FMDB: Flexible Distributed Mesh Database for Parallel Automated Adaptive Analysis

K.E. Jansen, A. Ovcharenko, O. Sahni,
M.S. Shephard, Ting Xie, Min Zhou,
with initial development by E.S. Seol

Scientific Computation Research Center

Rensselaer Polytechnic Institute

Outline

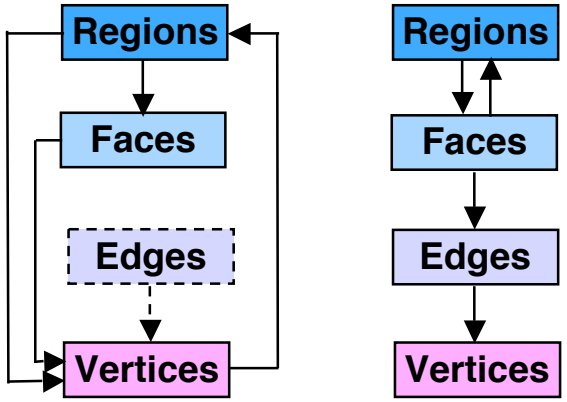
- Flexible Mesh Data Base (FMDB)
- Partition model to support FMDB for parallel adaptive analysis
- Recent developments to support adaptive applications
  - Definition of parts and objects to be partitioned
  - Importance of element and “FE node” balance
  - Parallel Performance
  - Application to large processor counts on large meshes

This work is supported by the DOE SciDAC program as part of the Interoperable Technologies for Advanced Petascale Simulations, and NSF through a petascale application grant

SCOREC

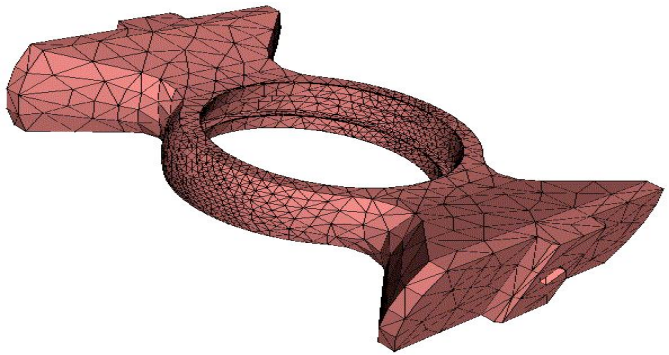
Meshes

- A piece-wise domain decomposition over which the simulation is to be run
- A mesh data structure provide services to create and/or use the mesh data
- Each application has its own needs of mesh representation in terms of levels of entities and adjacencies used
  ⇒ *"flexibility in mesh representations"*
- 3 approaches for mesh data structure design
  - Fixed, specific mesh representation
  - Fixed, general mesh representation
  - Flexible mesh representation



Flexible Mesh Data structure

- Switch between various representations for different needs of applications
- Achieving a good performance both in memory and computational cost



SCOREC

Nomenclature

G	the geometric model
P	the partition model
M	the mesh
$\{V\{V^d\}\}$	a set of topological entities of dimension d in model $V \in \{G,P,M\}$
V_i^d	the i^{th} entity of dimension d in model V. $d=0$ for a vertex, $d=1$ for an edge, $d=2$ for a face and $d=3$ for a region. Edges, faces, and regions are bounded by lower order entities. A shorthand for $V\{V^d\}_i$
$\{\partial(V_i^d)\}$	the entities on the boundary of V_i^d
$\{V_i^d\{V^q\}\}$	the set of entities of dimension q in model V that are adjacent to V_i^d
$V_i^d\{V^q\}_j$	the j^{th} entity in $\{V_i^d\{V^q\}\}$

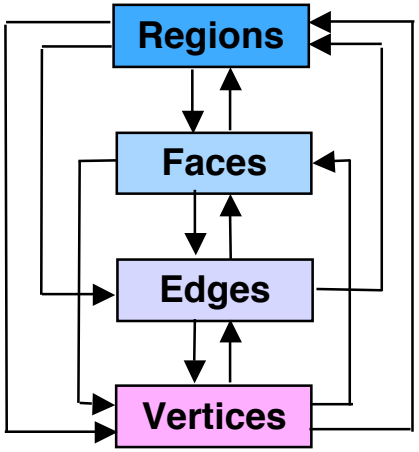
(Examples)

