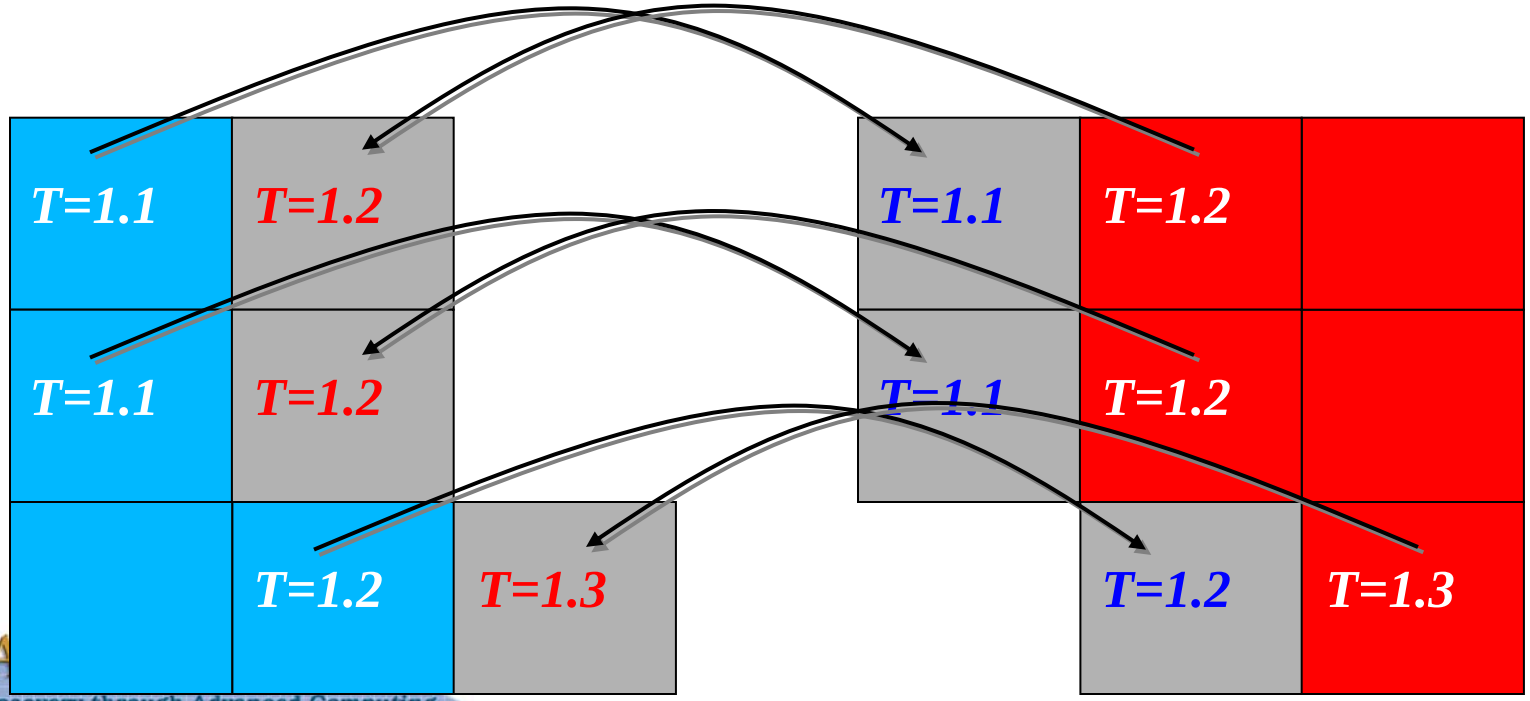
Large-Scale Migration

- In application, each part calls iMeshP to migrate (push) array of entities to new parts. (iMeshP_sendEntArr)
  - iMeshP computes and posts appropriate receives.
  - iMeshP sends entities to new parts.
  - iMeshP deletes entities from old parts.
  - iMeshP returns an iMeshP_Request.
- Application does something else for awhile.
- In application, each part calls iMeshP_Wait function with the iMeshP_Request returned by send.
  - iMeshP waits to receive messa
  - iMeshP adds entities to new parts and updates mesh.



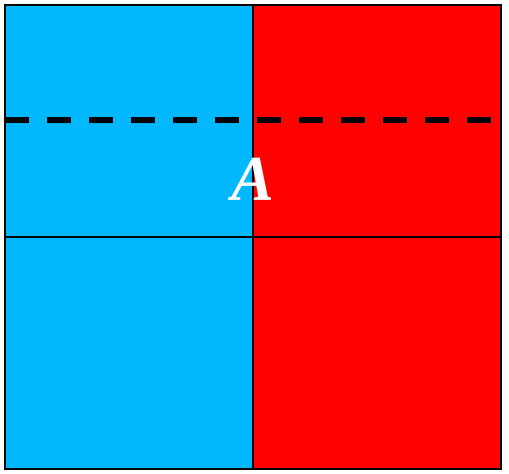
Exchange Entity Tag Data

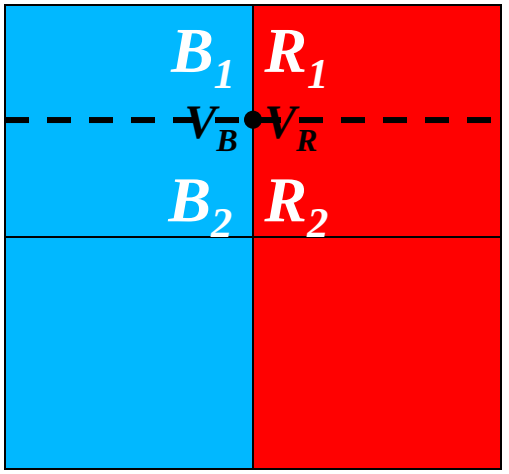
- Entity owners send tag data to copies.
- iMeshP API provides both blocking and non-blocking versions of tag-data exchange.
  - iMeshP_exchTagData and iMeshP_IexchTagData



Edge Splitting with Non-Blocking Update

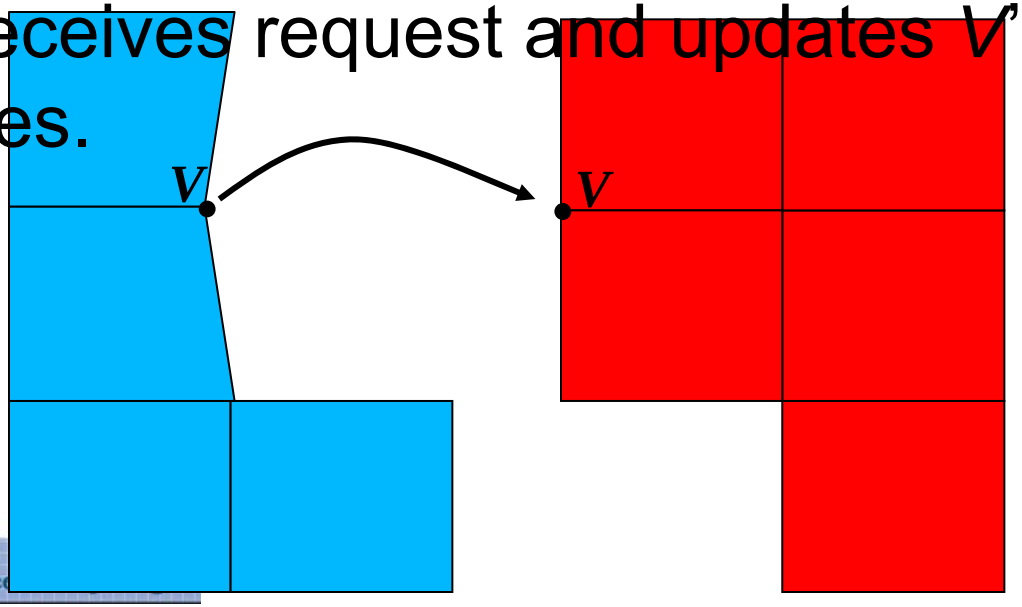
- Blue and red parts decide to split edge *A*.
- Red part creates edges R_1, R_2 and vertex V_R.
- Blue part creates edges B_1, B_2 and vertex V_B.
- Blue and red parts call iMeshP to request replacement of A with new edges and vertices on opposite part.
  - iMeshP_replaceOnPartBdry
- Blue and red parts call iMeshP to poll for requests; iMeshP receives updates and matches up $B_1 \Leftrightarrow R_1$, $B_2 \Leftrightarrow R_2$, and $V_B \Leftrightarrow V_R$.





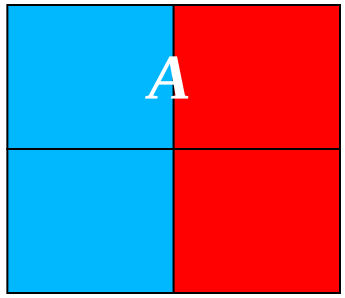
Mesh Smoothing with Non-Blocking Update

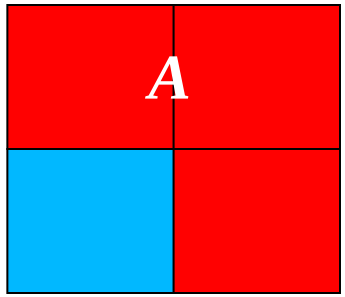
- Blue part decides to move vertex *V*.
- Blue part calls iMeshP to request update of *V*'s vertex coordinates on red part.
  - iMeshP_updateVtxCoords
- Red part calls iMeshP to poll for requests; iMeshP receives request and updates *V*'s coordinates.

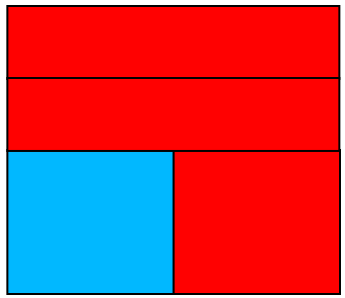


Micro-migration for Mesh Modification

- Blue part owns edge *A*.
- Red part needs edge *A* to do edge swapping.
- Red part calls iMeshP to request edge *A* from Blue part.
  - iMeshP_migrateEntity
- Blue part calls iMeshP to poll for requests; iMeshP receives request and sends *A* and its higher-order adjacencies to Red part.
- Red part calls iMeshP to wait for





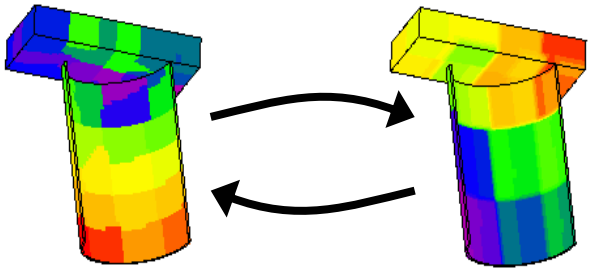


Updating Mesh Consistency

- After all mesh modification is done, application calls iMeshP_syncMeshAll.
  - A collective, blocking call that signals mesh modification operations are completed.
  - Polls for and processes outstanding requests.
  - Updates ghost entities for modified mesh.
  - Performs operations needed for parallel mesh consistency.

Future Work

- Add support for multiple partitions of a mesh.
  - Specify “primary” and “secondary” partitions.
  - Map mesh data between multiple partitions.



Graph-based region partition ***Geometric face partition***

- Parallel File I/O
  - Read/write 1, N<<P, or P files.
  - Provide initial partition of data.

ITAPS Geometry Interface: iGeom

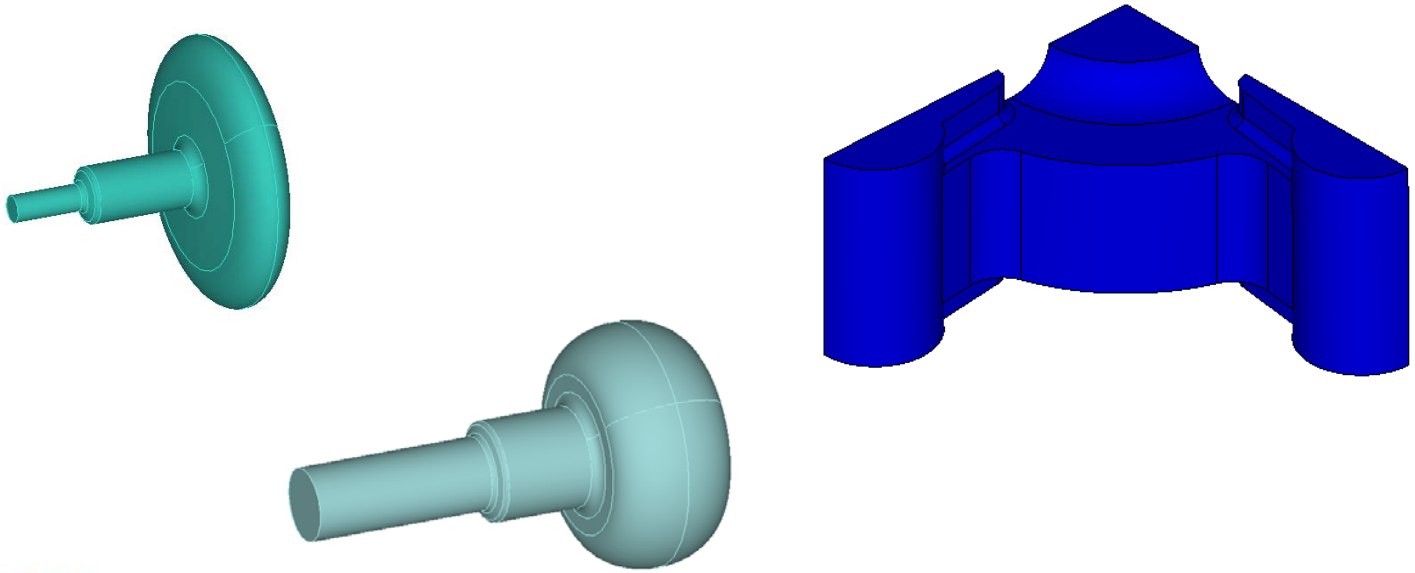
- Similar to mesh interface, but for geometric models
  - CAD models first, but really more than that (depending on implementation)
- Interface for accessing BREP-type geometric models
  - Uses iBase::EntityType's VERTEX, EDGE, FACE, REGION
- Provides same 4-typed data model used by iMesh
  - Entity, Entity Set, Interface, Tag
- Query methods for topology, geometry; modify interface also
- Somewhat less mature than iMesh
  - Implementation based on CGM (RPI implementing one too)
  - Interface not finalized by ITAPS group, but close
- CGM-based implementation:
  - Support shape optimization, parallel load
  - Initializes “complete” geometric models from CUBIT meshing toolkit
    - Tags, sets used to represent geometric groups, ids, boundary condition groupings, etc.

iGeom API Summary

- Logically arranged into interfaces, but not explicitly arranged as such in C
  - See iGeom.h or iGeom.sidl
- Basic (CoreQuery): load, save
- Topology: getEntities, getNumOfType, getEntType, getEntAdj, get2ndAdj, isEntAdj (also for Arr's)
- Shape: getEntClosestPt, getEntNrmlXYZ, getEntTgntXYZ, getFc/EgCvtrXYZ, etEgFcEvalXYZ, getEntBoundBox, getVtxCoord, getPntRayIntsct, getEntNrmlSense, getEgFcSense, getEgVtxSense, Measure
- Parametric: isParametric, forward/reverse parametric evaluation (getEntUVtoXYZ, getEntXYZtoUV, etc.), parameter-based normal/tangent, isEntPeriodic, isFc/EgDegenerate, etc.
- Tolerance: getTolerance, getEntTolerrance

iGeom API Summary (cont.)

- Iterator: initEntIter, etc.
- Modify:
  - Construct: Copy, SweepAboutAxis, Delete
  - Primitives: Brick, Cylinder, Torus
  - Transforms: Move, Rotate, Reflect, Scale
  - Booleans: Unite, Subtract, Section, Imprint, Merge



ITAPS Relations Interface: iRel

- An important part of the ITAPS interfaces philosophy is to have multiple interfaces
  - Makes it easier to mix & match components
  - Makes each component easier to understand
- Sometimes data in different components need to be associated together
  - Mesh-geometry
  - Fields-mesh
- ITAPS is developing “Relations” interface to do this
  - Currently, most of TSTTR is targeted to mesh-geometry classification
- 2 parts of relations process: building/restoring relations, querying them
  - Restoring relations done by matching entities, sets, tag(s)/value(s)
  - Can relate entity to entity, set to entity, set to set
  - For mesh-geometry, relate set-entity
- Implementation depends heavily on restoring data
  - For LASSO, done by reading CUBIT .cub save/restore files

iRel Interface Summary

- createAssocation, destroyAssociation
- getAssociatedInterfaces
- set/get EntEnt/EntArr/Arr Association
- create Vtx/Ent/VtxArr/EntArr AndAssociate, moveTo
- infer Ent/Arr/All Associations

iRel Simple Example: RelateEm

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

const int NOT_SET = 0, IS_SET = 1;

int main( int argc, char *argv[] )
{
  // Check command line arg
  if (argc < 2) {
    std::cerr << "Usage: " << argv[0] << " <geom_filename>" <<
        " <mesh_filename>" << std::endl;
    return 1;
  }

  // initialize the Geometry, Mesh, and Relate instances
  int err;
  iGeom_Instance geom;
  iGeom_newGeom( 0, &geom, &err, 0 );
  iMesh_Instance mesh;
  iMesh_newMesh( 0, &mesh, &err, 0 );
  iRel_Instance relate;
  iRel_newAssoc( 0, &relate, &err, 0 );

    // load geometry and mesh
  iBase_EntitySetHandle mesh_root_set, geom_root_set;
  iMesh_getRootSet( mesh, &mesh_root_set, &err ); ;
  iGeom_getRootSet( geom, &geom_root_set, &err ); ;
  iGeom_load( geom, argv[1], 0, &err, strlen(argv[1]), 0 );
  iMesh_load( mesh, mesh_root_set, argv[2], 0, &err,
                  strlen(argv[2]), 0 );

// create a relation
  iRel_RelationHandle rel;
  iRel_createAssociation( relate, geom, NOT_SET,
                          iRel_IGEOM_IFACE,
                          mesh, IS_SET, iRel_IMESH_IFACE,
                          &rel, &err ); ;

    // relate all entities
  iRel_inferAllAssociations(relate, rel, &err);

     // get all geom regions
  iBase_EntityHandle* gents = 0;
  int gents_alloc = 0, gents_size = 0;
  iGeom_getEntities( geom, geom_root_set, iBase_REGION,
                     &gents, &gents_alloc, &gents_size,
                     &err );

    // check for related sets
  iBase_EntityHandle* ments = 0;
  int ments_alloc = 0, ments_size = 0;
  int *offset = 0, offset_alloc = 0, offset_size = 0;
  iRel_getArrAssociation( relate, rel,
                          gents, gents_size,
                          0, 1, 0,
                          &ments, &ments_alloc, &ments_size,
                          &offset, &offset_alloc, &offset_size,
                          &err );

  int num_related = 0;
  for (int i = 0; i < ments_size; i++)
    if (NULL != ments[i]) num_related++;


  std::cout << "Number of geom regions = " << gents_size
            << ", related mesh sets = " << ments_size << std::endl;

