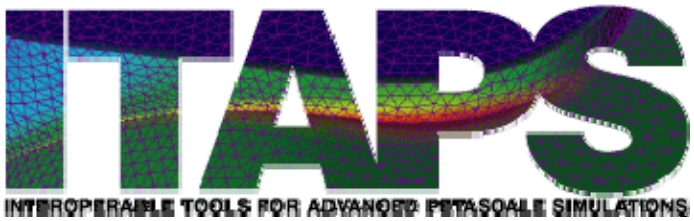
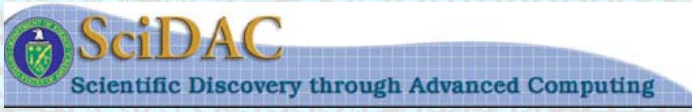


Interoperable Tools for Advanced Petascale Simulations (ITAPS)

TUTORIAL

November 2008





UCRL-XXXX-XXXXX

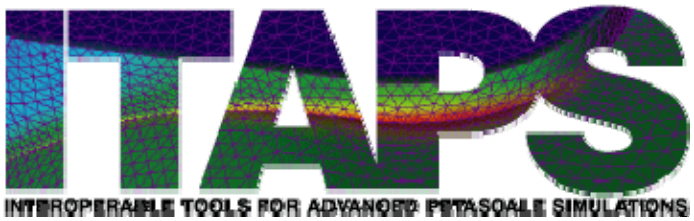
Tutorial Overview

Morning

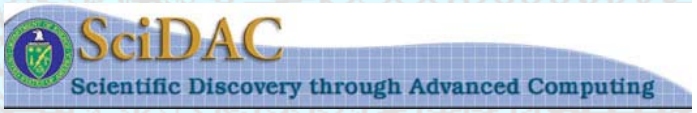
- Part 1: Introduction
- Part 2: Overview of ITAPS services
- Part 3: The ITAPS Data Model
- Part 4a: Basic ITAPS Interfaces
- *Hands on exercise*:
  - Hello iMeshP

Afternoon

- Part 4b: Advanced ITAPS Interfaces
- Part 5: ITAPS Software
- Part 6: Experiences and use in applications
- Part 7: Conclusion
- *Hands on exercises*:
  - Accessing mesh information
  - Partitioning services through iMeshP



PART 1: Introduction

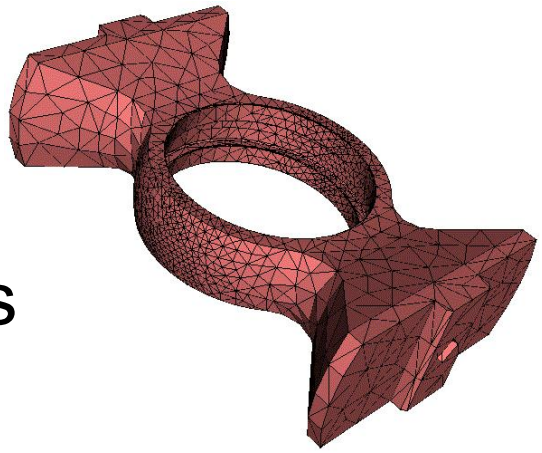




Unstructured meshes

- An unstructured mesh
  - A piece-wise space/time domain decomposition over which the simulation is to be run
  - General topology-based mesh representation consists of
    - 0-3 D topological entities (vertices, edges, faces and regions)
    - Connectivity between entities called *adjacencies*
  - Mesh data structure provide services to create and/or use the mesh data

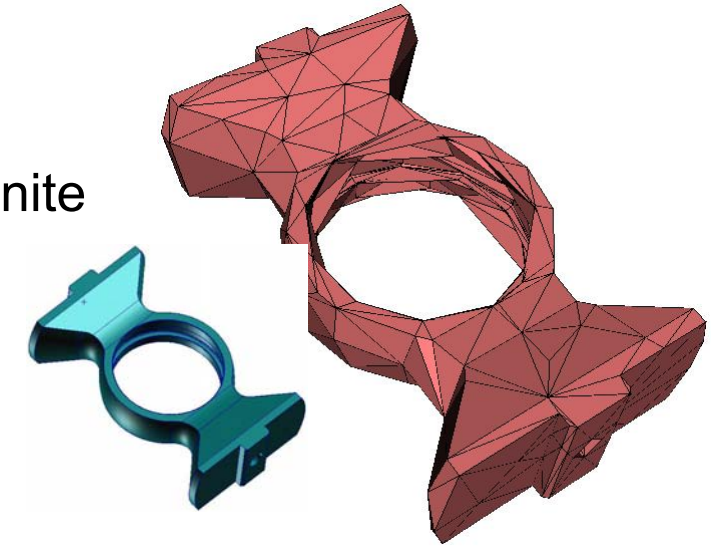
Geometric domain

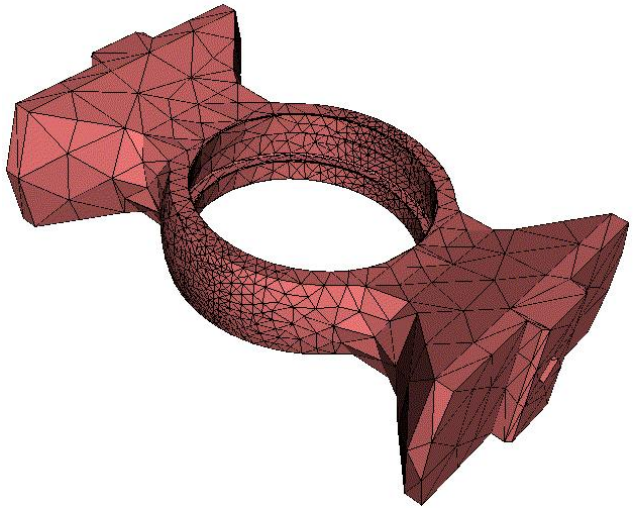


Mesh

Unstructured mesh methods offer key advantages for numerical simulation

- 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

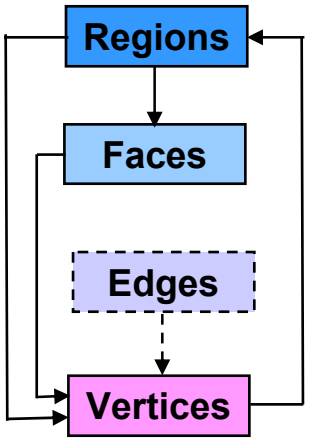


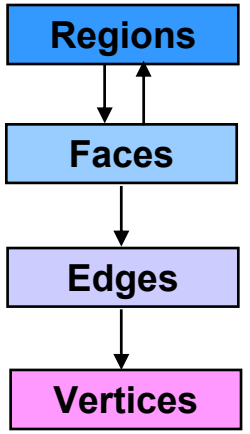


SciDAC
Scientific Discovery through Advanced Computing

Adjacency information is critical in managing unstructured meshes

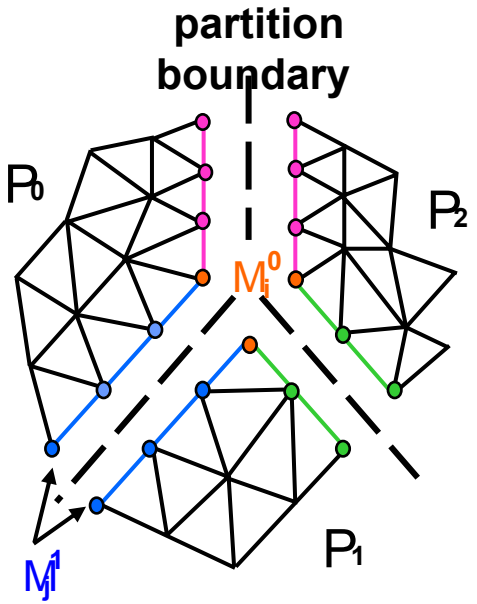
- Each particular mesh application has its own needs of mesh representation in terms of levels of entities and adjacencies used
- Approaches mesh data structure design
  - Fixed, specific mesh representation
  - Fixed, general mesh representation
  - Flexible mesh representation
- Different unstructured mesh databases will provide different entities and adjacencies





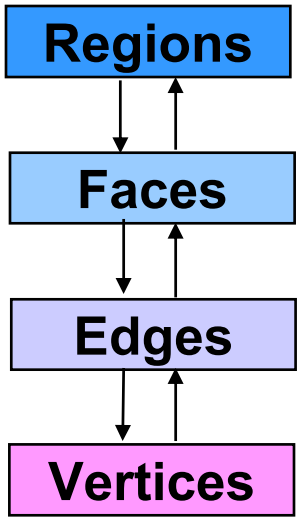
Unstructured Meshes on Parallel Computers

- Typically the mesh is distributed over independent memories
- A mesh partition groups mesh entities and places them into parts
- Applications using a partitioned mesh need
  - Communication links for between "shared" mesh entities on neighboring parts
  - Ability to move mesh entities between parts (while maintaining links)
  - Algorithms to maintain load balance of parts which minimizing communications

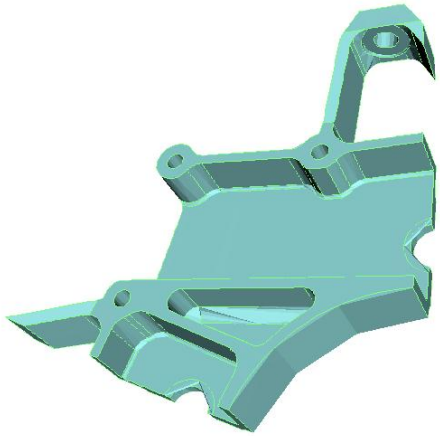


Unstructured Meshes on Parallel Computers

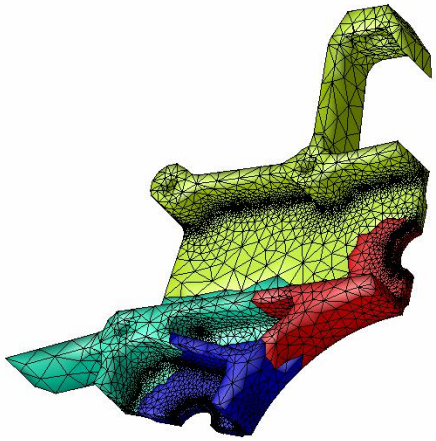
- Communication links for distributed meshes need to provide information associated with mesh entities and adjacencies
- Extending concepts of mesh entity topology and entity adjacency to parallel can address these communication needs
- Will describe the basics through the conceptual construct of a partition model



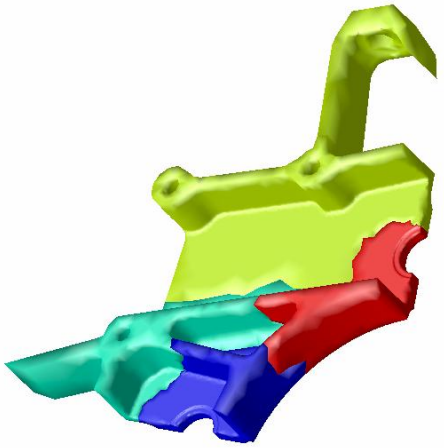
Conceptual view of an unstructured mesh partition model



Domain of Interest



Mesh distributed over four parts

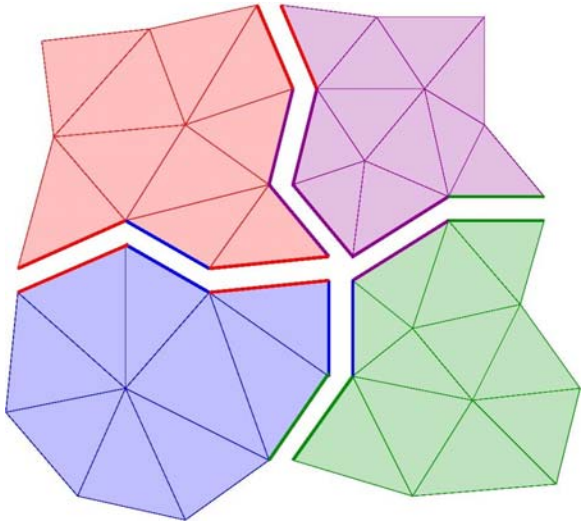


Partition model focuses on mesh entities between parts

- Mesh adjacency information can provide
  - Entities on part boundaries to define inter-part communications shared entities
  - Adjacencies to entities bounded by those entities (for ghosting, etc.)

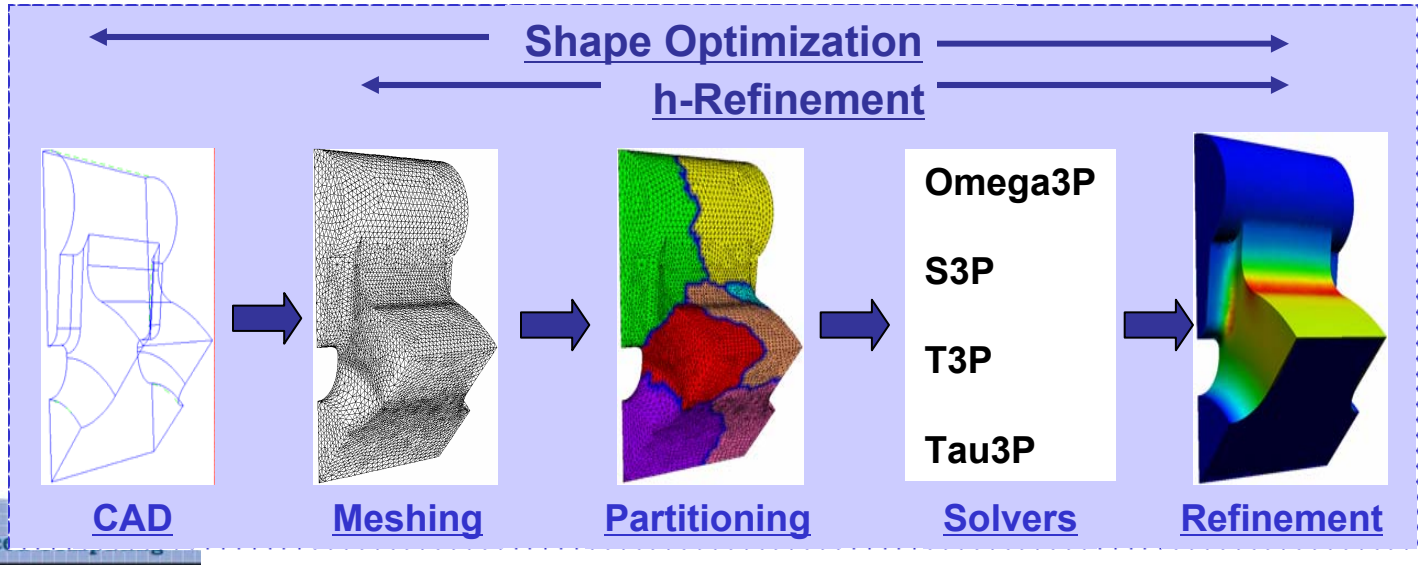
Distributed mesh representations have several functional requirements

- Entity ownership
  - Each mesh entity is owned by exactly one part
  - Ownership imbues right to modify
  - Ownership is not static during the course of a simulation
    - Repartitioning
    - Local micro-migration
  - Some entities have read-only copies on other parts (e.g. along part boundaries and ghosts)
- Communication links
  - Efficient mechanisms to update mesh partitioning and keep the links between parts are mandatory



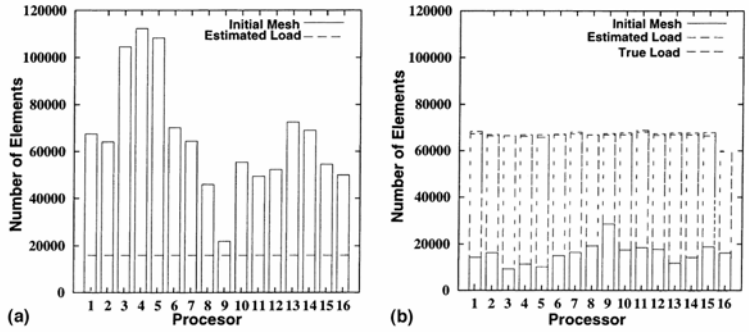
Basic parallel solution on unstructured meshes has several key steps

Construct the initial mesh (serial or parallel)
Improve the mesh using **smoothing** and **swapping**
If necessary, **(re)partition** the mesh across processors
Solve the PDE on mesh and estimate the error
While error > tolerance
  Refine, coarsen, improve and repartition the mesh
  Solve the PDE on the mesh and estimate the error
End

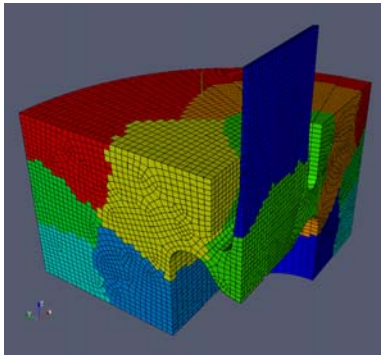


Key issues that must be addressed for parallel computation

- Scalability
  - Load balance as the mesh changes
  - Low communication overhead costs
  - Efficient use of distributed memory

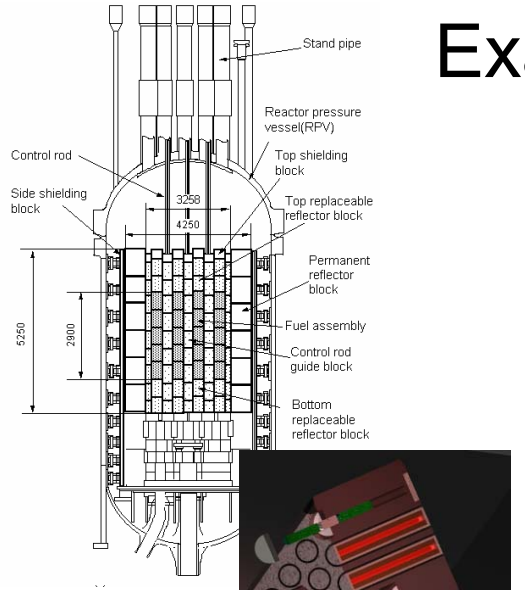


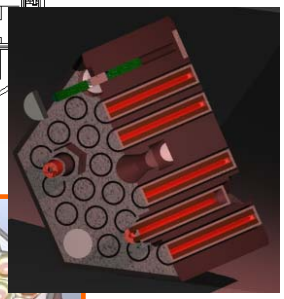
- Function
  - Consistent, correct mesh operations
  - Management of complex communication schedules

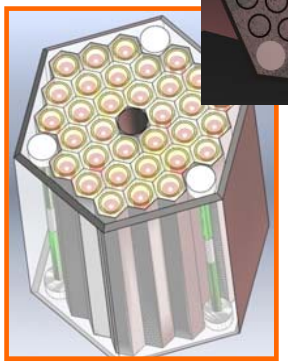


- Performance
  - Near optimal serial efficiency on each processor
  - Minimal overhead when using general tools relative to native implementations

Parallel solution is further complicated by the needs of advanced simulations

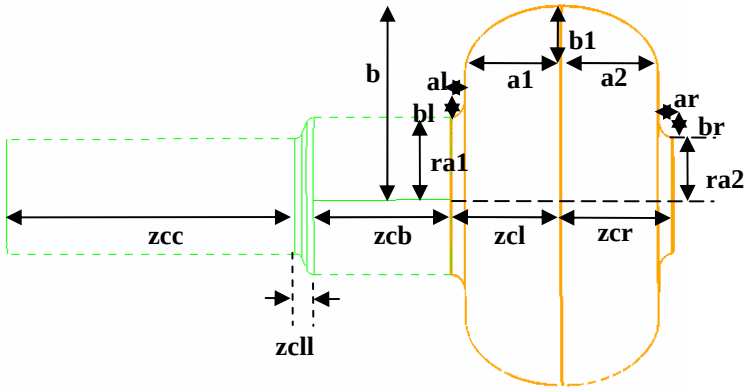


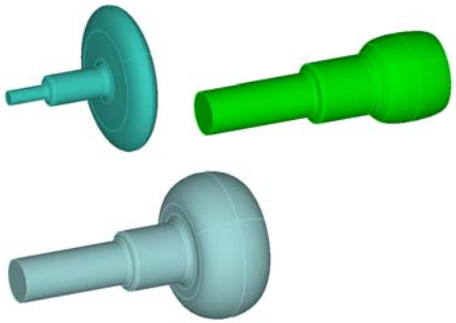




Examples:

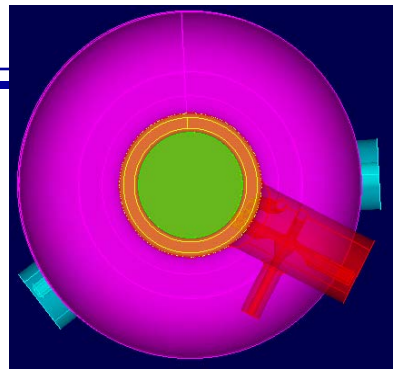
- Design optimization requires geometry modification, remeshing, derivative computations
- Multi-physics applications require mesh to mesh transfer, interpolation methods, sophisticated adaptive methods



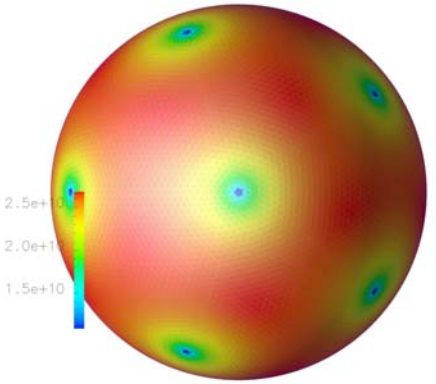


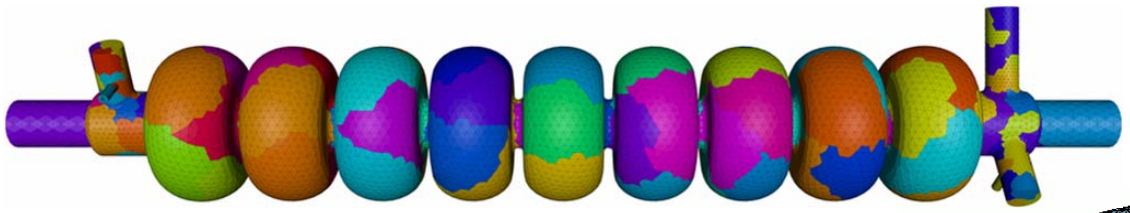
SciDAC
Scientific Discovery through Advanced Computing

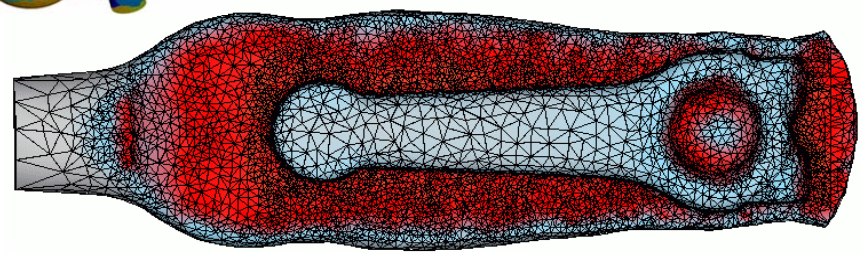
The ITAPS team has developed tools to address these needs



- CAD interaction: CGM
- Mesh generation: GRUMMP, NWGrid
- Mesh databases: FMDB, MOAB
- Mesh improvement: Mesquite, swapping tools
- Parallel Adaptive loops: FMDB, NWGrid
- Front tracking: Frontier
- Partitioning: Zoltan







While these tools exist, significant challenges remain

Developing and using these technologies requires significant software expertise from application scientists

- Difficult to improve existing codes
- Difficult to design and implement new codes

These tools all meet particular needs, but

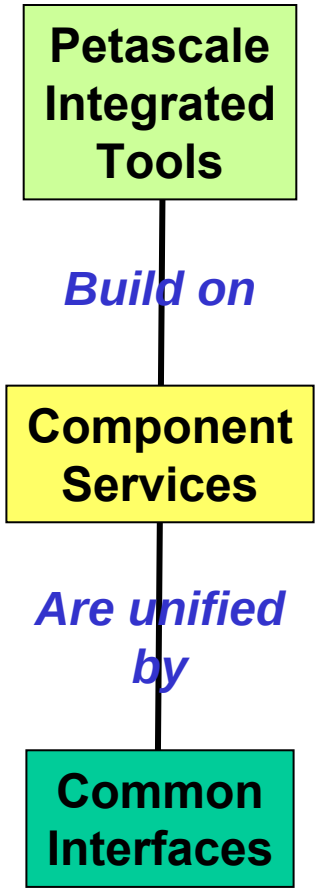
- They do not interoperate to form high level services
- They cannot be easily interchanged in an application

The ITAPS center is developing key technologies to ease the use of advanced meshing tools on large scale parallel computers

SciDAC Scientific Discovery through Advanced Computing

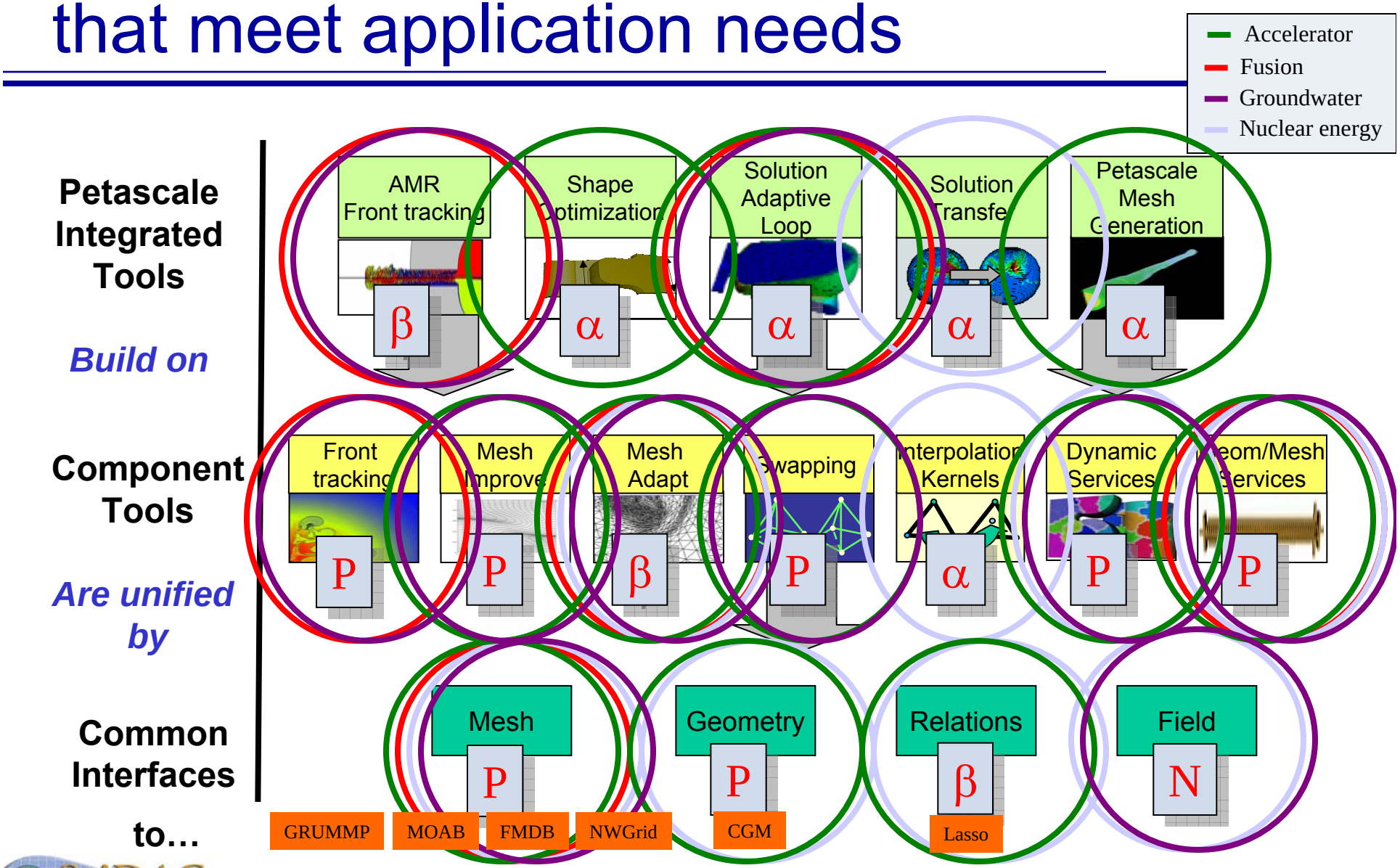
ITAPS uses a component-based approach to address these challenges

- 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* and *data model* to enable interoperability
  - Interfaces are implemented on top of existing mesh databases
- Work with key application teams to insert ITAPS technologies into simulations



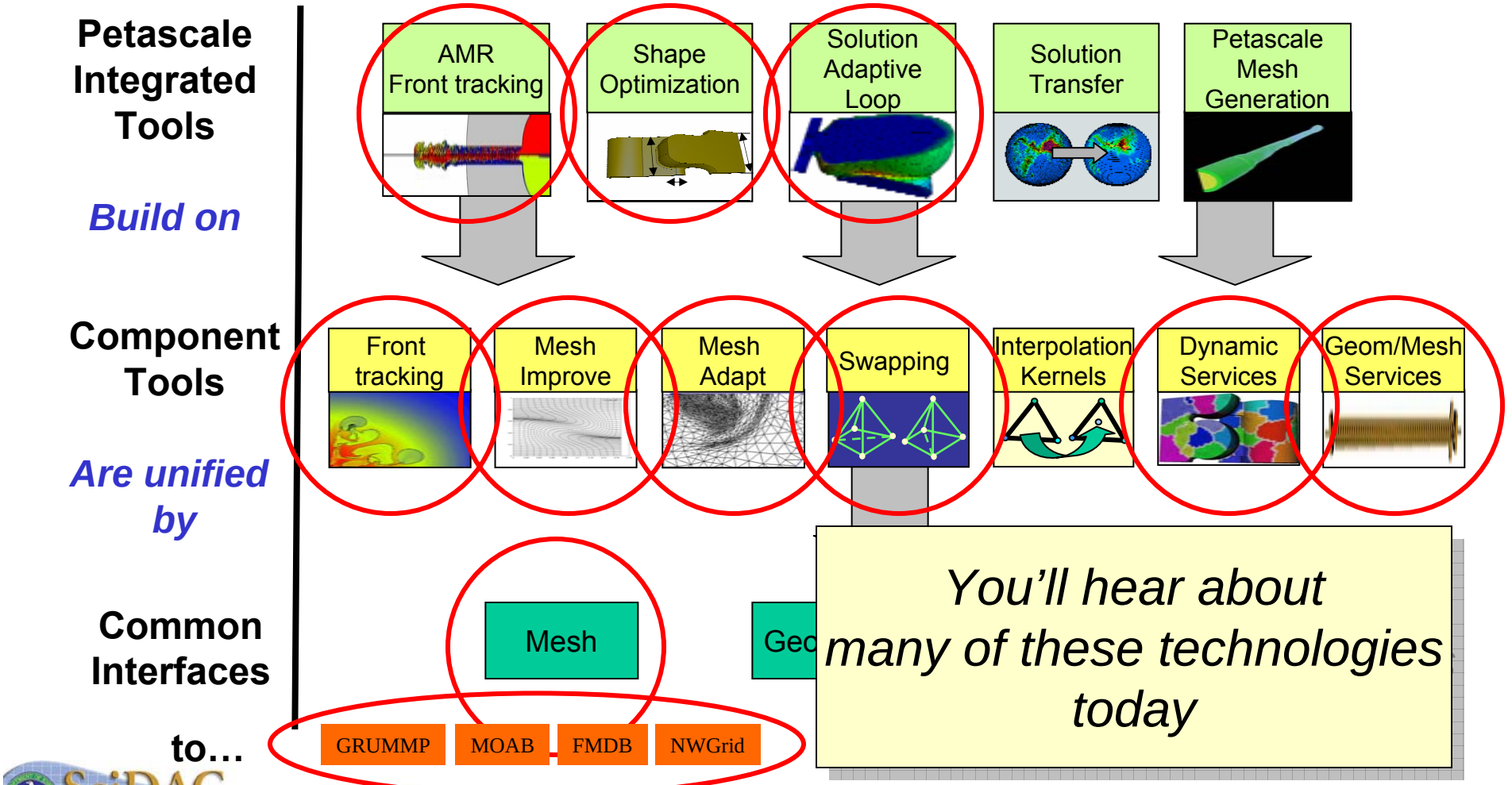
SciDAC
Scientific Discovery through Advanced Computing