$\{M\{M^2\}\}$	a set of faces in the mesh
$\{M_3^1\{M^3\}\}$	a set of mesh regions adjacent to the mesh edge M_3^1
$M_1^3\{M^1\}_2$	the 2^{nd} edge adjacent to the mesh region M_1^3

General Topology-based Mesh Data Structure[1]

Functional Requirements

- *Topological entities*
  - $\{M\{M^d\}\}$, *d=0,1,2,3*, for vertices, edges, faces, regions respectively
  - Edges, faces, and regions are bounded by the lower order entities
- *Geometric classification*
  - Unique association of entity $M_i^{d_i}$, to a geometric model entity, $G_j^{d_j}$, where $d_i \leq d_j$, is denoted by $M_i^{d_i} \sqsubset G_j^{d_j}$
  - $\sqsubset$ indicates the left-hand entity (or set) represents a portion of the right-hand entity in the discretization
  - Multiple $M_i^{d_i}$ classified on a $G_j^{d_j}$
- Adjacencies
  - Connectivity between entities forming a graph
  - $\{M_i^d\{M^q\}\}$, $d \neq q$
    - adjacent entities to M_i^d of dimension q
    - If $d > q$, *downward* adjacency
    - If $d < q$, *upward* adjacency

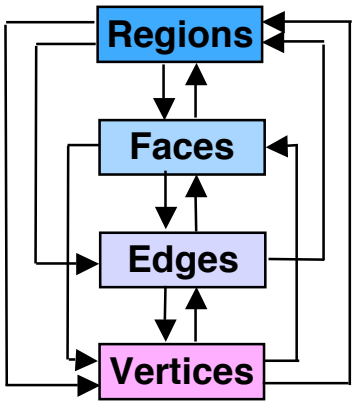


12 adjacencies

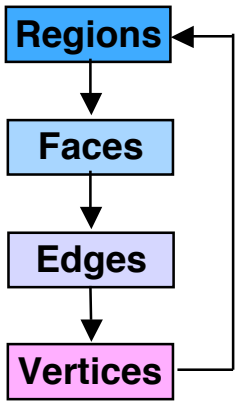
Mesh Representation Options

Categories

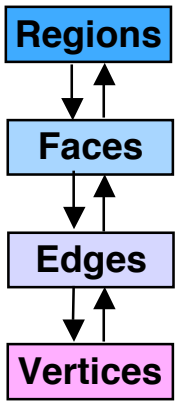
- Full vs. Reduced
  - If a mesh representation stores all *0* to *d* level entities, it is full, otherwise, reduced.
- Complete vs. Incomplete
  - If 12 adjacencies are obtainable in O(1) time, it is complete, otherwise, incomplete.



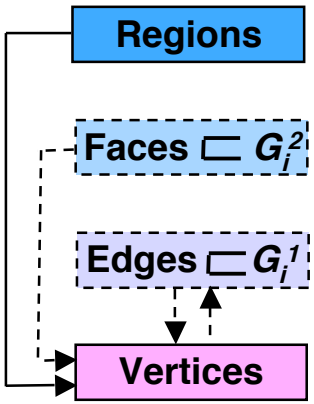
greedy



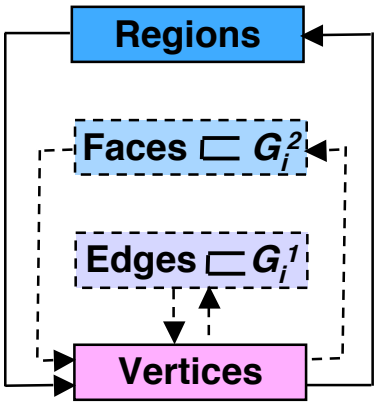
circular[1]



one-level[1]



MSR[2]
(incomplete)



Complete MSR[3]

---

1 M.W. Beall and M.S. Shephard (1997) Int. J. Numer. Mech. Engng
2 J.F. Remacle et al (2003) Int. J. Numer. Mech. Engng
3 W. Celes et al (2005) submitted, Int. J. Numer. Mech. Engng