    // clean up
  free( gents );
  free( ments );
  free( offset );
  iRel_destroyAssociation( relate, rel, &err );
  iRel_dtor( relate, &err );
  iMesh_dtor( mesh, &err );
  iGeom_dtor( geom, &err );

  return 0;
}
```

Advanced: Writing Your Own iMesh Implementation

- You can use iMesh-based tools like Mesquite, Zoltan by implementing iMesh on top of your mesh representation
- Steps (C):
  - Implement functions in iMesh.h based on your data structure
- Steps (SIDL):
  - Decide class name for your implementation, provide iMesh-<class>.sidl (see iMesh.sidl, iMesh_SIDL.sidl)
  1. Run Babel tool on iMesh.sidl and iMesh_<class>.sidl
  2. Produces function declarations and stub definitions in server/iMesh_<class-uppercase>_<class>_Impl.[hh,cc]
  3. Fill in your implementation
     - Lines with “DO-NOT-DELETE” delimit places you can fill in includes, implementation, etc.
  4. Link into your own iMesh server library
  5. Subsequent runs of Babel won’t overwrite blocks of code between those delimiters
- Link iMesh-enabled tool to your implementation

ITAPS

INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

MORNING SESSION HANDS ON

SciDAC

Scientific Discovery through Advanced Computing

Sandia National Laboratories

Rensselaer

STONY BROOK

THE UNIVERSITY OF BRITISH COLUMBIA

BROOKHAVEN NATIONAL LABORATORY

OAK RIDGE NATIONAL LABORATORY

Pacific Northwest National Laboratory

ITAPS

INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

PART 4: ITAPS Services and Tools

SciDAC

Scientific Discovery through Advanced Computing

Sandia National Laboratories

Rensselaer

STONY BROOK

THE UNIVERSITY OF BRITISH COLUMBIA

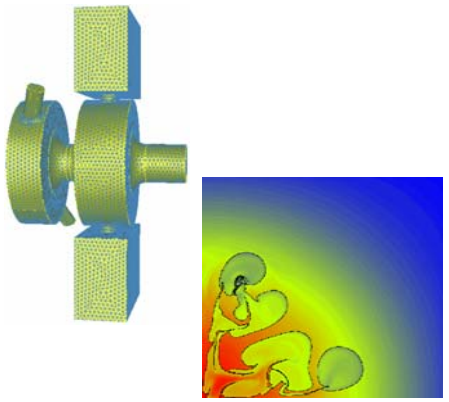
BROOKHAVEN NATIONAL LABORATORY

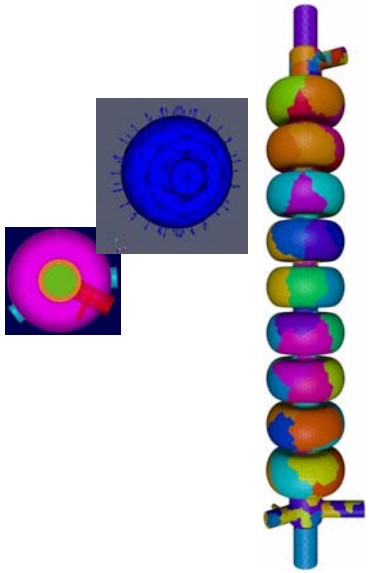
OAK RIDGE NATIONAL LABORATORY

Pacific Northwest National Laboratory

ITAPS impact SciDAC applications in three ways

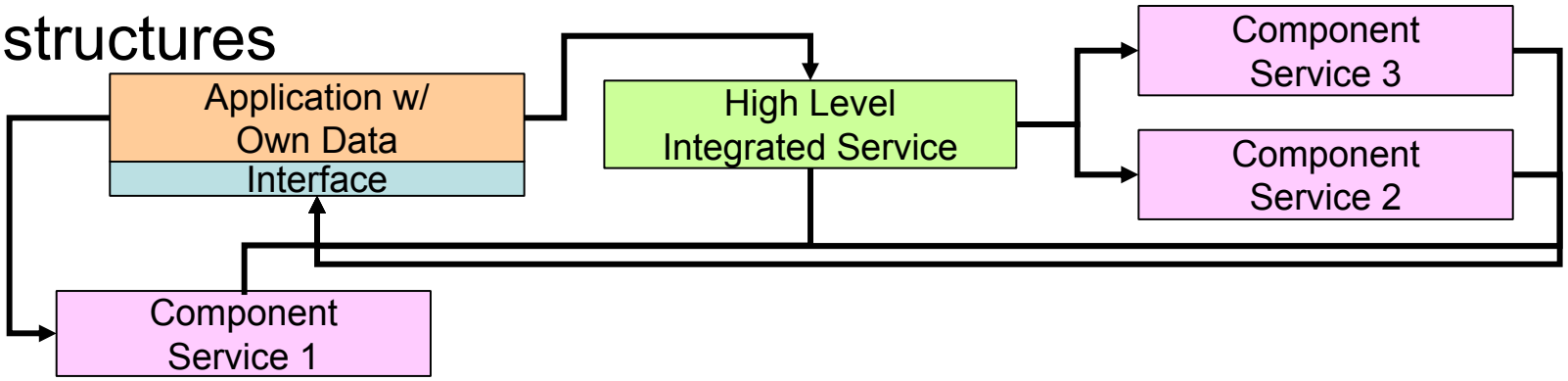
- Direct use of ITAPS technology in applications
  - Geometry tools, mesh generation and optimization for accelerators and fusion
  - Mesh adaptivity for accelerators and fusion
  - Front tracking for astrophysics and groundwater
  - Partitioning techniques for accelerators and fusion
- Technology advancement through demonstration and insertion of key new technology areas
  - Design optimization loop for accelerators (w/ TOPS)
  - Petascale mesh generation for accelerators
- Enabling future applications with ITAPS services and interfaces
  - Parallel mesh-to-mesh transfer for multi-scale, multi-physics applications
  - Dynamic mesh services for adaptive computations



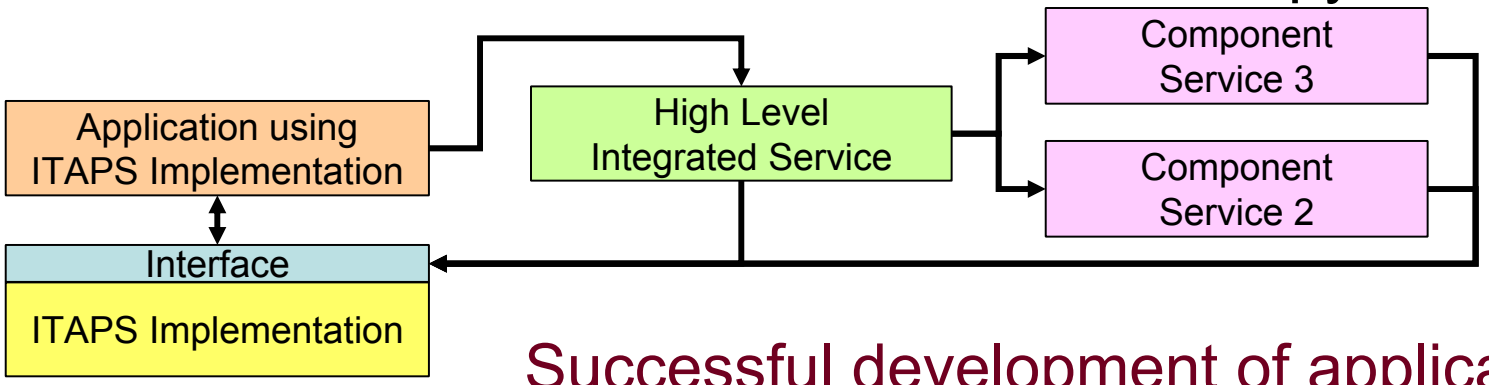


Applications can access ITAPS services in two ways

1. Implement ITAPS interfaces on top of application data structures



2. Use a reference implementation of the interfaces to provide access to ITAPS services at the cost of a data copy

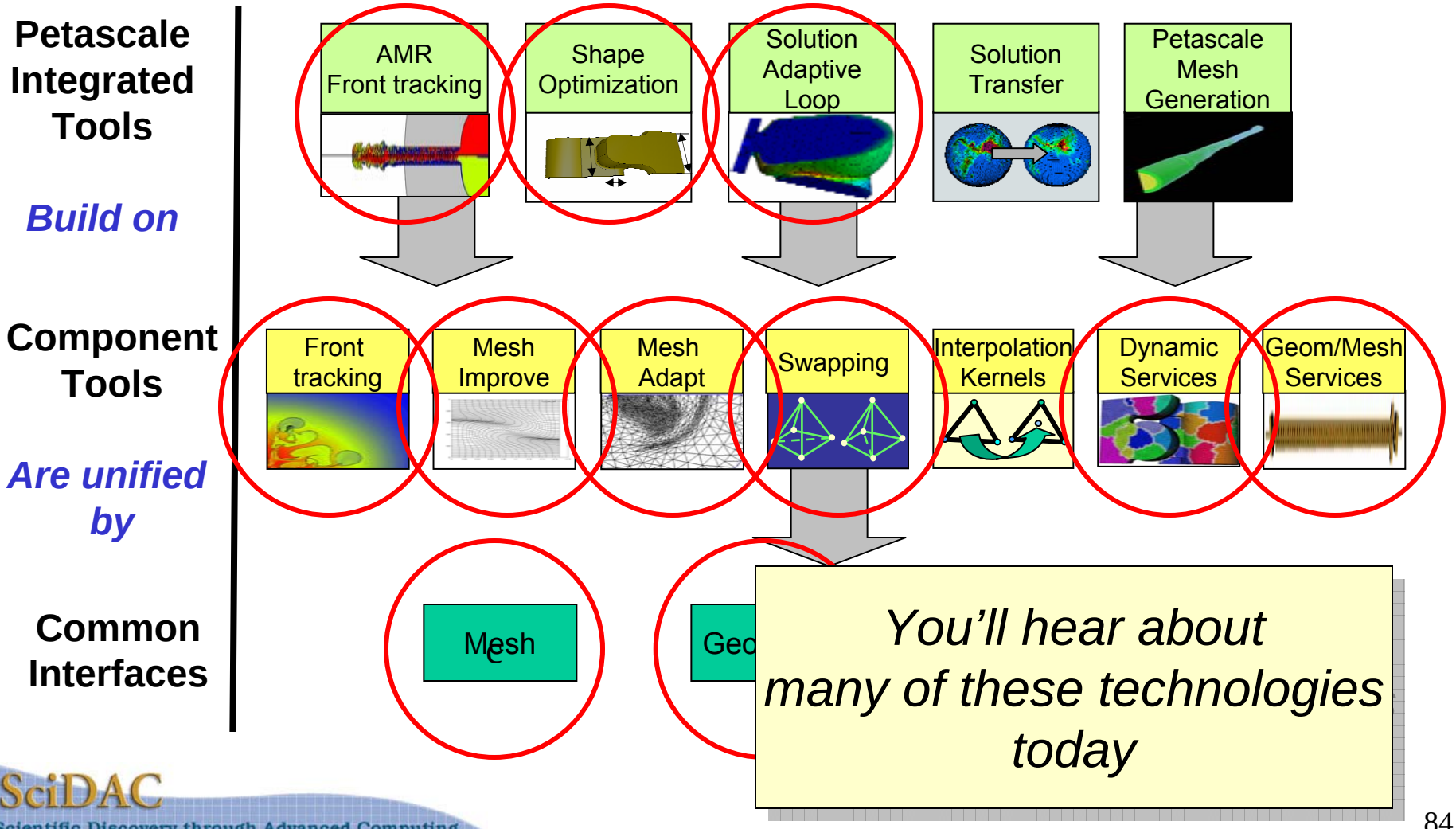


Successful development of applications requires close collaborations

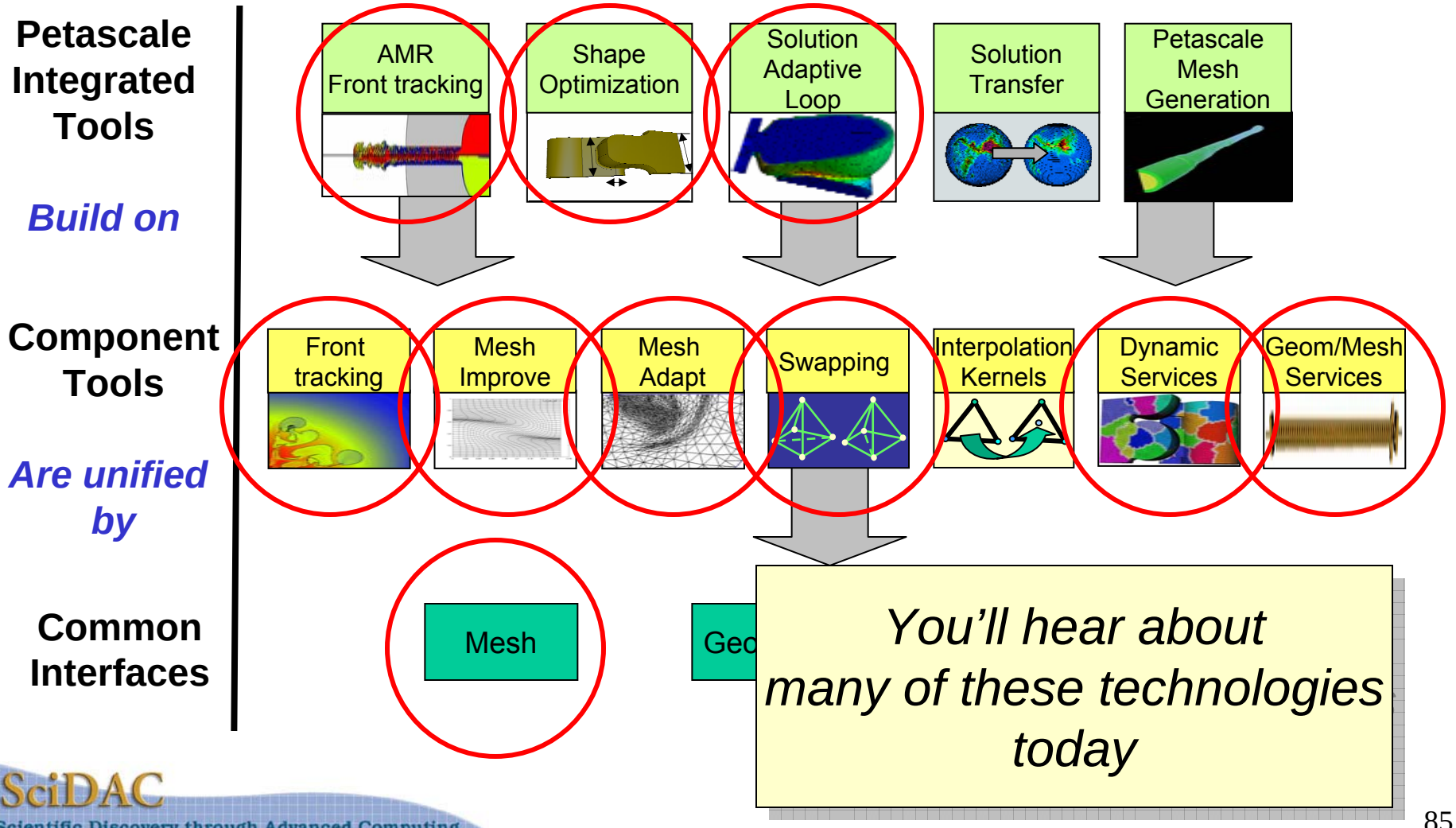
Apply ITAPS developed technologies to advance SciDAC applications

- Applications discussed today:
  - Front tracking for Pellet Ablation in Tokamak Fuelling
  - Parallel Adaptive Loop for Accelerator Design
  - Adaptive mesh control for Extended MHD Simulations in PPPL's M3D-C^1 Code
  - Element Curving Tool for Higher Order Finite Elements
  - Mesh Generation for Nuclear Reactor Simulation
  - Shape optimization

ITAPS Technologies

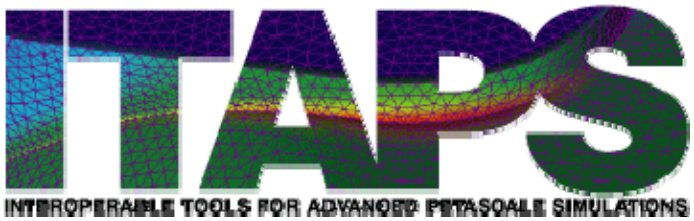


ITAPS Technologies



Provide interoperable services to speed the development of simulation technologies

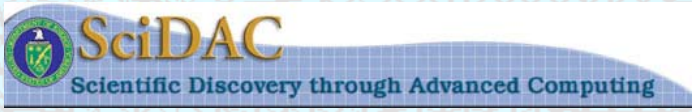
- **Need for Services:**
  - Provide SciDAC applications ability to take advantage of advanced tools for generation and control of meshes as part of their simulations
- **Focus:**
  - Initially provide prototype implementations for specific SciDAC needs
    - Mesh quality improvement
    - Shape optimization
    - Front tracking
    - Mesh adaptation loops
  - Current focus is on formalizing services to use ITAPS interoperable interfaces



The Mesquite Mesh Quality Improvement Toolkit

Patrick Knupp (SNL)

Lori Diachin (LLNL)

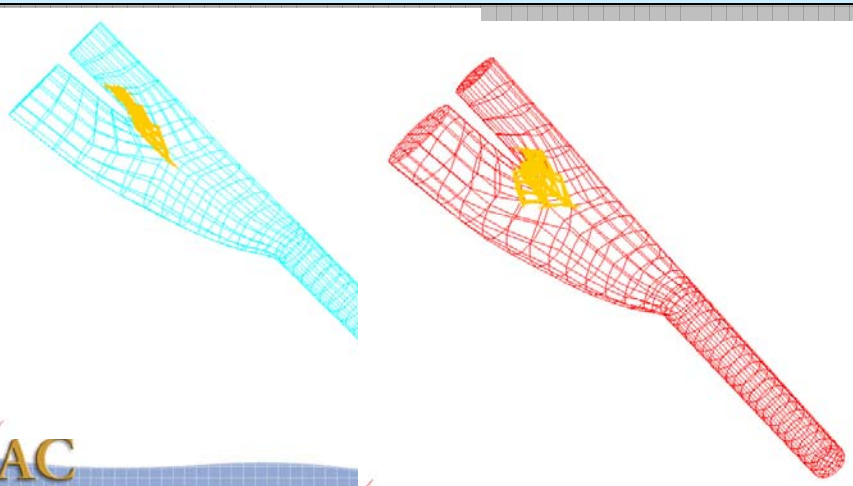



Improving mesh quality can dramatically affect time to solution

- Mesh quality is the geometric properties of a mesh such as shape, size, and orientation of elements
- Mesh quality affects solution accuracy, efficiency, stability

Arteriovenous Graft Turbulent Flow Simulation

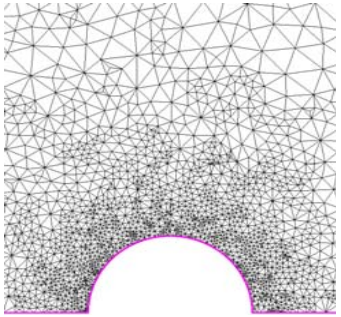
Optimized mesh traded four hours of applications solver time 19 minutes of mesh smoothing time.

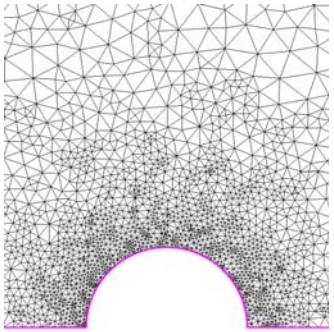


Compressible Flow

- Mesh Optimized w/ Active set solver
  - Improved the convergence rate by 25%

Mesh improvement cost less than one multigrid iteration





Freitag and Ollivier-Gooch, 1998

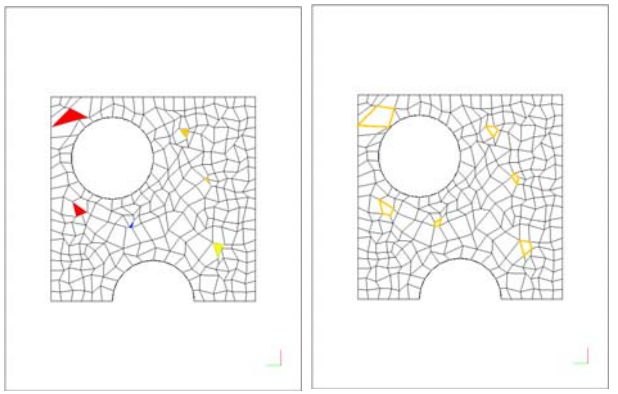
SciDAC Scientific Discovery through Advanced Computing

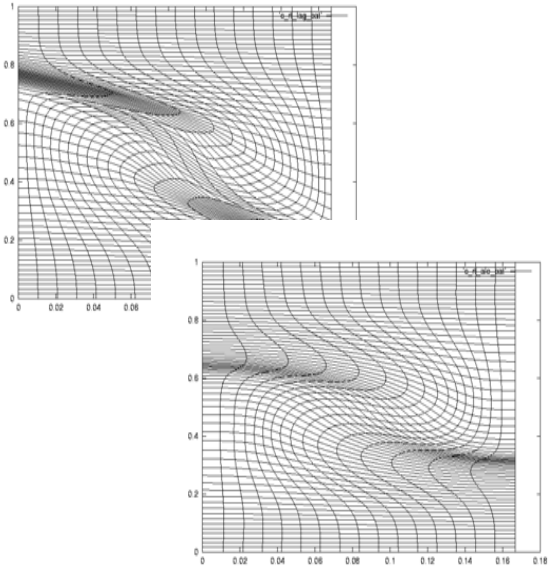
Mesquite provides advanced mesh smoothing capabilities

- Mesquite is a comprehensive, stand-alone library for mesh quality improvement with the following capabilities
  - Shape Quality Improvement
  - Mesh Untangling
  - Alignment with Scalar or Vector Fields
  - R-type adaptivity to solution features or error estimates
    - Maintain quality of deforming meshes
    - Anisotropic smoothing
    - Control skew on mesh boundaries
- Uses node point repositioning schemes

Mesquite is versatile and comprehensive

- 2/3D Structured, Unstructured, Hybrid Meshes
- Element Types
  - Triangular, Tetrahedral, Quadrilateral, Hexahedral
  - Prismatic, Pyramidal planned, Polyhedral easily added
  - Linear currently, higher-order planned
- Customizable
  - User-defined metrics, objective functions, and algorithms
- Callable as a library
  - Mesh and geometry information obtained through ITAPS accessor functions



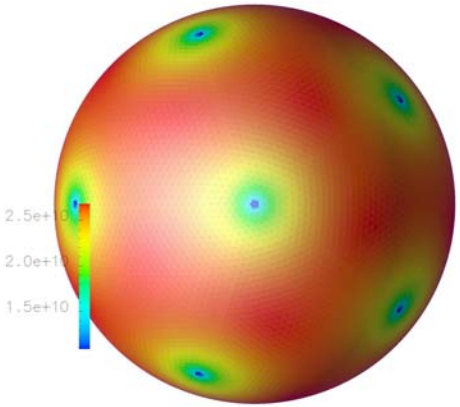


The Mesquite infrastructure provides advanced solution techniques

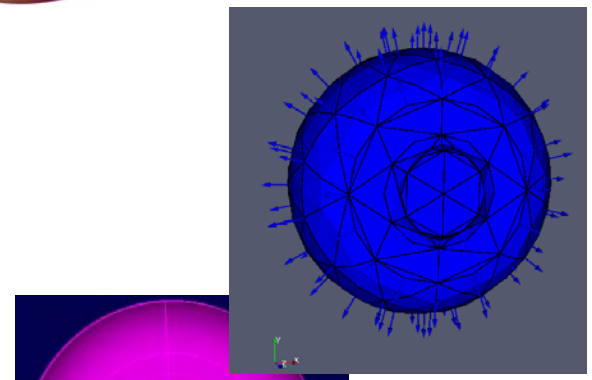
- State-of-the-art algorithms and metrics
  - Simple (Laplace) to complex (optimization) smoothers
  - Untangle, smooth, size, shape metrics
    - Feasible Newton techniques
    - Active set solvers
  - Single-vertex and all-vertex solvers
  - Mesh culling to eliminate un-needed operations
  - Some metrics permit anisotropic smoothing
- Combined approaches
  - Increase effectiveness and efficiency
- Efficient to run
  - Kernels written with inline functions and array-based access
  - Light-weight mesh data structure

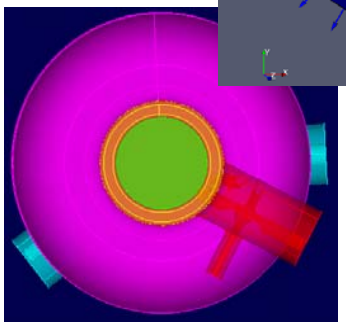
Mesquite has been used extensively in SciDAC settings

- SciDAC meshing groups
  - MOAB (SNL)
  - Cubit (SNL)
  - Overture (LLNL)
  - FMDB (RPI)
  - NWGrid (PNNL)
- SciDAC applications
  - SLAC (Tau3p), stability studies
  - Climate Group (CSU), geodesic meshes
  - SLAC, design optimization
- Non-SciDAC applications
  - Nek5000 (ANL), arteriovenous graft
  - ALEGRA (SNL), magneto-hydrodynamics
  - ASCI ASAP Rocket Center, rocket booster simulations (UIUC)



Mesquite used to smooth geodesic mesh for CSU SciDAC Climate Group In mesh quality vs. accuracy study





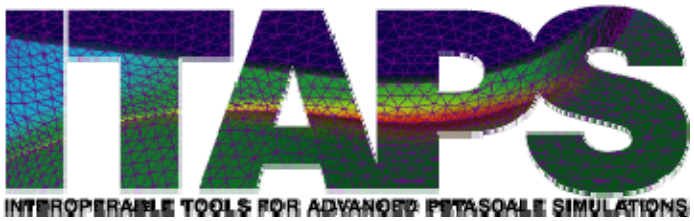
Mesquite used to adapt meshes to new geometry in SLAC design optimization

ITAPS mesh interfaces necessary for using Mesquite

- Entity sets specify which elements to improve, entity access functions, vertex coordinates, geom dim
- Adjacency information
  - Build sets of vertices from an input set of elements
  - Get elements from a vertex
  - Get element connectivity
- Iterators over entity sets to populate local mesh information in Mesquite
  - Avoids large array copies
- Tags
  - indicate which vertices are fixed
  - saving state information
  - setting reference ideal element information

ITAPS geometry interfaces needed for surface smoothing

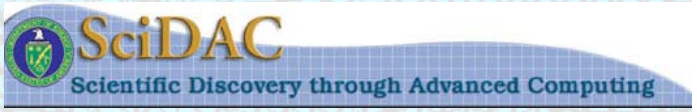
- Relax node positions to geometric domain
- Get surface normals
  - Avoids large array copies
- Get the geometric entity type to determine how many normals to cache



Triangular and Tetrahedral Mesh Swapping Using ITAPS

Carl Ollivier-Gooch

University of British Columbia





Bird's Eye View

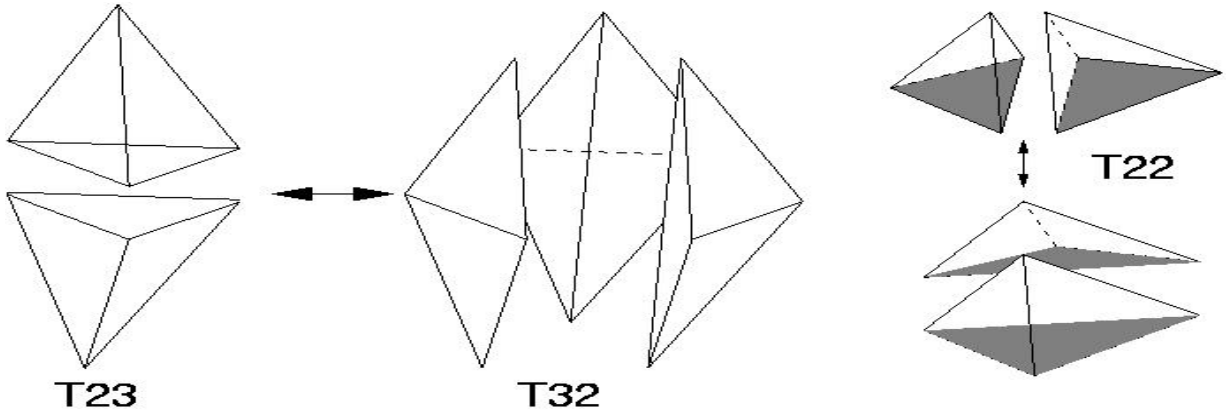
- 2D / 3D swapping with variety of criteria
- Single edge/face, mesh subset, or entire mesh
- Recursive swapping available
- User-defined swapping criteria
- Boundary modification optional in 3D
- Most typical usage will use four calls:

```
setMeshData
setAssocData
setQualMeasure
swapAll
```

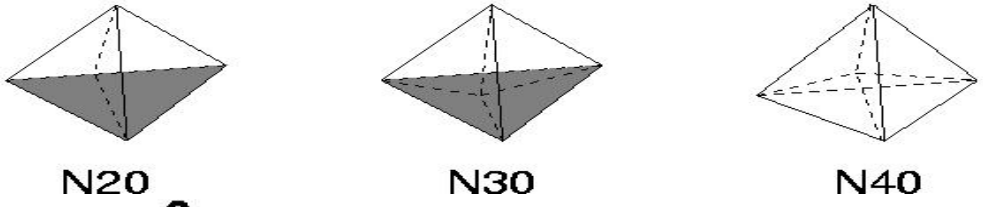
Swapping Operations

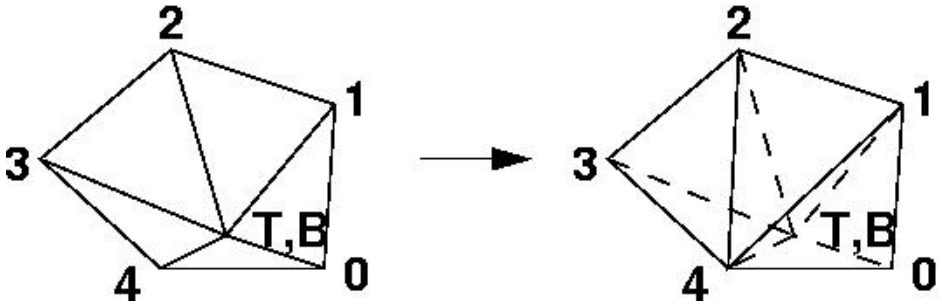
Face Swapping in 3D

Swappable



Unswappable





Tets Before	3	4	5	6	7
Tets After	2	4	6	8	10
Configs	1	2	5	14	42
Tets x Configs	2	8	30	112	420
Unique Tets	2	8	20	40	70

Swapping Criteria

- Built-in criteria
  - Delaunay, maxmin (dihedral) angle, minmax sine (dihedral) angle
- User-defined criteria
  - Any single-valued criterion depending only on vertex coordinates

```
double calcQuality(double coords[],
  int coords_size);
```

  - Can be minimized or maximized