ITAPS produces mesh services that meet application needs



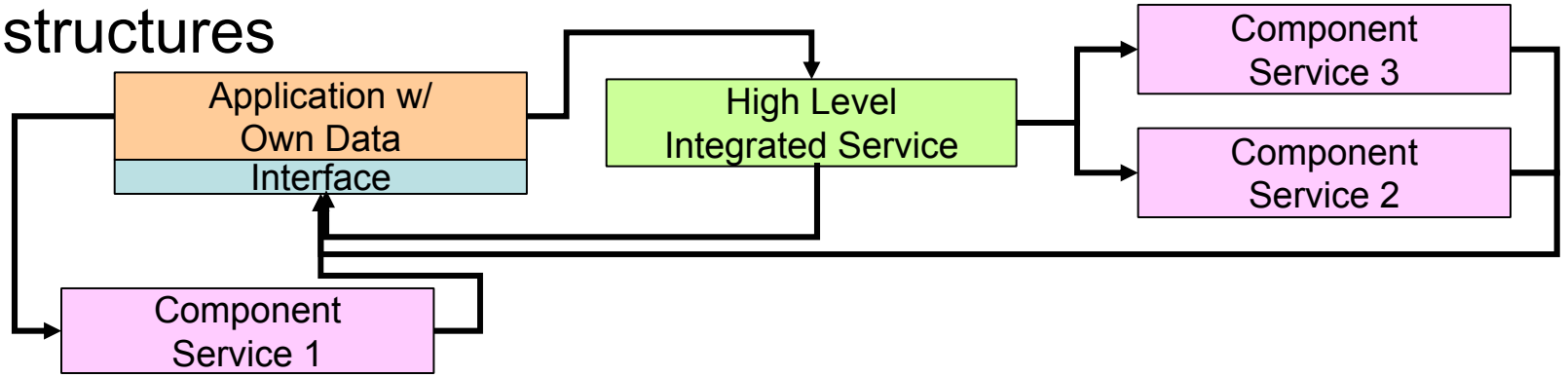
SciDAC
Scientific Discovery through Advanced Computing

ITAPS is developing meshing services available as stand-alone components

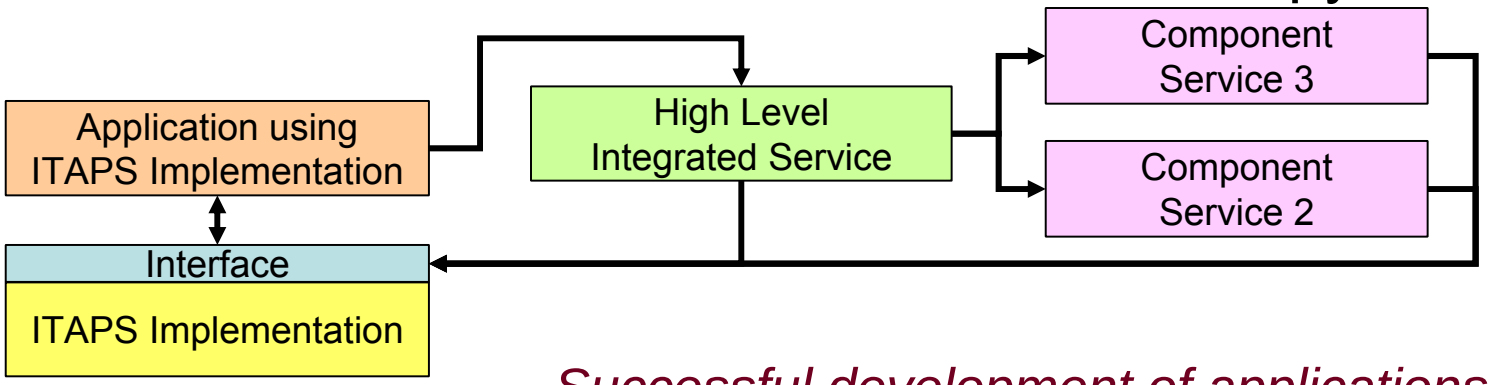


Applications can access ITAPS services in two ways

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 has been accelerated by close collaboration.

There are several advantages of the component-based approach ITAPS uses

- Focus on interfaces not on data structures or file formats
- Use independent interfaces for distinct data model abstractions to make adoption easier
- Incremental adoption for applications; only service dependent interfaces need to implemented
- Finer granularity of interoperable functionality reduces the need to mix huge libraries together

ITAPS

INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

PART 2: An Overview of ITAPS Services

SciDAC

Scientific Discovery through Advanced Computing

Sandia National Laboratories

Rensselaer

STONY BROOK

THE UNIVERSITY OF BRITISH COLUMBIA

BROOKHAVEN NATIONAL LABORATORY

OAK RIDGE NATIONAL LABORATORY

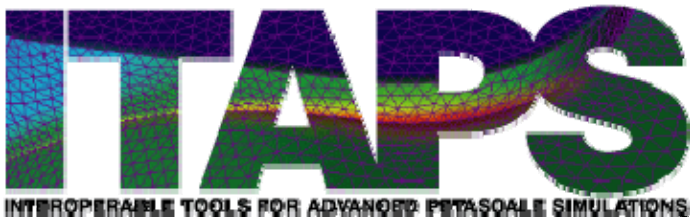
Pacific Northwest National Laboratory

Interoperable services speed the development of simulation technologies

- ITAPS provides stand-alone services as libraries
- Improve applications' ability to leverage advanced tools
  - **Mesh quality improvement**
  - **Mesh adaptation loops**
  - **Front tracking**
  - **Mesh partitioning**
  - **Visualization**

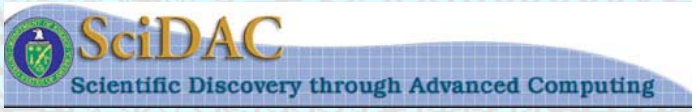
We'll provide a brief overview of each of these tools; more information can be found on www.itaps-scidac.org

SciDAC Scientific Discovery through Advanced Computing



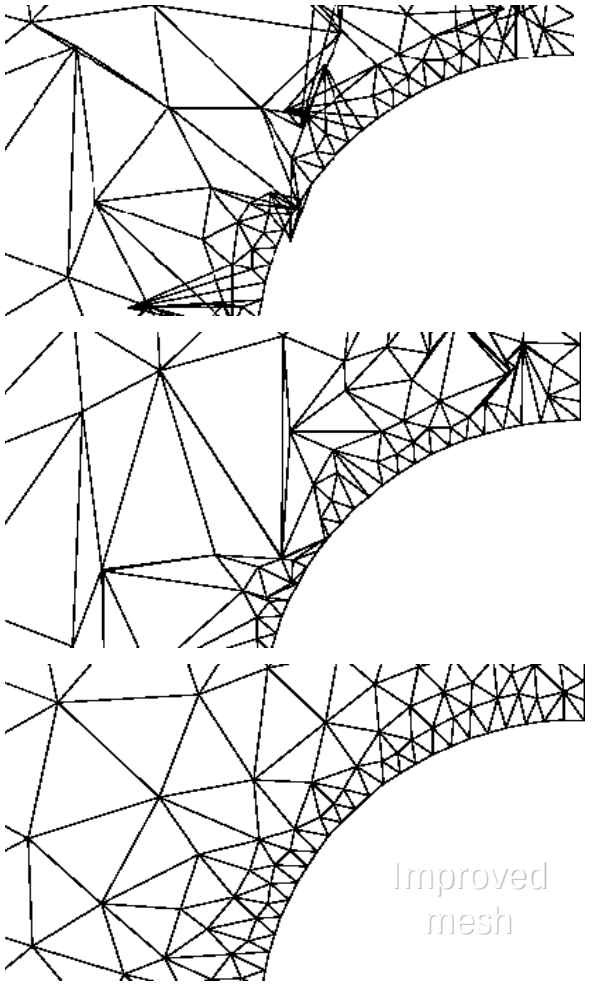
Mesh Quality Improvement

Lori Diachin (LLNL)
Patrick Knupp (SNL)
Carl Olivier-Gooch (UBC)



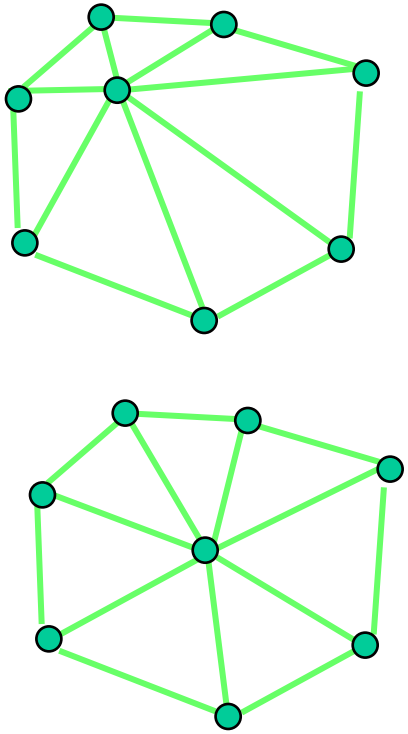

Unstructured mesh quality is a critical factor in solution efficiency and accuracy

- In general, mesh size and quality affects
  - **Solution efficiency** (e.g. Axelsson, 1976; Fried 1972; Axelsson and Barker, 1984)
    - Iterations grow as a function of N
    - Iterations grow as a function of minimum angle
  - **Solution accuracy**
    - Solution from iterative solver is less accurate
    - For isotropic fields, discretization error adversely affected by distorted elements (e.g. Babuska and Aziz, 1976)
- Understanding application solution characteristics is critical
  - stretched elements are more accurate than equilateral elements for boundary layer flow



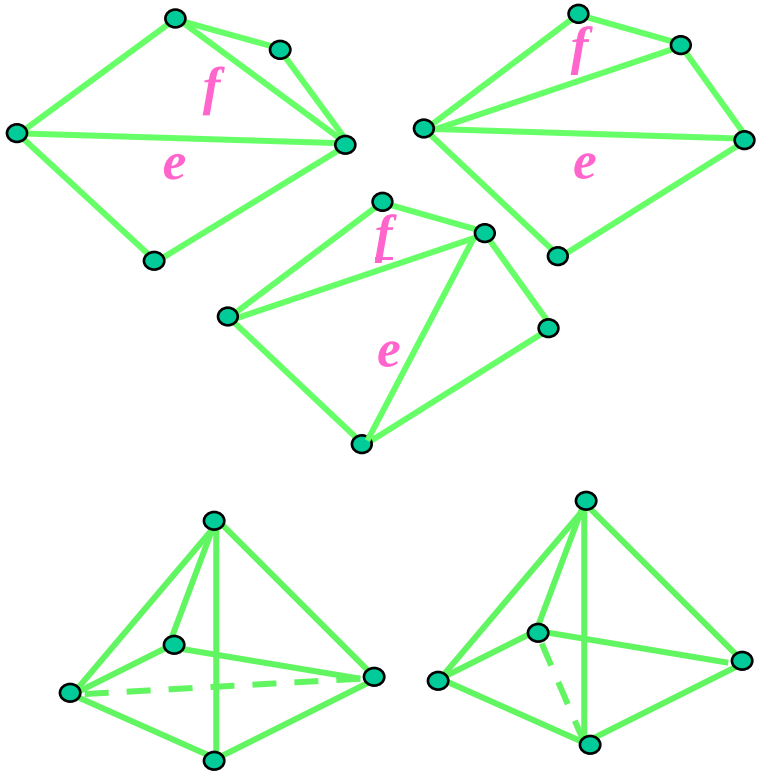
Mesh quality can be improved using a variety of methods

Node Movement



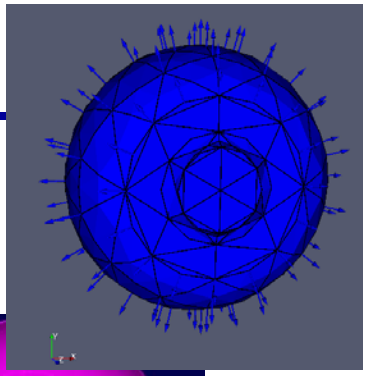
Moving grid points without changing mesh topology

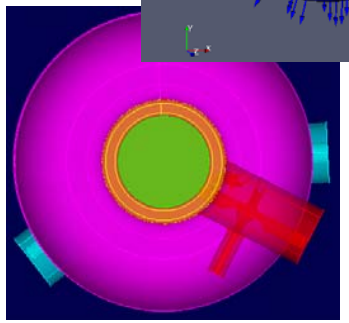
Edge or Face Flipping

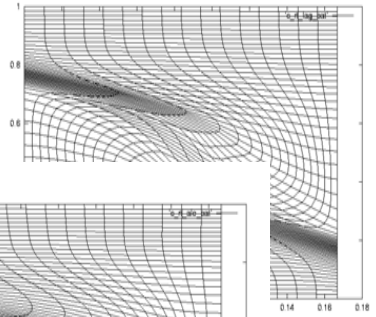


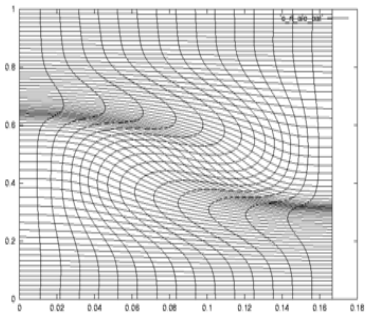
Modify topology without changing grid point location

Mesquite provides advanced mesh smoothing capabilities





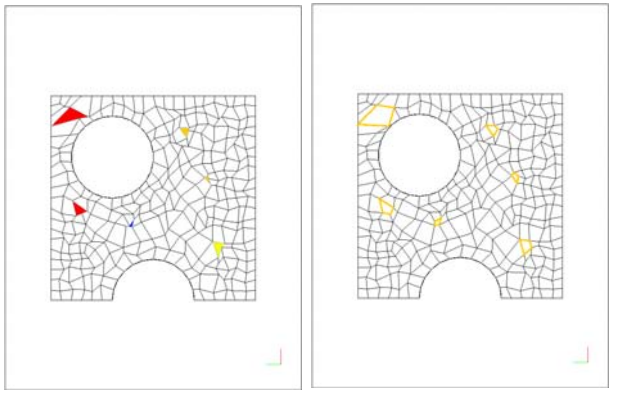


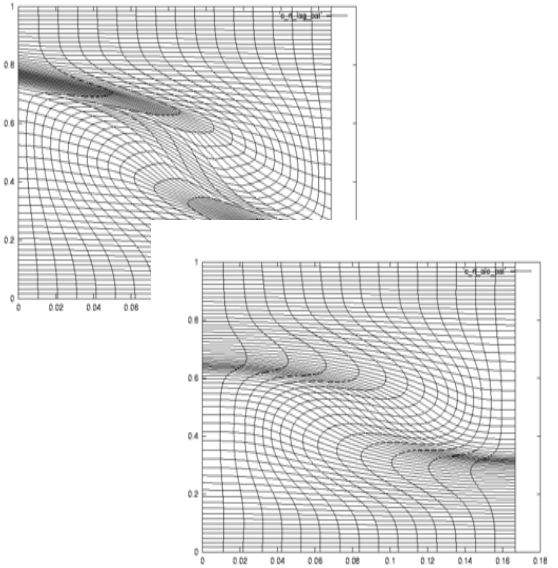


- Mesquite is a comprehensive, stand-alone library for mesh quality improvement with the following capabilities
  - Shape Quality Improvement
  - Mesh Untangling
  - Alignment with Scalar or Vector Fields
  - R-type adaptivity to solution features or error estimates
    - Maintain quality of deforming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

Mesquite is versatile and comprehensive

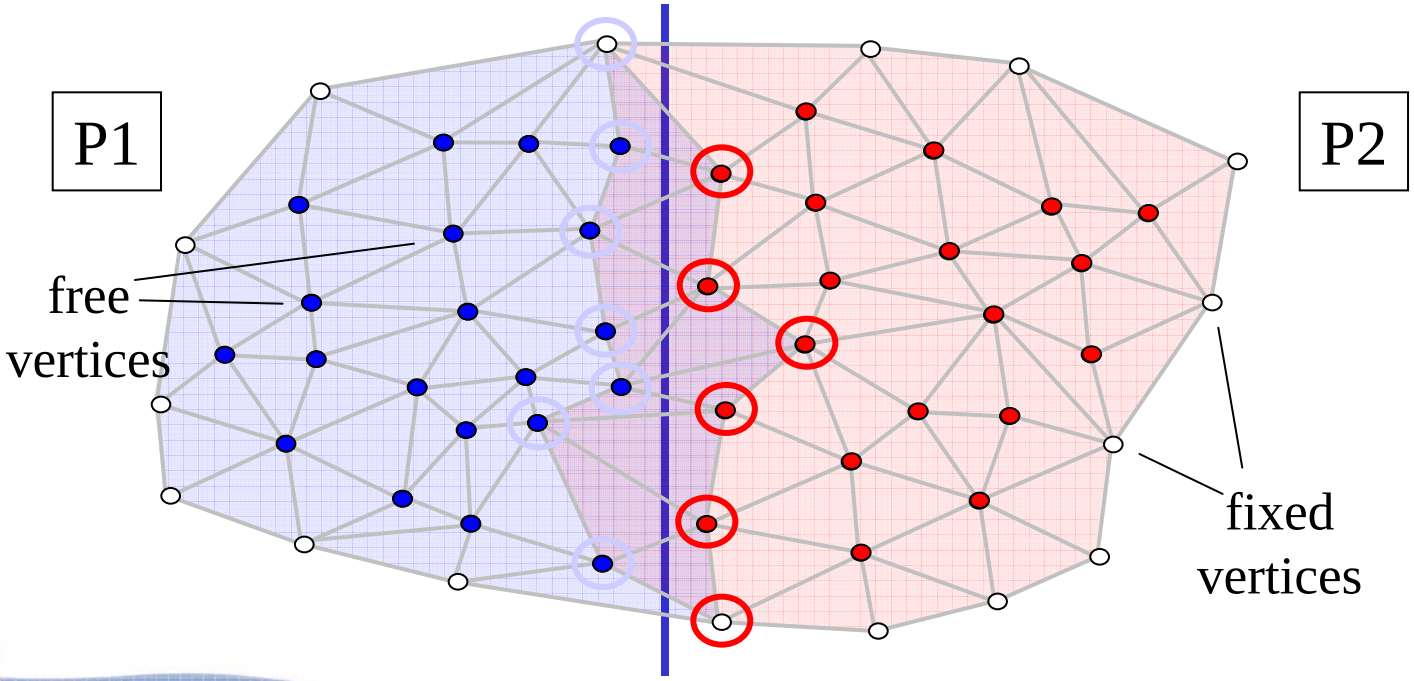
- 2/3D Structured, Unstructured, Hybrid Meshes
- Many different element types
- State-of-the-art algorithms and metrics
- Efficient to run
- Customizable
  - User-defined metrics, objective functions, and algorithms



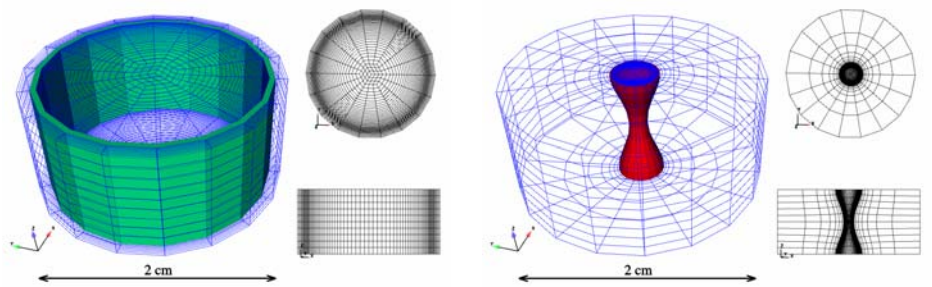


Parallel smoothing requires special attention at partition boundaries

Incorrect execution of the smoothing algorithm can occur unless partition boundaries maintain consistent updates of vertex locations; requires synchronization with partition neighbors



Our Mesh Quality Improvement Work has Impacted Many DOE Applications



Application: Plasma implosion using ALE methods

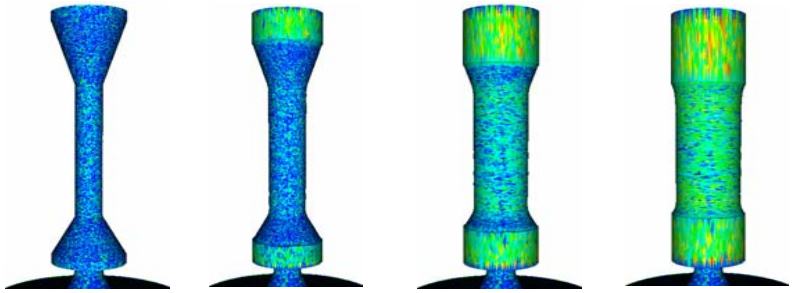
Challenge: Maintain good mesh quality and biasing during deformation of plasma.

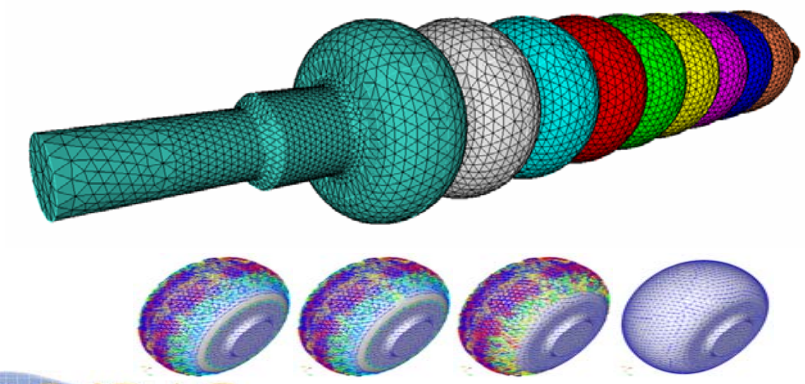
Impact: Prior to use of Mesquite, this calculation could not be performed by Alegra due to ineffective mesh rezoning algorithm.

Application: Burn of rocket propellants in a time-deforming domain

Challenge: Maintain good tetrahedral element shape quality as domain deforms

Impact: Condition number smoother (through ShapeImprovementWrapper) enabled many burn simulations at CSAR/UIUC.





Application: Shape optimization for accelerator cavities to minimize losses

Challenge: Rapidly and smoothly update the mesh to conform to trial geometries

Impact: Used the deforming mesh metric to prototype geometry & mesh update model for potential use in SLAC accelerator design studies.

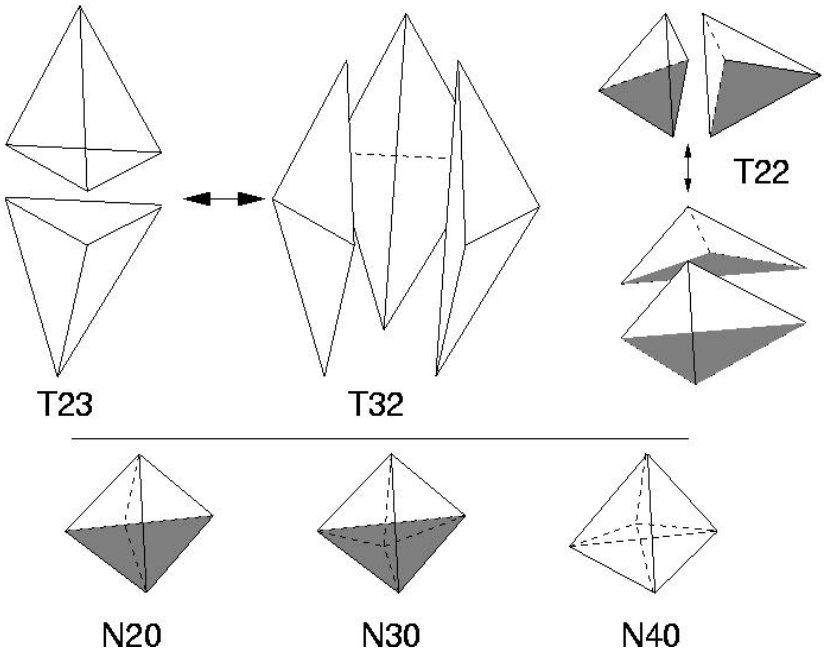
SciDAC
Scientific Discovery through Advanced Computing

Swapping complements node point movement in improving mesh quality

- Changing topology can eliminate poorly-shaped mesh entities directly
- Particularly effective in combination with node point movement (Freitag and Ollivier-Gooch, IJNME, 1997)
- Swapping decisions and (especially) topology change challenging to implement correctly
- Service using standard interface can handle all the difficult, error-prone aspects

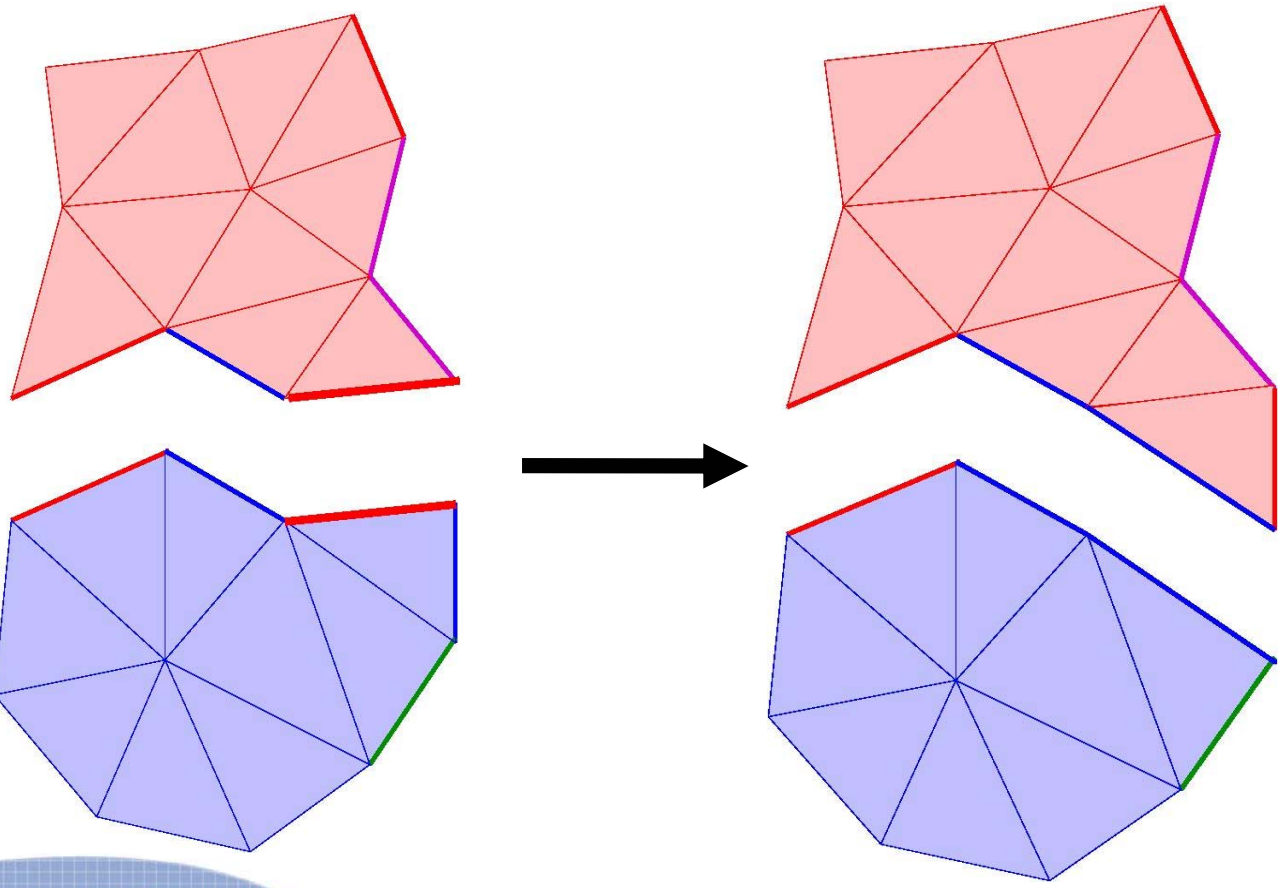
Mesh swapping capabilities are available using ITAPS

- Triangular/tetrahedral swapping with variety of criteria
- Single edge/face, mesh subset, or entire mesh
- Boundary modification optional in 3D
- Built in and user-defined swapping criteria provide both ease of use and flexibility

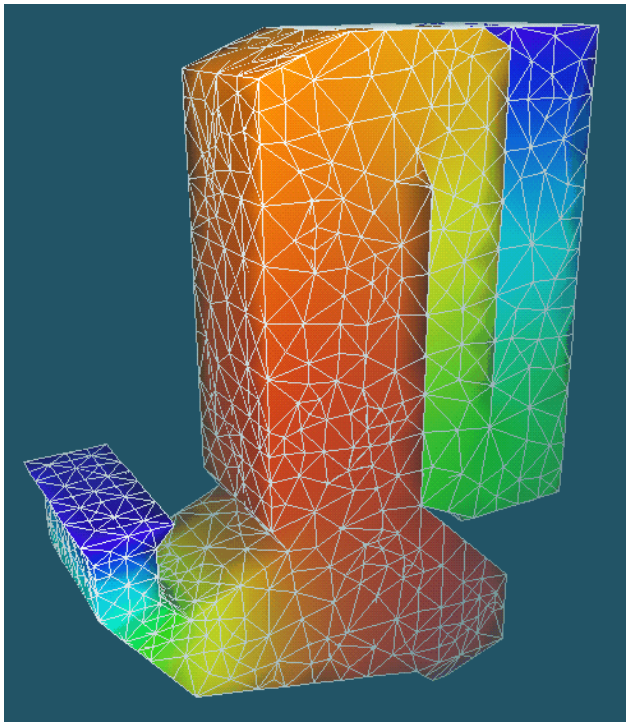


Issues with swapping in parallel

- Must migrate data onto single processor to modify
- Load balance and communication overhead issues