SCOREC

Analysis of Mesh Data Structure

Memory Efficiency

N_0 : # entities of level *i* in a mesh

S_{ent} : cost for entity data

S_{adj} : cost for adjacency

- Storage(M) = $\sum_{d=0}^{3}\sum_{i=1}^{N_d} Storage(M_i^d)$

$$= \sum_{d=0}^{3}\sum_{i=1}^{N_d}\left(S_{ent} + \sum_{q=0}^{3,q\neq d} \left|\{M_i^d\{M^q\}\}\right| \times S_{adj} \right)$$

  - Storage(greedy) = $25N_0S_{ent}+342N_0S_{adj}$
  - Storage(circular) = $25N_0S_{ent}+93N_0S_{adj}$
  - Storage(one-level) = $25N_0S_{ent}+155N_0S_{adj}$

- Example: 1m Tet, N_0=194k, S_{ent}=70 bytes, S_{adj}=4 bytes

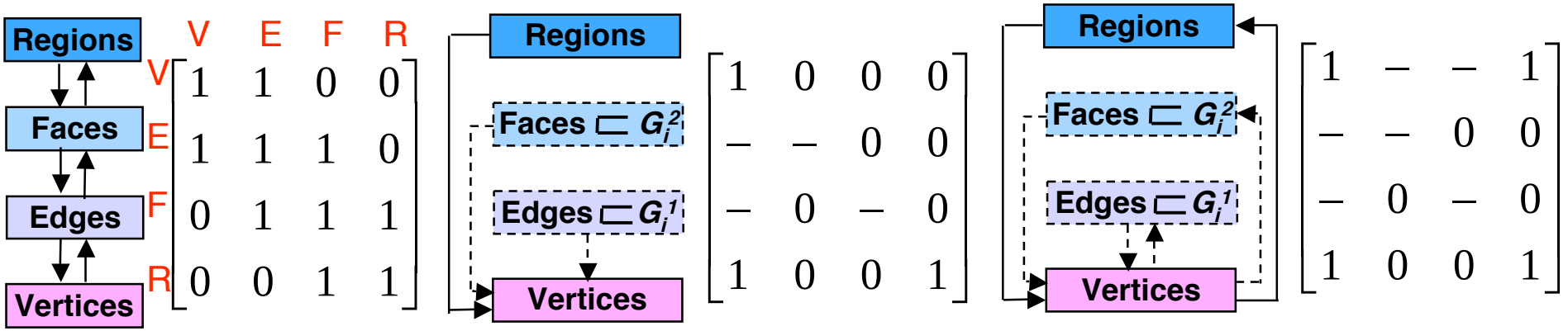
Representation	Storage Requirements (MB)		Storage decrease (%)	
	By computation	By experiment	By computation	By experiment
One-level	421.9	420.7	-	-
complete MSR	122.7	116.4	75.3	77.1
MSR	104.2	96.4	70.9	72.3

- Analysis of 22 mesh operators
  (*ex*) *12 adjacencies, entity creation, entity existence check*
  - Greedy – 1,685 steps
  - Circular – 16,771 steps
  - One-level – 3,359 steps
  - Complete MSR – 148,008 steps

Flexible Mesh Data Structure

Mesh Representation Matrix (MRM) $\mathcal{R}$

- A 4×4 matrix representing the needs of mesh representation
- *Diagonal element* $\mathcal{R}_{i,i}$
  - 1, if {M{M^i}} is present in the representation
  - –, if only {M{M^i}} classified on G_j^i is present in the representation
  - 0, if {M{M^i}} is not present in the representation
- *Non-diagonal element* $\mathcal{R}_{i,j}$
  - 1, if $\mathcal{R}_{i,i}=\mathcal{R}_{j,j}=1$ and {M^i{M^j}} is present in the representation
  - –, if {M^i{M^j}} present only for stored {M^i{M^i}} and {M^j{M^j}}
  - 0, if {M^i{M^j}} is not stored at all
- Examples



One-level **MSR** **Complete MSR**

Flexible Mesh Data Structure

Requirements of the Flexible Mesh Data Structure

- The representation should be maintained even with mesh modification
  - Mesh entity creation
  - Mesh entity deletion
  - Mesh entity migration
- Restored implicit entities and adjacencies should be *valid*
- Cost of any mesh operator should not be a function of mesh size *except non user-requested adjacencies*