```
bool shouldMinimize();
```

Driver for Swapping Entire Mesh

Initialize: create an iterator for all d-1–dimensional entities

Iterate: while swaps are still occuring ...

Iterate over mesh:

get next d-1–dimensional entity

swap that entity, if appropriate

recursively swap other nearby entities

Reset:

reset iterator for next pass

Finalize: delete iterator

Deciding Whether to Actually Swap

Finding Local Topology

2D: Edge → triangles → vertices

3D: Triangle → tetrahedra → vertices

Swapping is A Geometric Decision

Based on orientation and quality of entities

Getting Coordinate Data

Could fetch one vertex at a time

- Too much call overhead

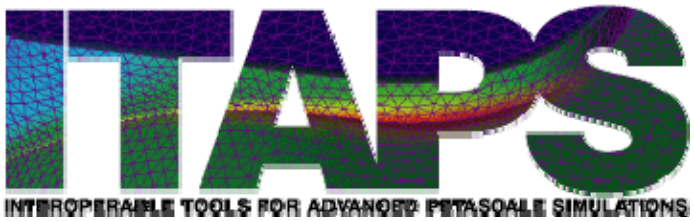
Instead, use a block call

Changing Mesh Topology

- Out with the Old
  - Remove interior of old configuration
  - Delete d-dimensional entities first
  - Then d-1-dimensional entities
- In with the New
  - Create new d-1-dimensional entities from vertices
  - Create new d-dimensional entities from d-1-dimensional entities

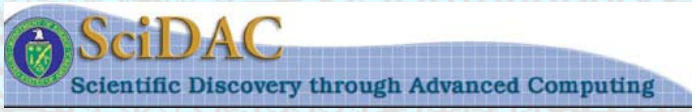
Summary

- Full swapping implementation is non-trivial
- Service using standard interface can handle all the tricky bits
- Implementation of approx ten functions can allow anyone, with any mesh data structure, to take advantage of this
- Most difficult is correct implementation of iterators
- Performance is a bit slow compared with native implementation, but not disastrously so



ITAPS Mesh Adaptation Service

M. S. Shephard and X. J. Luo
Rensselaer Polytechnic Institute

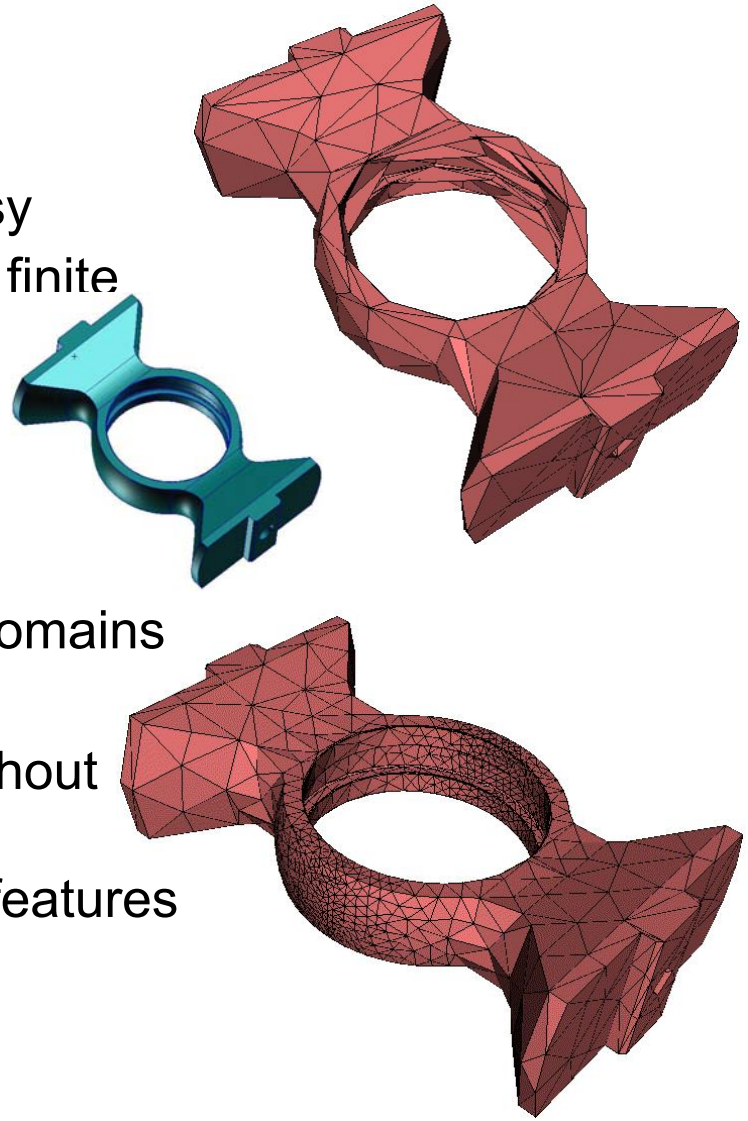



Overview of ITAPS Adaptive Loop

- Operational components for an adaptive loop
  - FEA solver
  - Error estimation
  - Mesh adaptation
  - Field solution transfer
  - Attributes mapping
- Size field definition
  - Anisotropic size field
    - Anisotropic size field tensor at each mesh vertex
  - Isotropic size field
    - Use a double value to define the edge length at each mesh vertex

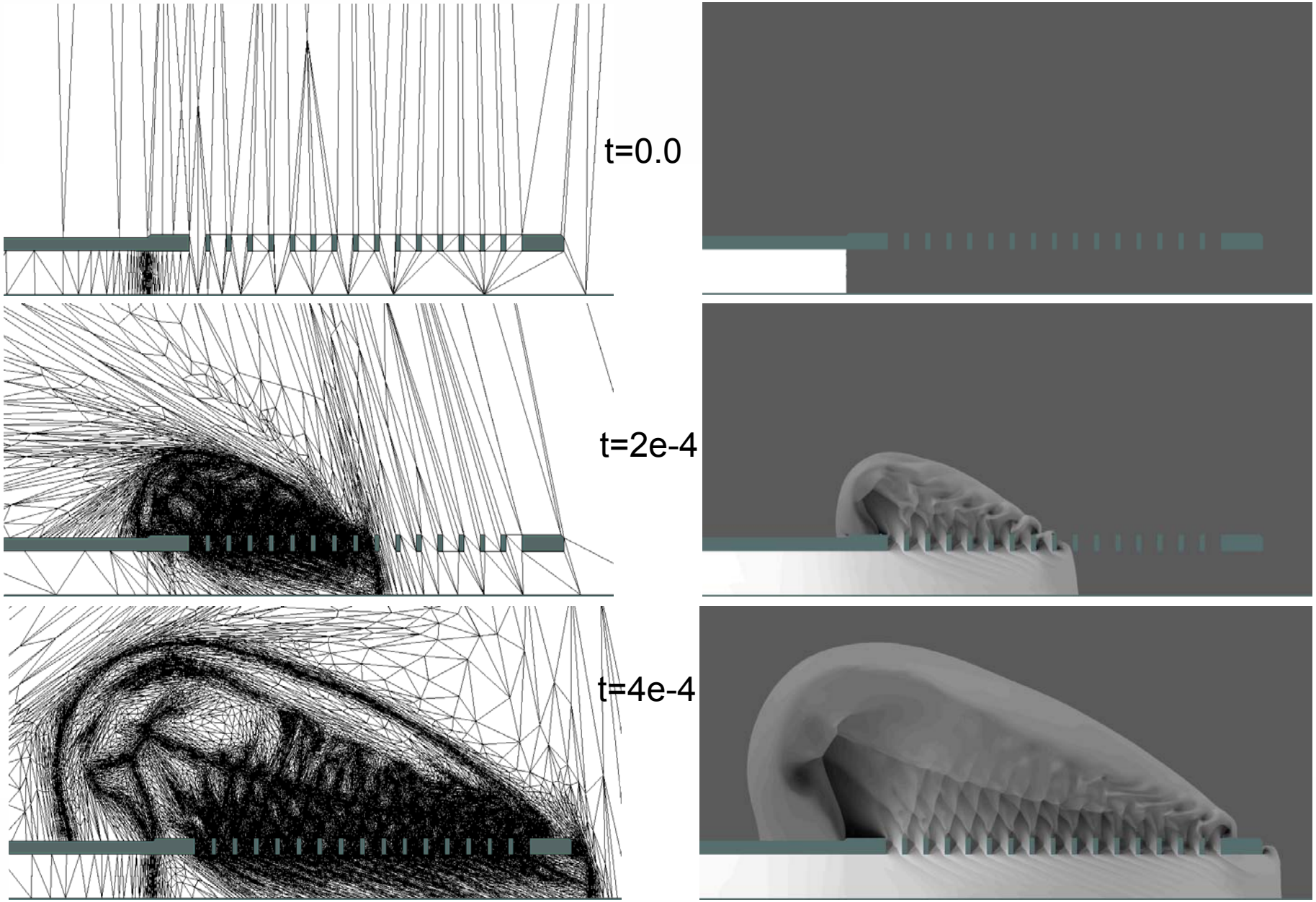
Adaptive Unstructured Mesh Methods: Characteristics and Application

- ITAPS has unstructured mesh tools
- Some Basic Characteristics
  - Meshes of mixed topologies and order easy
  - Commonly used spatial decomposition for finite element discretizations
  - Data structures larger and more complex
  - Solution algorithms can be more complex
- Some Advantages
  - Mesh adaptation can account for curved domains
  - General mesh anisotropy can be obtained
  - Easy to create strong mesh gradations without special numerical techniques
  - Alignment with multiple curved geometric features



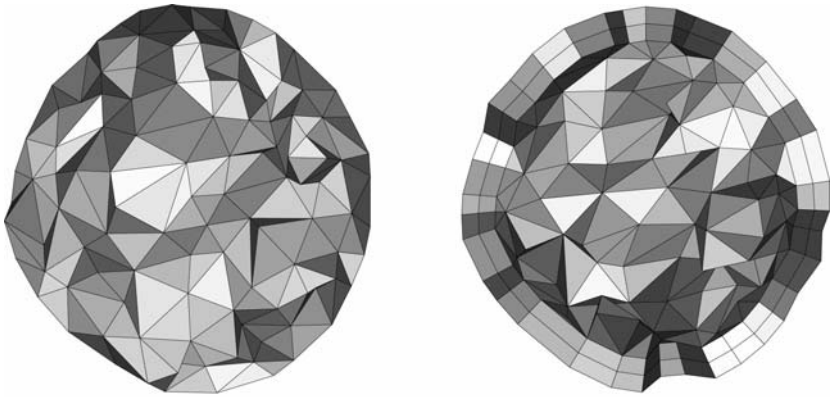
SciDAC
Scientific Discovery through Advanced Computing

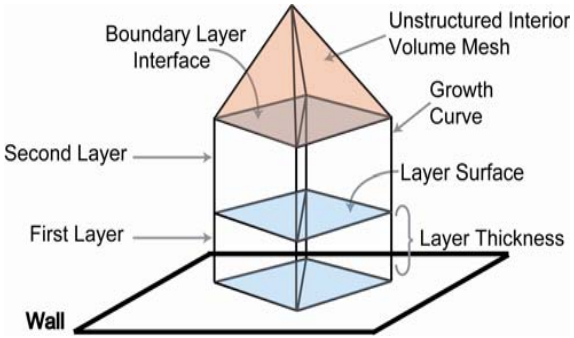
Muzzle Blast Example

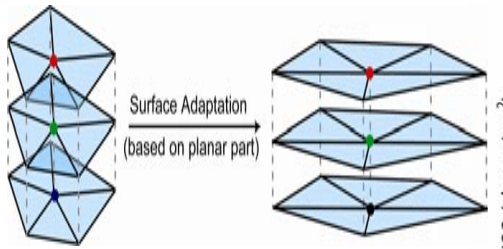


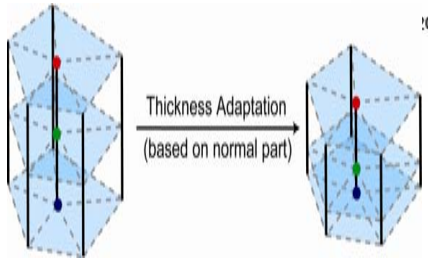
Maintaining "Structure" is of Value for Pointwise Derivative Recovery

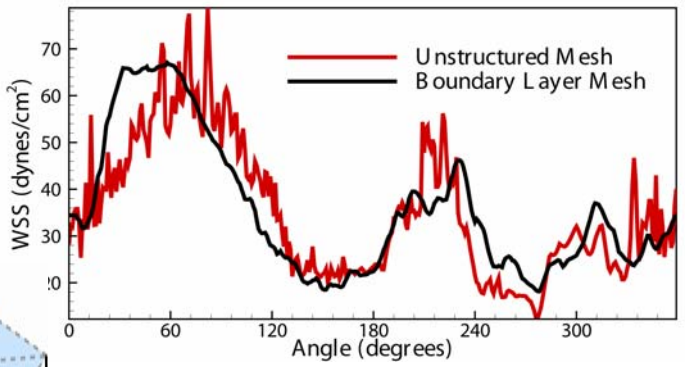
- Post-processing procedure for recovering conservative wall shear stress has been observed to be sensitive to near wall mesh "structure".
  - Coarse example of arterial cross section
- Semi-structured mesh adaptation added







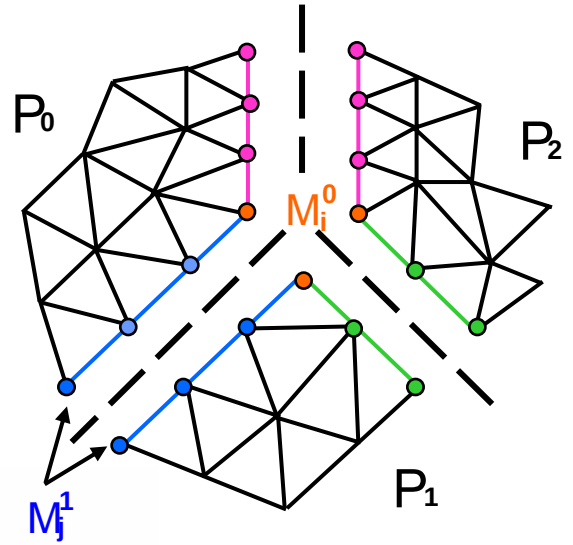


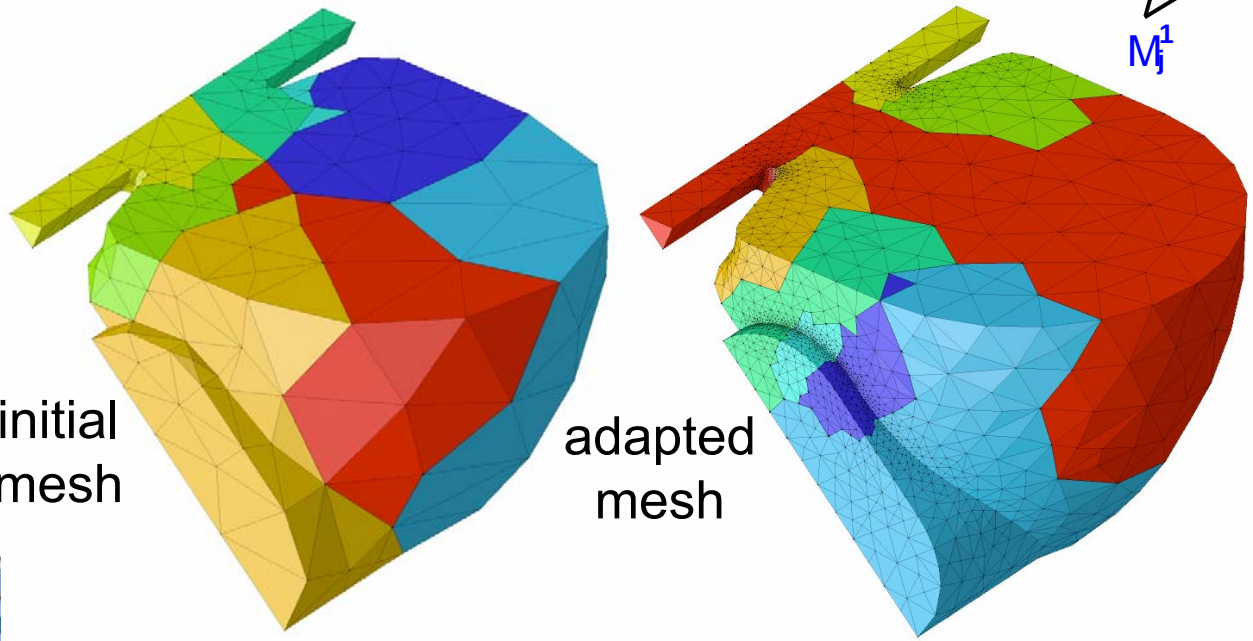


SciDAC
Scientific Discovery through Advanced Computing

Parallel Adaptive Simulation

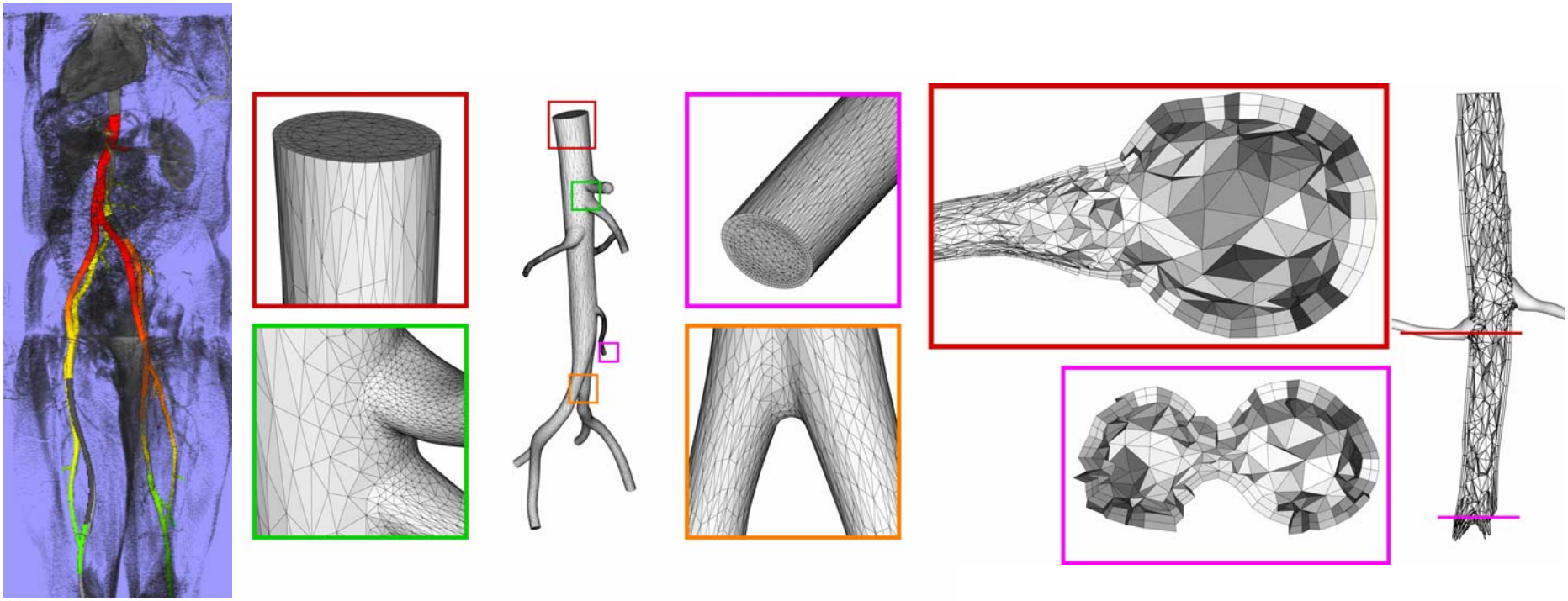
- Parallel Support Operations
  - Interprocessor links
  - Dynamic parallel partitioning
  - Mesh migration
  - Predictive balancing
  - Parallel mesh adaptation





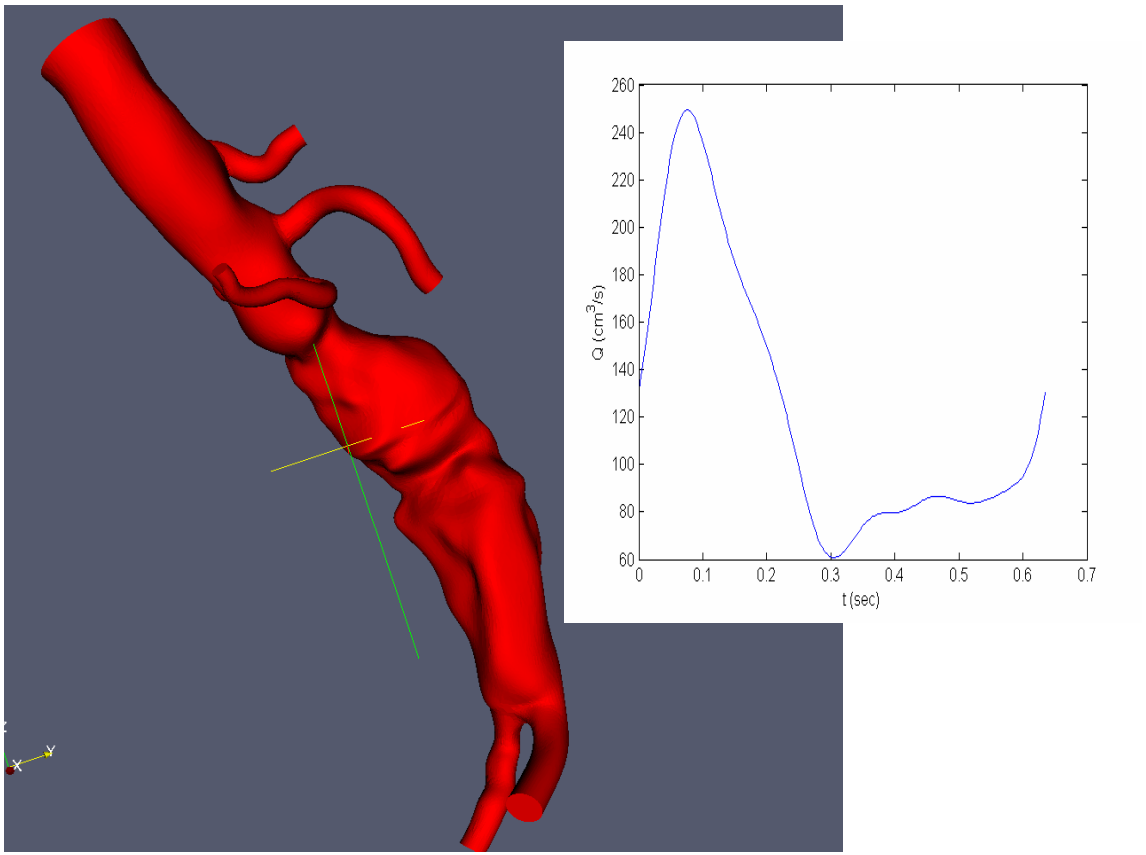
Patient Specific Vascular Surgical Planning

- Virtual flow facility for patient specific surgical planning
- High quality patient specific flow simulations needed quickly
- Image patent, create model, apply adaptive flow simulation



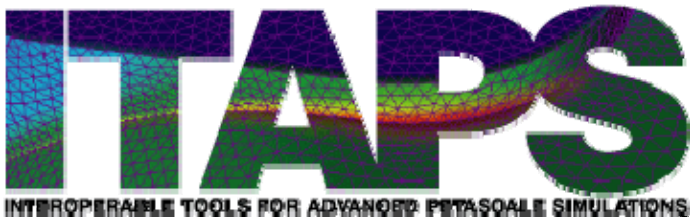
Abdominal Aorta Aneurysm

- Exercise conditions



QuickTime™ and a
H.264 decompressor
are needed to see this picture.

SciDAC
Scientific Discovery through Advanced Computing



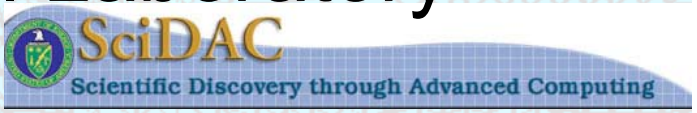
The FronTier Lite ITAPS service

Brian Fix, Xiaolin Li, Y. Li and James Glimm

Stony Brook University

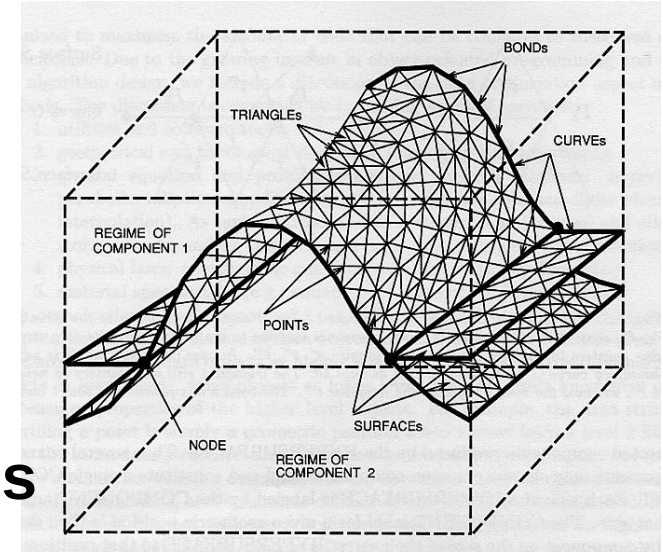
Zhiliang Xu and Roman Samulyak

Brookhaven National Laboratory



The front tracking method and FronTier code

FronTier **tracks a dynamically moving manifold representing a discontinuity or a moving boundary**



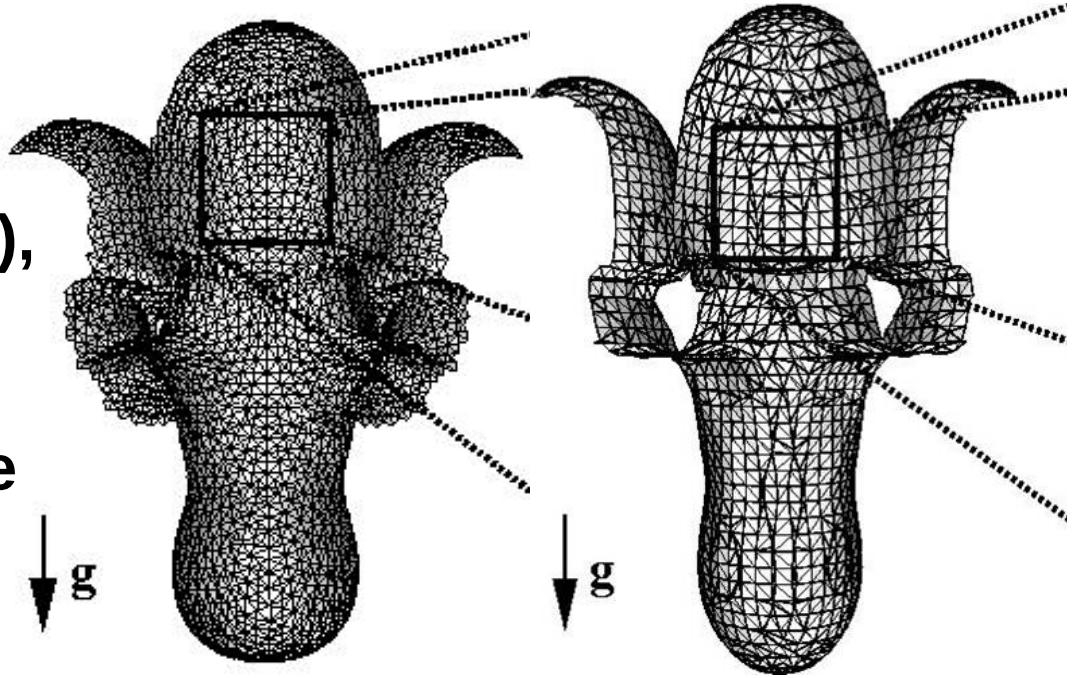
FronTier **features a meshed front separating computational subdomains**

FronTier **can be combined with PDE solvers and other scientific software packages in which a moving front is of scientific importance**

FronTier **automatically resolves topological bifurcation**

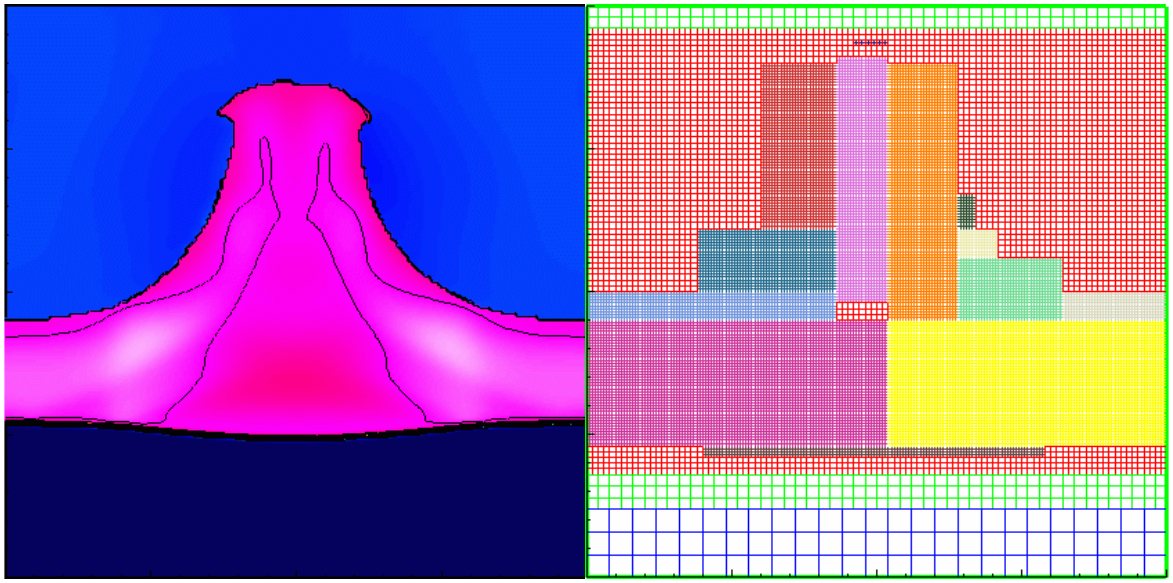
FronTier mesh and geometry

***FronTier* supports independent surface mesh (grid-free mesh), and grid-based mesh which interacts with the structured volume mesh.**



Above left: independent surface mesh; above right: grid-based surface mesh. Both for the simulation of gravity-driven Rayleigh-Taylor instability.

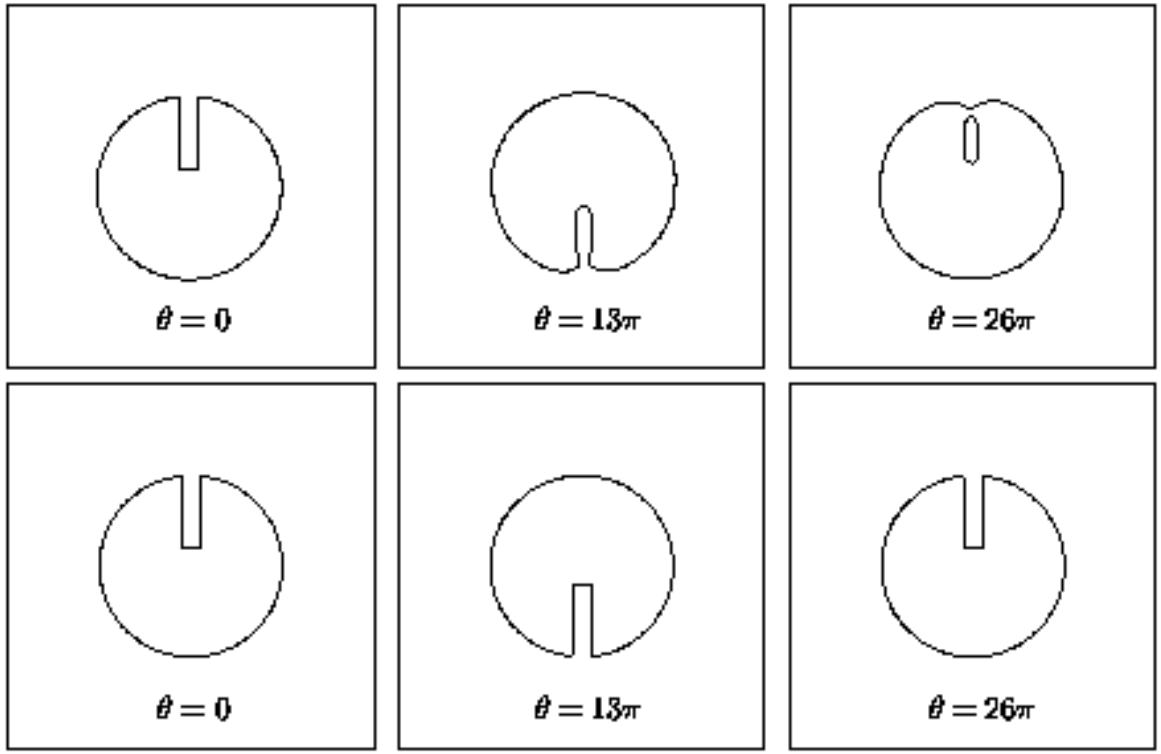
FronTier interoperability



***FronTier* inter-operation with the Overture code from LLNL**

***FronTier* usesthe adaptive mesh refinement from Overture code for the simulation of a Richtmyer-Meshkov instability.**

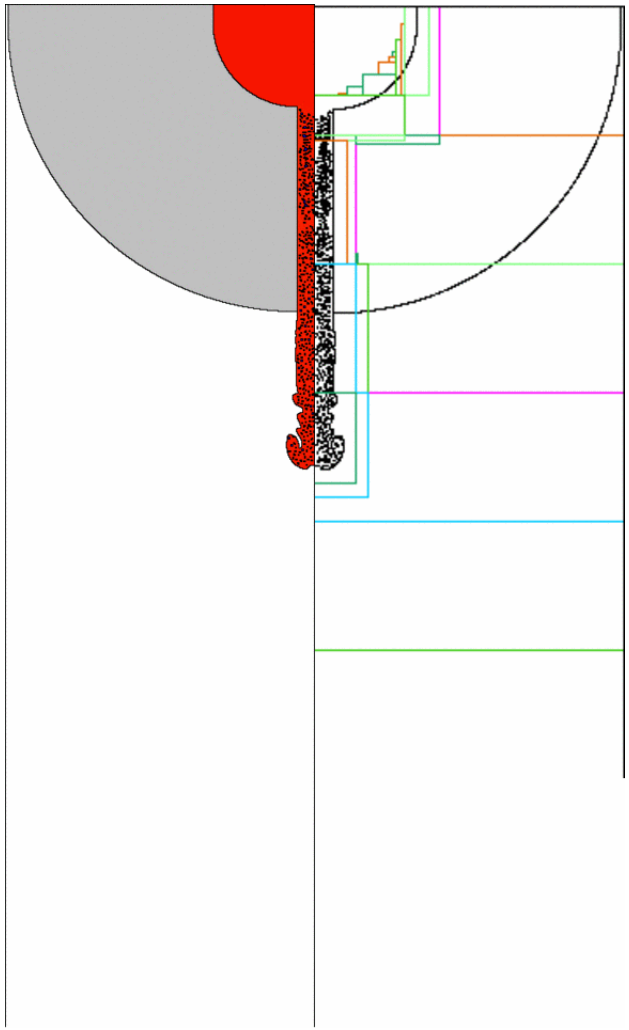