Smoothing and swapping improvement operations work well in concert

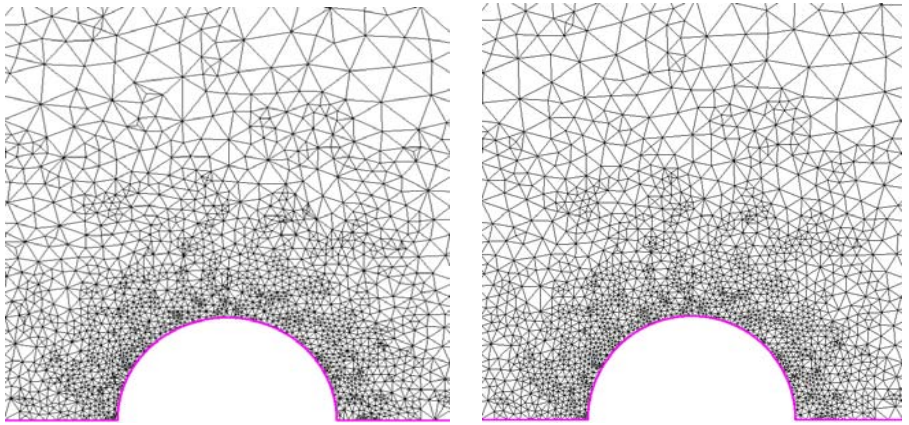


Compressible Flow Over a Cylinder Combining smoothing and swapping improved the convergence rate by 25% compared to smoothing alone

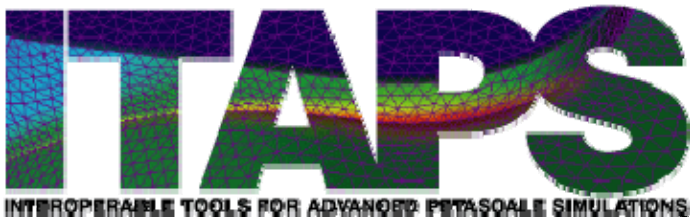
Overall mesh improvement cost less than one multigrid iteration

Original mesh:	Min: .56	Max: 178.86
Smoothing only:	Min: 12.3	Max 145.6
Both:	Min: 23.2	Max: 131.9

Swapping only:	Min: .66	Max: 178.9
Both:	Min: 4.34	Max: 174.3

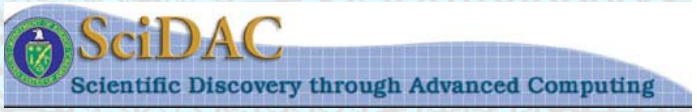


SciDAC Scientific Discovery through Advanced Computing



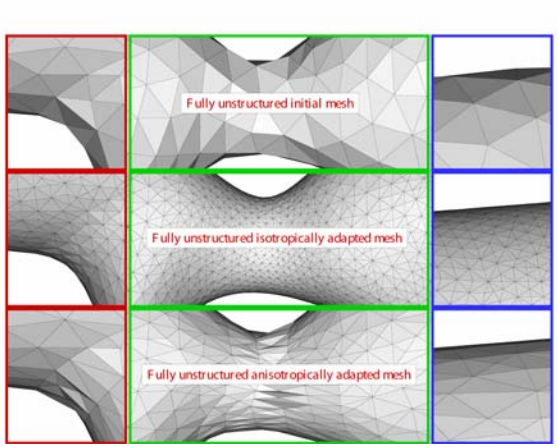
ITAPS Adaptive Mesh Service

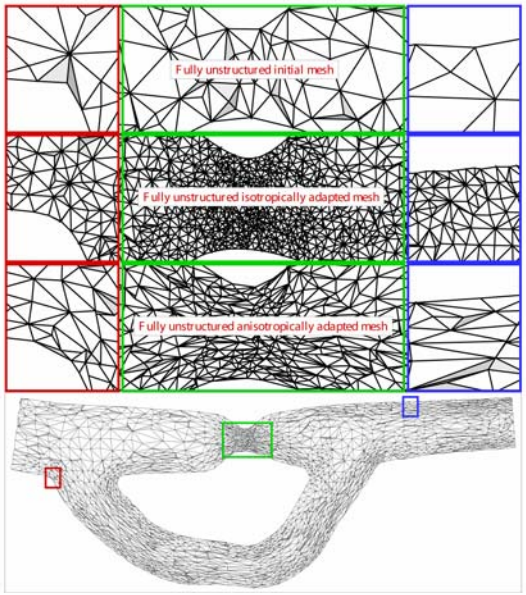
Mark Shephard (RPI)
Ken Jansen (RPI)
RPI SCOREC Team




Motivation

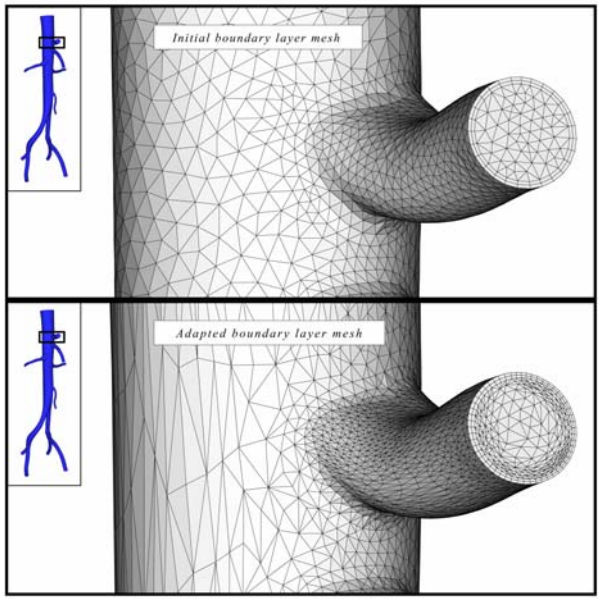
- Complex geometry and/or complex physics generate solutions with large variation in length scales
- Assigning near optimal mesh sizes for initial mesh generation in such cases is not possible
- ITAPS Mesh Adapt Service starts with an arbitrary initial mesh with a solution and alters the mesh via local mesh modifications (may be isotropic or anisotropic)
- CFD example:
  - Isotropic adaptivity yields same accuracy as uniform with one order of magnitude fewer elements
  - Anisotropic adaptivity yields same accuracy as uniform with two orders of magnitude less than uniform

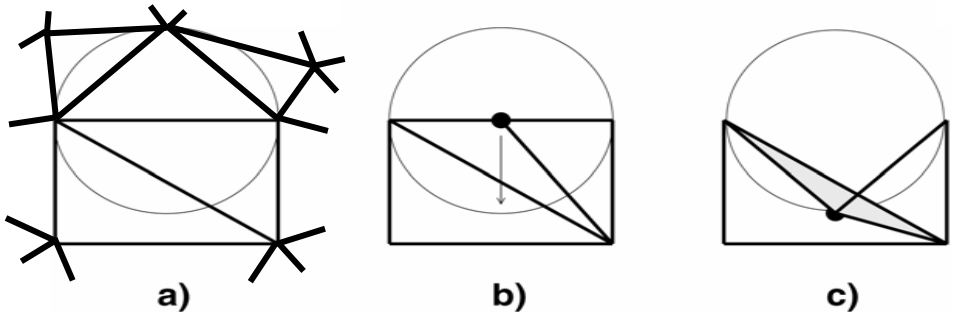




ITAPS Mesh Adapt Service

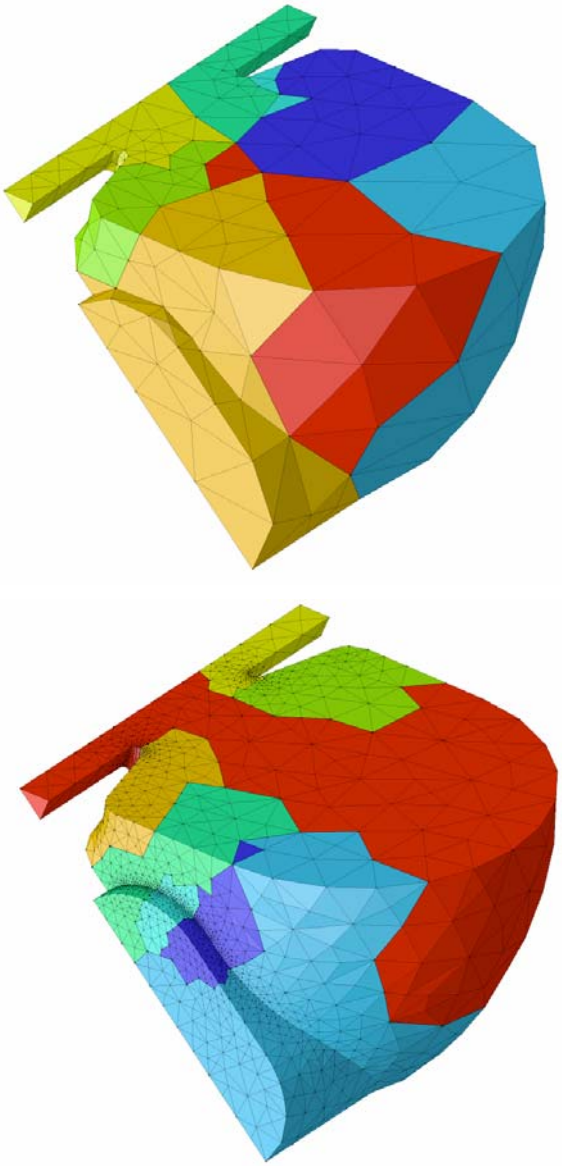
- Components of an adaptive analysis loop
  - PDE solver
  - Error estimation procedure
  - Mesh adaptation - ITAPS Mesh Adapt Service
  - Field solution transfer - ITAPS Mesh Adapt Service can support
- ITAPS Mesh Adapt Service
  - Input is a mesh and new mesh size field definition: isotropic or anisotropic
  - Mesh adaptation uses local mesh modification to support:
    - Curved Boundaries
    - Anisotropy
    - Parallel mesh adaptation





Parallel Mesh Adaptation

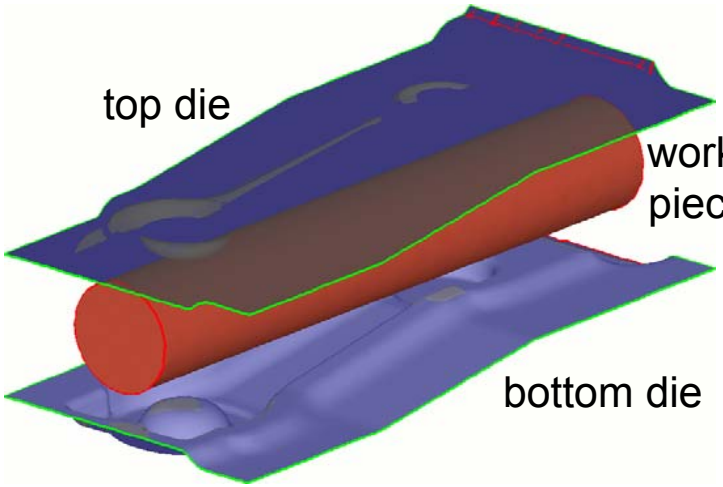
- Operates with a partitioned mesh
- Control parallel communication and operations - iMeshP
- Part boundaries do not introduce artificial constraints on execution of mesh modification operations
- By including dynamic load balancing (Zoltan), can be easily integrated with parallel solvers that also operate with partitioned meshes

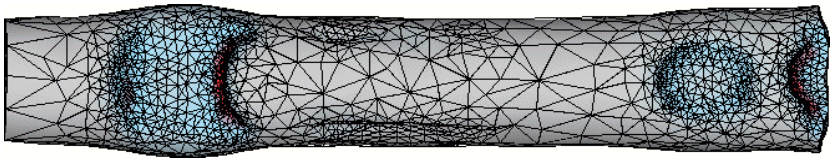


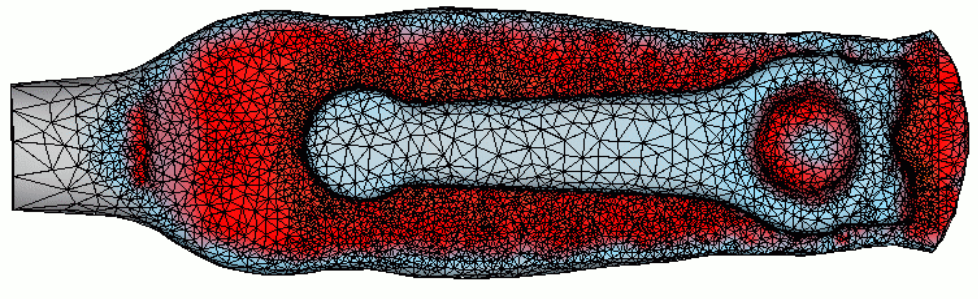
SciDAC
Scientific Discovery through Advanced Computing

Example Application: Forming Simulation

- Large plastic deformation - Meshes become invalid
- Components of automated adaptive simulation
  - Model topology update based on contact evolution
  - Mesh adapted using automation tools to account for
    - Discretization errors
    - Geometric approximation
    - Element shape control

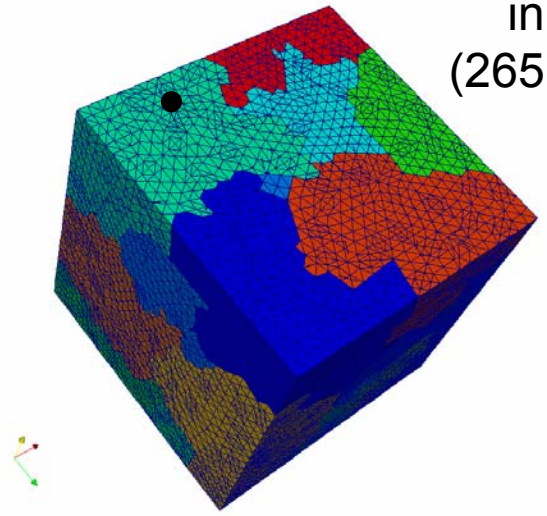




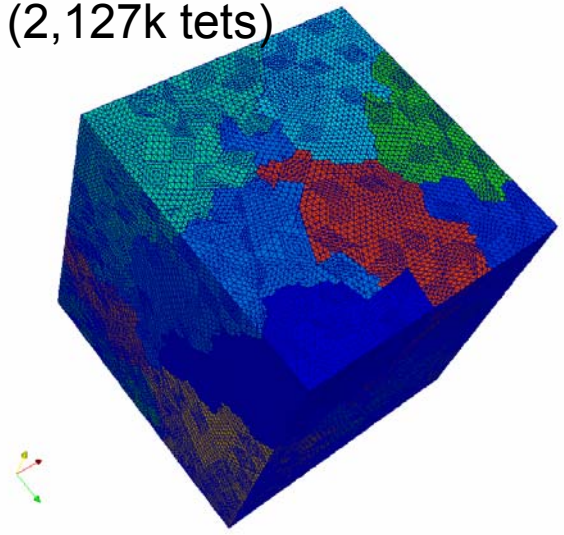


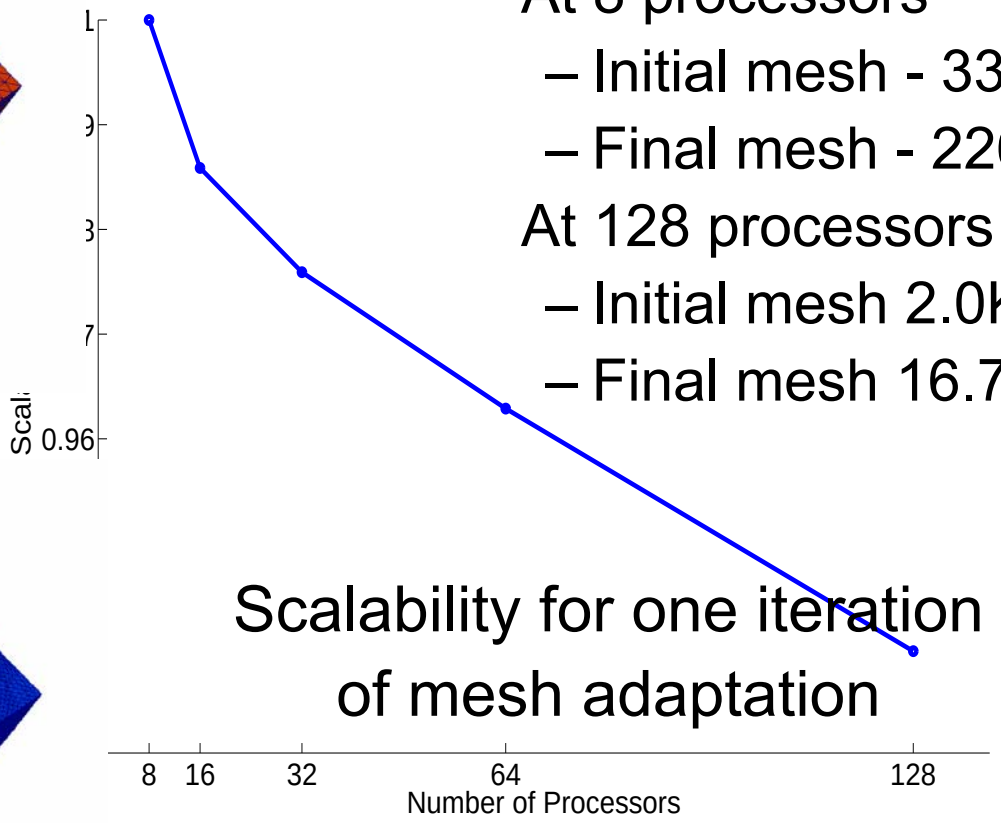
Parallel Refinement

initial
(265k tets)



adapted
(2,127k tets)





Scalability for one iteration of mesh adaptation

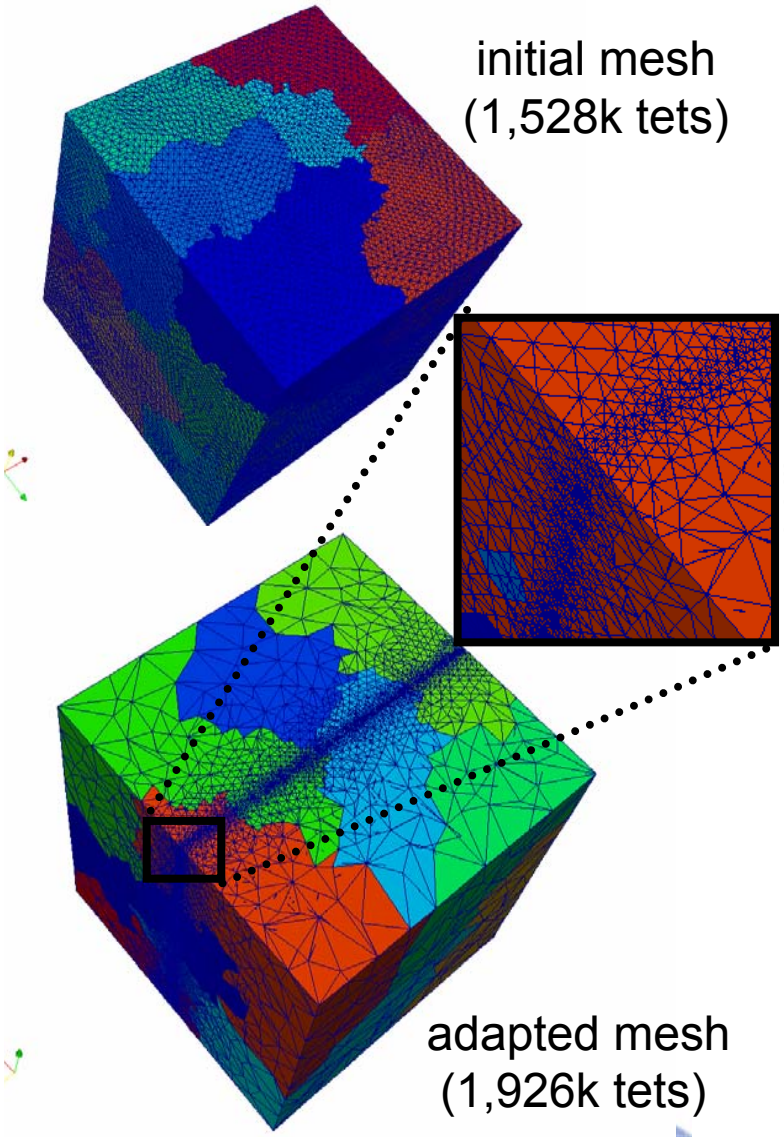
Tests run on IBM Blue Gene/L

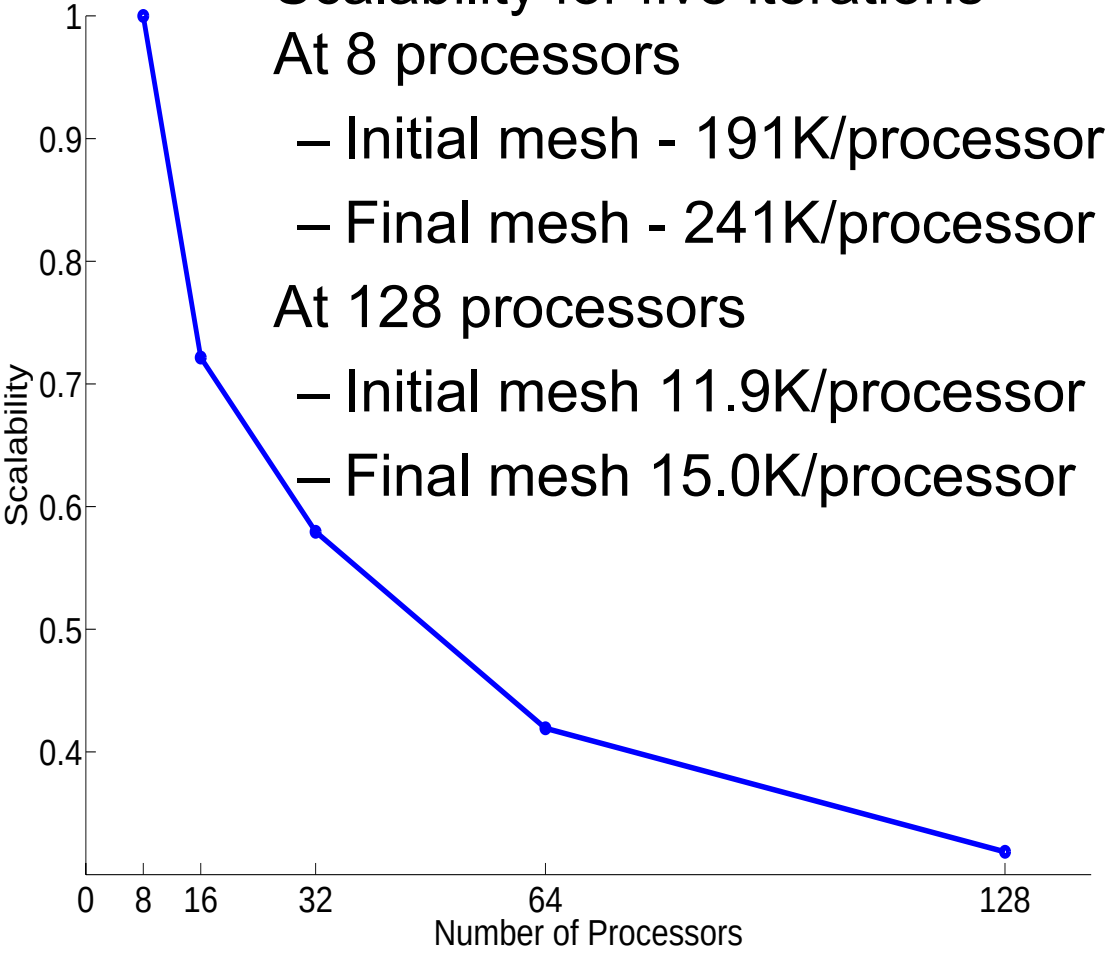
At 8 processors
- Initial mesh - 33K/processor
- Final mesh - 226K/processor

At 128 processors
- Initial mesh 2.0K/processor
- Final mesh 16.7K/processor

$$\text{Scalability} = \frac{\text{run-time}_{base} \bullet n\ \text{processors}_{base}}{\text{run-time} \bullet n\ \text{processors}}$$

Parallel Refinement and Coarsening





Scalability for five iterations

At 8 processors

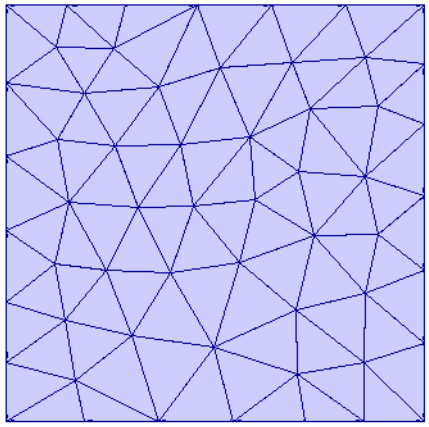
- Initial mesh - 191K/processor
- Final mesh - 241K/processor

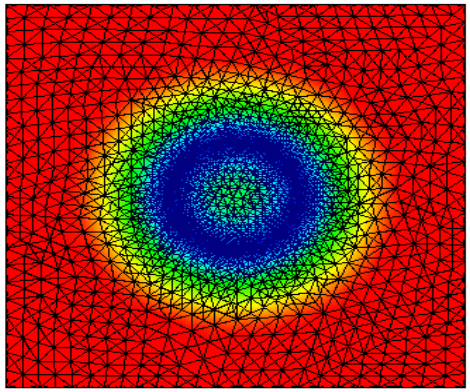
At 128 processors

- Initial mesh 11.9K/processor
- Final mesh 15.0K/processor

$$\text{Scalability} = \frac{\text{run-time}_{\text{base}} \bullet n\ \text{processors}_{\text{base}}}{\text{run-time} \bullet n\ \text{processors}}$$

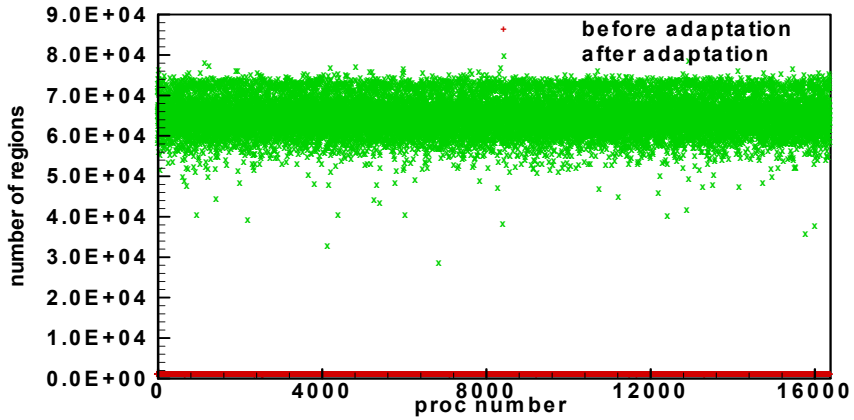
Mesh Adaptation for 1 Billion Element Mesh





Initial and adapted mesh at one bubble - colored by magnitude of mesh size field

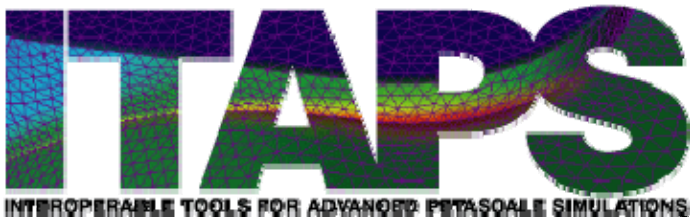
Mesh size field of air bubbles distributing in a tube (segment of the model)



Number of regions of adapted mesh among 16k parts

- Initial mesh: uniform, 17,179,836 mesh regions
- Adapted mesh: 160 air bubbles 1,064,284,042 mesh regions
- Multiple predictive load balance are used to make the adaptation possible
- Larger meshes possible (not out of memory) but this element count is appropriate for solver

SciDAC
Scientific Discovery through Advanced Computing



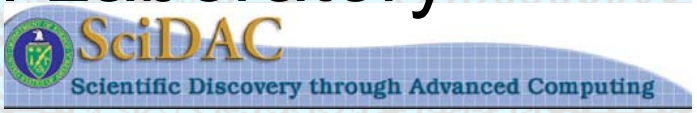
The FronTier Lite ITAPS service

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

Stony Brook University

Zhiliang Xu and Roman Samulyak

Brookhaven National Laboratory



Motivation

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

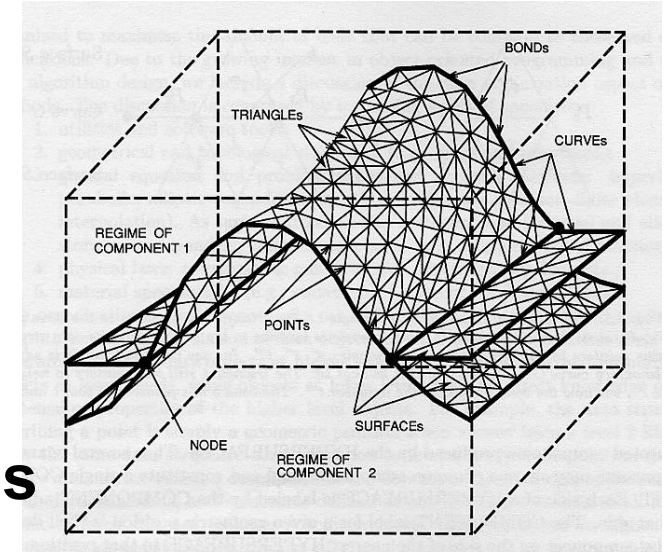
We provide a toolkit for scientific applications in which a dynamically moving front plays an important role in physical problems.

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.