Shape mesh data structure by *setting mesh modification operators to the proper ones that keep the representation correctly.*

(It satisfies all 3 requirements listed)

SCOREC

Distributed Mesh Data Structure

Distributed Mesh

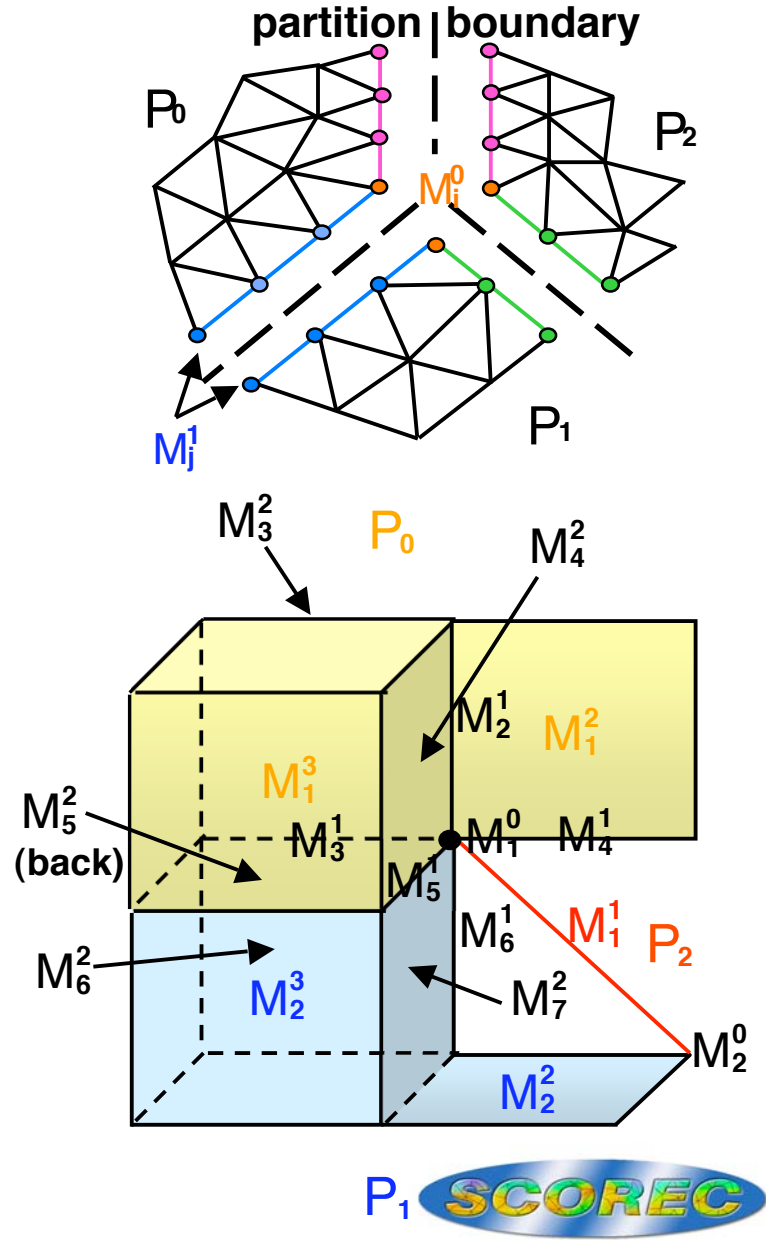
- Mesh divided into parts for distribution on parallel computers
- Part P_i consists of a set of mesh entities assigned to the i^{th} part.

Partition Object

- The basic unit to which a part is assigned.
- A mesh entity to be partitioned
  - *Mesh entities to be partitioned in the example mesh are* $\{M_1^3, M_1^2, M_2^3, M_2^2, M_1^1\}$
- An EntityGroup.