FronTier vs. level set method



The comparison of a rotating slotted disk using the 5th order level set and the *FronTier (4th order)* code.

FronTier application: diesel jet

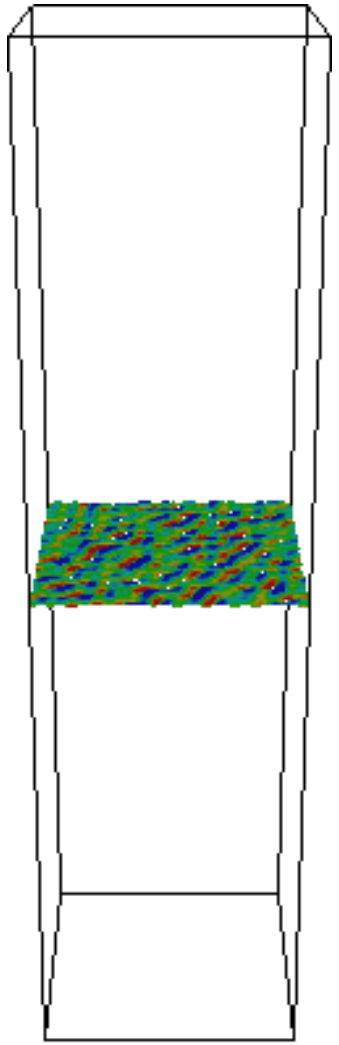
FronTier **application to the simulation of diesel jet injection. The simulation features the phase transition at the jet interface and the insertion of bubbles due to atomization in the nozzle.**



FronTier application: chaotic mixing

***FronTier* application to the simulation of fluid interface instability, induced by the acceleration field.**

The tracking of the fluid interface provides sharp, high resolution at the interface and yields the best result in agreement with experiment.



FronTier service Capabilites:

- Tracking a meshed and dynamically moving front
- Optimization of mesh geometry and automatic topological bifurcation of tangled front
- Implemented in 1D, 2D, and 3D
- In 1D and 2D, the tracking is grid-independent
- In 3D, choice of several tracking algorithms including grid based tracking, grid free tracking, and locally grid based tracking (recommended for both quality and robustness).
- Can be run as a fluid code (gas dynamics) or called as a toolkit library.

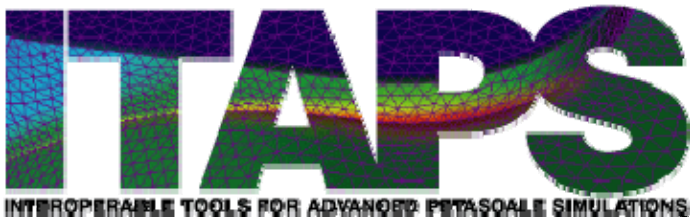
Conclusions:

The ITAPS *FronTier* service is simple to use and can be a useful tool

We provide a toolkit for scientific applications in which a dynamically moving front plays an important role in physical problems.

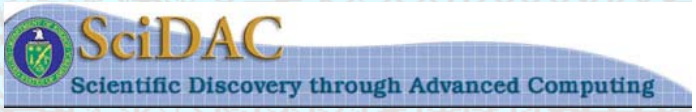
The *FronTier* library provides Lagrangian lovers an accurate package for surface propagation without experiencing the complexity and repeating the work by themselves.

The ITAPS interface standardizes the *FronTier* Mesh description and operation of the *FronTier* functions.



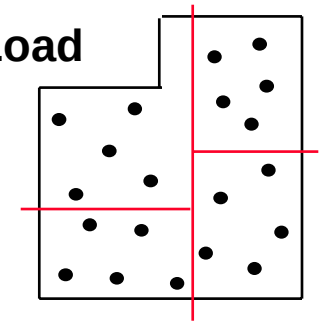
Dynamic Services (Zoltan)

Vitus Leung and Karen Devine
Sandia National Laboratories

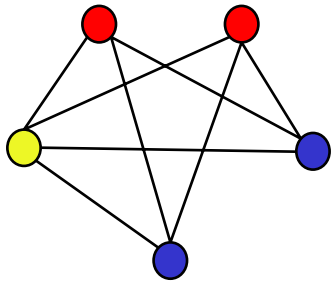



Zoltan Toolkit: Data Services for Dynamic Applications

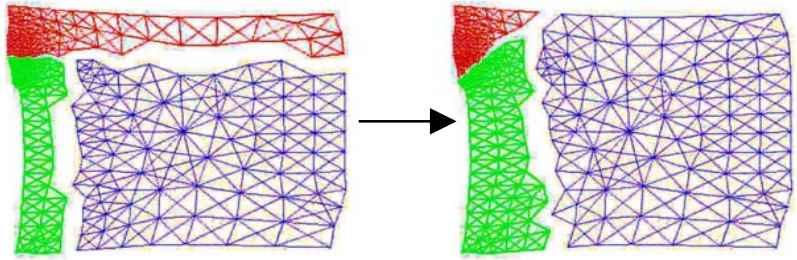
Dynamic Load Balancing



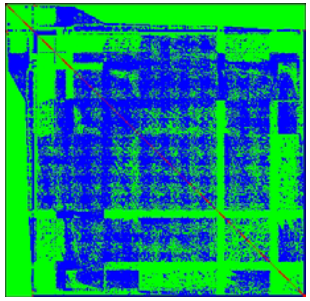
Graph Coloring



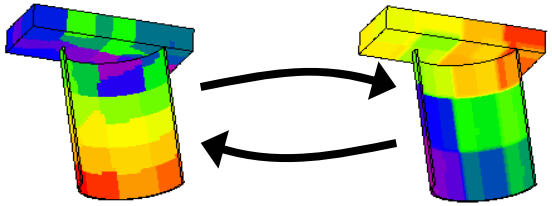
Data Migration



Matrix Ordering



Unstructured Communication



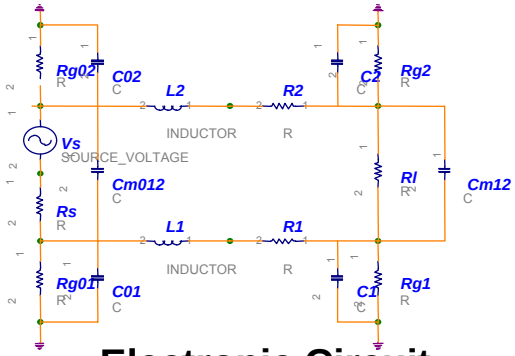
Distributed Data Directories

A	B	C	D	E	F	G	H	I
0	1	0	2	1	0	1	2	1

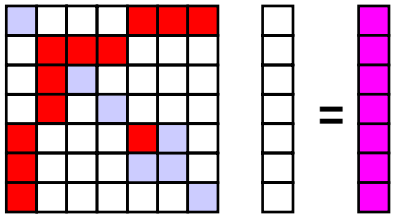
Dynamic Memory Debugging

SciDAC
Scientific Discovery through Advanced Computing

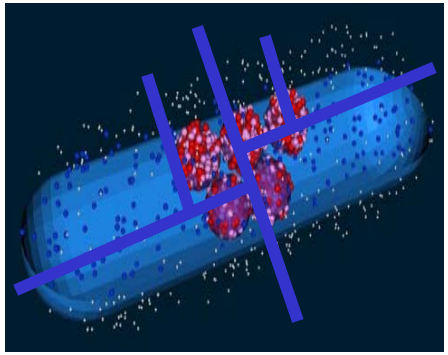
Applications using Zoltan



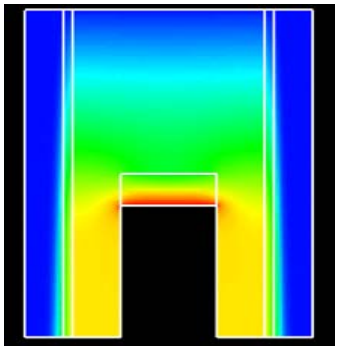
Electronic Circuit Simulations



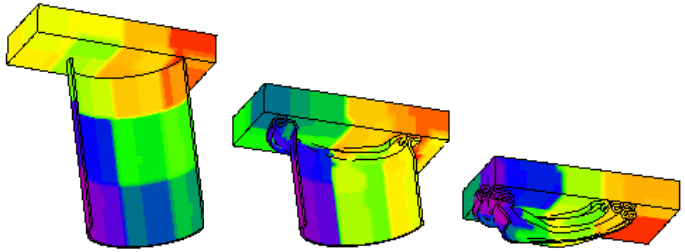
Linear Solvers & Preconditioners



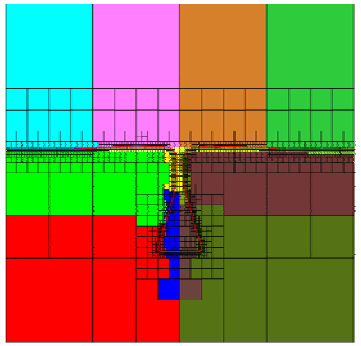
Particle-based Biological Cell Simulations



Multiphysics Simulations



Contact Detection & Crash Simulations

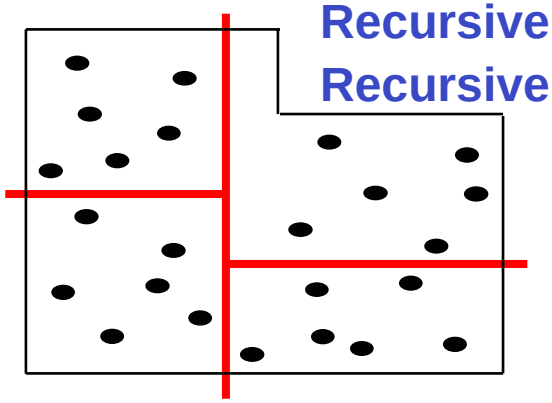


Adaptive Mesh Refinement

SciDAC
Scientific Discovery through Advanced Computing

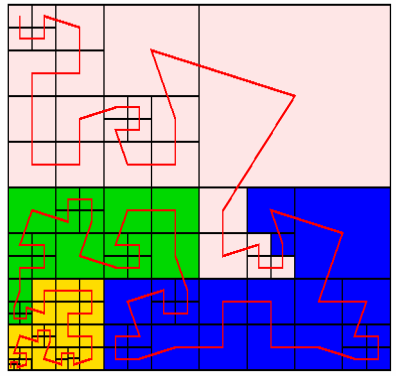
Zoltan Suite of Partitioners

Geometric (coordinate-based) methods



Recursive Coordinate Bisection (Berger, Bokhari)

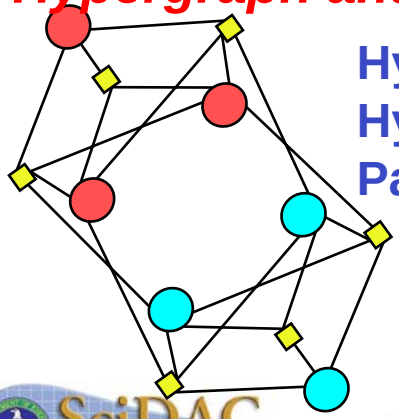
Recursive Inertial Bisection (Taylor, Nour-Omid)



Space Filling Curves (Peano, Hilbert)

Refinement-tree Partitioning (Mitchell)

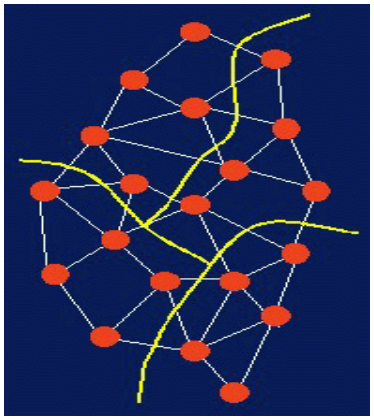
Hypergraph and graph (connectivity-based) methods



Hypergraph Partitioning

Hypergraph Repartitioning

PaToH (Catalyurek)



Zoltan Hypergraph Partitioning

ParMETIS (U. Minnesota)

Jostle (U. Greenwich)

SciDAC
Scientific Discovery through Advanced Computing

ITAPS C Interface to Partitioners

- Given a loaded iMesh_Instance, mesh, with iBase_EntitySetHandle, root_set
- Construct ITAPSZoltan object, itapsz
- Call partitioner from itapsz with number of partitions, method, sub-method
- Partitions contained in mesh as tagged entity sets upon return from partitioner
- Compile and run with MPI

ITAPS
INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

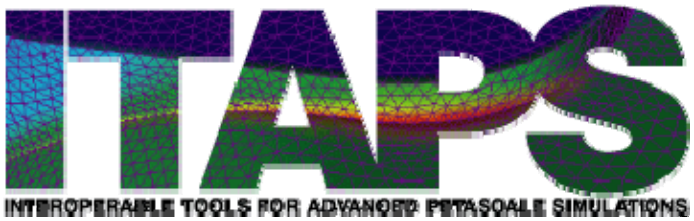
PART 5: Using ITAPS: Approaches and Experiences

SciDAC
Scientific Discovery through Advanced Computing

Sandia National Laboratories
Rensselaer
STONY BROOK
THE UNIVERSITY OF BRITISH COLUMBIA
BROOKHAVEN NATIONAL LABORATORY
OAK RIDGE NATIONAL LABORATORY
Pacific Northwest National Laboratory

NEEDED

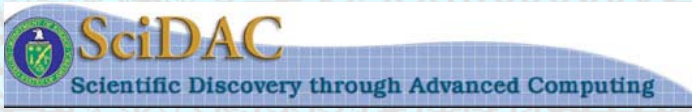
- Case Study 1: New application for nuclear reactor modeling



Mesh Generation for Nuclear Reactor Simulation

Ahmed Khamayseh and
Valmor de Almeida

Oak Ridge National Laboratory




Goals and People Involved

- Technical goals – Utilize and integrate the ITAPS Geometry, Meshing, and Adaptivity Services (GMAS) for Nuclear Reactor Simulations

- People involved (Ahmed Khamayseh and Valmor de Almeida—ITAPS, Glen Hansen and Kevin Clarno---GNEP)

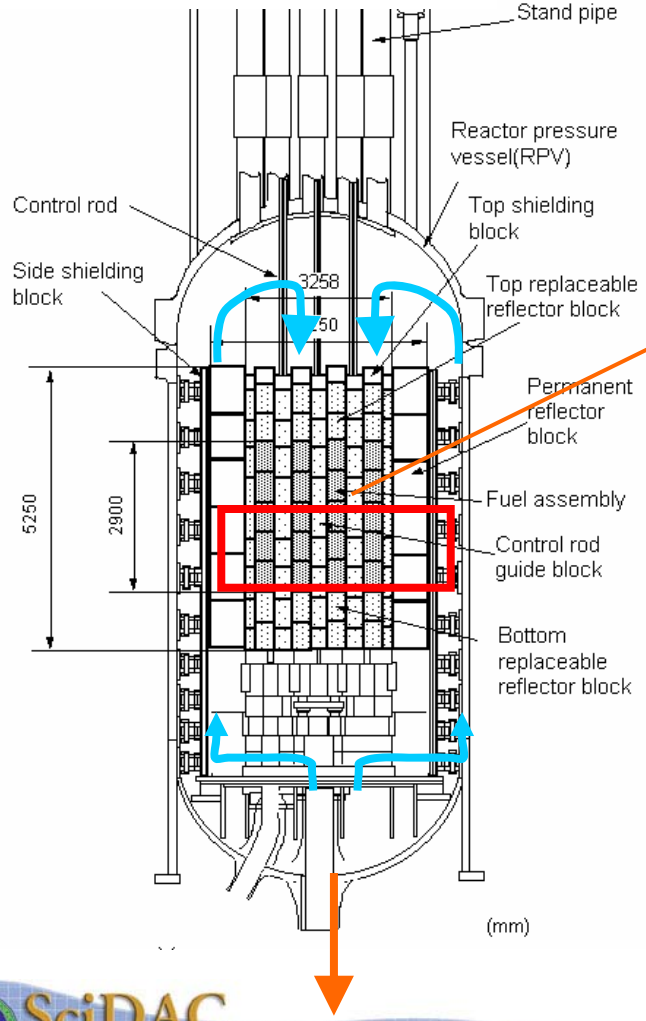
- ITAPS Services Used:

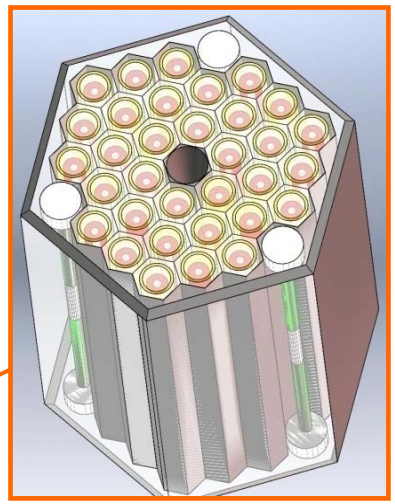
Modern Reactor Modeling and Simulation

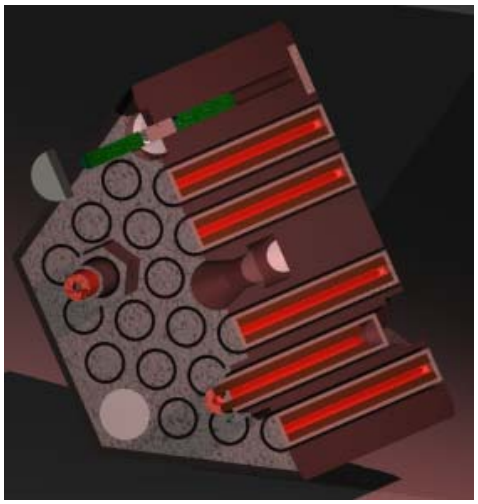
- **Full-core geometry realism**
  - A multi-material, multi-region mathematical domain
- **Coupled multi-phenomena modeling**
  - Particle transport, heat & fluid flow, solid mechanics
- **Multi-meshing**