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**

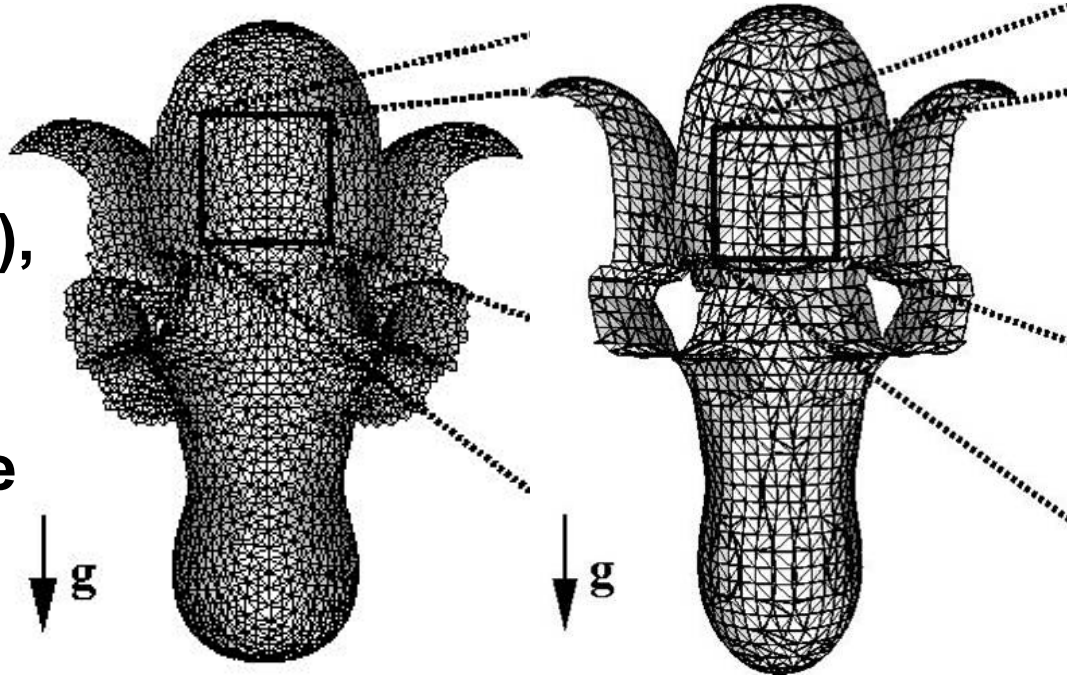
FronTier **automatically resolves topological bifurcation**

FronTier service Capabilities:

- 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.

FronTier mesh and geometry

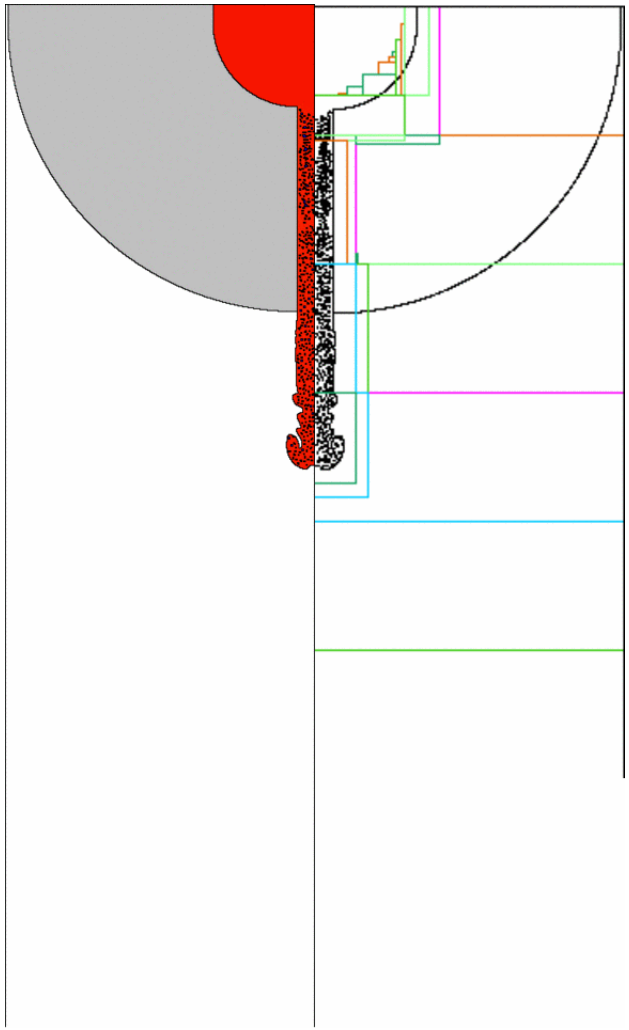
FronTier **supports independent surface mesh (grid-free mesh), and grid-based mesh which interacts with the structured volume mesh.**



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

FronTier application: diesel jet

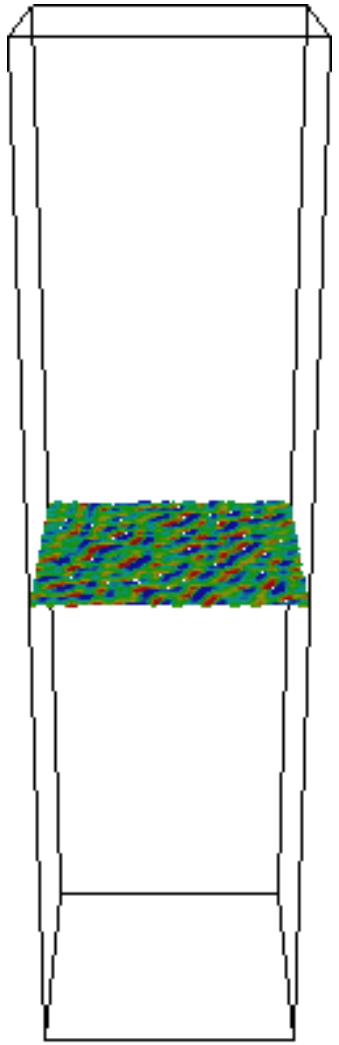
FronTier application to the simulation of diesel jet injection. The simulation features the phase transition at the jet interface and the insertion of bubbles due to atomization in the nozzle.

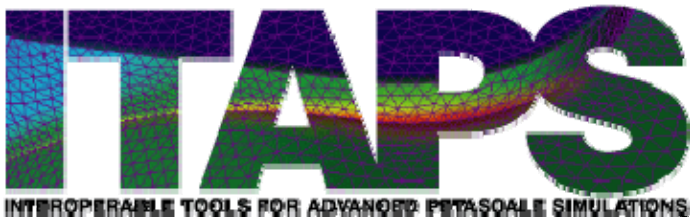


FronTier application: chaotic mixing

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

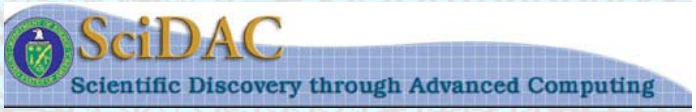
The tracking of the fluid interface provides sharp, high resolution at the interface and yields the best result in agreement with experiment.





Dynamic Services (Zoltan)

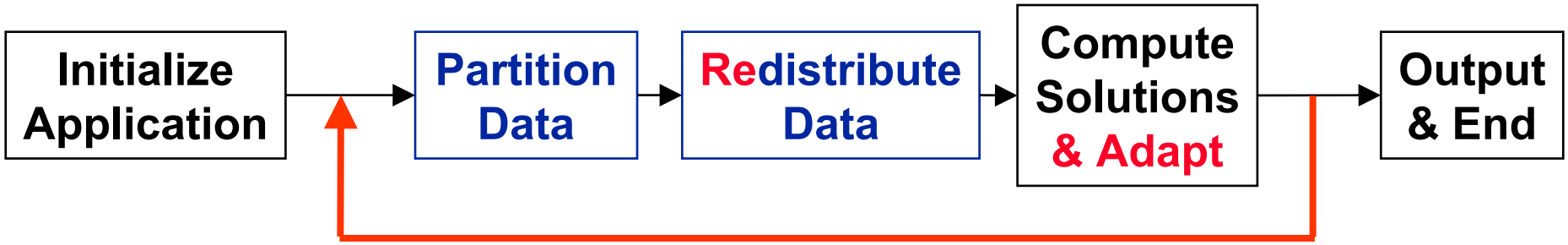
Vitus Leung and Karen Devine
Sandia National Laboratories




Static Partitioning

- Assign mesh entities to processors for parallel computation.
  - Minimize processor idle time.
  - Maintain low inter-processor communication costs.
- Static partitioning in an application:
  - Data partition is computed.
  - Data are distributed according to partition map.
  - Application computes.

Dynamic Repartitioning (a.k.a. Dynamic Load Balancing)

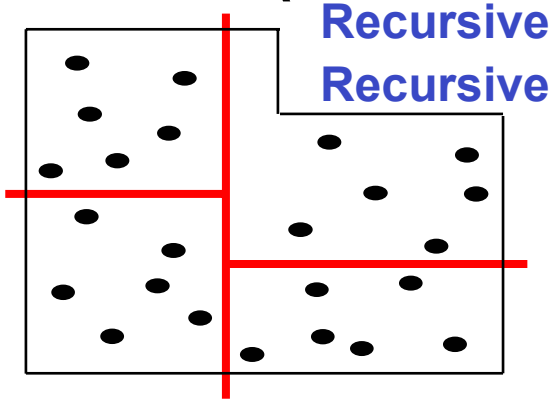


- Redistribute mesh entities among processor to adjust for changes in the computation.
  - Changing workloads in adaptive mesh refinement.
  - Changing geometric locality in crash simulations/contact detection.
- Dynamic repartitioning (load balancing) in an application:
  - Data partition is computed.
  - Data are distributed according to partition map.
  - Application computes and, perhaps, adapts.
  - Process repeats until the application is done.

Zoltan Suite of Parallel Partitioners and Repartitioners

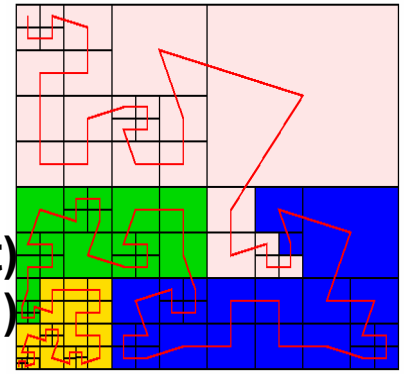
No single method is appropriate for all applications.

Geometric (coordinate-based) methods



Recursive Coordinate Bisection (Berger, Bokhari)

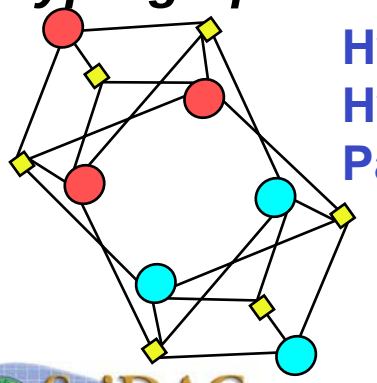
Recursive Inertial Bisection (Taylor, Nour-Omid)



Space Filling Curves (Peano, Hilbert)

Refinement-tree Partitioning (Mitchell)

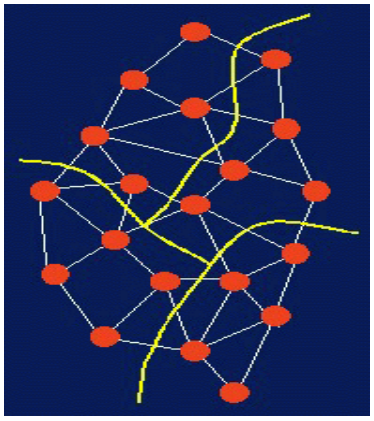
Hypergraph and graph (connectivity-based) methods



Hypergraph Partitioning

Hypergraph Repartitioning

PaToH (Catalyurek)



Zoltan Graph Partitioning

ParMETIS (U. Minnesota)

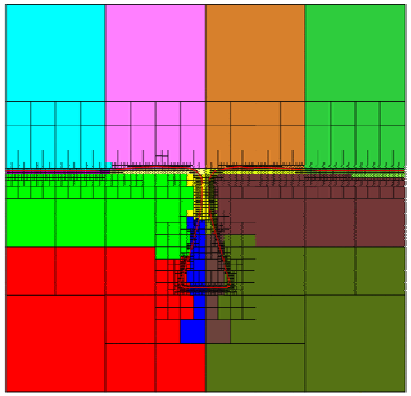
PT-Scotch (U. Bordeaux)

SciDAC
Scientific Discovery through Advanced Computing

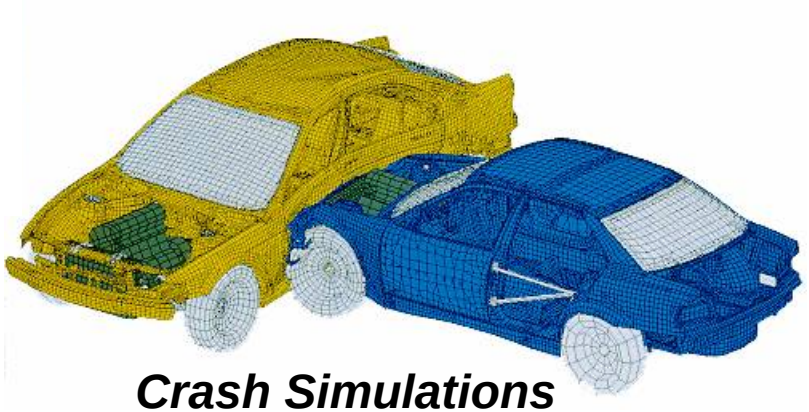
Geometric Partitioning and Repartitioning in Zoltan

- Partition with respect to entities' geometric coordinates.
  - Assign physically close entities to the same processor.
- Advantages:
  - Easy to use; need only coordinate info.
  - Fast and inexpensive; suitable for adaptive mesh refinement with frequent repartitioning.
  - Grouping of physically close entities is essential for contact detection and particle simulations.
  - All processors can inexpensively know the entire partition; needed for global search in contact detection.
- Disadvantages:
  - Mediocre partition quality; no explicit control of communication costs.
  - Can generate disconnected subdomains for complex geometries.

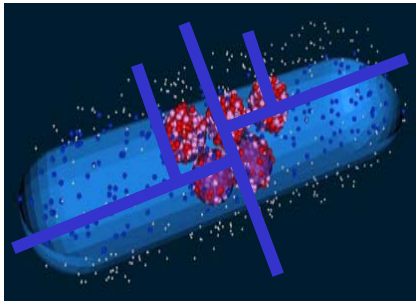
Applications of Geometric Repartitioners

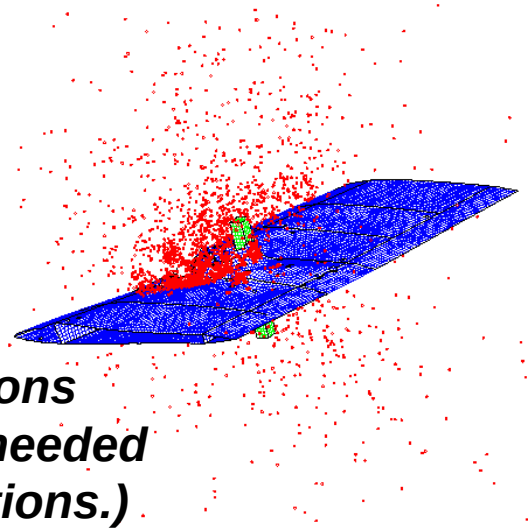


Adaptive Mesh Refinement (Frequent adaptation requires fast repartitioning.)



Crash Simulations and Contact Detection (Geometric locality needed for global search.)



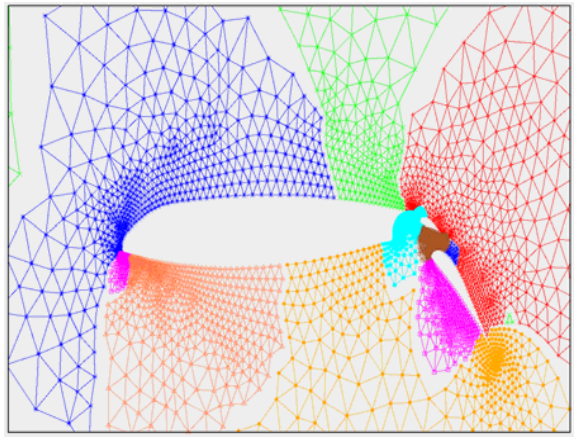


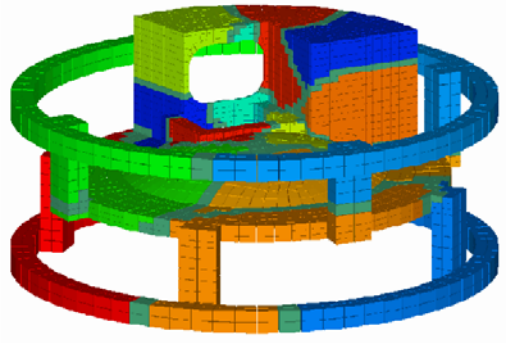
Particle Simulations (Geometric locality needed for particle interactions.)

Connectivity-based Partitioning and Repartitioning in Zoltan

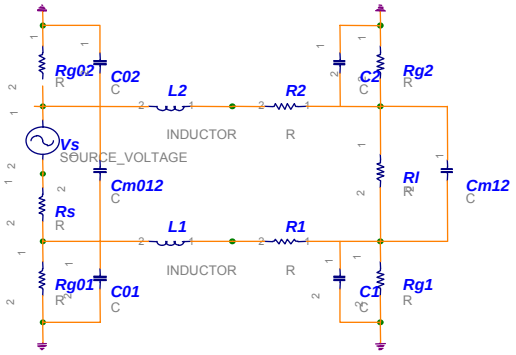
- Partition with respect to entities' data dependencies.
  - Assign entities that depend on each other to the same processor.
  - Graph/Hypergraph representation:
    - Entities are graph/hypergraph vertices.
    - Dependencies are edges/hyperedges.
- Advantages:
  - Proven highly successful for mesh-based PDE problems.
  - Explicit control of communication volume gives higher partition quality than geometric methods.
- Disadvantages:
  - More expensive than geometric methods.
  - More difficult to use than geometric methods.

Applications of Connectivity-based Partitioners

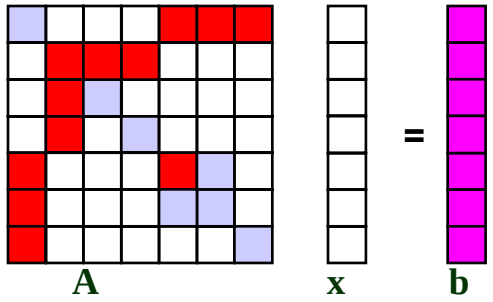




Finite Element Simulations
(Explicit control of communication costs enables efficient computation.)

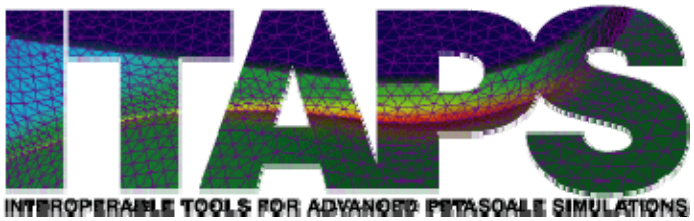


Electronic Circuit Simulations
(Network-based data does not have coordinate info.)



Linear Solvers & Preconditioners
(Partitioners' communication metrics are designed for matrix operations.)

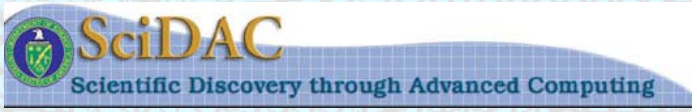
SciDAC
Scientific Discovery through Advanced Computing



Visualization using VisIt

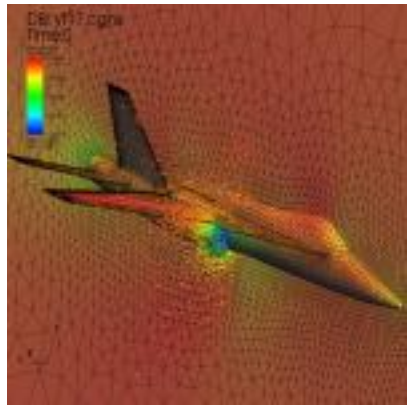
Mark Miller

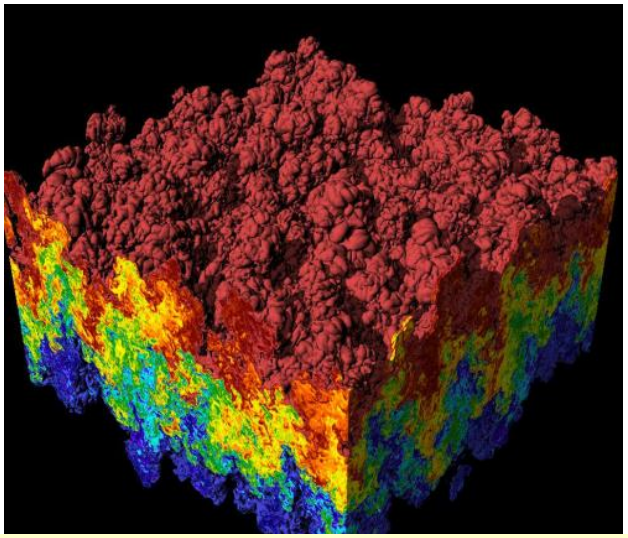
Lawrence Livermore National Laboratories




VisIt provides many advanced capabilties

- Free interactive parallel visualization and graphical analysis tool for scientific visualization
- Useful for scalar, vector, and tensor fields in multiple dimensions
- Can visualize data on point, rectilinear, curvilinear and unstructured grids
- Powerful, full featured graphical user interface
- Parallel architecture for visualizing terascale data sets
- Extensible with dynamically loadable plug-ins

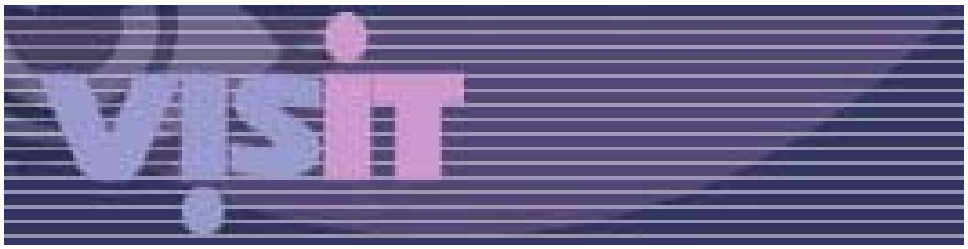




https://wci.llnl.gov/codes/visit/home.html

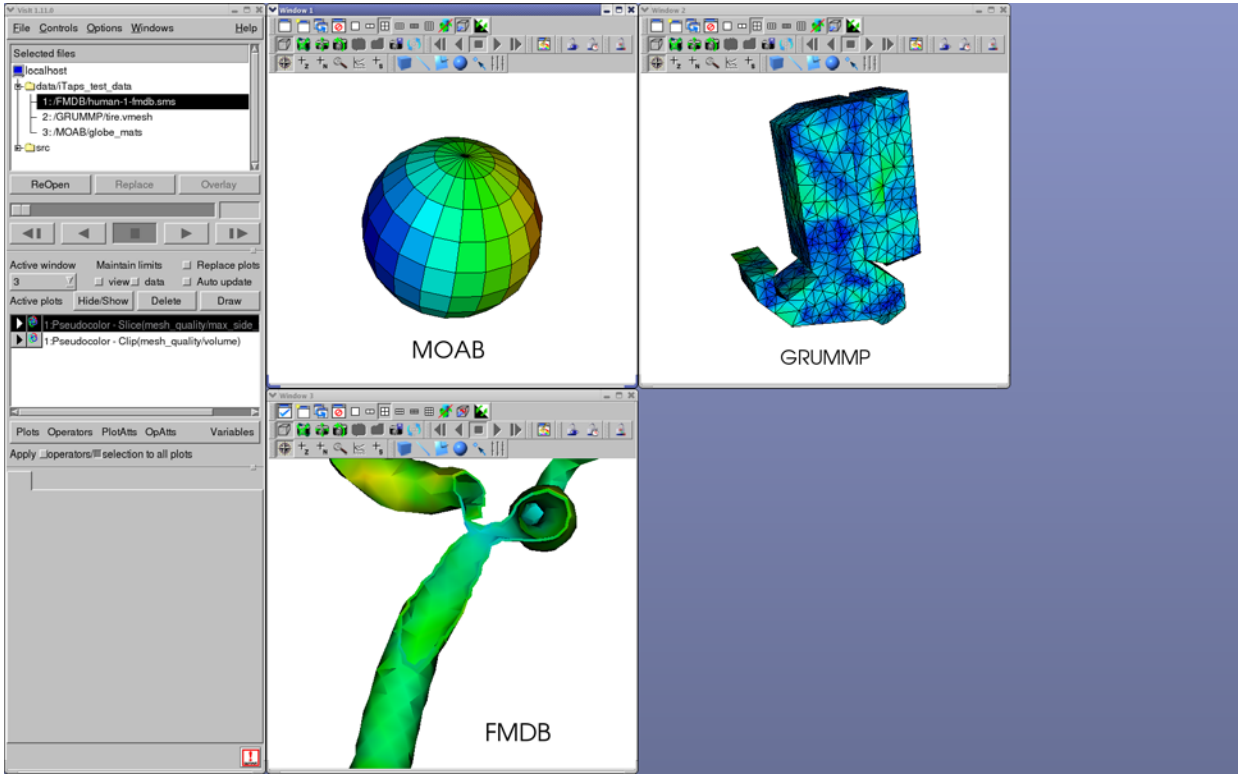
ITAPS has been integrated with VisIt as a database plug-in

- A single plug-in supports multiple ITAPS implementations
- Supports all entity types
- Supports subset and and tag data visualization
- Future integration will use VisIt's in-situ 'simulation' interface
  - Will enable any ITAPS-compliant software to integrate with VisIt at run-time



VisIt can be used to display test data from multiple implementations

Example of VisIt displaying test data from multiple ITAPS implementations simultaneously



ITAPS
INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

PART 3: ITAPS Data Model

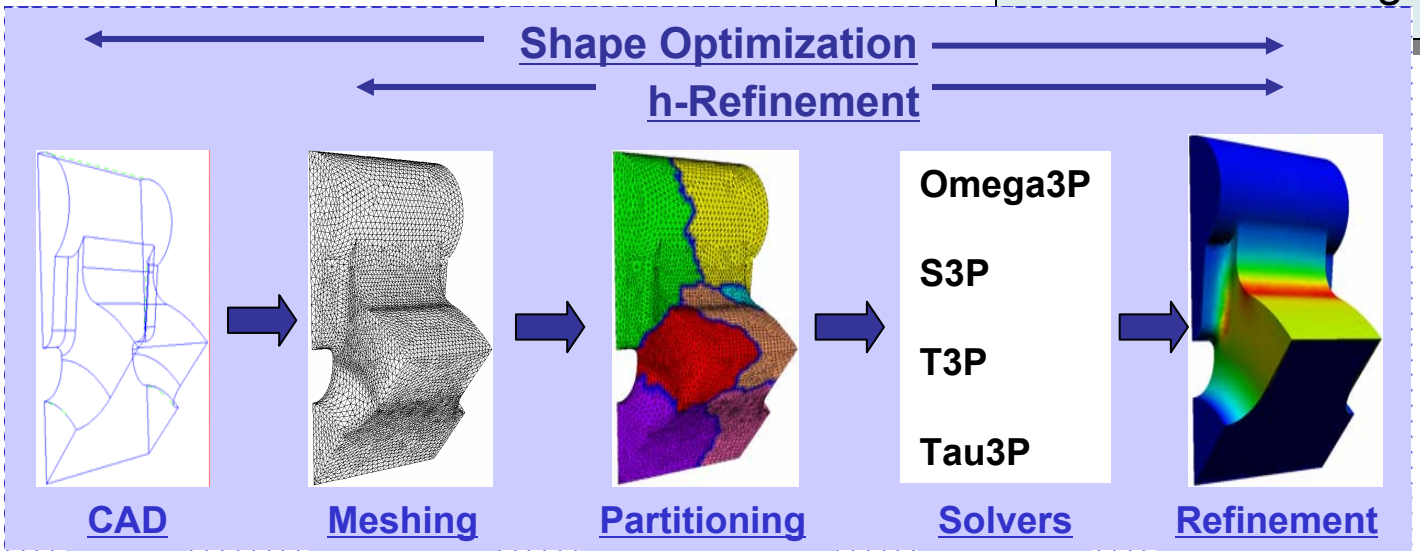
SciDAC
Scientific Discovery through Advanced Computing

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

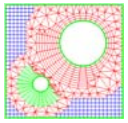
- The data model must encompass a broad spectrum of mesh types and usage scenarios
- A set of common interfaces
  - Implementation and data structure neutral
  - Small enough to encourage adoption
  - Flexible enough to support a broad range of functionality



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, iMeshP
- Geometry: iGeom
- Fields: iField
- Relations: iRel

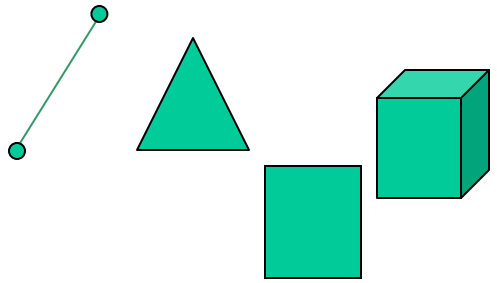
- Data Relation Managers
  - Provides control of the relationships among the core data types
  - Resolves cross references between entities in different groups
  - Provides functionality that depends on multiple core data types

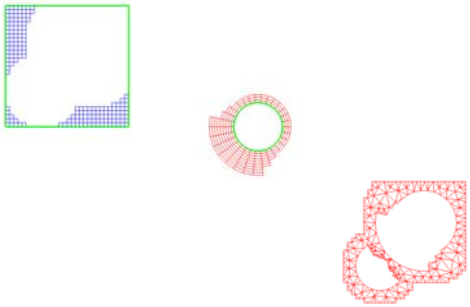
The ITAPS data model has four fundamental "types"

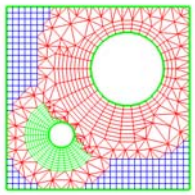
- *Entity:* fine-grained entities in interface (e.g., vertex, face, region)
- *Entity Set:* arbitrary collection of entities & other sets
  - Parent/child relations, for embedded graphs between sets
- *Interface Instance:* object on which interface functions are called and through which other data are obtained
- *Tag:* named datum annotated to Entities and Entity Sets

These core data types blend abstract and specific concepts

- Entity Definition
  - Unique type and topology
  - Canonical ordering defines adjacency relationships
- Entity Set Definition
  - Arbitrary grouping of ITAPS entities a other sets
  - There is a single “Root Set”
  - Relationships among entity sets
    - Contained-in
    - Hierarchical
- These objects are accessed using opaque (type-less) “handles”

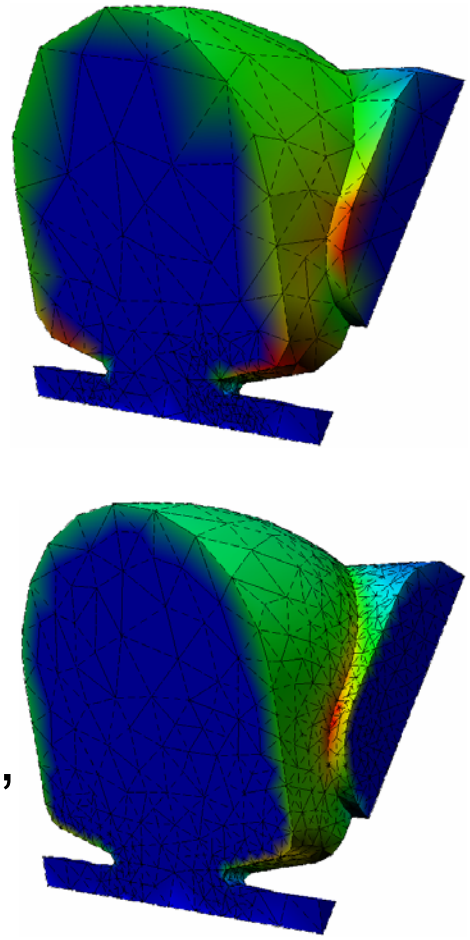






iMesh(P) provides access to the discrete representation of the domain

- iMesh supports local access to the mesh
- iMeshP complements iMesh with parallel support
- 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)
- 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