Residence Part

- Operator $\mathcal{P}[M_i^d]$ returns a set of part id's where M_i^d exists. (e.g. $\mathcal{P}[M_1^0] = \{P_0, P_1, P_2\}$)
- Residence part of M_i^d *on* P_i
  - If a partition object, $\mathcal{P}[M_i^d] = \{P_i\}$
  - Otherwise, $\mathcal{P}[M_i^d] = \mathrm{U}\ \mathcal{P}[M_j^q | M_i^d \in \{\partial(M_j^q)\}]$



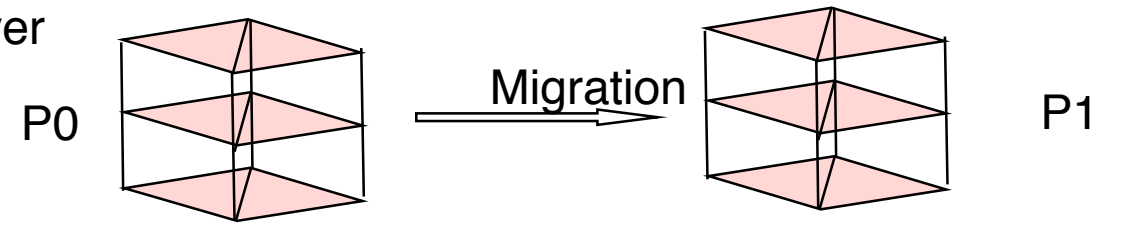
Partition Object

Mesh entity to be partitioned:

- Mesh entity that is not part bounded by any higher order mesh entity.

EntityGroup:

- A group of *mesh entities to be partitioned* that need to stay together during the lifetime of the EntityGroup as defined by the needs of an application.
  - Example - stack of prismatic elements in a boundary layer to support the adaptation of the layer



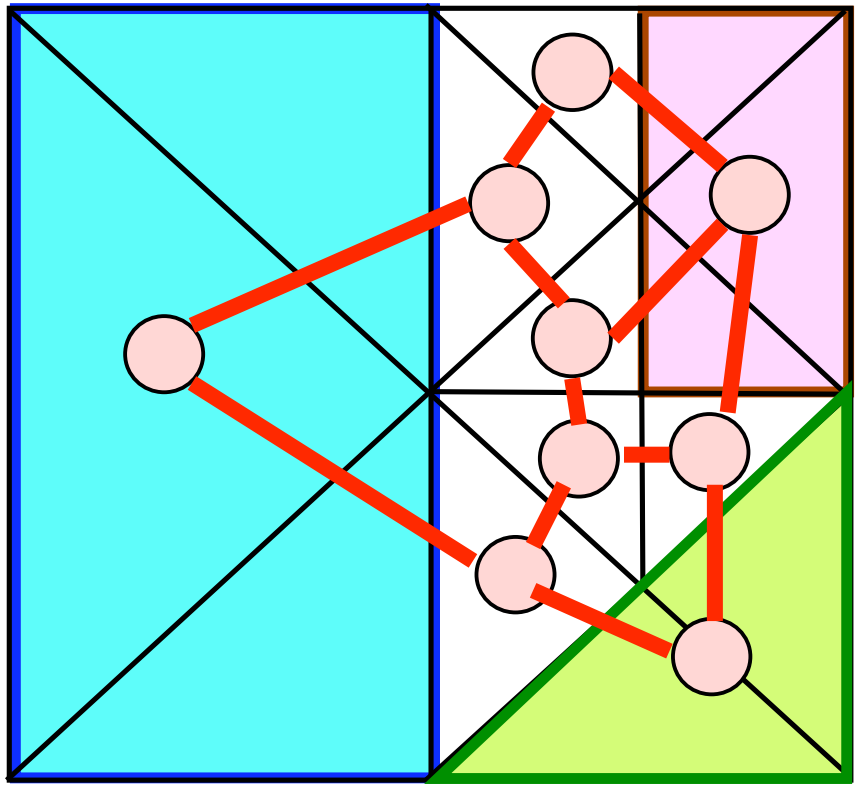
Entity group rules

- Mesh entities in a group stay as a group during the life time of EntityGroup
- A mesh entity can only be in a single EntityGroup, and is defined once in the EntityGroup
- EntityGroup information maintained before and after migration
- EntityGroup is dynamic as defined by the application which can create and destroy an EntityGroup, or add/remove mesh entities in an EntityGroup

Mesh Partition through Zoltan

Perform mesh partition through **Zoltan** **graph-based** partitioning tool, and will use **Zoltan hypergraph-based** partitioning tool in the near future.