  - Application-based, interoperable meshes
- **Advanced parallel algorithms**
  - Petascale computing

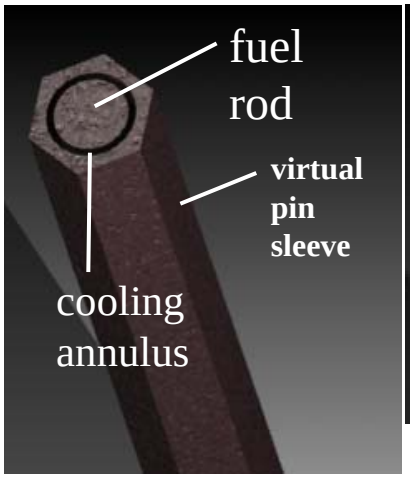
Modeling Challenges – Geometric Complexity

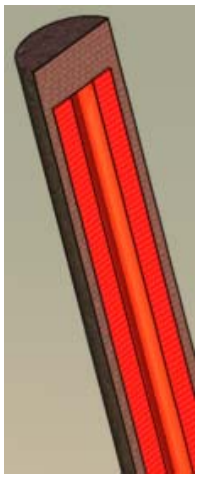
Japan's HTTR 30MWt







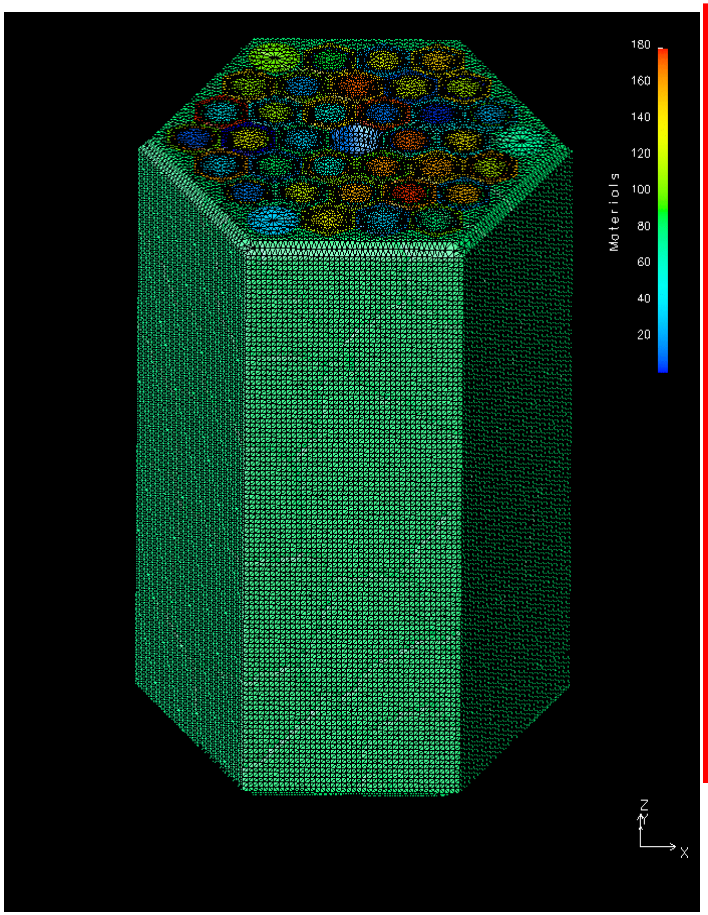




Fuel rod

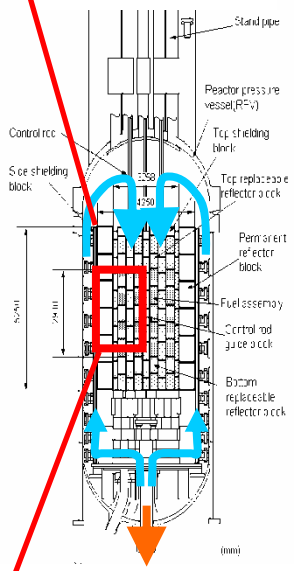
SciDAC
Scientific Discovery through Advanced Computing

Meshing Challenges – Problem Scale



- 150 fuel elements (assemblies)
- 120 replaceable reflector blocks
- 72 permanent reflector blocks
- 144 control rod blocks
- 27 irradiation blocks
- Over 30,000 parts
- 3+ M elements per fuel block
- 1B elements full core

HTTR



Modern Reactor Modeling and Simulation

Geometry, Meshing, and Adaptivity Services

- Intend to provide geometry, meshing and adaptivity services for multiphysics applications
- Handle multiple meshes for multiple PDE solvers for a given geometry
- Follows the CCA and will provide services through ports
- Written in C++ and will incorporate C++/C libraries directly, whereas FORTRAN libraries will be integrated via CCA
- Has been used to provide meshing services for a neutron transport simulation and a solvent extraction fluid flow code in development

Technology Developments- GMAS

Description of ITAPS technologies developed/applied

- Integrate geometry, meshing, and adaptivity software (SciDAC, public domain, etc.) and provide services to multiphysics, coupled, PDE's solvers

- Developed or extended functionality of software in the area of geometry modeling, meshing and adaptivity. On-demand-basis development (as of now through short term projects)

- Aimed at large scale parallel computing

Results

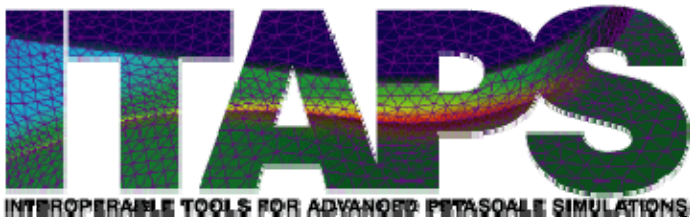
Results obtained and benefits afforded the application

- Applications experience in gas-cooled reactors, materials science, neutronics.

- Code and theory base in mesh adaptation, multiphysics modeling, spatial searching, and remapping.

NEEDED

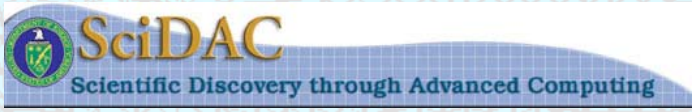
- Case Study 2: Inserting adaptive mesh refinement into a fusion application



Adaptive Mesh Control for Extended MHD Simulations in PPPL's M3D-C^1 Code

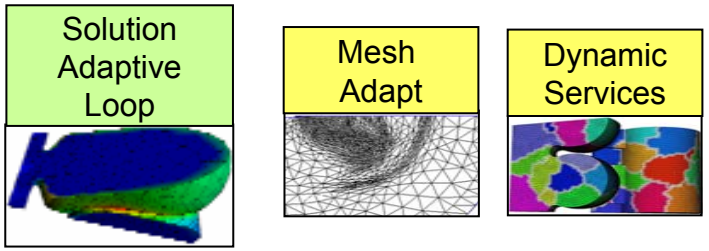
SCOREC, RPI

PPPL




Goals and People Involved

- Develop parallel adaptive simulation technologies for extended MHD simulations
- Andrew Bauer, Kenneth Jansen, Mark Shephard, SCOREC, RPI
- Stephen Jardin, Nathaniel Ferraro, PPPL
- ITAPS Services Used:

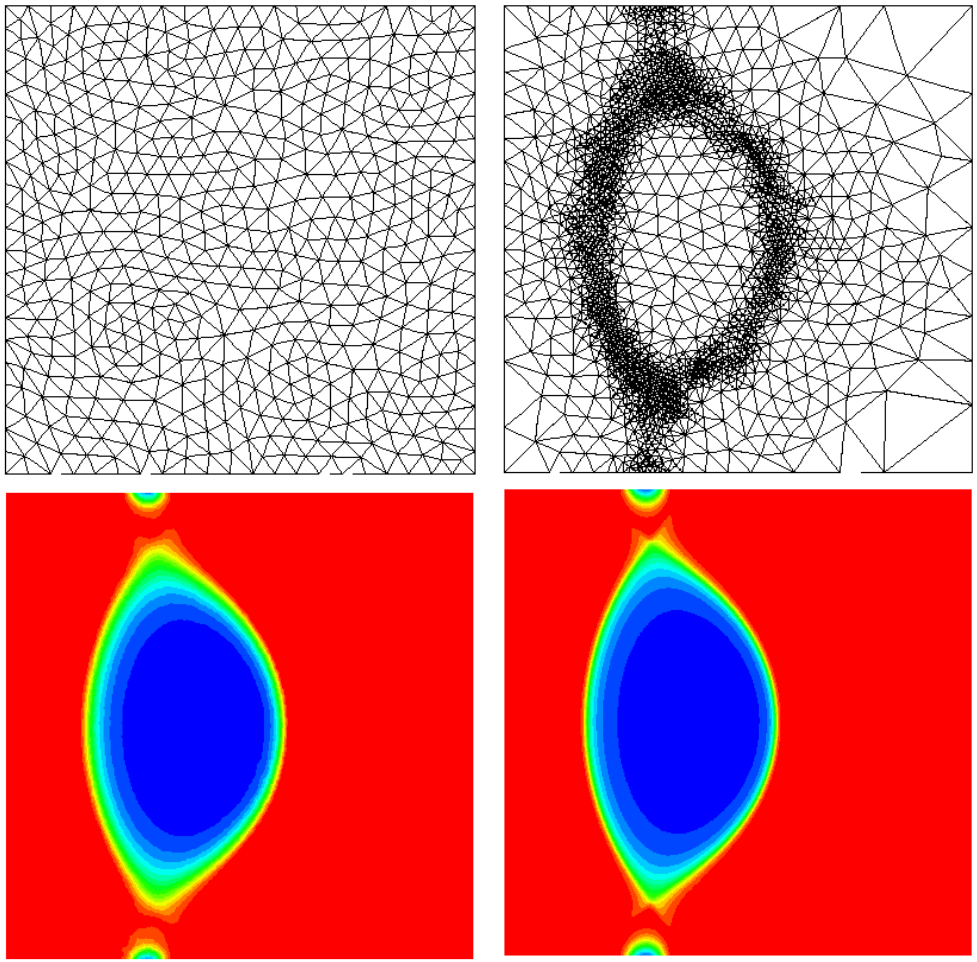


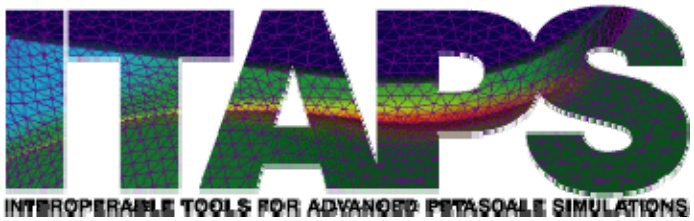
Technology Developments

- Modified M3D-C^1 structured code to use unstructured meshes
  - Interface to efficient unstructured mesh search algorithm
  - Efficient use of C^1 continuous shape functions for unstructured meshes
  - Reduced interprocessor communication bottlenecks
- Interfaced with mesh adapt software
  - Size field based mesh adaptation based on two methods:
    - Integrated field quantities
    - Inter-element flux jump of Laplacian

Adaptive mesh provides substantial improvement in results

Toroidal equilibrium problem that converges to steady state

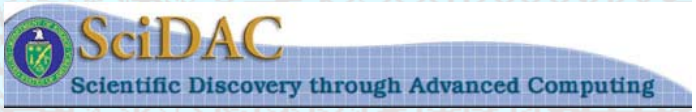




SciDAC Application Area: Pellet Ablation for Tokamak Fuelling

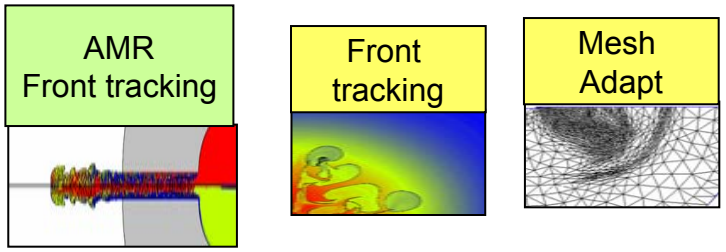
Roman Samulyak

Brookhaven National Laboratory




Goals and People Involved

- Technical goals of the ITAPS involvement
  - Provide Adaptive Meshing and Front Tracking technologies and software for “microscale” studies of pellet ablation in tokamaks
  - Provide subgrid models for multiscale coupling with macroscale pellet simulation codes and transport codes
- People involved:
  - R. Samulyak, T. Lu, BNL (ITAPS and OASCR base funding)
  - P. Parks, General Atomics (Application)
- ITAPS Services Used:



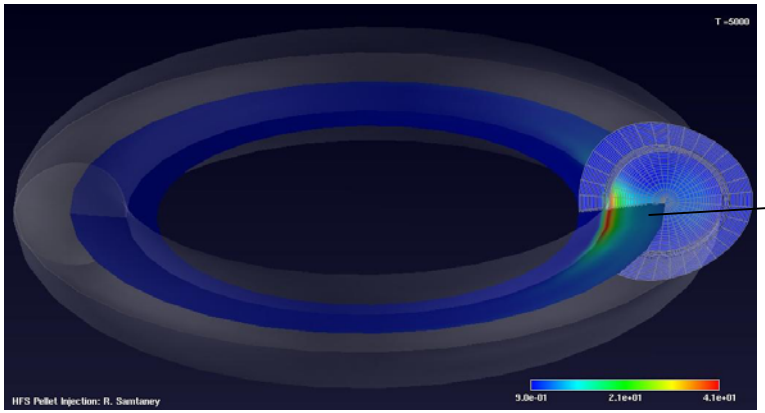
Technology and Software Developments

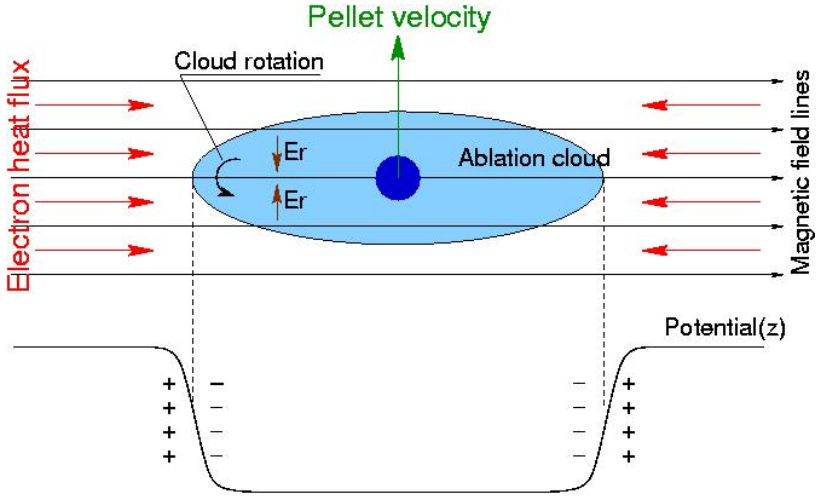
ITAPS
INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

- Further development of Adaptive Meshing technologies and Front Tracking for phase boundaries

- Based on Front Tracking, an MHD code for the detailed study of the pellet ablation physics has been developed and validated. It includes
  - Kinetic model for interaction with hot electrons
  - Surface ablation model
  - Equation of state with atomic processes
  - Cloud charging and rotation models
  - New conductivity model (ionization by electron impact)

Relation to Other Projects





Macroscale Model: AMR simulation of ablation flow in plasma (Ravi Samtaney)

- Focus on plasma flow
- Ablation physics not resolved (simplified analytical models)

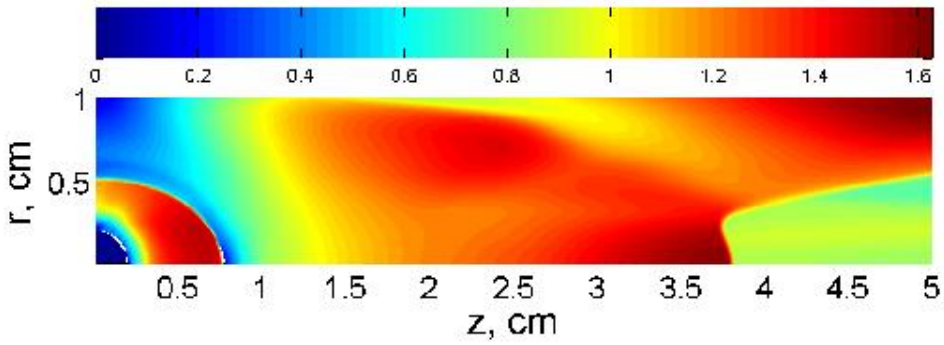
Microscale Model: Front Tracking simulation of pellet ablation

- Focus on detailed ablation physics, ablation rates etc.
- Far field plasma evolution not resolved

Both approaches are complementary. Future goal is their coupling.

Simulation Results

- Code validation and benchmarks with other hydro studies
- First systematic "microscale" MHD studies of pellet ablation physics
- Revealed new properties of the ablation flow (supersonic rotation of the ablation channel)
- Explained the factor of 2.2 reduction of the ablation rate in hydrodynamic models with directional heating
  - In the literature, it was incorrectly attributed to the directional heating; we showed it was caused by Maxwellian electron heat flux vs. monoenergetic