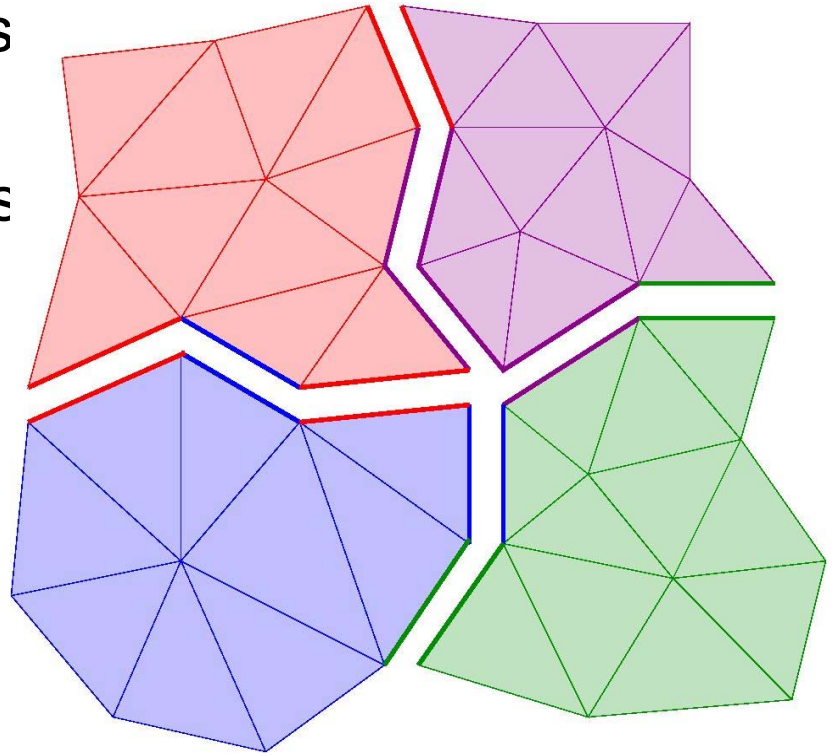


The iMesh interface supports basic and advanced local operations

- 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
  - Access to all entities in a set as single entities, arrays of entities, or entire set
  - Set/remove/access tag data
- Set functionality
  - Boolean operations (union, subtract, intersect)
  - Hierarchical relationships
- Mesh modification
  - Adding / Deleting entities
  - Vertex relocation
  - No validity checks

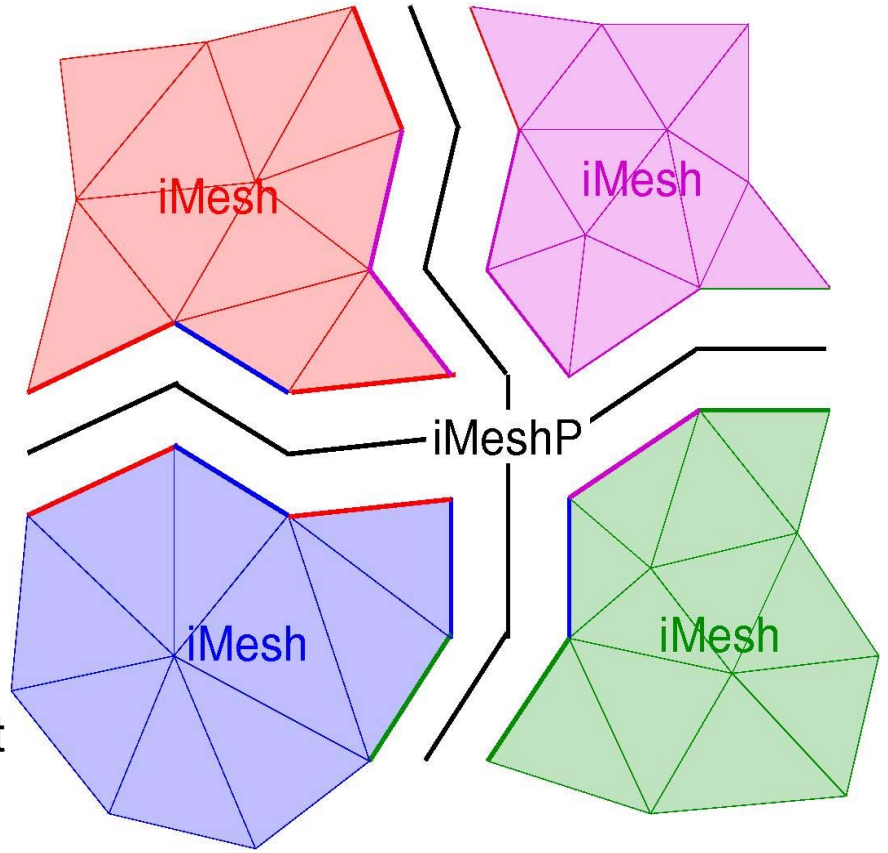
iMeshP extends iMesh to support parallel computations

- Focus on distributed memory
  - For example: use application's MPI communicators
  - But allow use of global address space and shared memory paradigms
- Leverage serial iMesh
  - Works as expected within a process
  - Works as expected for global address space and shared memory programs



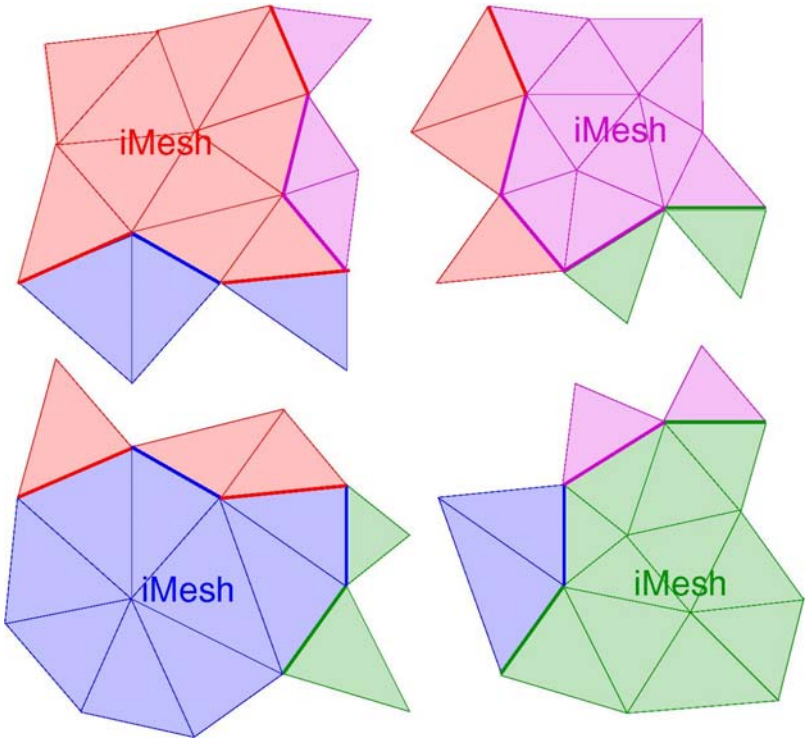
The iMeshP parallel interface defines a partition model

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



The Partition Model

- *Ownership*: right to modify an entity
- *Internal entity*: Owned entity not on an interpart boundary.
  - E.g., all triangles w/ same color as iMesh label for part
- *Part-Boundary entity*: Entity on an interpart boundary
  - E.g., bold edges
  - *Shared* between parts (owner indicated by color; other parts have copies).
- *Ghost entity*: Non-owned, non-part-boundary entity in a part
  - E.g., triangles whose color is different from iMesh label
  - Needed for adjacency and/or solution data
- *Copies*: ghost entities + non-owned part-boundary entities.

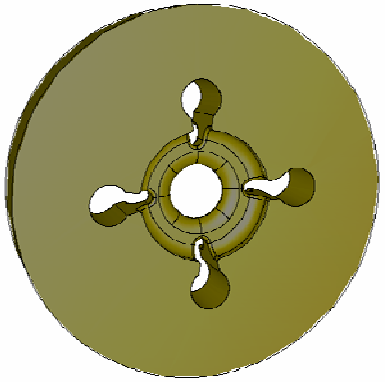


Partition Characteristics

- Maps entities to parts
  - Part assignments computed with respect to a set of entities
  - Computed assignments induce 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

The geometry interface provides access to the computational domain

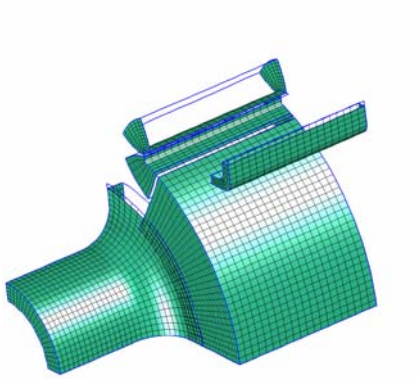
- Must support
  - automatic mesh generation
  - mesh adaptation
  - tracking domain changes
  - relating information between alternative discretizations

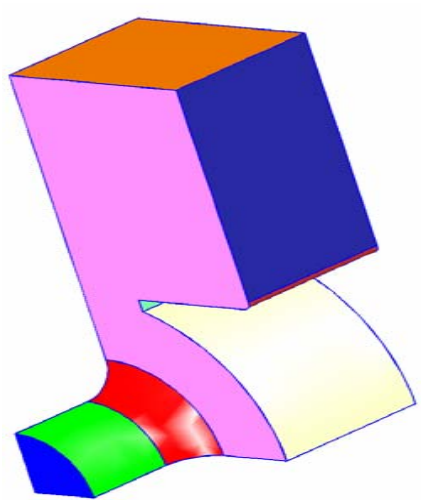


- Builds on boundary representations of geometry
- Used to support various underlying representations
  - Commercial modelers (e.g., Parasolid, ACIS)
  - Modelers that operate from standard files (e.g. IGES, STEP)
  - Models constructed from an input mesh

Basic and advanced functionalities are supported in the geometry interface

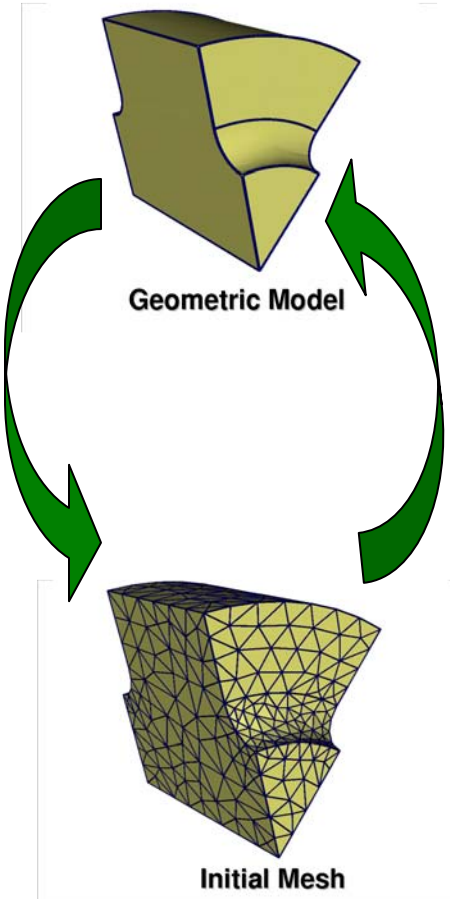
- Model loading and initiation
- Topological queries of entities and adjacencies
- Pointwise geometric shape interrogation
- Parametric coordinate systems
- Model topology modification





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



Multiple approaches can be used to embed application specific data

- Boundary conditions
  - Approach 1:
    - Boundary conditions associated with geometric model
    - Mesh classified on geometric model connects boundary conditions to mesh entities
  - Approach 2:
    - Mesh entities grouped together into an entity set that is tagged to represent boundary conditions
- Solution data
  - Tags associated with the appropriate mesh entities

Summary of the ITAPS data model

- The data model abstracts the information flow in PDE simulations: Geometry, Mesh, Fields and their Relations
- Each core abstraction has a separate interface definition: iMesh, iGeom, iField, iRel
- The core building blocks of the data models are entities, entity sets, interfaces and tags
- The parallel data model defines partitions, parts, and entity ownership concepts

ITAPS
INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

PART 4a: Basic ITAPS Interfaces

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

Outline

- iMeshP ITAPS Interface (w/ examples)
- Best Practices (for Performance)
- Language Interoperability
  - C-binding interface
  - SIDL/Babel

An overarching philosophy guides our interface definition efforts

- Create a small set of interfaces that existing packages can support
  - Should be able to implement the interfaces based on your representation of data structures
  - Allows incremental adoption
- Keep interfaces for the different core data model abstractions separate
- Balance performance and flexibility
- Work with a large tool provider and application community to ensure applicability
- Language Interoperability
  - C interface callable from C, C++ and Fortran
  - Use SIDL/Babel for Java and python support

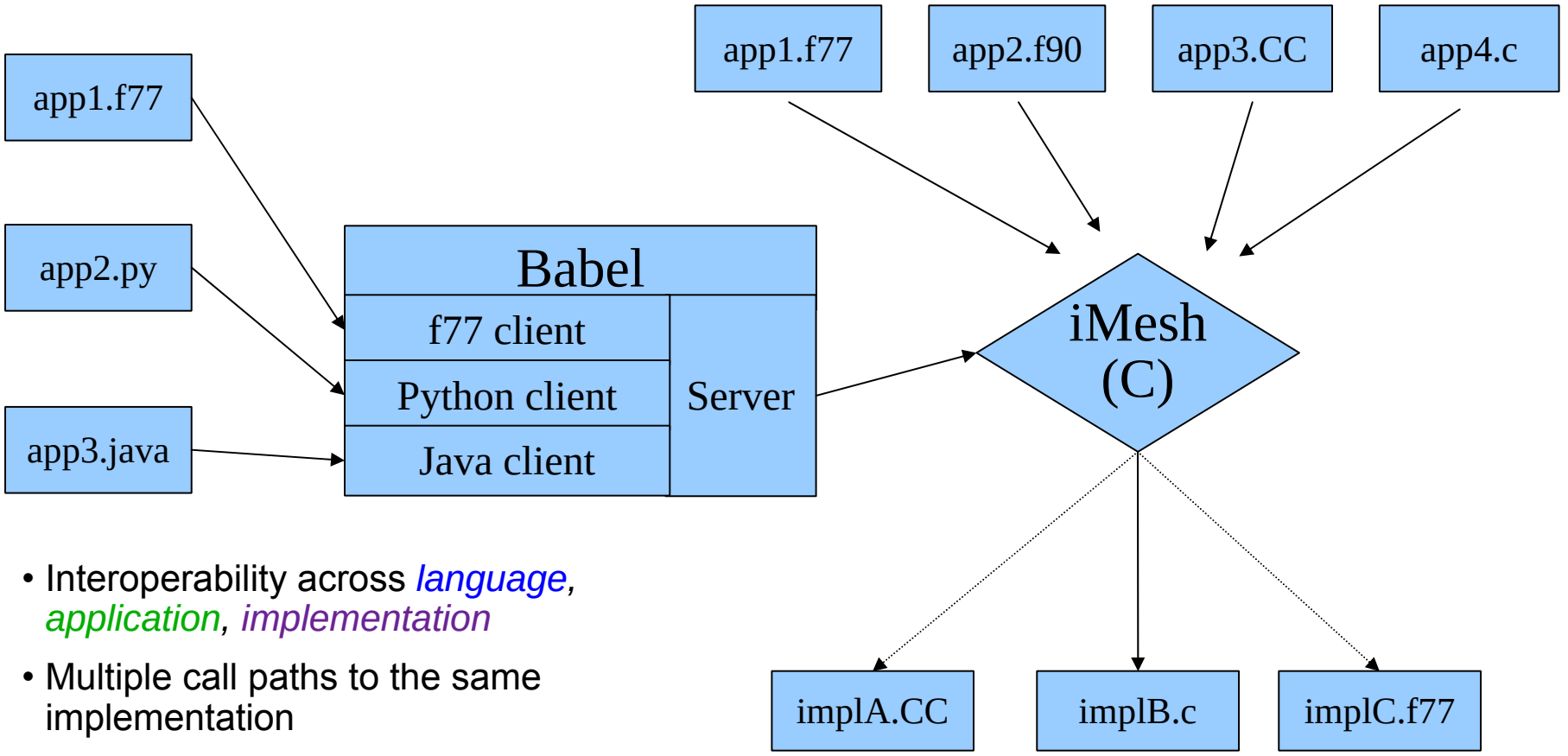
Lowers the burden for adoption of the interface

Performance is critical for kernel computations involving mesh access; flexibility is critical for covering a broad usage spectrum

CCA provides infrastructure and guidance for domain-specific interface definition efforts

SciDAC Scientific Discovery through Advanced Computing

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

Enumerated types used in the ITAPS interface

- 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

Simple Example: HELLO iMesh

```
#include "iMesh.h"
int main(int argc, char *argv[]){
  char *options = NULL;
  int ierr, dim, num, options_len = 0;
  iMesh_Instance mesh;
  iBase_EntitySetHandle root;

  /* create the Mesh instance */
1 iMesh_newMesh(options, &mesh, &ierr, options_len);
  iMesh_getRootSet(mesh, &root, &ierr);

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

  /* report the number of elements of each dimension */
  for (dim = iBase_VERTEX; dim <= iBase_REGION; dim++) {
3   iMesh_getNumOfType(mesh, root, dim, &num, &ierr);
    printf("Number of %d-D elements = %d\n", dim, num);
  }
  iMesh_dtor(mesh, &ierr);
  return 1;
}
```

Simple application that

1) Instantiates iMesh interface

2) Reads mesh from disk

3) Reports # entities of each dimension

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

HELLO iMesh Makefile

```
CC = cc -g
CXX = c++ -g

FMDB_TOP = /usr/local/itaps/visit/fmdb
LIBDIR += -L$(FMDB_TOP)/lib
LIBS += -lFMDB -lSCORECModel -lSCORECUtil
INCPATH += -I$(FMDB_TOP)/include

hello: hello.c
       $(CC) $(INCPATH) -c hello.c
       $(CXX) $(LD_FLAGS) -o $@ hello.o $(LIBDIR) $(LIBS)
```

Simple Example: HELLO iMeshP

Parallel Version: HELLO iMeshP

1) Instantiates Partition

2) Reads mesh into mesh instance and Partition

3) Reports # parts in Partition

```
#include "iMesh.h"
#include "iMeshP.h"
#include <mpi.h>
int main(int argc, char *argv[]) {
  char *options = NULL;
  iMesh_Instance mesh;
  iMeshP_PartitionHandle partition;
  int me, dim, num, ierr, options_len=0;
  iBase_EntitySetHandle root;
  /* create the Mesh instance */
  iMesh_newMesh(options, &mesh, &ierr, options_len);
  iMesh_getRootSet(mesh, &root, &ierr);

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &me);
  /* Create the partition. */                                   (1)
  iMeshP_createPartitionAll(mesh, MPI_COMM_WORLD, &partition, &ierr);

  /* load the mesh */                                           (2)
  iMeshP_load(mesh, partition, root, argv[1], options, &ierr,
              strlen(argv[1]), options_len);

  /* Report number of Parts in Partition */                     (3)
  iMeshP_getNumParts(mesh, partition, &num, &ierr);
  printf("%d Number of Parts = %d\n", me, num);
. . .
```

HELLO iMeshP Makefile

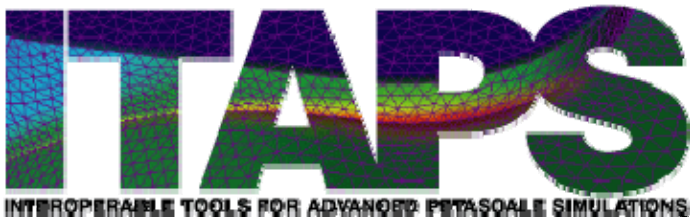
```
CC = mpicc -g
CXX = mpicxx -g

FMDB_TOP = /mnt/disk2a/txie/tmp
LIBDIR += -L$(FMDB_TOP)/lib \
          -L../Zoltan/Obj_linux64 \
          -L../ParMETIS3_1 -L../FMDB_iMeshP/lib/linux64
LIBS += -lFMDB -lSCORECModel -lSCORECUtil \
        -lzoltan -lparmetis -lmetis -lautopack
INCPATH += -I$(FMDB_TOP)/include/FMDB
CC += -DIMESHP -DPARALLEL

helloP: helloP.c
        $(CC) $(INCPATH) -c helloP.c
        $(CXX) $(LD_FLAGS) -o $@ helloP.o $(LIBDIR) $(LIBS)
```

ITAPS API's: Argument Handling Conventions

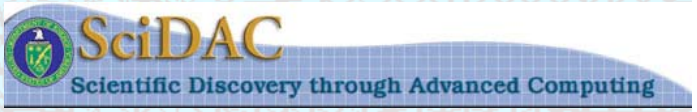
- ITAPS API's are C-like and can be called directly from C, Fortran, C++
- Arguments pass by value (in) or reference (inout, out)
  - Fortran: use %VAL extension
- Memory allocation for lists done in application *or* implementation
  - If inout list comes in allocated, length must be long enough to store 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*

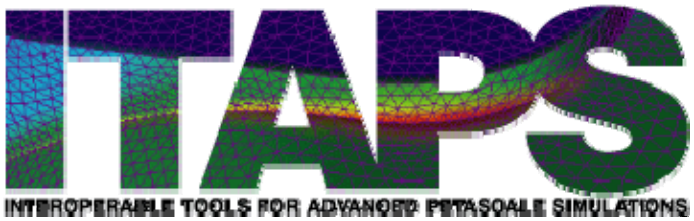
Argument Handling Conventions

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

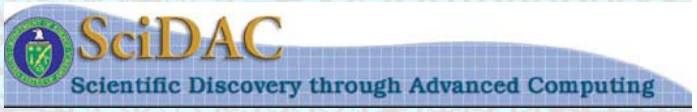


MORNING SESSION HANDS ON
HELLO iMeshP



ITAPS Tutorial: Afternoon Session



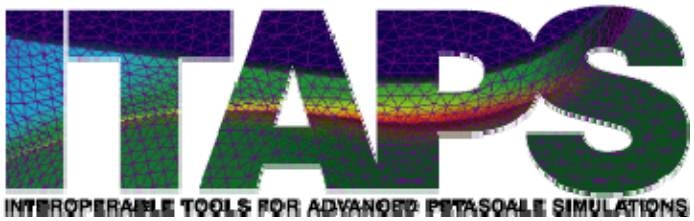

Tutorial Overview

Morning

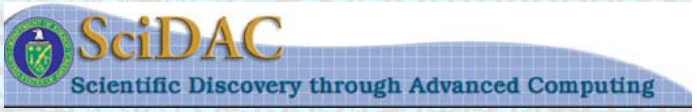
- Part 1: Introduction
- Part 2: Overview of ITAPS services
- Part 3: The ITAPS Data Model
- Part 4a: Basic ITAPS Interfaces
- *Hands on exercise*:
  - Hello iMeshP

Afternoon

- Part 4b: Advanced ITAPS Interfaces
- Part 5: ITAPS Software
- Part 6: Experiences and use in applications
- Part 7: Conclusion
- *Hands on exercises*:
  - Accessing mesh information
  - Partitioning services through iMeshP



PART 4b: Advanced ITAPS Interfaces




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.)

Slightly More Complicated Example: FindAdjacency (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

FindAdjacency(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: FindAdjacency (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

FindAdjacency (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: FindAdjacency (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

SciDAC Scientific Discovery through Advanced Computing

FindAdjacency (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

FindAdjacency 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

FindAdjacencyS: FindAdjacencyS.o
$(CXX)  $(CXXFLAGS) -o $@ FindAdjacency.o ${IMESH_SIDL_LIBS_LINK}

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

FindAdjacnecyF: FindAdjacencyF.o
$(CXX) -o $@ FindAdjacencyF.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 (C)

```
#include "iMesh.h"

int main( int argc, char *argv[] )
{
  char *filename = argv[1];

    // create the Mesh instance
  iMesh_Instance mesh;
  int err;
  iMesh_newMesh(NULL, &mesh, &err, 0);

  iBase_EntitySetHandle root_set;
  iMesh_getRootSet(mesh, &root_set, &err);

    // load the mesh
  iMesh_load(mesh, root_set, filename, NULL,
    &err, strlen(filename), 0);

    // get all sets
  iBase_EntitySetHandle *sets = NULL;
  int sets_alloc = 0, sets_size;
  iMesh_getEntSets(mesh, root_set, 1, &sets,
        &sets_alloc, &sets_size, &err);

// iterate through them
  iBase_TagHandle *tags = NULL;
  int tags_alloc = 0, tags_size;
  int i, j;
  for (i = 0; i < sets_size; i++) {
      // get connectivity
    iMesh_getAllEntSetTags(mesh, sets[i],
    &tags, &tags_alloc, &tags_size, &err);
    if (0 != tags_size) {
      printf("Set %ld: Tags:", sets[i]);

        // list tag names on this set
      for (j = 0; j < tags_size; j++) {
        char tname[128];
        int int_val, tname_size, tag_type;
        double dbl_val;
        iMesh_getTagName(mesh, tags[j], tname,
                         &err, tname_size);
        printf("%s", tname);
        iMesh_getTagType(mesh, tags[j],
                         &tag_type, &err);
        if (iBase_INTEGER == tag_type) {
          iMesh_getEntSetIntData(mesh, sets[i],
                       tags[j], &int_val, &err);
          printf("(val = %d);", int_val);
        }
        else if (iBase_DOUBLE == tag_type) {
          iMesh_getEntSetDblData(mesh, sets[i],
                       tags[j], &dbl_val, &err);
          printf("(val = %f);", dbl_val);
        }
        else printf("; ");
      }
    }
    printf("\n");
    free(tags); tags = NULL; tags_alloc = 0;
  }

  return 0;
}
```

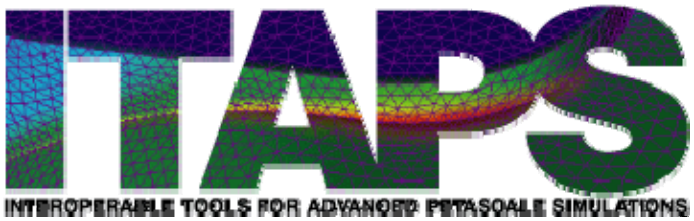
1

ListSetsNTags Example Notes

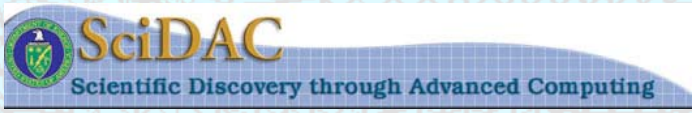
- Enumerated variables passed as int, compared to enum types (e.g. iBase_SUCCESS, iBase_INTEGER)

iMesh Interface 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 works 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



Parallel Mesh Interfaces




iMeshP extends iMesh to support parallel computations

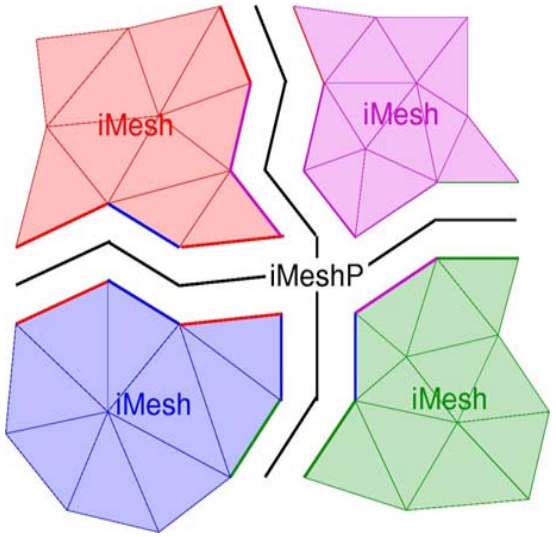
Example functionalities directly supported by iMeshP:

- Loading meshes onto parallel computers
- Maintaining relationships between mesh entities on different parts including boundary and ghost data
- Communicating tag data between parts
- Moving mesh entities between parts and re-establish communication links

Sufficient for building services such as:

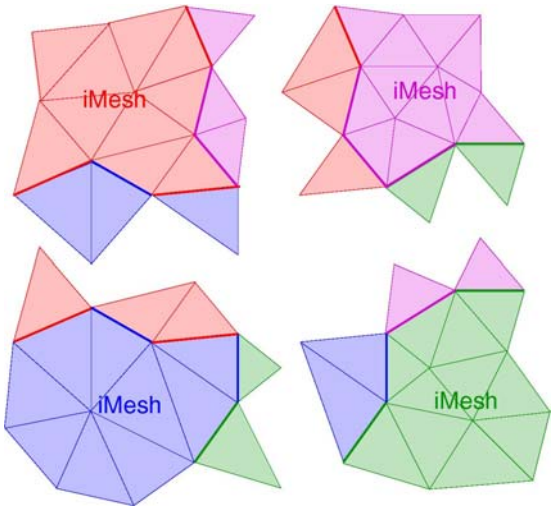
- Parallel mesh generation/improvement/modification
- Partitioning and repartitioning

Recall the partition model defined by the iMeshP parallel interface



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

- *Ownership*: right to modify an entity
- *Internal entity*: Owned entity not on an interpart boundary
- *Part-Boundary entity*: Entity on an interpart boundary
- *Ghost entity*: Non-owned, non-part-boundary entity in a part
- *Copies*: ghost entities + non-owned part-boundary entities



iMeshP Partition API Functions

- Create partition
  - Accepts pointer to MPI Communicator or NULL
  - iMeshP_createPartitionAll
- Destroy partition
  - iMeshP_destroyPartitionAll
- Sync partition
  - After parts are added/updated, compute and store global information about the partition
  - iMeshP_syncPartitionAll
- Return number of parts in partition
  - iMeshP_getNumParts
- Return global number of entities of given type/topology in partition
  - May require collective communication
  - iMeshP_getNumOfTypeAll, iMeshP_getNumOfTopoAll

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 accessed through part handles

iMeshP Part API Functions

- Create part and add to partition.
  - iMeshP_createPart
- Remove part and destroy it.
  - iMeshP_destroyPart
- Return part neighbors.
  - Parts A and B are neighbors if Part A has copies of entities owned by Part B or vice versa.
  - iMeshP_getNumPartNbors, iMeshP_getPartNbors
- Iterate over part-boundary entities.
  - iMeshP_initPartBdryEntIter

iMeshP Part API Functions

- Part handles can be used as set handles.
  - iMesh functions overloaded to take part handles.
- Examples:
  - Add and remove entities from part with iMesh_addEntToSet and iMesh_rmvEntFromSet.
  - Get number of entities in part with iMesh_getNumOfType, iMesh_getNumOfTopo.
  - Get info for entities in part with iMesh_getEntities, iMesh_getVtxArrCoords, iMesh_getAdjEntities, etc.