In mesh partitioning, a partition object can be either a mesh entity to be partitioned, or an EntityGroup.



Different colors represent different EntityGroup's
(3 EntityGroup's in the 2D mesh).

Construct graph for mesh partition:

- **Graph nodes** = objects to be partitioned (partition object).
- **Graph Edges** = mesh edge-based dependencies between two objects.
- **Weights** = Set graph node and graph edge weights.

Distributed Mesh Representation

Functional Requirements

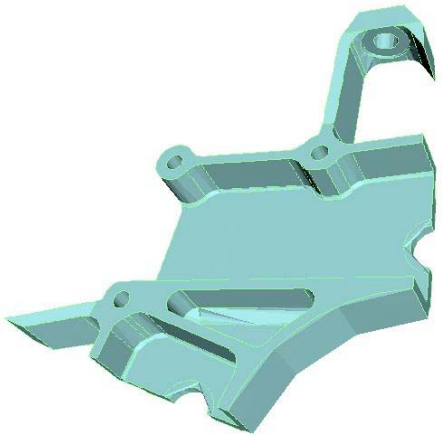
- Communication links
  - Remote part: non-self part where an entity is duplicated
  - Remote copy: the memory location of the entity duplicated on the remote part
  - Efficient mechanisms to update mesh partitioning and keep the links between partitions are mandatory
- Entity ownership
  - Control communications and computations between remote copies
  - Static ownership
    - Owner part of an entity is fixed to the specific partition regardless of mesh partitioning
    - Not suitable for adaptive analysis due to severe load imbalance
  - Dynamic ownership
    - Owner part of an entity is determined dynamically depending on mesh partitioning
    - *poor-to-rich part ownership*

SCOREC

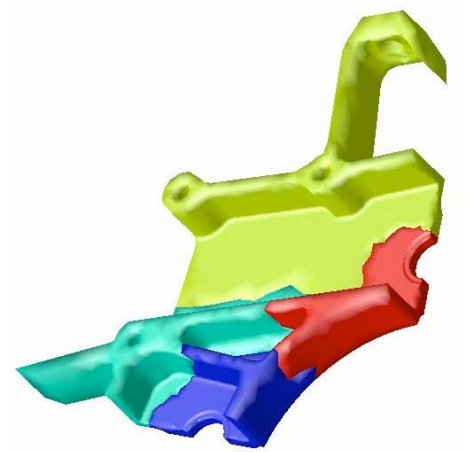
A Partition Model

Partition Model

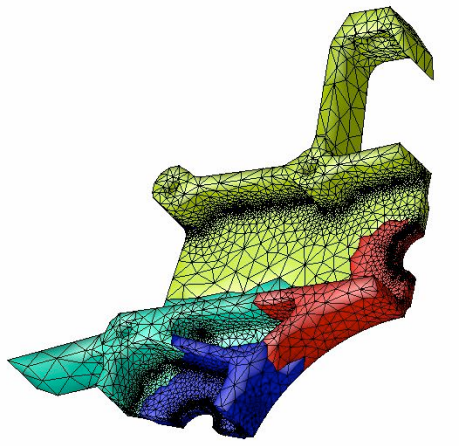
- Purpose
  - Represent mesh partitioning in topology
  - Support mesh-level inter-partition communications
- An intermediary model located between the geometric model and mesh
- A set of topological entities (*0* to *d* dimensional entities) representing the collection of mesh entities that lie on part boundaries.



Geometric model



Partition model



Partitioned mesh

SCOREC

Distributed Mesh Data Structure

Partition Classification

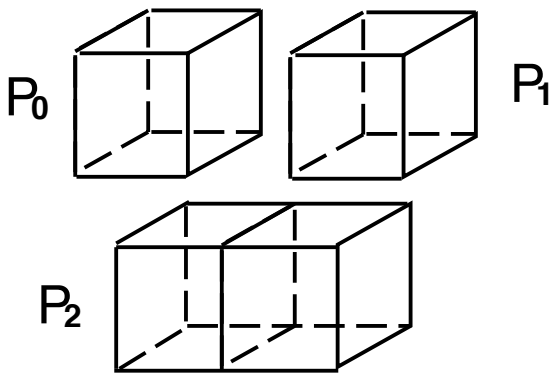
- The unique association of mesh entities to the partition model entities
- In a hierarchy of domain decomposition,