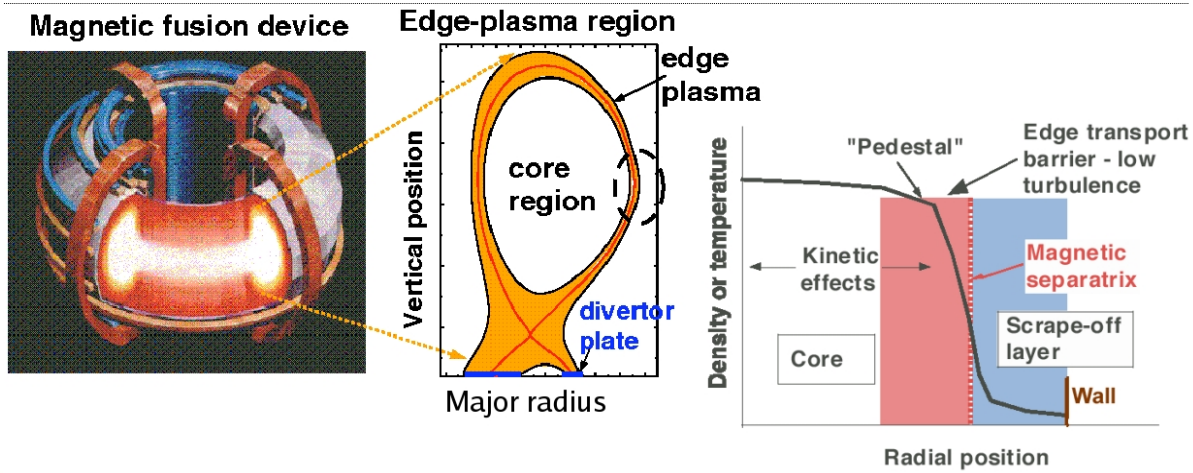


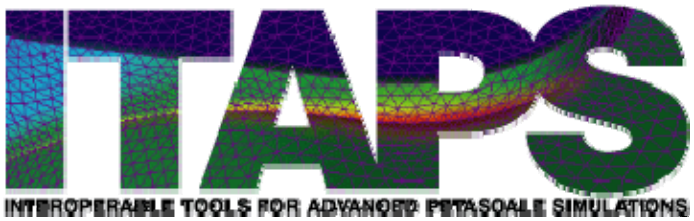
Distributions of the Mach number of the ablation flow near the pellet

Simulation Results

Critical observation:

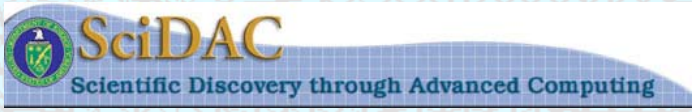
- **Formation of the ablation channel and ablation rate strongly depends on plasma pedestal properties and pellet velocity**
  - Simulations suggest that novel pellet acceleration technique (laser or gyrotron driven) are necessary for ITER





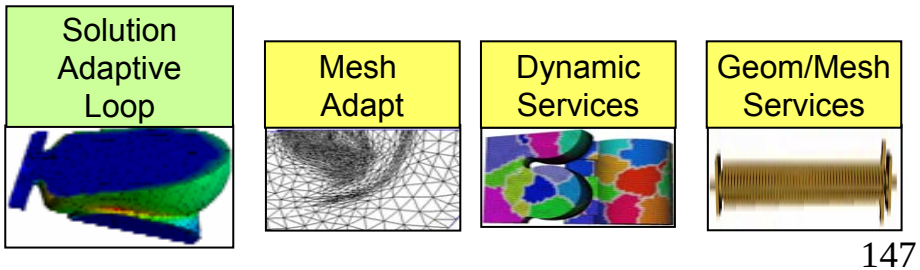
Parallel Adaptive Loop for Accelerator Design

SCOREC, RPI
SLAC

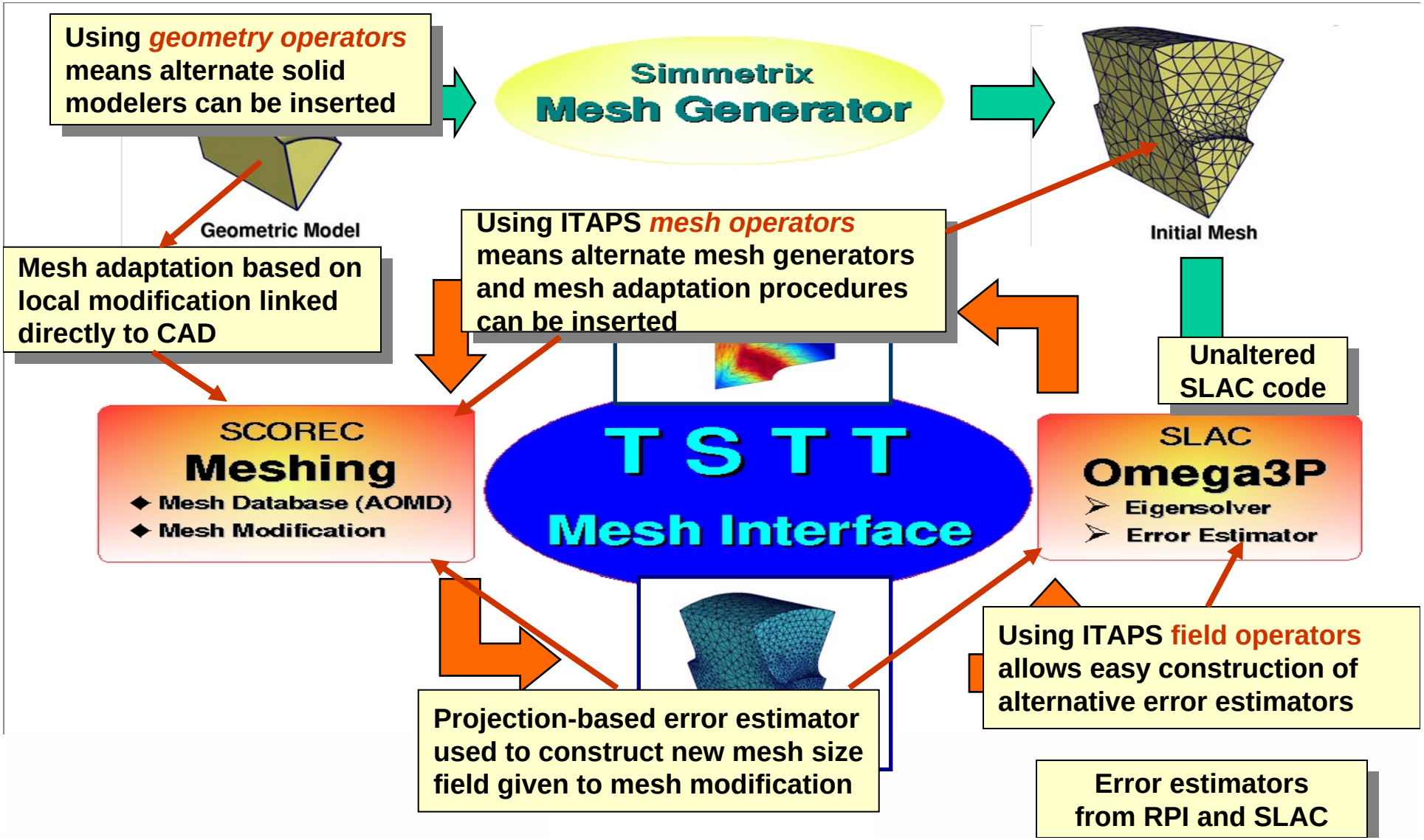



Goals and People Involved

- Provide automated adaptive mesh control for the Omega3P FE eigensolver used to calculate RF cavity frequencies
- Several people from RPI SCOREC
- Simmetrix Inc.
- Lixin Ge, Liequan Lee, Zenghai Li, Cho Ng, Inam Ur Rahman, Yong Sun and Kwok Ko from SLAC
- ITAPS Services Used:

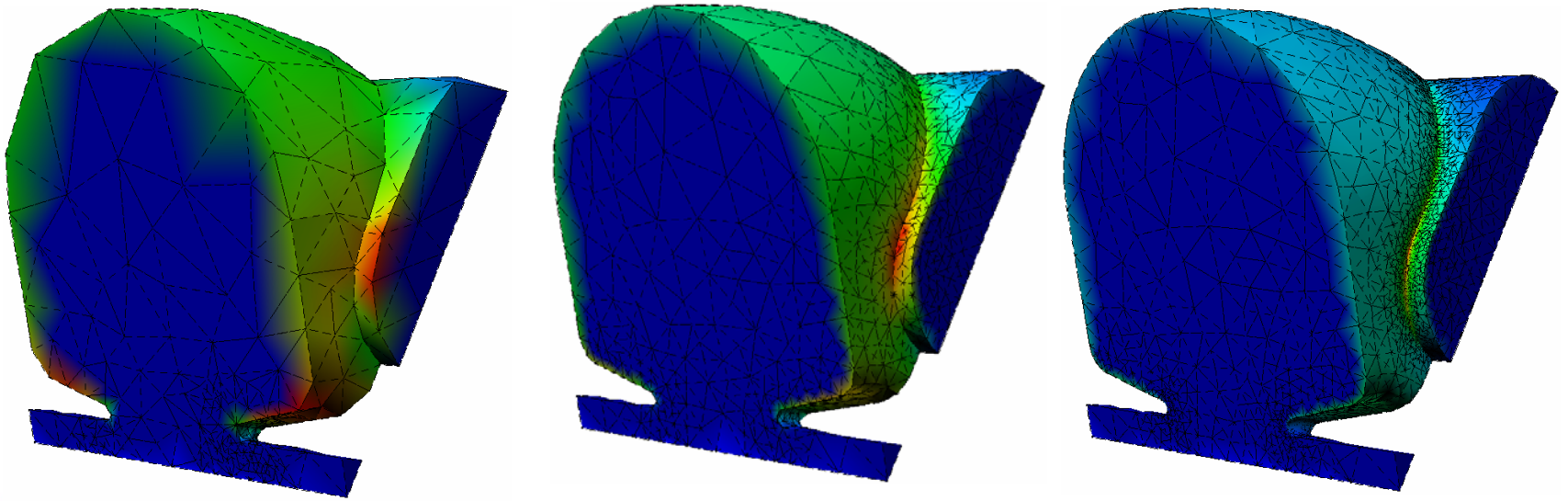


Components in adaptive loop for SLAC



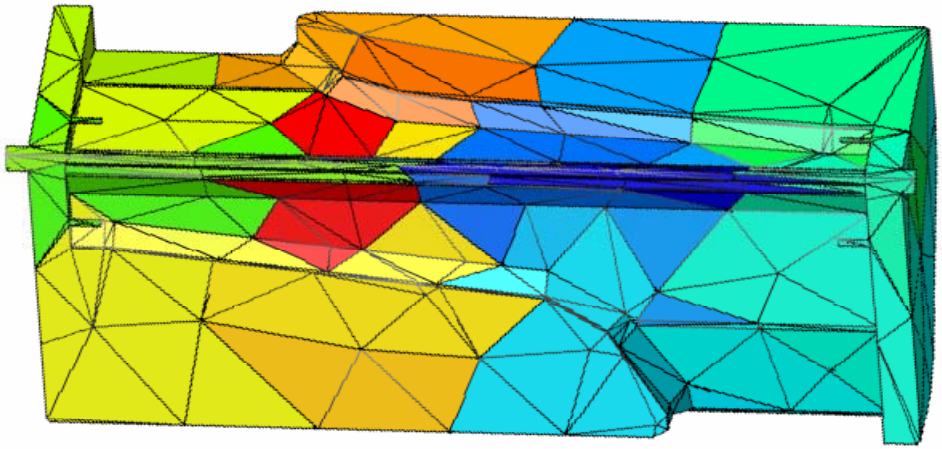
Adaptive Loop for SLAC Accelerator Design

- Geometry defined in CAD modeler
- Omega3P code from SLAC
- High level modeling accuracy needed
- Adaptive mesh control to provide accuracy needed
- Adaptive loop runs in serial and parallel

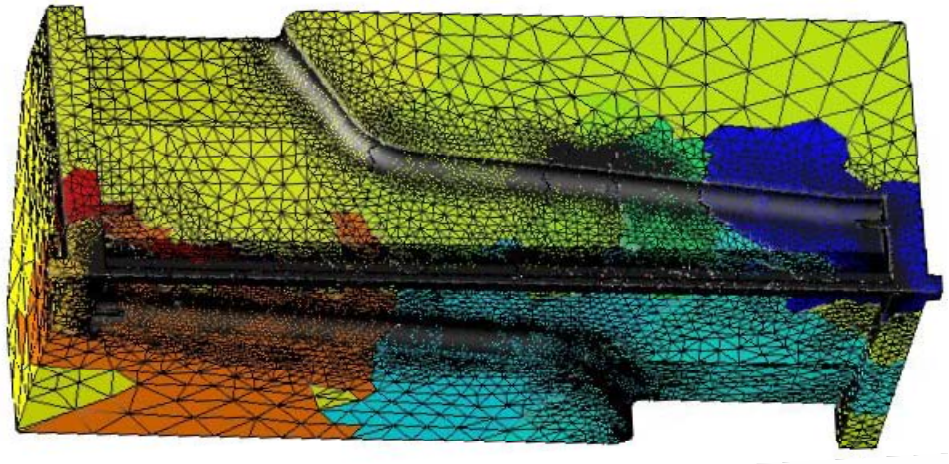


SciDAC
Scientific Discovery through Advanced Computing

Adaptive Loop for Accelerator Design



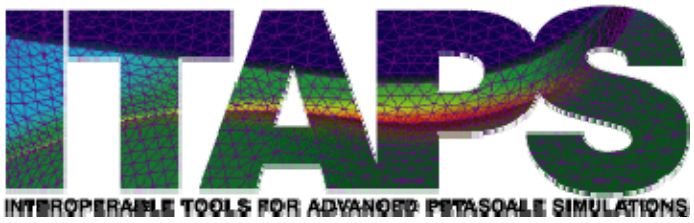
Initial mesh (1,595 tets)



Adapted mesh (23,082,517 tets)

NEEDED

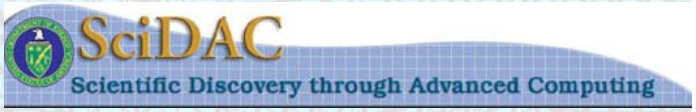
- Case Study 3: Using Several tools in concert for shape optimization in accelerator cavities



DDRIV: Geometry & Mesh Services for Shape Optimization

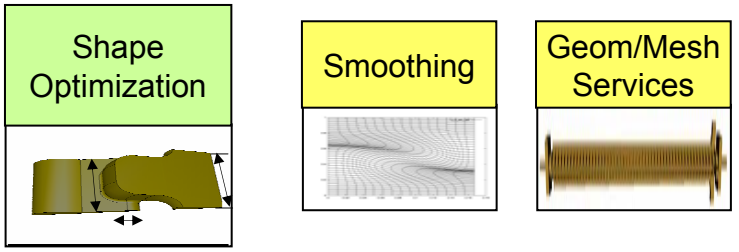
Tim Tautges, ANL

ITAPS, SLAC, TOPS



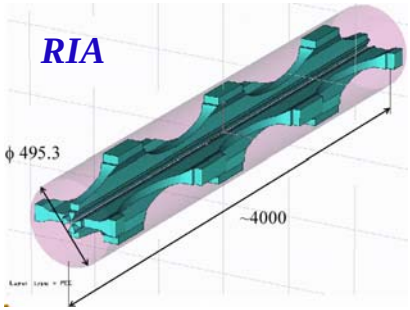

Goals and People Involved

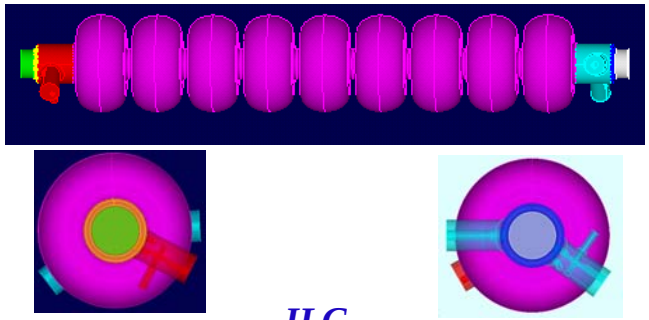
- Technical goals:
  - Develop shape optimization for accelerator cavity tuning and de-tuning, and for reverse engineering as-assembled cavity shape
  - ITAPS provides the geometry control and meshing components
- People involved:
  - SLAC: Rich Lee, Cho Ng, Volkan Ancelik
  - ITAPS: Tim Tautges and Pat Knupp
  - TOPS: Volkan Ancelik and Omar Ghattas
- ITAPS Services Used:

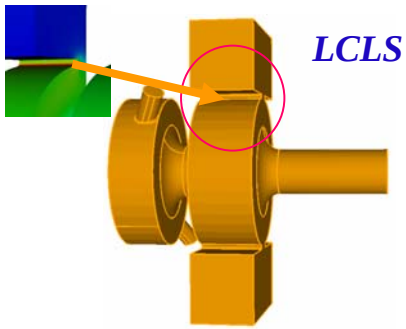


Shape Optimization for Accelerator Cavity Design

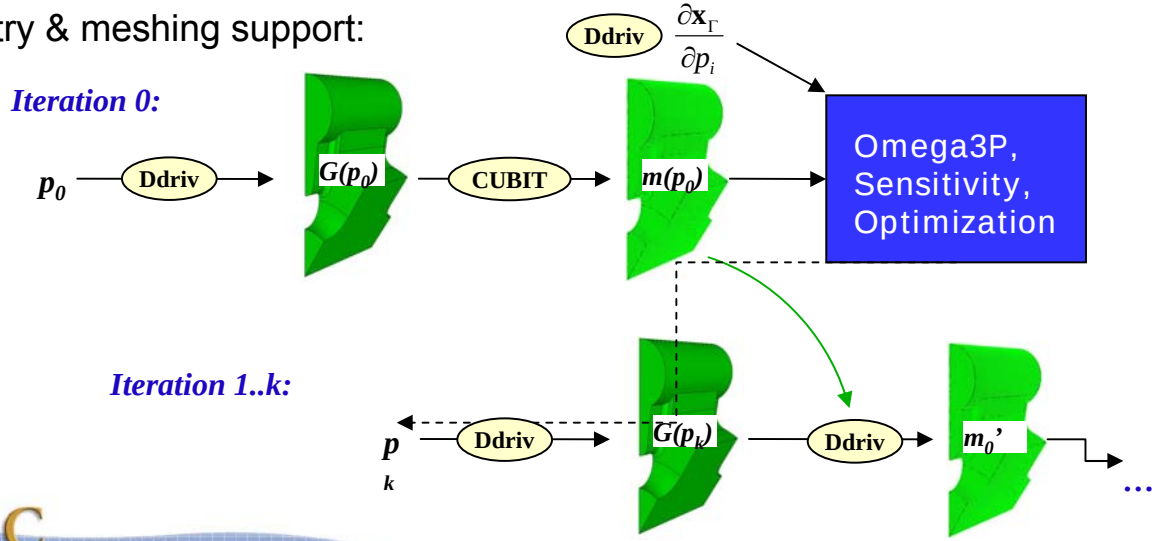
- Optimizing a cavity design is still mostly a manual process
- Future accelerators employ complex cavity shapes that require optimization to improve performance







- Geometry & meshing support:

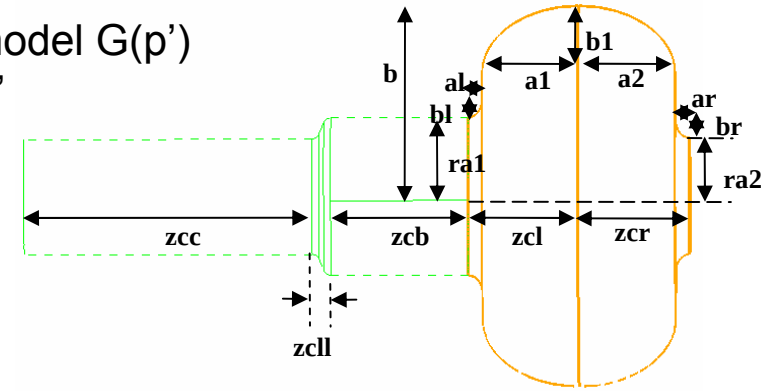


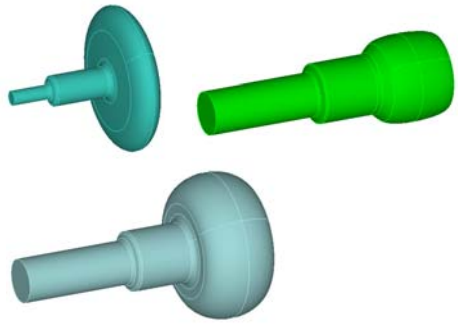
Fixed mesh topology:
Convergence
No re-meshing
Re-use factorization

SciDAC
Scientific Discovery through Advanced Computing

Shape Optimization for Accelerator Cavity Design

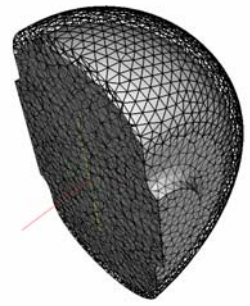
- Generate new geometric model G(p') given a parameter vector p'
  - *MkILCCell function*
    - DDRIV
    - CGM (iGeom)

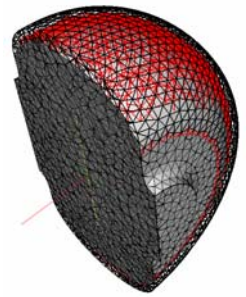




- Associate old mesh m(p_o) to new geometry G(p'), project to CAD
  - DDRIV
  - CGM (iGeom)
  - MOAB (iMesh)
  - LASSO (iRel)

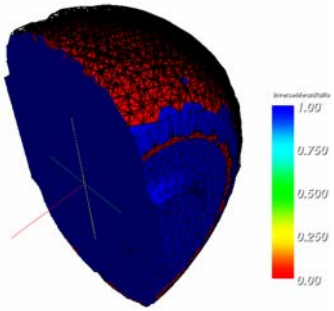
New geom, old mesh



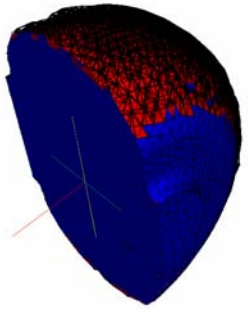


Project to CAD, inverted elements

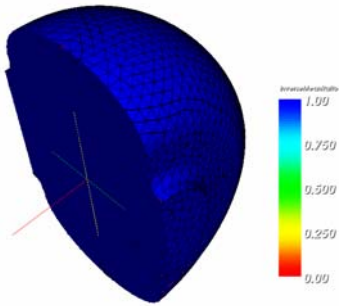
- Smooth mesh
  - Mesquite



Smooth Curves



Smooth Surfaces

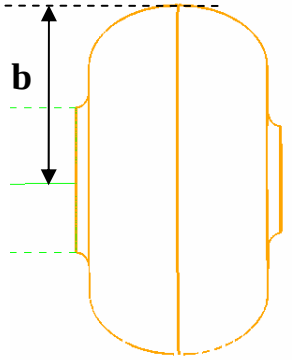


Smooth Volume

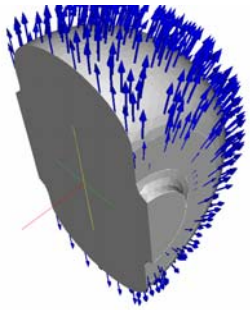
SciDAC
Scientific Discovery through Advanced Computing

Shape Optimization for Accelerator Cavity Design

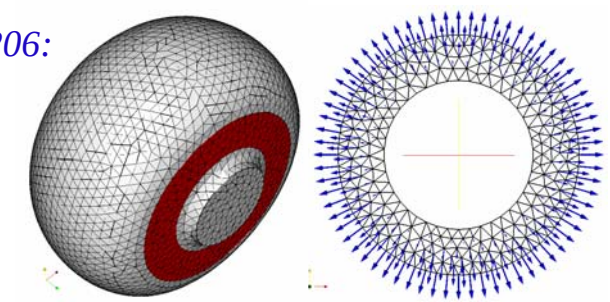
- Compute "design velocity" $\partial \mathbf{x}_\Gamma / \partial p_i$
  - Deform each parameter & smooth affected surfaces
  - Compute $\partial \mathbf{x}_\Gamma / \partial p_i$ using finite difference



Whole model:

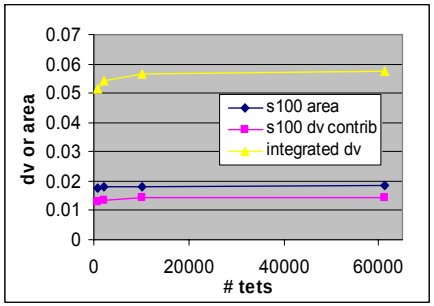


Surf 206:

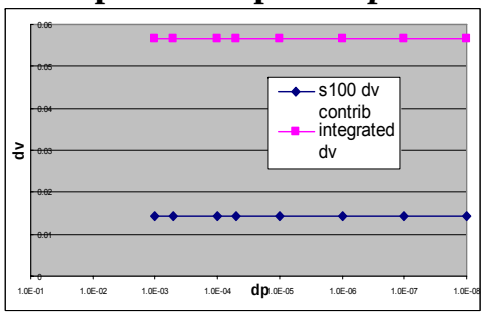


- Convergence of $\int (V \bullet n) dS$:

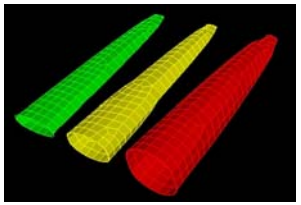
vs. *h*

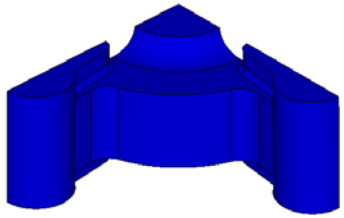


vs. param step size *dp*



- Why not use parametric surfaces & analytic derivatives?
  - *Parameter bounds change*
  - *Some surfaces not analytic*
  - *Not as flexible as current approach*





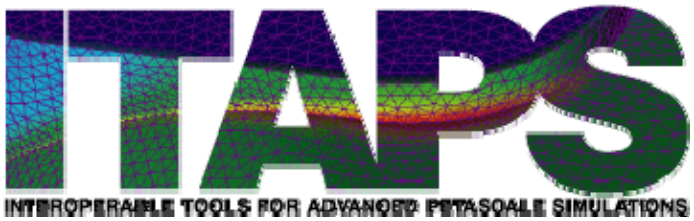
SciDAC
Scientific Discovery through Advanced Computing

Services Provided by DDRIV

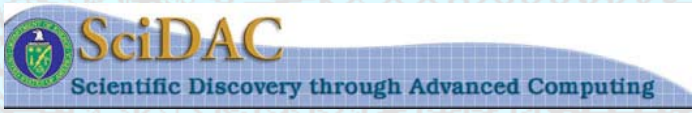
- Parameterized geometric model construction
  - You write function which constructs model using iGeom
  - DDRIV acts as driver and handles IO
- Coordination of mesh smoothing on geometric model
- Re-classification of “old” mesh on “new” model
- Target matrix-based smoothing of re-classified mesh
- Computation of design velocities & embedding on mesh using iMesh Tags

NEEDED

- Best practices – is this really the best place? Get slides from earlier



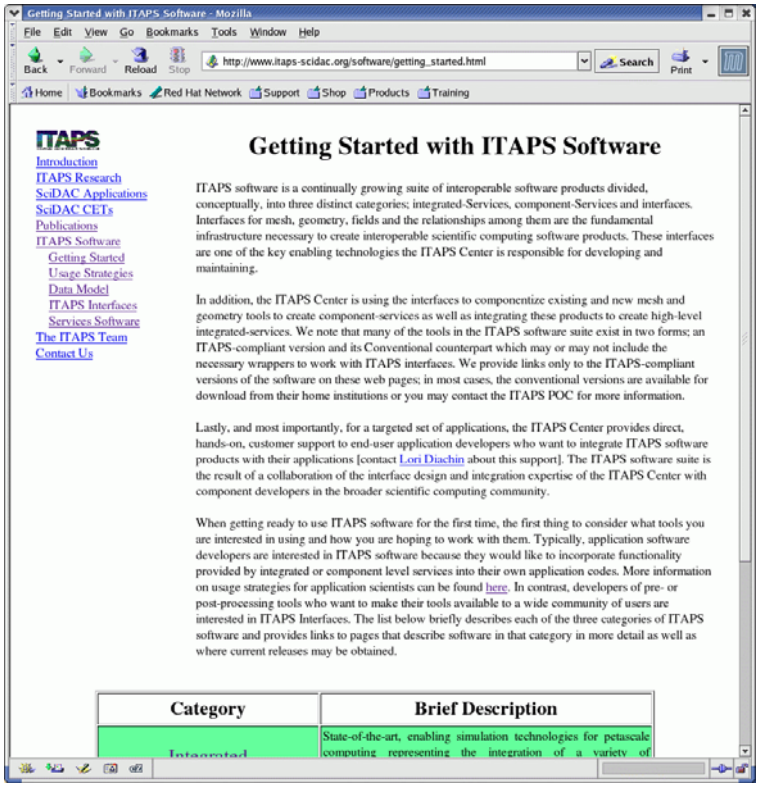
PART 6: ITAPS Software




ITAPS Software Web Pages

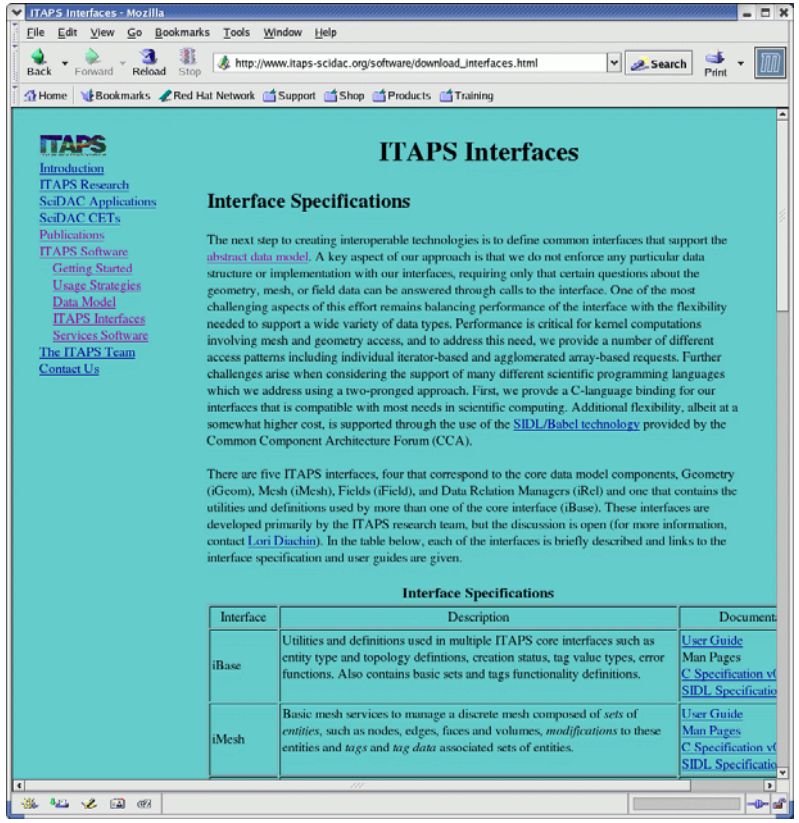
http://www.itaps- scidac.org/software/

- Provides help getting started
- Usage strategies
- Data model description