Entity Characteristics

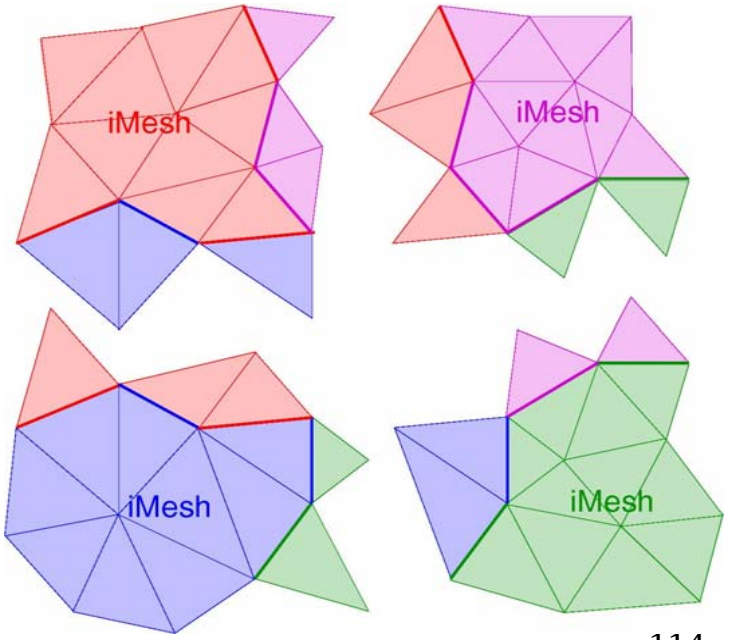
- No globally unique entity IDs are required or supplied.
  - Can be constructed by user as pair: [part ID, entity handle].
- Each entity is owned by only one part per partition.
  - Ownership grants right to modify.
- Entities may be copied on other parts.
- Duplicated information for copies:
  - Owner of an entity knows remote part and remote entity handle of all its copies.
  - All boundary entities know all remote parts and remote entity handles of all copies.
  - All ghost copies of entity know the entity owner's part and the entity handle on the owner.
- 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.
  - iMeshP_getEntOwnerPart, iMeshP_isEntOwner
- Determine whether entity is internal, boundary, ghost.
  - iMeshP_getEntStatus
- Return remote part and remote entity handles for copies of entity.
  - iMeshP_getNumCopies, iMeshP_getCopyParts, iMeshP_getCopies, iMeshP_getCopyOnPart
- Return owner part and owner entity handle for an entity.
  - iMeshP_getEntOwnerPart, iMeshP_getOwnerCopy

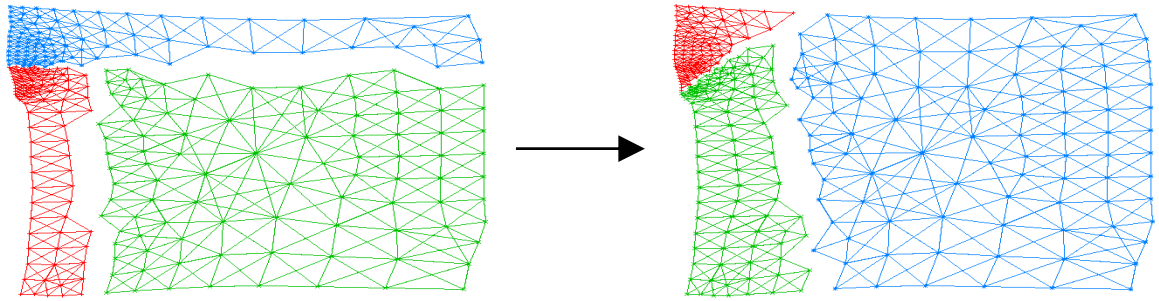
Ghost Entities

- Create ghost entities for each part.
  - iMeshP_createGhostEnts
  - 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.
- Delete all ghost entities.
  - iMeshP_deleteGhostEnts



Inter-part Mesh Operations

- iMeshP provides functions for inter-part operations on mesh entities.
  - Exchange tag values.
  - Migrate large numbers of entities for, say, load balancing.
  - Migrate small numbers of entities for, say, mesh modification.
  - Update mesh database during mesh modification.



Inter-part Mesh Operation Requests

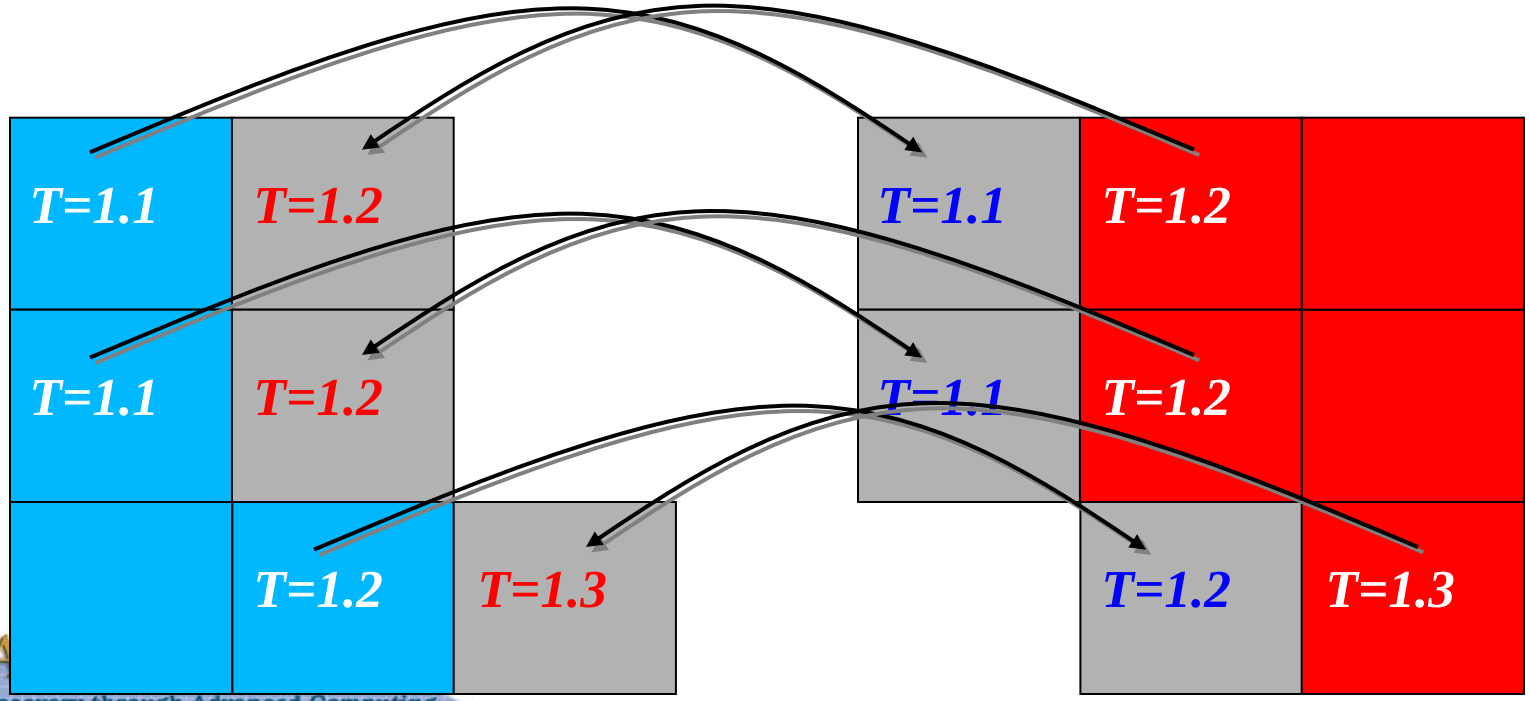
- Inter-part mesh operations are coordinated via iMeshP_Requests.
  - More than an MPI_Request!
  - Indicate status of a given iMeshP mesh operation.
    - Migrate entity.
    - Update vertex coordinates.
    - Update part-boundary entities.
    - Exchange tag data.
  - iMeshP encodes type of request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

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.

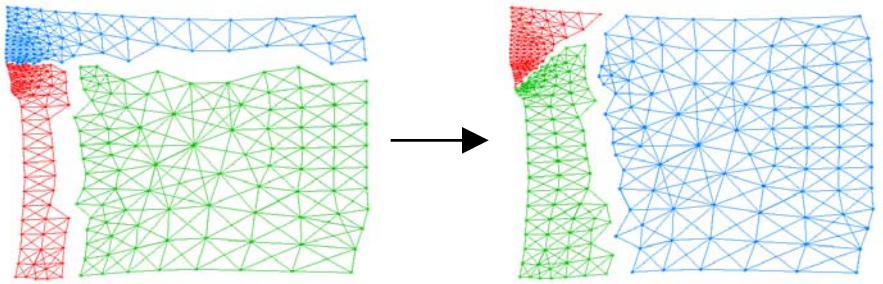
Communicate Entity Tag Data

- Entity owners send tag data to copies.
- iMeshP API provides both blocking and non-blocking versions of tag-data exchange.
  - iMeshP_pushTagData, iMeshP_iPushTagData



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 messages.
  - iMeshP adds entities to new parts and updates mesh.

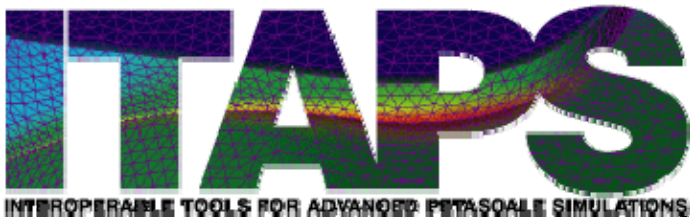


Updating Mesh Consistency

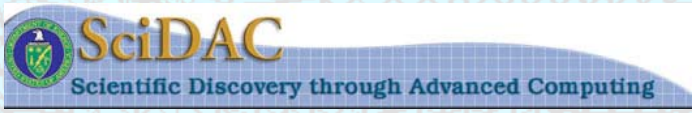
- After all mesh modification is done, application calls iMeshP_syncMeshAll.
  - A collective, blocking call that signals mesh modification operations are completed.
  - Polls for and processes outstanding requests.
  - Updates ghost entities for modified mesh.
  - Performs operations needed for parallel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

Parallel I/O

- Load a mesh from files.
  - Populate a mesh instance and a partition handle.
  - iMeshP_Load
- Write a mesh to files.
  - Store mesh data and partition data in files.
  - iMeshP_Save
- Options specified through parameter strings.
  - File names, file format, number of files, initial partitioning.
  - Analogous to iMesh load and save functions.



PART 5: ITAPS Software




ITAPS Web Pages

http://www.itaps- scidac.org

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

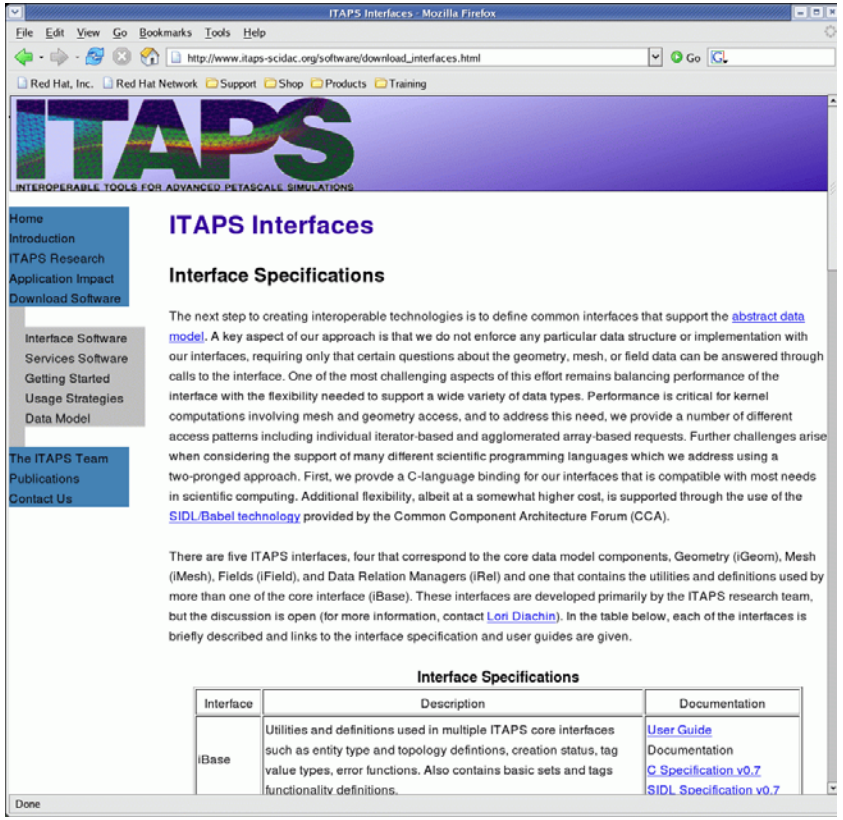
SciDAC
Scientific Discovery through Advanced Computing

Interface implementations are well underway

- iMesh 0.7 Interface complete
- iMeshP 0.5 specified and alpha implementations underway
- iGeom and iRel 0.7 interfaces complete
- Implementations
  - iMesh: FMDB, GRUMMP, NWGrid, MOAB
  - iMeshP: FMDB, MOAB
  - iGeom: CGM (Acis, OpenCasCade)
  - iRel: Lasso
- Interfaces have been used to build services and test interoperability
- Analyzing performance ramifications of interface usage

Interface Software Access

- Links to the interface user guides and man pages where available
- 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



FMDB Implementation

- Supports general set of mesh topologies
- Strengths are: adaptively changing meshes, flexible adjacency information
- Formal partition model supports parallel operations
- iMesh implementation is complete
- iMeshP implementation is alpha; current successful tests include functions required for some mesh adapt functions, partitioning, and migration
- Application use includes Fusion, Accelerators, and CFD

MOAB Implementation

- Supports hex, tet, prism, pyramid, tri, quad
  - Reads meshes generated by CUBIT, including geometric topology and a few other common formats
- Strengths are efficient memory management
- Includes tools for
  - OBB and kd tree construction and query
  - Read CGM/iGeom model & save as mesh-based model
  - Solution coupling
- Complete implementation of iMesh, iMeshP
- Application use includes nuclear reactor modeling, neutron transport, accelerator design optimization

GRUMMP Implementation

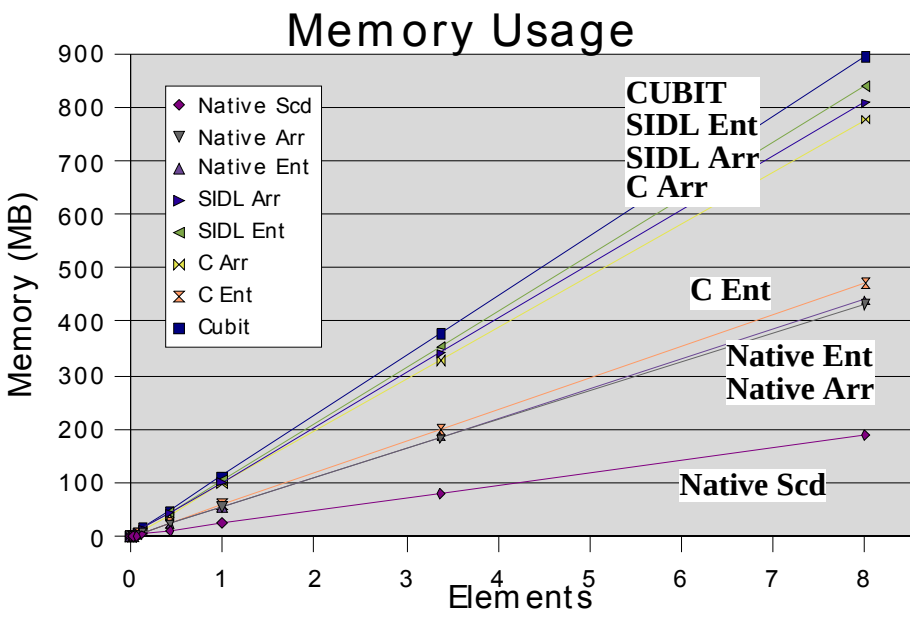
- Supports tri, quad, tet
- Strengths are: tri/tet meshing, mesh improvement and adaptation
- iMesh implementation is complete
- iMeshP implementation is planned
- Application use includes CFD, especially aerodynamics and non-Newtonian fluids

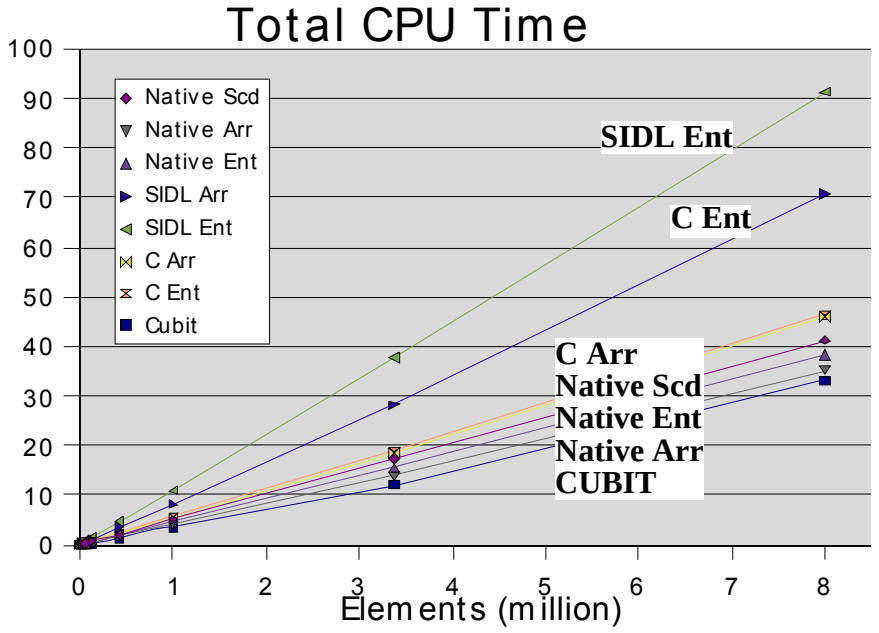
NWGRID Implementation

- Supports hex, tet, pyramid, tri, quad and hybrid meshes
- Strengths are: adaptive mesh refinement and dynamic mesh reconnections for simplex meshes
- Supports shared memory parallelism using Global Arrays
- iMesh implementation is complete
- iMeshP implementation is in process
- Application use includes biology, CFD/CMM/CEM, subsurface transport

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





Performance in building a finite element stiffness matrix

- Set up a simple stiffness matrix for a 2D diffusion equation
- Examine costs of entity access via native data structures, arrays, entity iterators and workset iterators
- Arrays minimize time overhead but require a data copy
- Entity iterators are straightforward to program, minimize memory overhead, but maximize time cost
- Entity array iterators balance time/memory tradeoffs but are the most difficult to program

$$\nabla^2 u = f$$

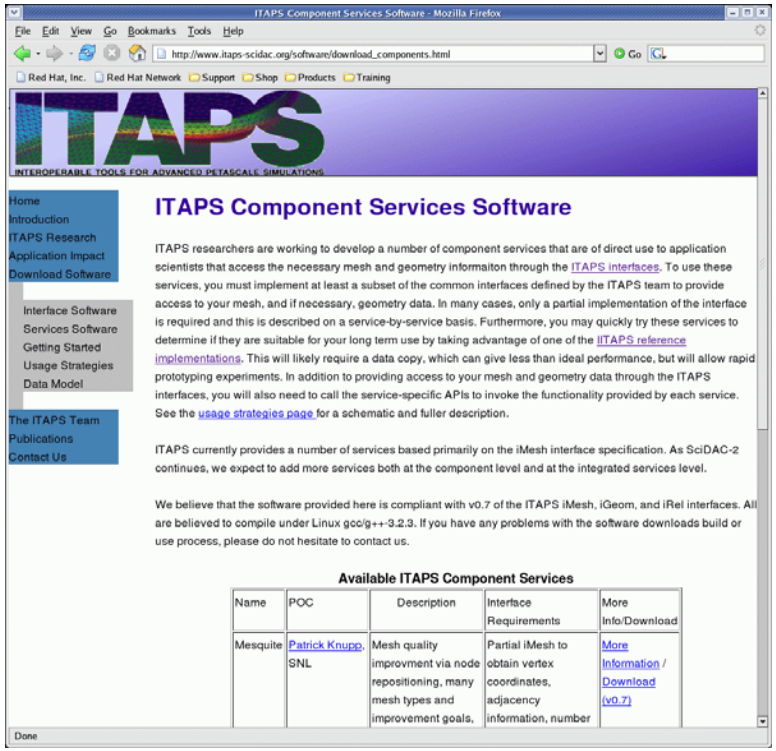
$$u(x=0)=1 \; u(x=1) = 1$$

$$u_y(x=0, x=1) = 0$$

	Time (ms)	Percent Diff
Native	10479	
Array-based	10774	2.8%
Entity Iterator	11642	11.1%
Workset Iterator (1)	11351	8.3%
Workset Iterator (3)	11183	6.7%
Workset Iterator (5)	11119	6.1%
Workset Iterator (10)	11095	5.8%
Workset Iterator (20)	11094	5.8%

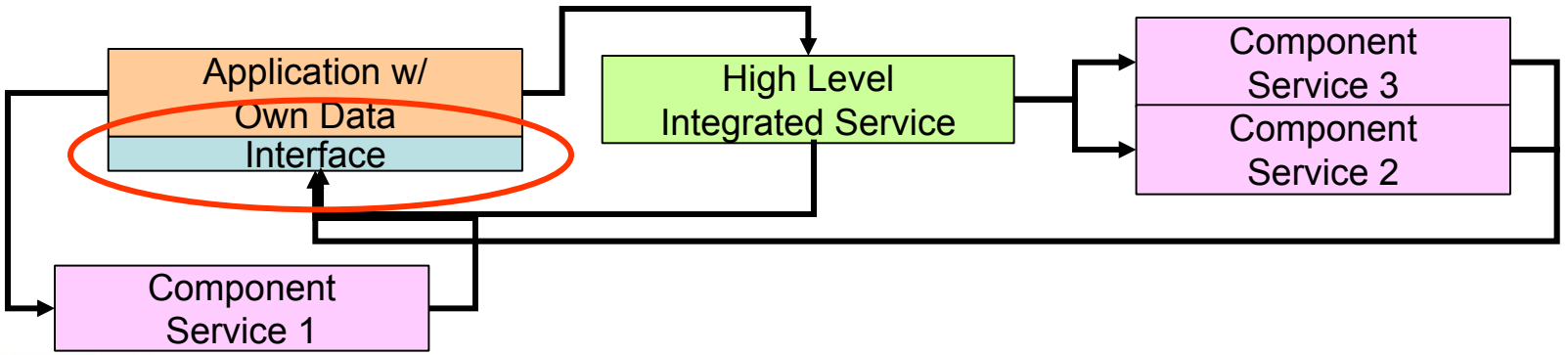
Services Software Access

- 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



Implementing ITAPS interfaces allows use of services on your data structures

- Need to implement some ITAPS interface functions using your data structures – BUT NOT ALL
- Most ITAPS functions reflect things you already do with your databases; most implementation tasks are a thin wrapper
- Compliance testing tools ensure correctness
- Can use a reference implementation at the cost of a data copy to experiment with services



Interface functions needed by the various services

	Coordinates	Adjacency	Other queries	Iterators	Modification	Basic Sets	Tags	Parallel	iGeom/iRel	Total
Mesquite	1	1	4	4	1	2	13		6	32
Swapping	1	1	4	4	2					12
Mesh Adapt	1	2	5	3	3		7	13		34
FronTier Lite	3	1	5	3	3		3			18
Zoltan	1	2	5					14		22
VisIt	1	1	9	3		1	16			31

Best practices for implementing for services

- Basic functionality such as vertex query, adjacency information, get type/topo are widely used
- Swapping needs efficient, correct iterators under modification
- Mesquite relies on tags and set boolean operations that may impact performance
- VisIt visualization routines require tag implementations to retrieve data to be analyzed
- MeshAdapt and Zoltan both use parallel interface functions for setting and getting partition data as well as exchanging data among processors.

Performance of iMesh Swap for 3D Meshes

- Comparing GRUMMP native implementation to service with GRUMMP iMesh implementation
- Most remaining overhead is in transcribing data to return format expected by iMesh

Case	# of Tets	Native		iMesh	
		Swaps	Rate (1/s)	Swaps	Rate (1/s)
Rand1	5104	10632	29500	10838	21300
Rand2	25704	65886	27700	67483	22100
Airplane	251140	25448	3380	28629	2800
Rocket	464080	53331	3540	59330	2790

Compliance testing is your friend

- Why bother?
  - Confirm an implementation complies with iMesh spec --- enhances interoperability
  - Confirm that a partial implementation supports a service --- ensures plug-and-play with services
- About the compliance test suite
  - Covers all functions
  - Supplements documentation with unambiguous software interpretation of interface
  - Coordinate and adjacency checks rely on consistency between different retrieval strategies
  - Set and tag functionality testable directly
  - Builds just like any other iMesh application
  - Work in progress on testing partial implementations effectively

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 and iMesh compliant services
  - Partitioning, IO, parallel communication, (parallel) file readers
- Try different implementations: they are tuned for different application uses – Experiment!
- Implement iMesh on top of your data structure
  - Take advantage of tools that work on iMesh API
- Let us help you
  - Not all best practices are easily described or self-evident

ITAPS
INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

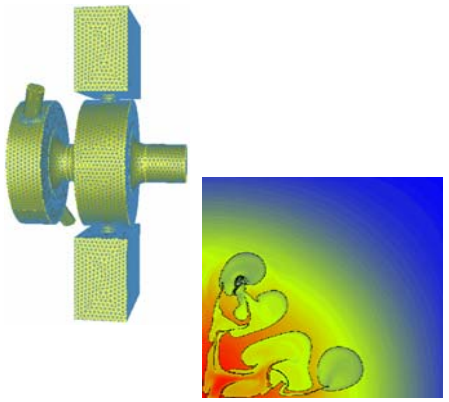
PART 6: Using ITAPS: Approaches and Experiences

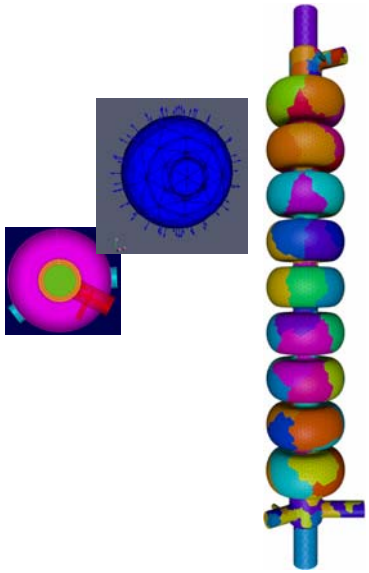
SciDAC
Scientific Discovery through Advanced Computing

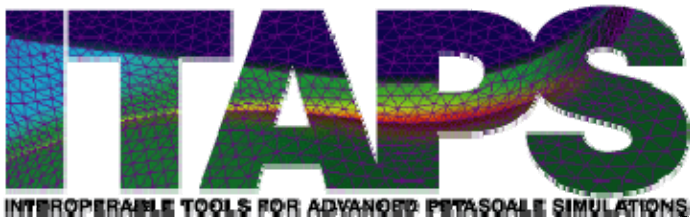
Sandia National Laboratories
Rensselaer
STONY BROOK
THE UNIVERSITY OF BRITISH COLUMBIA
BROOKHAVEN NATIONAL LABORATORY
OAK RIDGE NATIONAL LABORATORY
Pacific Northwest National Laboratory

ITAPS impact SciDAC applications in three ways

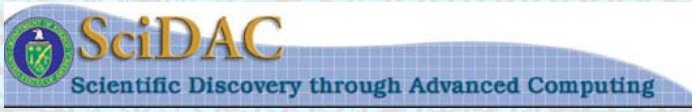
- Direct use of ITAPS technology in applications
  - Geometry tools, mesh generation and optimization for accelerators and fusion
  - Mesh adaptivity for accelerators and fusion
  - Front tracking for astrophysics and groundwater
  - Partitioning techniques for accelerators and fusion
- Technology advancement 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







Case Study 1: Incorporating Parallel Adaptive Applications




Parallel adaptive loops

Accelerator

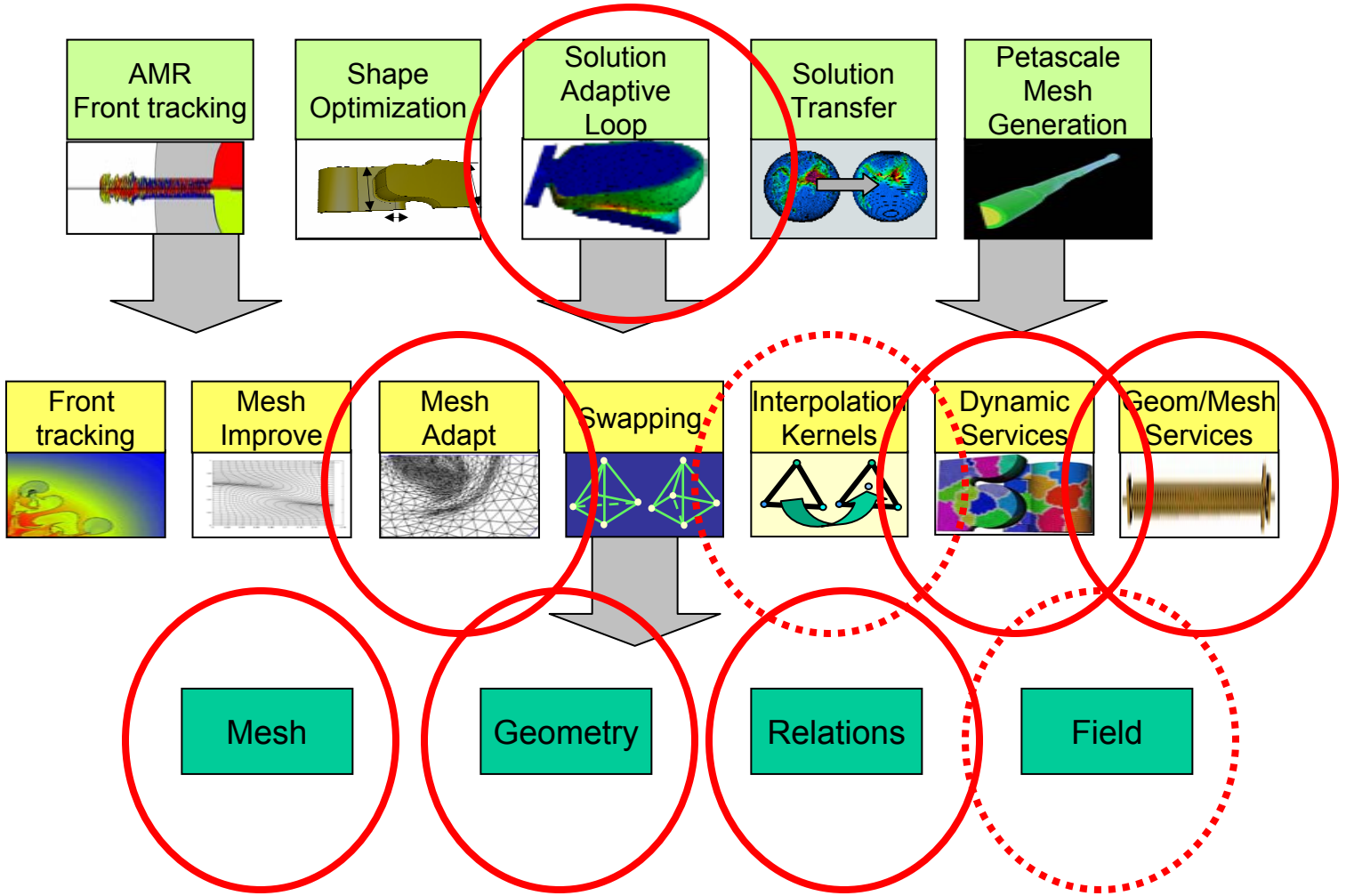
Petascale Integrated Tools

Build on

Component Tools

Are unified by

Common Interfaces



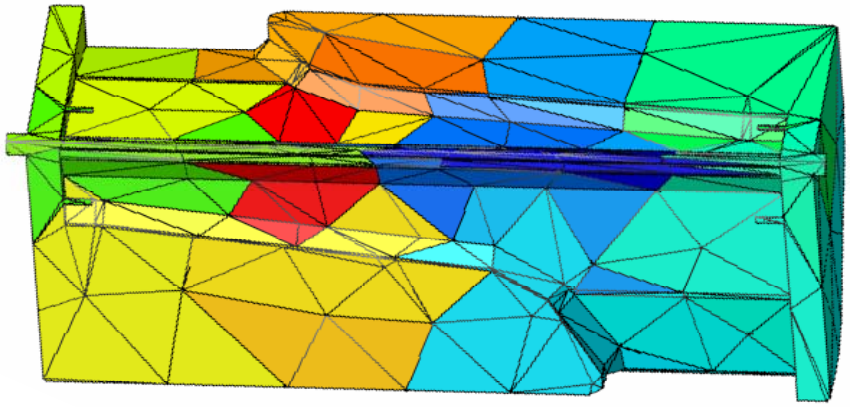
SciDAC
Scientific Discovery through Advanced Computing

Parallel Adaptive Applications

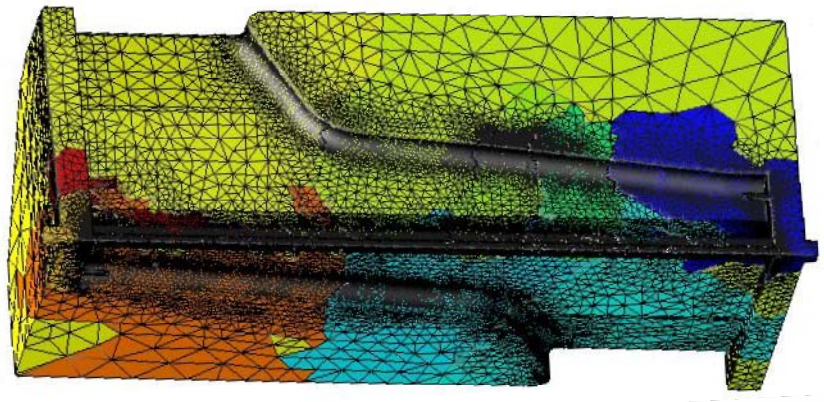
- Parallel adaptive application requires integration of parallel components for
  - Unstructured grid solver
  - Error estimation and mesh size field construction
  - Mesh adaptation
  - Dynamic load balancing with repartitioning
- With these components ITAPS has constructed parallel adaptive applications

Adaptive Loop for SLAC Accelerator Design

- Components:
  - ACIS CAD modeler, Omega3P
  - ITAPS/SCOREC mesh adaptation
  - Error estimations
- Parallel adaptive loop control



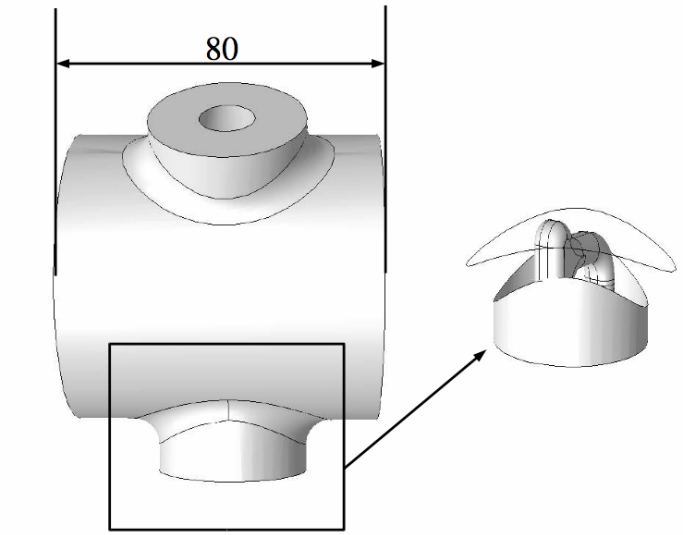
Initial mesh (1,595 tets)

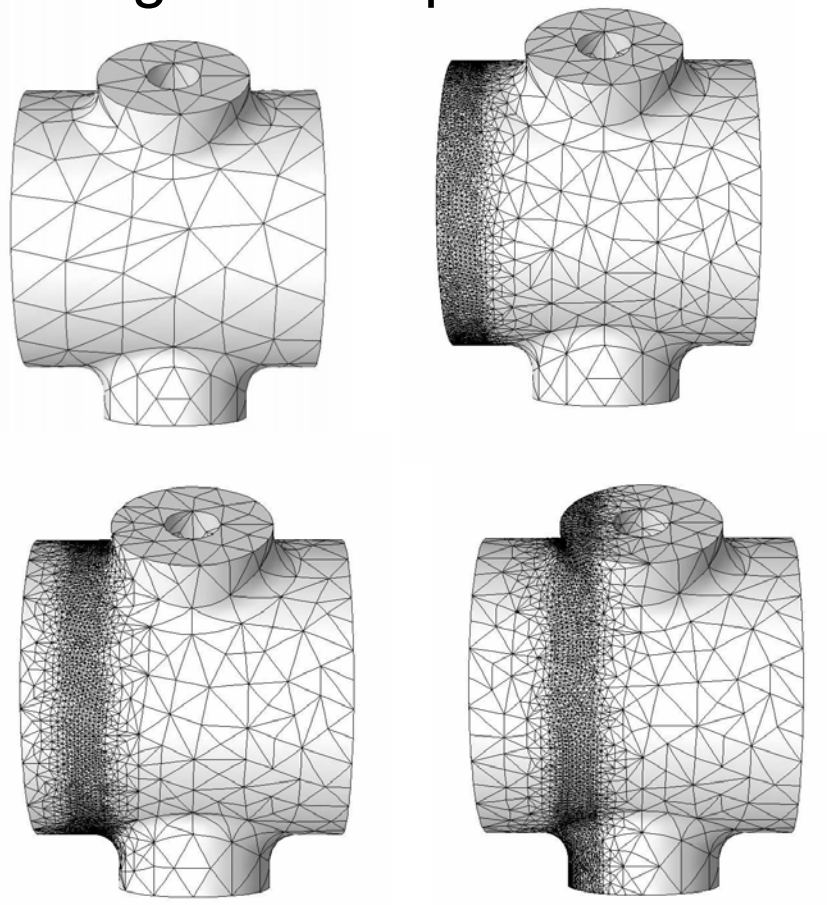


Adapted mesh (23,082,517 tets)

Adaptive Mesh Refinement for SLAC Accelerator Design

- Higher order finite element time domain analysis in SLAC
- Require local mesh refinement around the particle domain
- The mesh refinement moves along with the particle



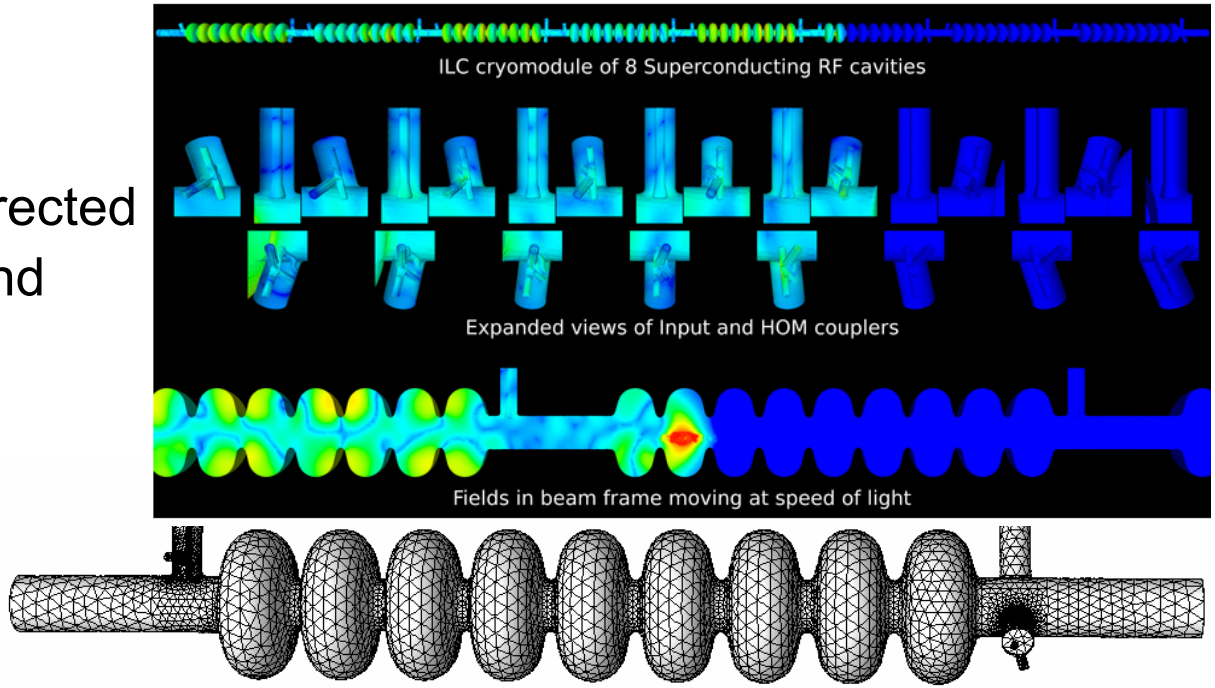


- Controlled refined meshes 1-1.5M
- Uniformly refined meshes 6 M

Adaptive Mesh Curving for SLAC Accelerator Design

- Higher order finite element analysis engine - Omega3P from SLAC
- Valid higher order curvilinear mesh - adaptive mesh curving correction tools from ITAPS/SCOREC
- Improved solution accuracy and efficiency

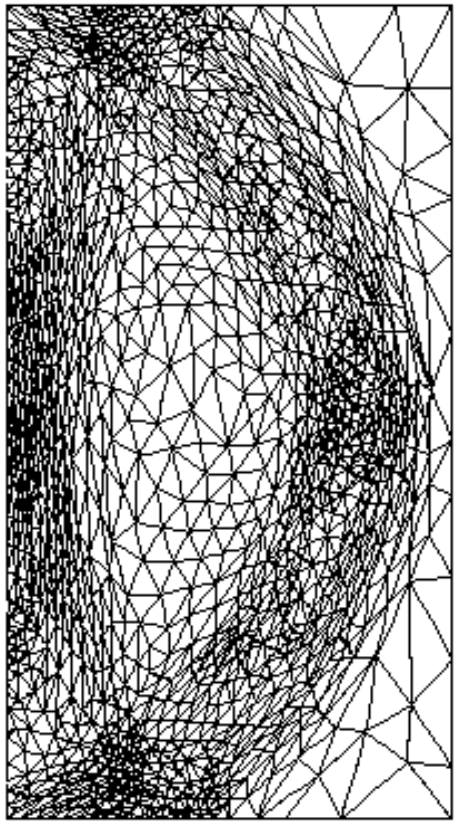
- 2.97 million curved regions
- 1,583 invalid elements corrected
- lead to stable simulation and execute 30% faster

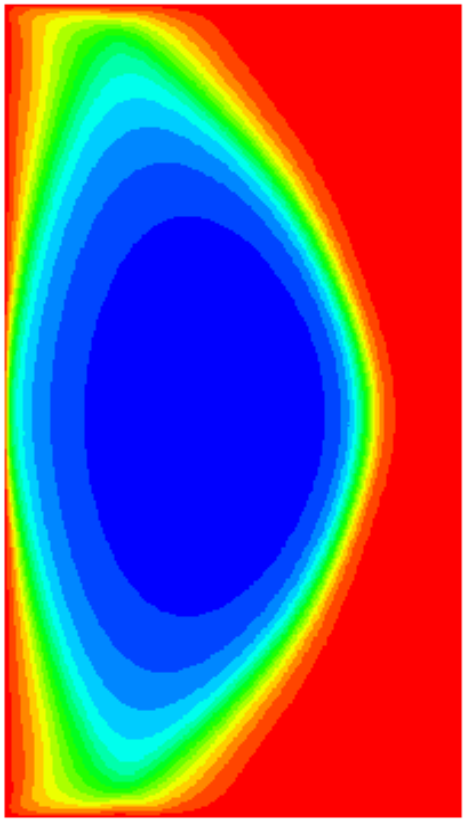


8-cavity cryomodule simulation

Adaptive Loop for PPPL CEMM M3D-C1 MHD Simulations

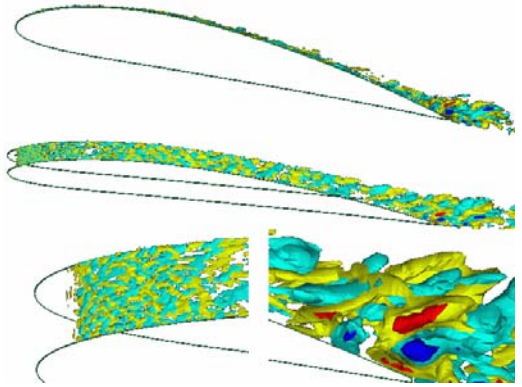
- Restructured M3D-C1 to support general unstructured meshes
- Provided API level for interacting with mesh and solvers (three-way interactions between CEMM, ITAPS and TOPS)
- Developed initial anisotropic mesh adaptation procedure
- Extending code to deal with complex boundary conditions on curved domains using the high order M3D-C1 elements



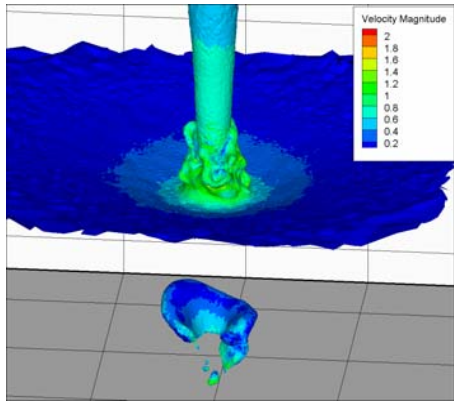


SciDAC Scientific Discovery through Advanced Computing

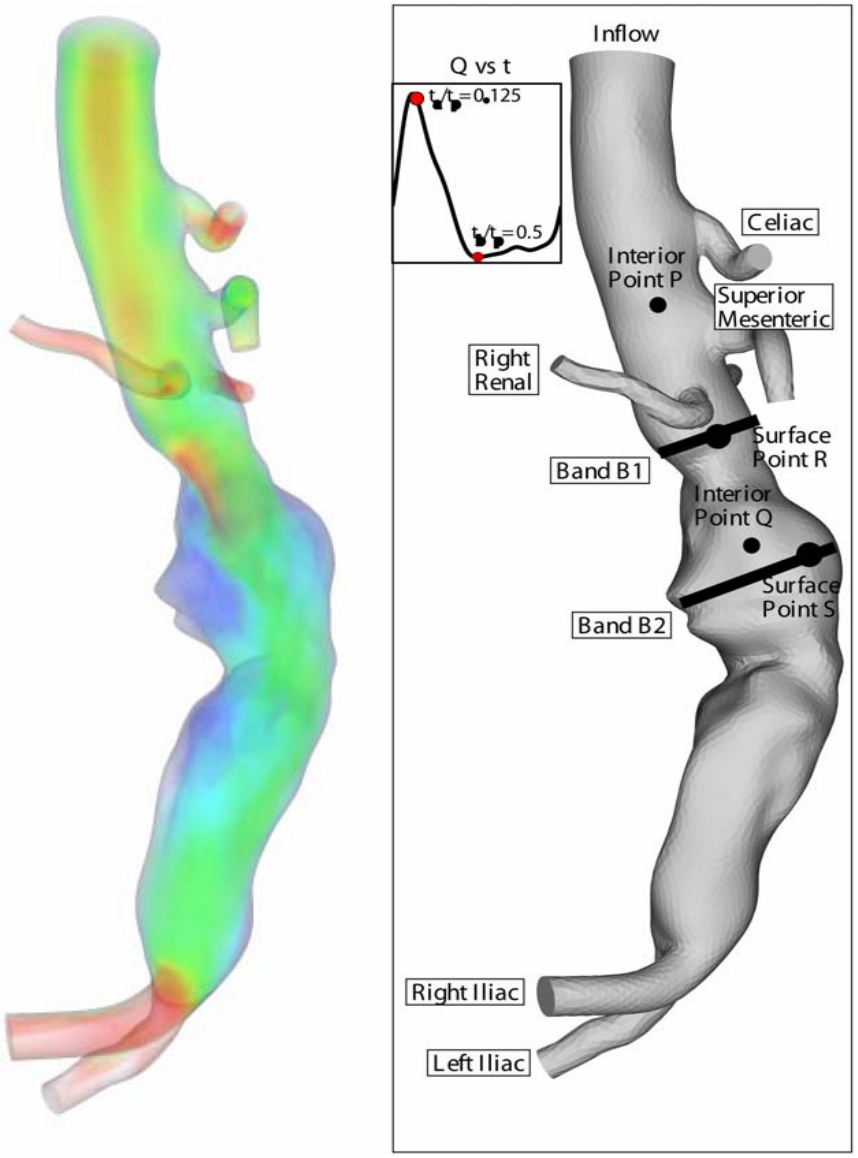
PHASTA Flow Simulation Code



- Parallel finite element flow solver
  - compressible and incompressible flows
  - Turbulence DNS, LES, RANSS/DES
  - Two-phase (immiscible) fluids
- Implicit time integration -
  - solution of linear algebra at each time step using iterative solvers
- Breaks the total domain into parts with roughly the same number of elements on each processor.
- Work can be characterized as requiring:
  - Substantial floating point operations to form equations
  - Organized, substantial, and regular communication between partitions that touch each other
  - For each iteration (O(10) iterations per solve), there is a required ALL-REDUCE communication



Scalability of the Solver

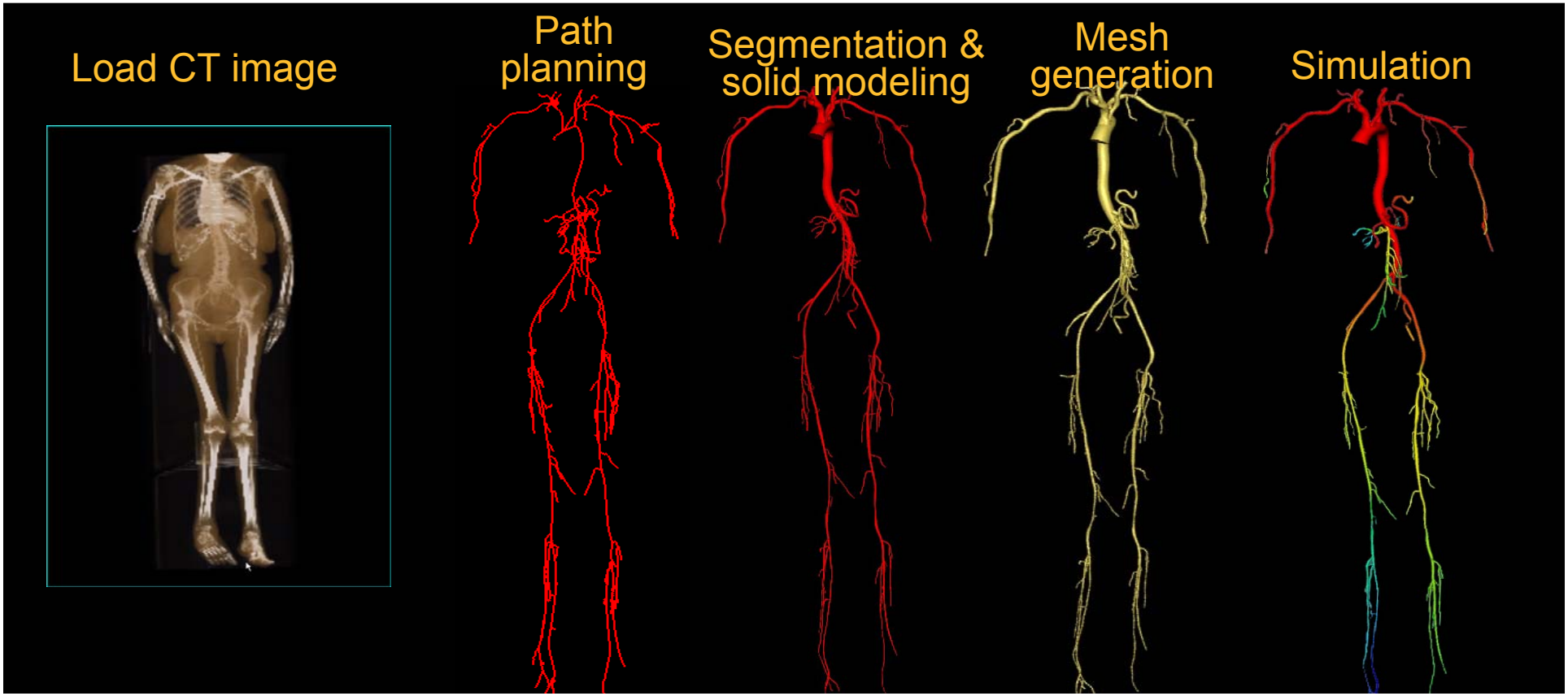


- Adapted mesh had > 50M dof
- Must be solve in 10 min.
  - Implicit FE flow solve scales

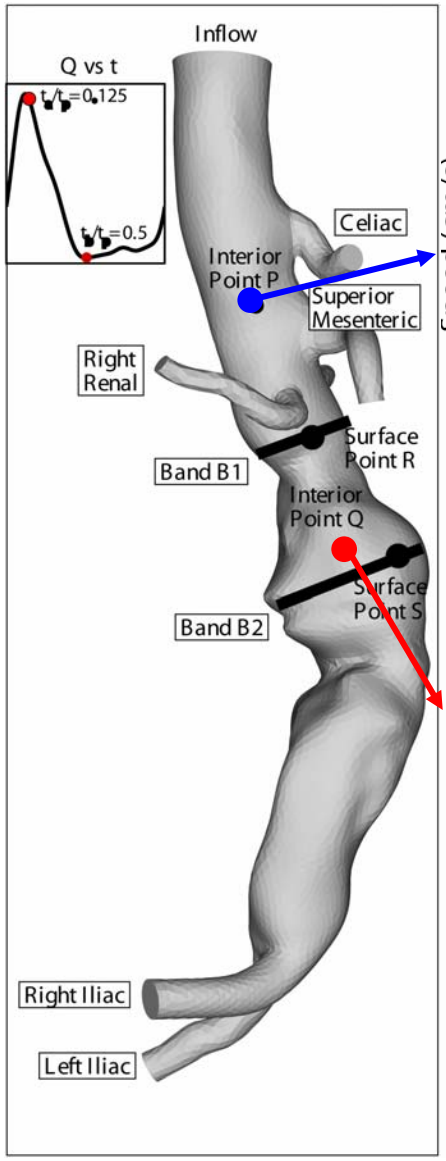
# Proc.	t (sec)	scale
16384	60.6	1.04
8192	131.7	0.957
4096	241.6	1.04
2048	502.3	1.00
1024	1008.7	1.00

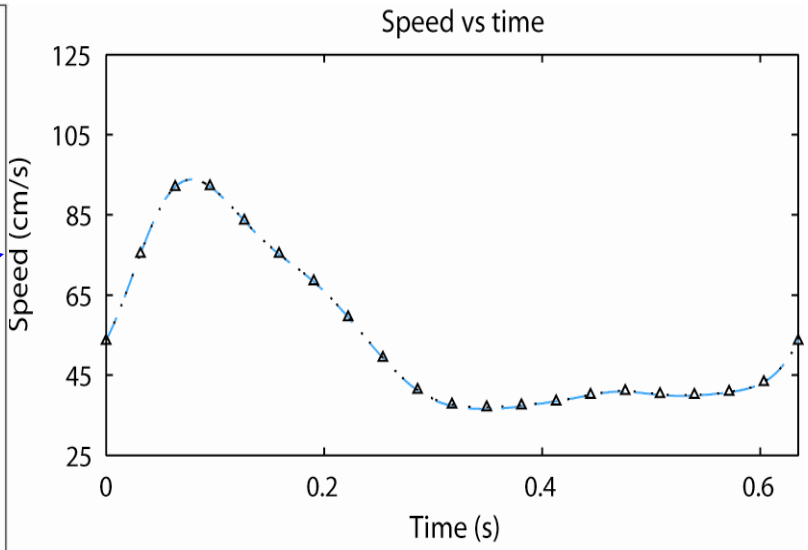
Patient Specific Vascular Surgical Planning (Stanford, RPI)

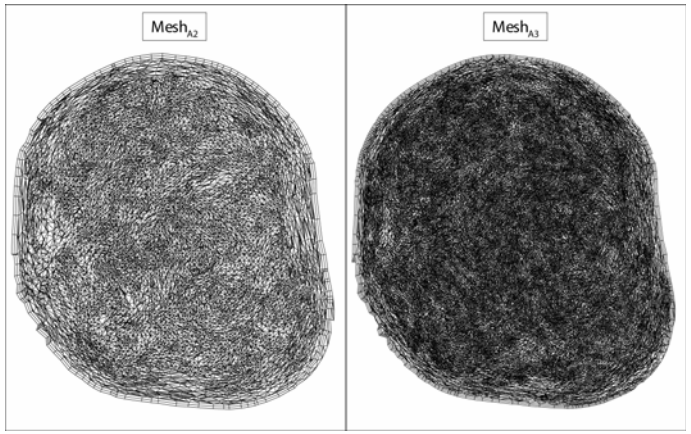
- Virtual flow facility for patient specific surgical planning
- High quality patient specific flow simulations needed quickly
- Image patent, create model, apply adaptive flow simulation



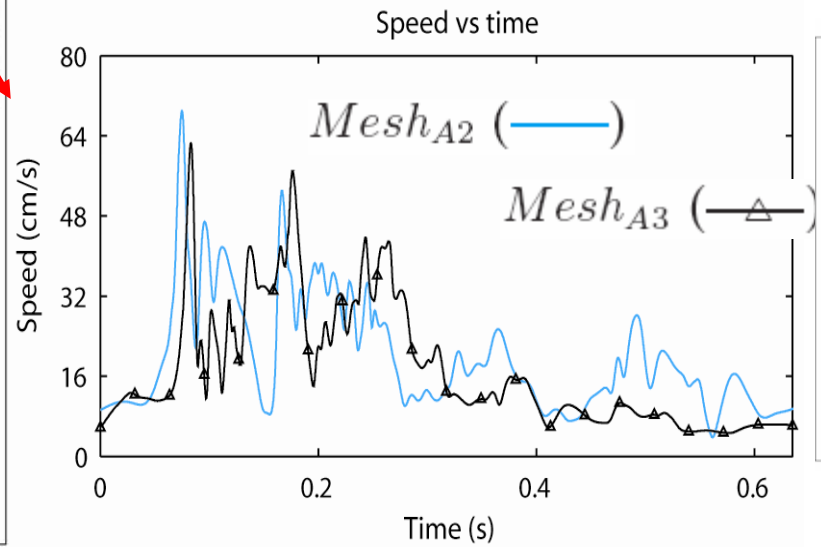
Abdominal Aortic Aneurysm

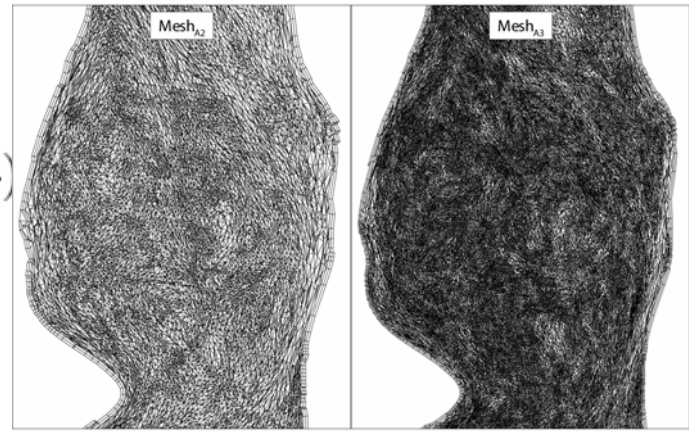






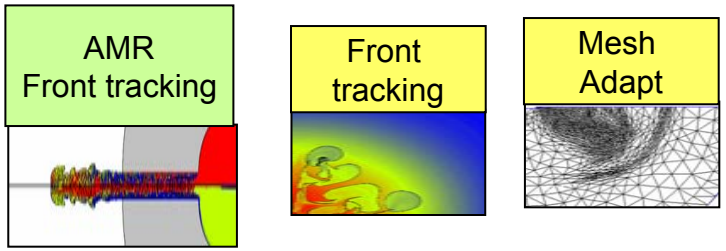
$Mesh_{A2}$ (——) $Mesh_{A3}$ (—△—)





Pellet Ablation for Tokamak Fuelling

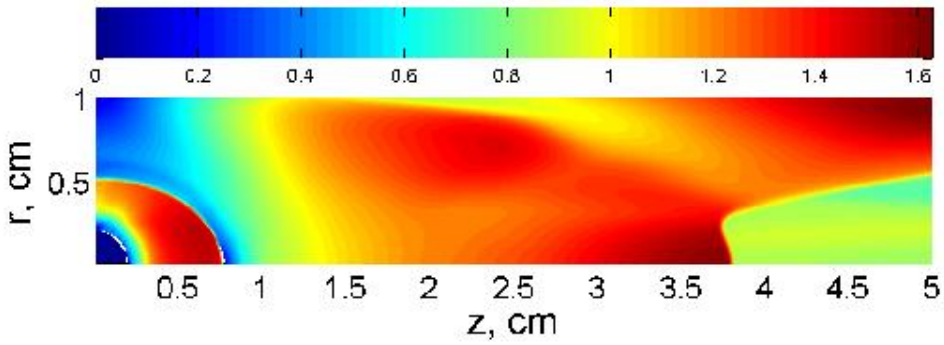
- Goals: Adaptive Meshing and Front Tracking for “microscale” studies of pellet ablation in tokamaks with subgrid models for with macroscale pellet simulation
- People involved: R. Samulyak, T. Lu, BNL P. Parks, General Atomics (Application)
- Based on Front Tracking. Code Includes:
  - Kinetic model 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)
- ITAPS Services Used