- Access to interface specifications, documentation, implementations
- Access to compatible services software



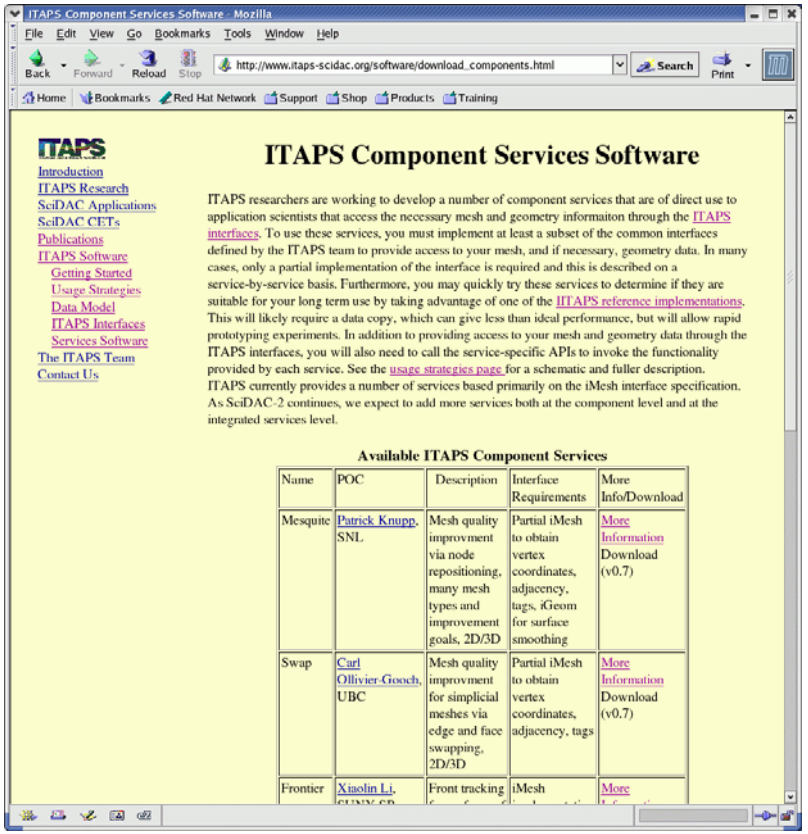
Interface Software Access

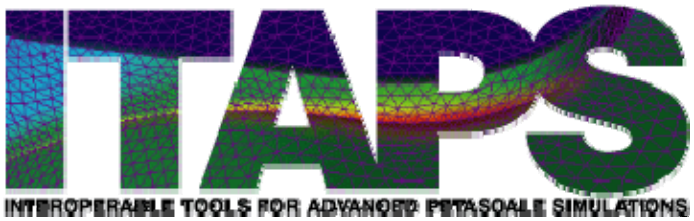
- Links to the interface user guides and man pages where available
- Links to the C-binding and SIDL-binding files
- Links to implementations for iMesh, iGeom, iRel
  - Version 0.7 compatible software
  - Links to the home pages for more information
- Simple examples, compliance testing tools and build skeletons coming soon



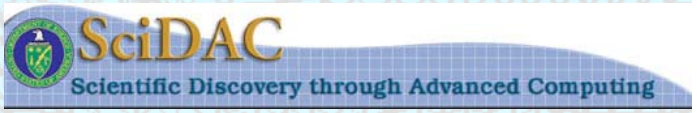
Services Software Access

- Links to the services built on the ITAPS interfaces
- Currently or very soon to be available
  - Mesquite (C, SIDL)
  - Zoltan (C, SIDL)
  - Swapping (SIDL)
  - Frontier (SIDL)
  - VisIt Plug In (C, SIDL)
- Links to home pages for more information
- Instructions for build and links to supporting software





Future Directions



Terascale computing support is a near term consideration

- Requirements document is nearly complete
- Interfaces will build on existing, supported functionality
  - In the data model (e.g., entity sets)
  - In the underlying TSTT tools (e.g., FMDB, MOAB)
    - Supports distributed mesh representation
    - Available services
      - Invariant entity handles (unique and unchanged even with mesh migration and mesh modification in parallel)
      - Mesh migration: global or local
  - In existing services (e.g., Zoltan)
- Requires additional interface work
  - Local/Global map conventions and behavior
  - Convenience functions for ghost node support, etc.
  - Mechanisms for mesh and user-data transfer

Requirements document in place for parallel interfaces

- Targeting Distributed Memory (MIMD) programming models while not preventing use of shared-memory/global address space models
- Will define the concept of a ‘partition’ in the ITAPS interfaces to describe
  - distribution of data across processors
  - Basic information about partition assignment of entities
  - Interprocessor communication patterns
  - Ghost entities
  - Boundary entities
  - Global Mesh Characteristics
- Will support reading and writing of parallel files from arbitrary numbers of processors
- Provide initial distribution of data to processors
- Will make interface calls asynchronous whenever possible

Future plans

- Distribution/repository of implementations / services (GForge)
- PR, tutorials, etc., to gain mindshare
- Follow-on activities to build on current success:
  - Fields interface
  - Solution transfer
  - Fine-grained AMR service
  - Parallel mesh/data migration
  - Shape optimization services
  - Hybrid mesh representation tools
  - Lots of application-focused efforts

Acknowledgments

We thank all those who have contributed to the ITAPS interface definition effort and software!

- **ANL: Tim Tautges**
- **LLNL: Lori Diachin, Mark Miller, Kyle Chand, Tammy Dahlgren**
- **PNNL: Harold Trease**
- **RPI: Mark Shephard, Ken Jansen, Eunyoung Seole, Andy Bauer, Xiaojnan Luo**
- **SNL: Vitus Leung, Karen Devine**
- **SUNY SB: Xiaolin Li, Brian Fix, Ryan Kaufman**
- **UBC: Carl Ollivier-Gooch**
- **U Wisconsin: Jason Kraftcheck**

Contact Information

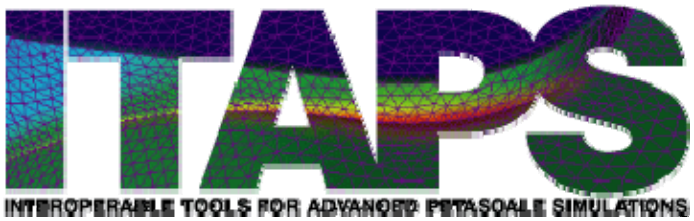
- ITAPS Web Page: http://www.itaps-scidac.org
- ITAPS Software Page: http://www.itaps-scidac.org/software
- Email: itaps-mgnt@llnl.gov
- Tutorial Presenters:

Lori Diachin, LLNL
diachin2@llnl.gov

Mark Shephard, RPI
shephard@scorec.rpi.edu

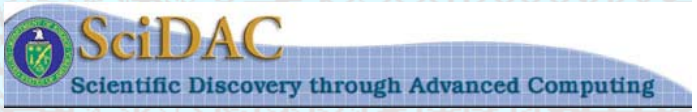
Tim Tautges, ANL
tautges@mcs.anl.gov

EXTRA SLIDES



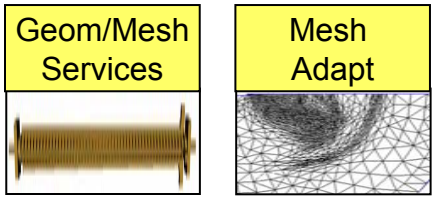
Element Curving Tool for Higher Order Finite Elements

SCOREC, RPI
SLAC



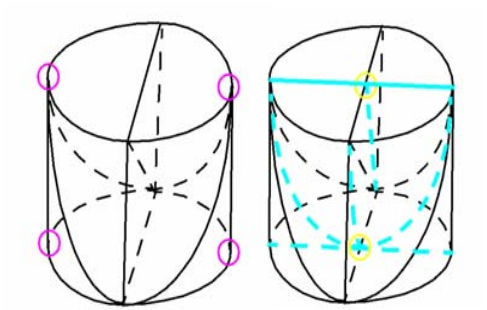

Goals and People Involved

- Build a stand alone tool to construct valid curvilinear meshes for the higher order finite element method applied in SLAC for accelerator design
- Xiaojuan Luo (RPI)
  Mark S. Shephard (RPI)
  Lie-Quan Lee (SLAC)
- ITAPS Services Used:

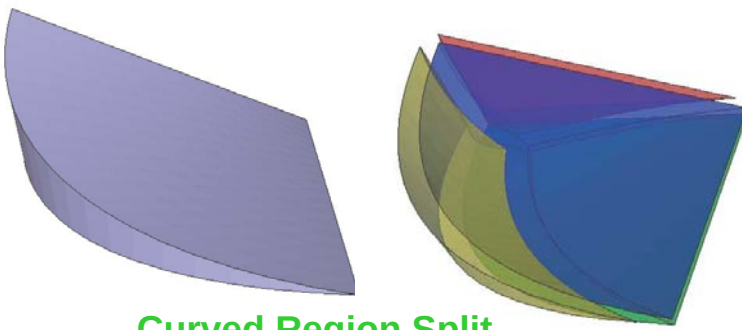


Technology Developments

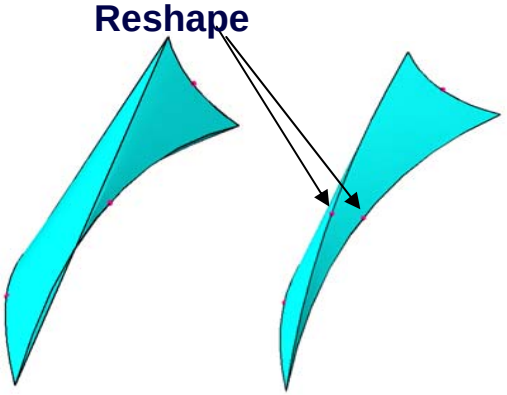
- Bezier higher order mesh shape representation
  - Analytical validity determination to ensure validity of all points in the element closure
  - Determine key mesh entity causing invalidity
- Apply curved local mesh modifications to correct invalid mesh entities
  - Reshape, split, collapse, swap and refinement



Curved Edge Split to Fix Model Tangency

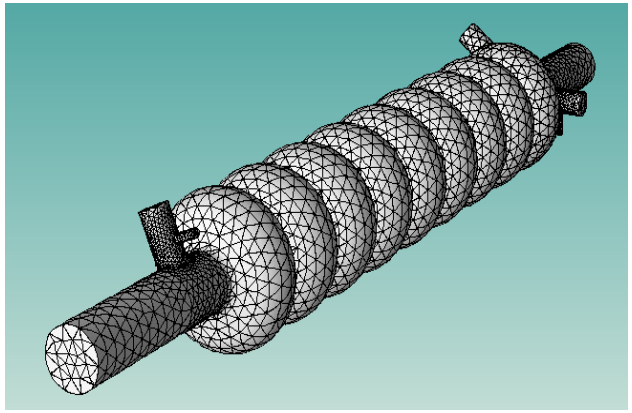


Curved Region Split



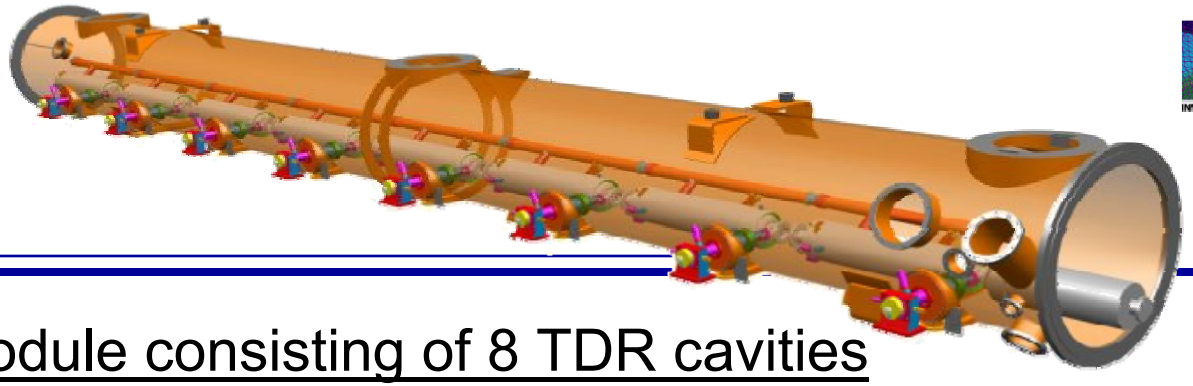
Results

- Initial mesh
  - 108k mesh regions
  - 250 invalid regions
  - Solution blows-up unless negative contribution removed
- Corrected mesh
  - No invalid regions
  - Solution process 37.8% faster (CG iterations per time step reduced)



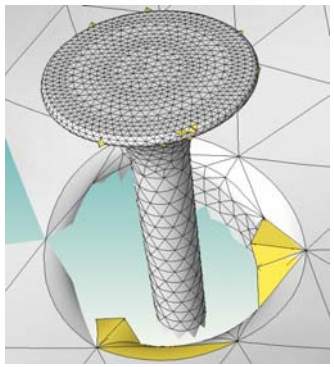
Valid curved mesh after operations
Electromagnetic analysis for SLAC

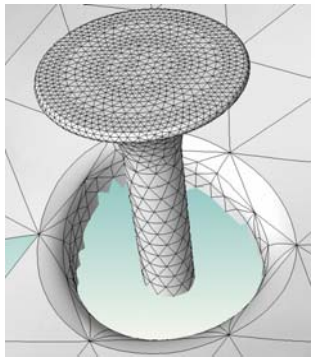
Results

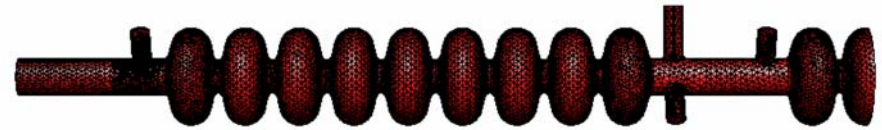


ILC cryomodule consisting of 8 TDR cavities

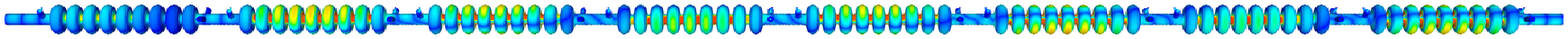
2.97M regions by correcting 1,357 invalid curved regions







Mesh before and after correcting the invalid regions marked as yellow



A dipole mode from Omega3P

~22 hours on 1024 CPUs on Seaborg at NERSC for 16 modes with about 20 million DOFs