SciDAC
Scientific Discovery through Advanced Computing

Pellet Ablation for Tokamak Fuelling

- 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

Adaptive Loop Construction Experiences

- ITAPS interfaces provide needed functionality to support adaptive mesh modifications
  - iMeshP supports mesh interactions
  - Zoltan supports needed dynamic load balancing
  - Interacting with solver for error estimation requires fields
- Reasonable straight forward to construct adaptive loops is solver used mesh partitions
- Approaches taken to date
  - Leave solver unaltered and communicate with files - easy implementation, not optimal efficiency
  - Tight integration into solver - improved efficiency, more learning and development needed to optimize

ITAPS
INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

Case Study 2: Mesh Generation for Nuclear Reactor Simulations

Tim Tautges (ANL)

SciDAC
Scientific Discovery through Advanced Computing




Sandia National Laboratories

Rensselaer

STONY BROOK
STATE UNIVERSITY OF NEW YORK

THE UNIVERSITY OF BRITISH COLUMBIA
BROOKHAVEN NATIONAL LABORATORY
OAK RIDGE NATIONAL LABORATORY
Pacific Northwest National Laboratory

Goals and people involved

- Technical goals – Utilize and integrate the ITAPS Geometry, Meshing, and various mesh services for Nuclear Reactor Simulations

- People involved – Tim Tautges and Valmor de Almeida—ITAPS, Andrew Siegel, Glen Hansen and Kevin Clarno---Nuclear Energy)

ITAPS tools used in nuclear energy

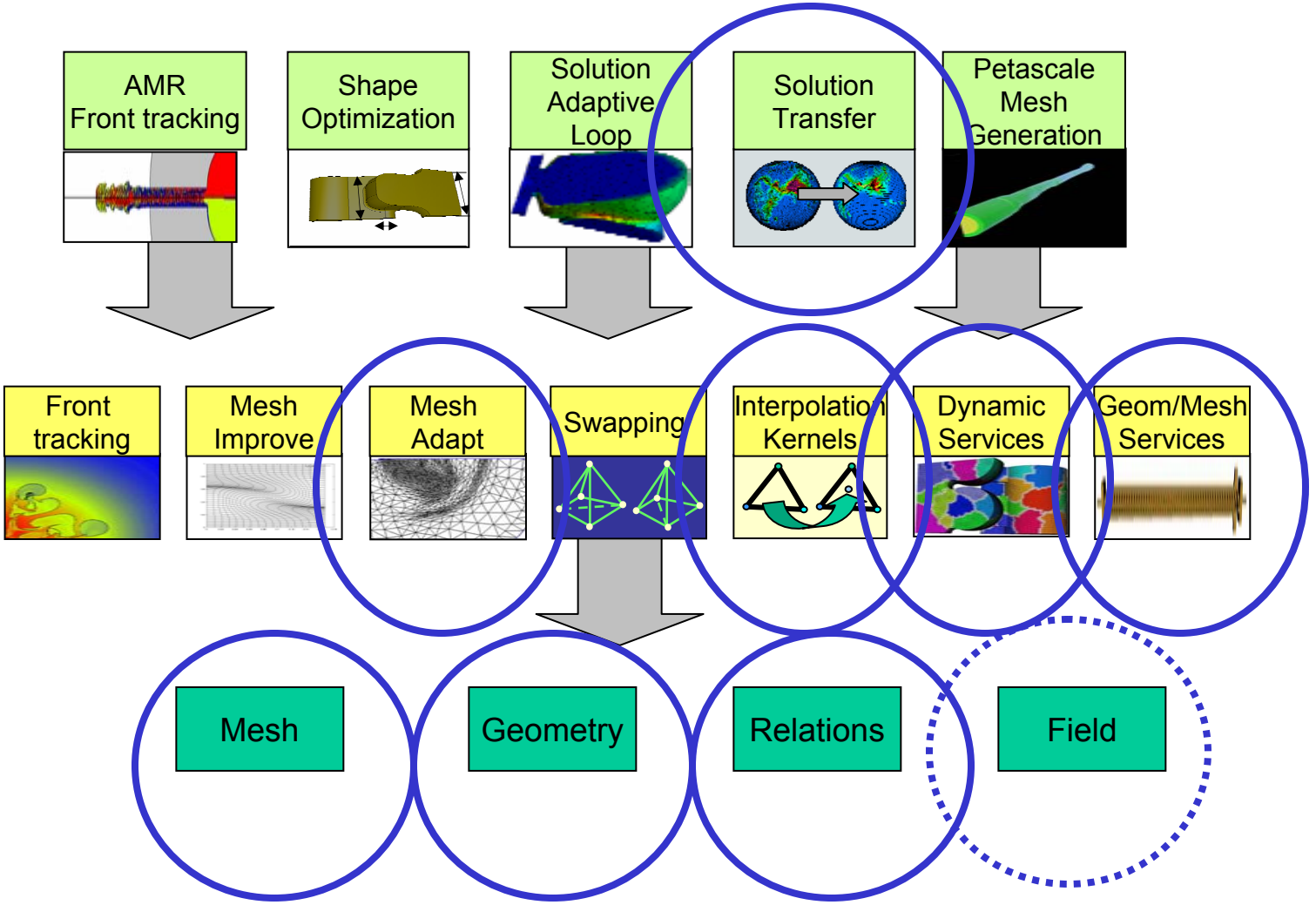
Petascale Integrated Tools

Build on

Component Tools

Are unified by

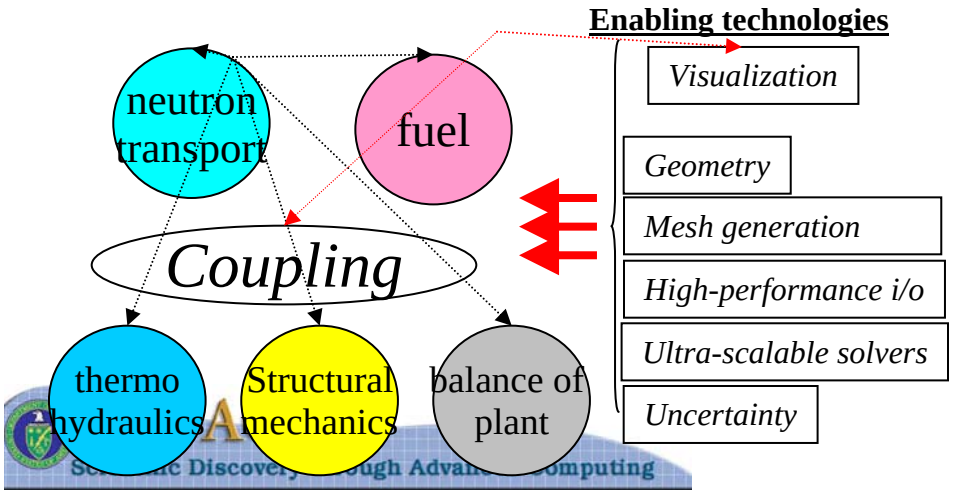
Common Interfaces

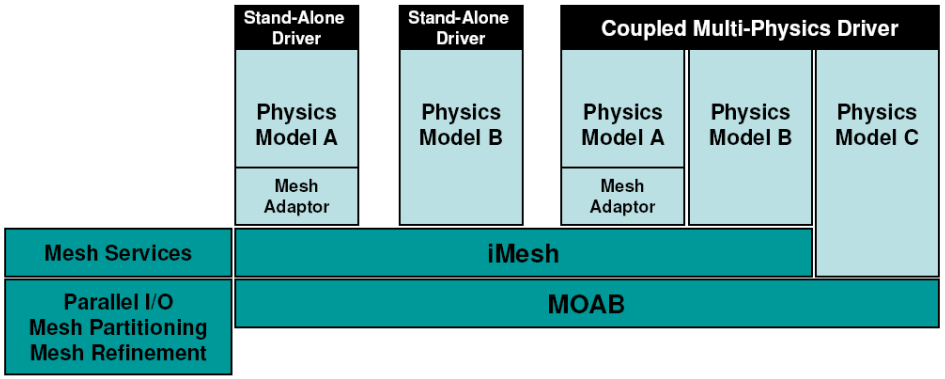


SciDAC
Scientific Discovery through Advanced Computing

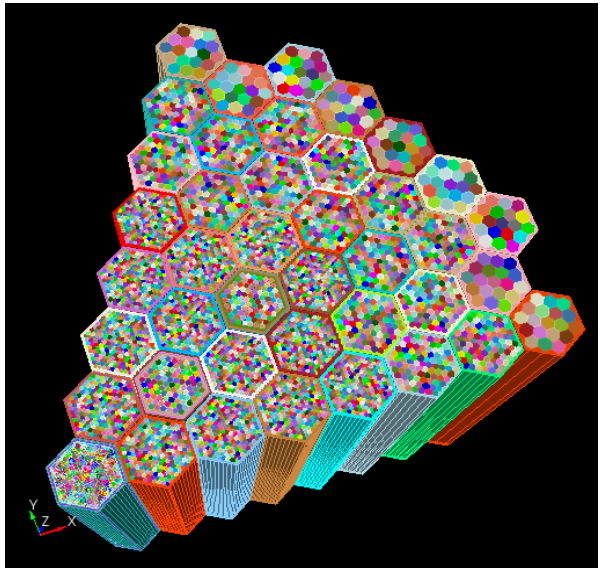
SHARP framework: Multi-Physics Reactor Simulation

- Reactor safety depends on thermal/hydrualics-structural mechanics-neutronics feedback
- Interoperability crucial
  - Interacting with multiple modules
  - Need variety of tools to operate on mesh & field data
- Coupling requires meshes & results in the same place
- Centralized parallel mesh infrastructure needed, to support coupling while allowing modules freedom
- Metadata needed to coordinate coupling (normalization/conservation)



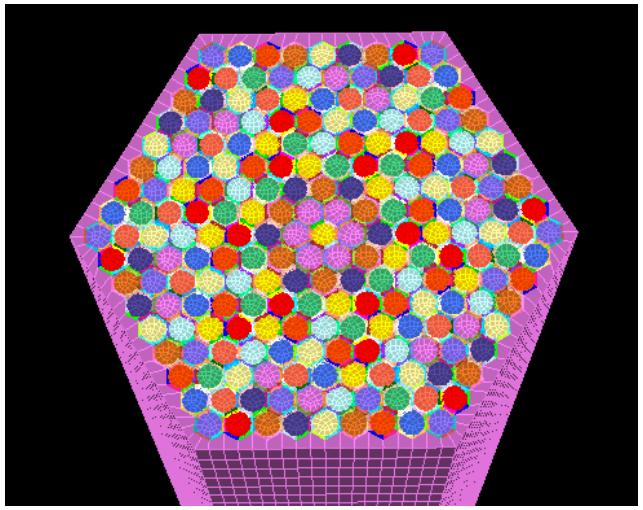


Advanced Burner Test Reactor Mesh Generation



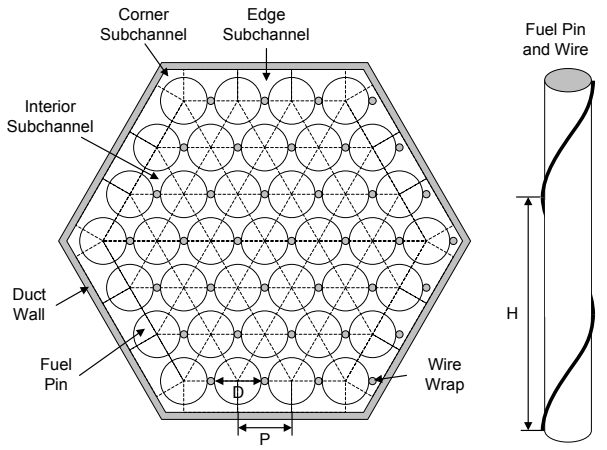
← **1/6 ABTR core**
• 7k volumes (core, ctrl, reflect, shield)
• 43k-5m hex elements
•~6 GB to generate using CUBIT

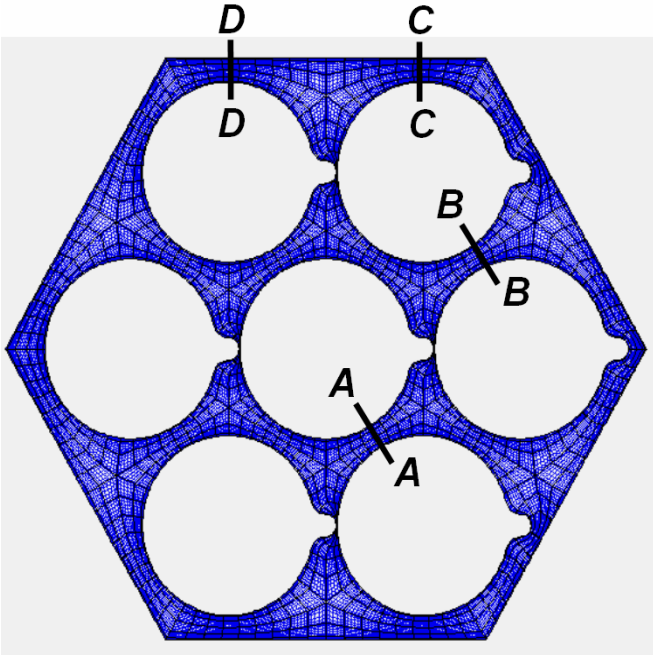
217-pin fuel ass'y →
• Conformal hex mesh
• 1520 vols
• Multiple homogenization options, e.g. pins resolved



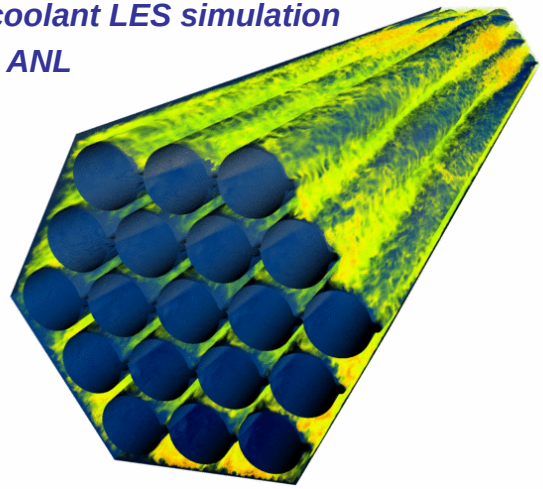
- Varying fidelity geometry, mesh
- Need scalable geometry & mesh generation
- Parallel mesh IO, representation to support ANL UNIC neutron transport code
- Need for mixed quad/tri extrusion, unavailable in CUBIT
- Customized mesh generation would make this easy (simple swept model)

ABTR Mesh Generation
Wire-Wrapped Fuel Pin Assembly





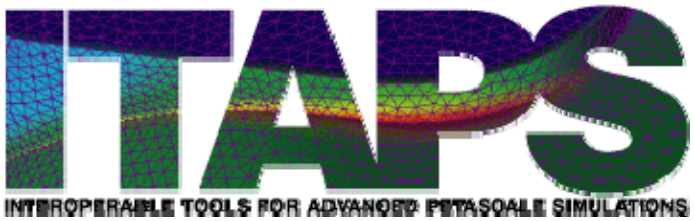
Sodium coolant LES simulation
Nek5000, ANL



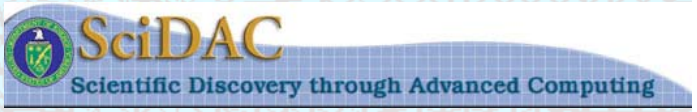
- Currently using MATLAB for meshing
- Wire-pin junction smoothed using fillets
- Mesh “slides” over wires through sweep

Reactor Simulation
Required Mesh Services

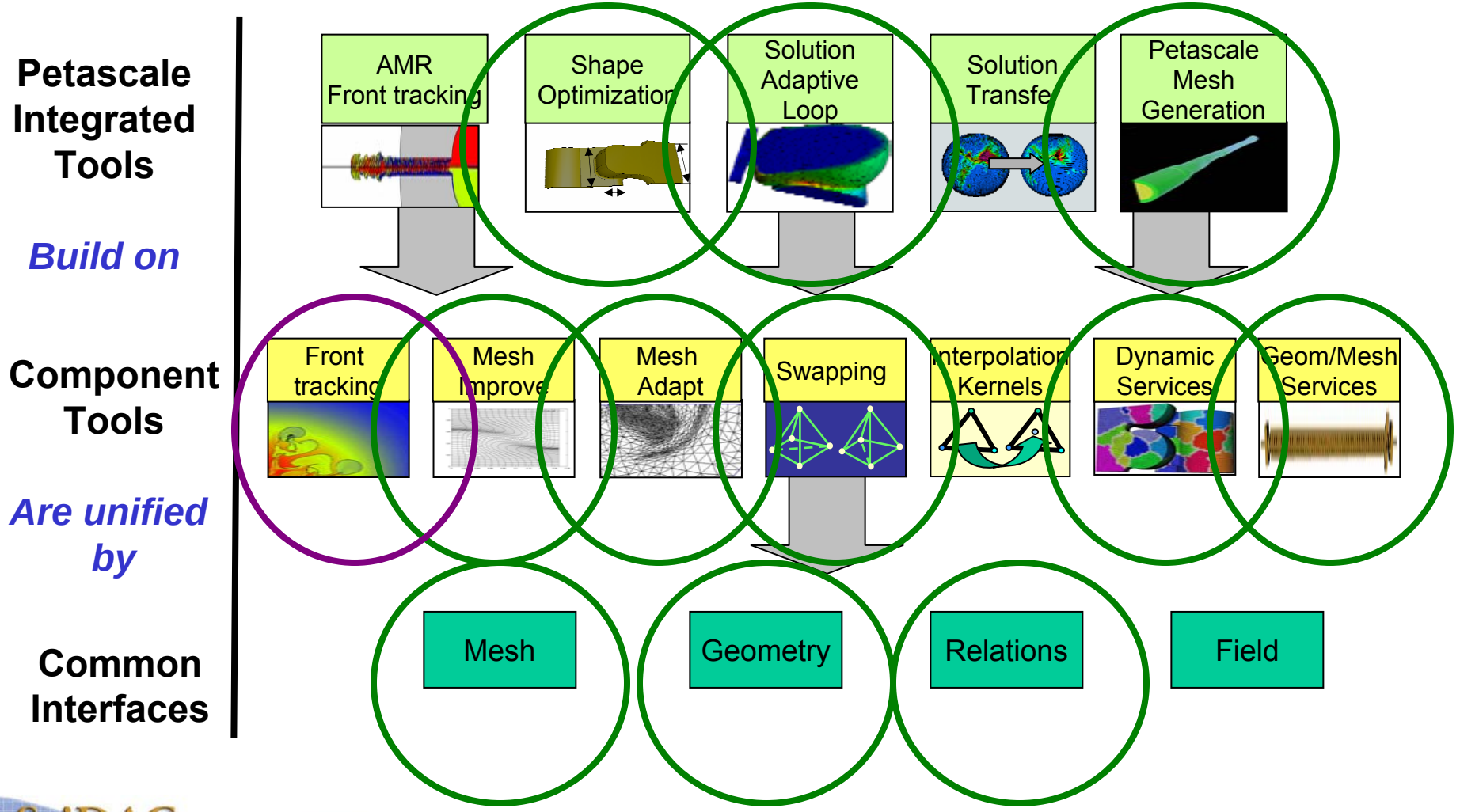
- Robust mesh generation for complex & large geometric models
  - Developing MeshKit, based on iGeom, iMesh, iRel
- Assembly of SHARP framework, by connecting physics modules to iMesh “backbone”
- Coupling of solution results through iMesh
  - Implemented using MOAB data searching functions
  - Will provide data searching as iMesh service eventually
- Partitioning, visualization, other iMesh-based services



Case Study 3: Geometry & Mesh Services for Shape Optimization

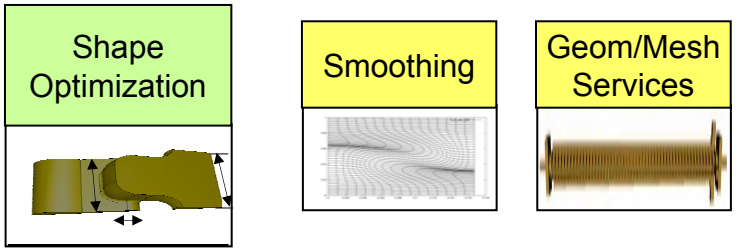



Design optimization for accelerators



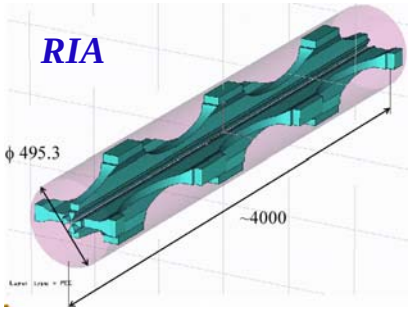
Goals and People Involved

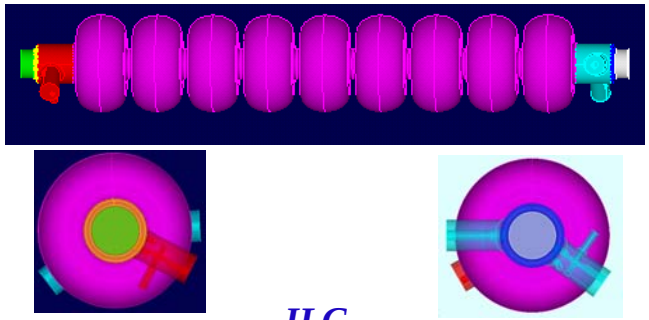
- Technical goals:
  - Develop shape optimization for accelerator cavity tuning and de-tuning, and for reverse engineering as-assembled cavity shape
  - ITAPS provides the geometry control and meshing components
- People involved:
  - SLAC: Rich Lee, Cho Ng, Volkan Ancelik
  - ITAPS: Tim Tautges and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

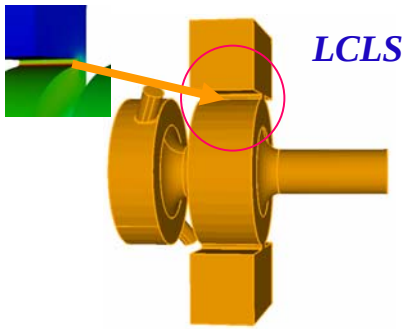


Shape Optimization for Accelerator Cavity Design

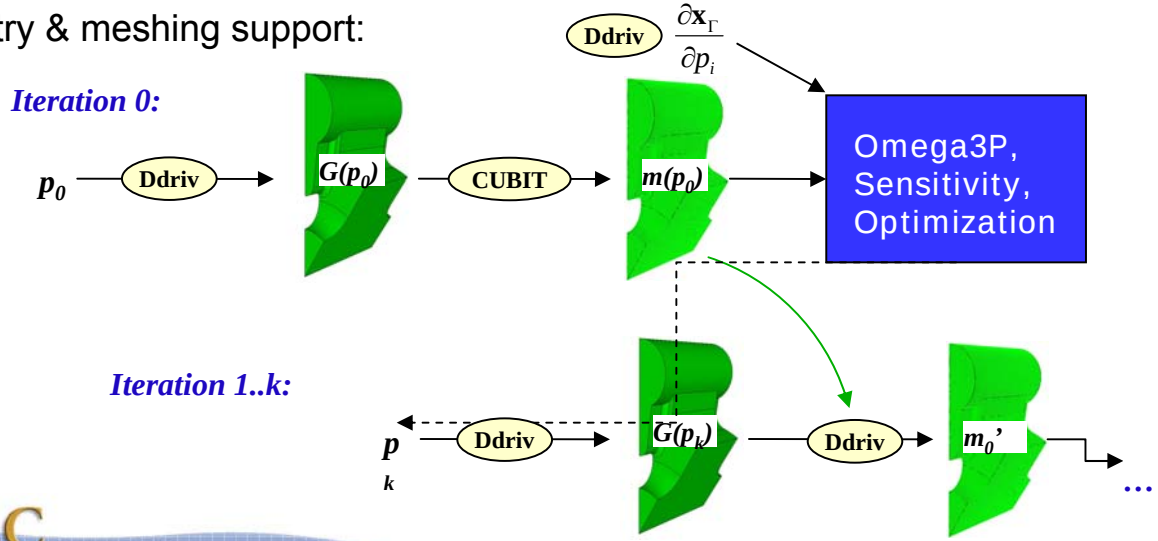
- Optimizing a cavity design is still mostly a manual process
- Future accelerators employ complex cavity shapes that require optimization to improve performance







- Geometry & meshing support:

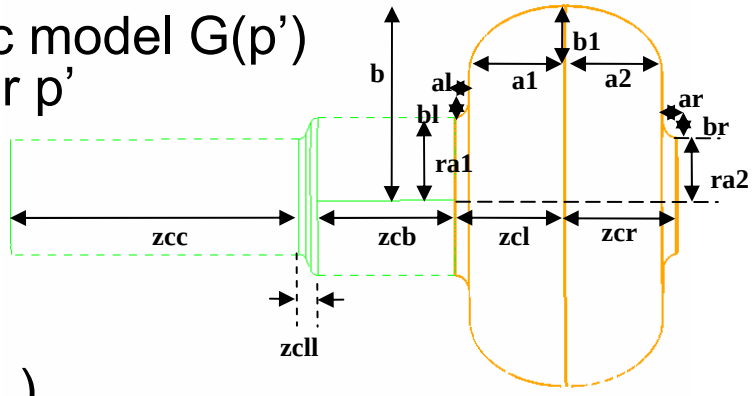


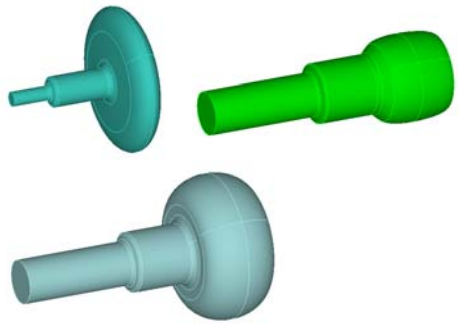
Fixed mesh topology:
Convergence
No re-meshing
Re-use factorization

SciDAC
Scientific Discovery through Advanced Computing

Shape Optimization for Accelerator Cavity Design

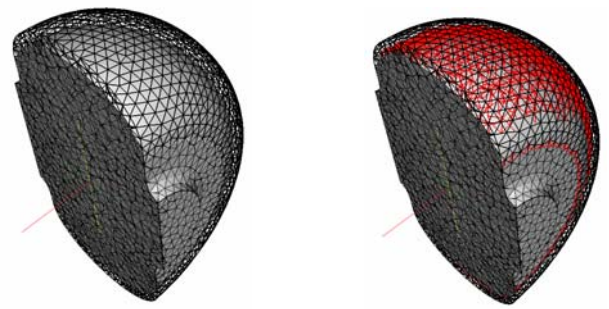
- Generate new geometric model G(p') given a parameter vector p'
  - *MkILCCell function*
  - DDRIV
  - CGM (iGeom)





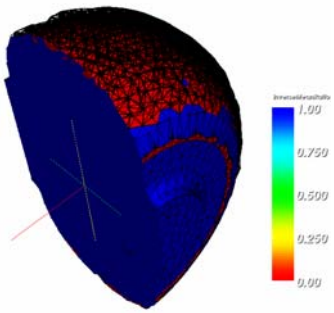
- Associate old mesh $m(p_o)$ to new geometry G(p'), project to CAD
  - DDRIV
  - CGM (iGeom)
  - MOAB (iMesh)
  - LASSO (iRel)

New geom, old mesh

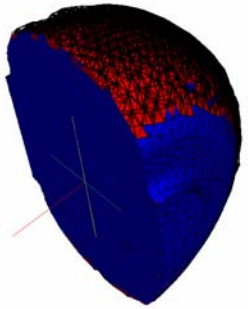


Project to CAD, inverted elements

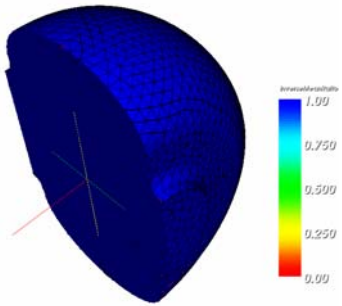
- Smooth mesh
  - …
  - Mesquite



Smooth Curves



Smooth Surfaces

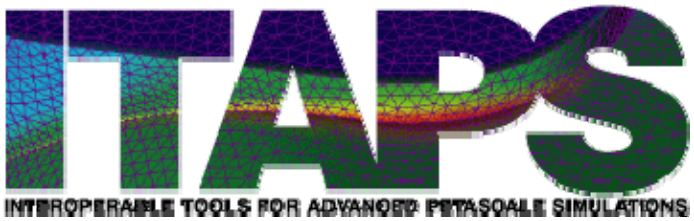


Smooth Volume

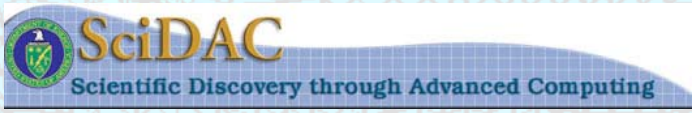
SciDAC
Scientific Discovery through Advanced Computing

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



PART 7: Conclusions




What you learned today

- ITAPS is using a component-based approach to providing mesh services and tools
- The ITAPS interfaces provide an avenue to leverage many existing technologies and a path to incremental adoption

None of this would be possible without a strong team of contributors

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, Martin Isenburg**
- **PNNL: Harold Trease**
- **RPI: Mark Shephard, Ken Jansen, Eunyoung Seol, Xiaojnan Luo, Ting Xie, Onkar Sahni**
- **SNL: Vitus Leung, Karen Devine**
- **SUNY SB: Xiaolin Li, Brian Fix, Ryan Kaufman**
- **UBC: Carl Ollivier-Gooch**
- **U Wisconsin: Jason Kraftcheck, Jane Hu**

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:

Karen Devine, SNL
kddevin@sandia.gov

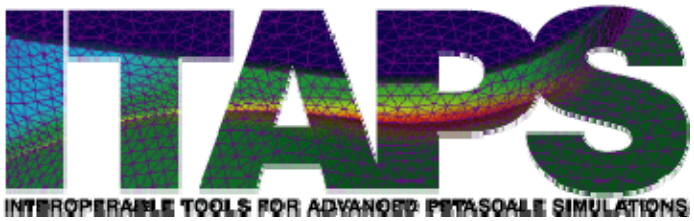
Lori Diachin, LLNL
diachin2@llnl.gov

Mark Shephard, RPI
shephard@scorec.rpi.edu

Tim Tautges, ANL
tautges@mcs.anl.gov

Auspices and disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.



AFTERNOON SESSION HANDS ON