, $d_i \le d_j \le d_k$

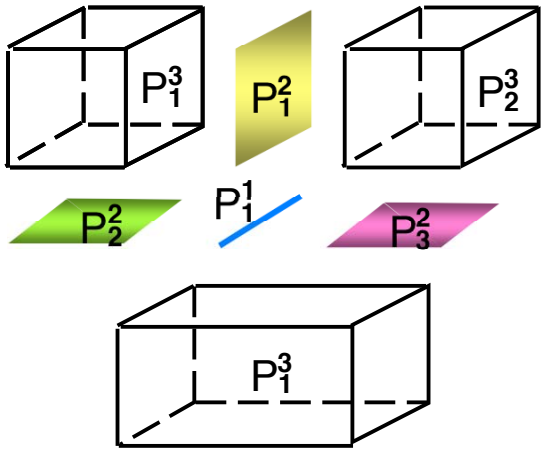
$$M_i^{di} \sqsubset P_j^{dj} \sqsubset G_k^{dk}$$

Rules to Construct a Partition Model

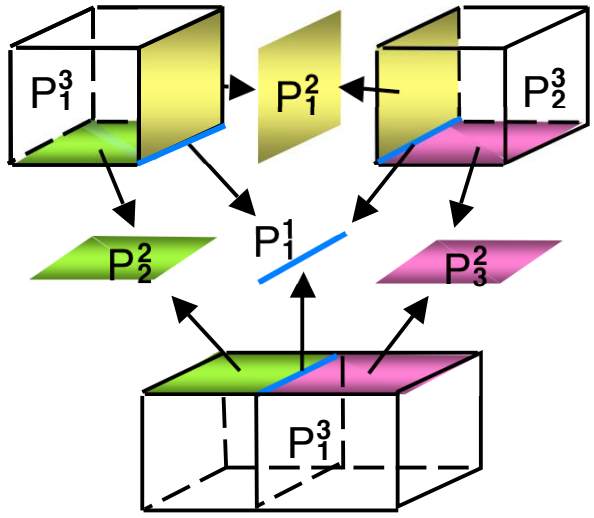
- High-to-low mesh entity traversal
- Inheritance-first
- Connectivity-second
- New partition entity creation-last



3D partitioned mesh



partition model



partition classification

Distributed Mesh Data Structure

Mesh Migration with Full Complete Representations

Given a list of pair <partition object, destination part id>

- STEP 1: Preparation – collect entities to be updated and reset their $\mathcal{P}$ and partition classification
- STEP 2: Determine $\mathcal{P}$ of partition objects and downward entities
- STEP 3: Based on $\mathcal{P}$, update the partition classification (partition model) and collect entities to remove from each part
- STEP 4: Exchange entities and update remote copies
- STEP 5: Remove unnecessary, migrated entities collected in STEP 3
- STEP 6: Update the owner part of partition model entities based on *poor-to-rich ownership*

Flexible Distributed Mesh Data Structure

Representational Requirement - Must be complete

- For each partition object M_i^d
  - Downward adjacent entities $\in \{\partial(M_i^d)\}$

    $\Rightarrow$ *Efficient interior entities restoration with validity*

    $\Rightarrow$ *HOWTO: For non-existing $\mathcal{R}_{i,j}$, set $\mathcal{R}_{0,j}$ to –*
  - For each downward adjacent $M_j^p \in \{M_i^d\{M^p\}\}$,
    - Upward adjacent partition objects to determine if M_i^p will exist on the current local after migration

  $\Rightarrow$ *Instead, use neighboring partition objects (Residence partition theorem)*

  $\Rightarrow$ *HOWTO: For each partition object M_i^d, store $\{M_j^0\{M_i^d\}\}$, $M_j^0 \in \{\partial(M_i^d)\}$*

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ - & - & 0 & 0 \\ - & 0 & - & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & \bigcirc\!\!- & \bigcirc\!\!- & 0 \\ - & - & 0 & 0 \\ - & 0 & - & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & - & - & \bigcirc\!\!1 \\ - & - & 0 & 0 \\ - & 0 & - & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

(ex) MRM adjustment for 3-D distributed mesh

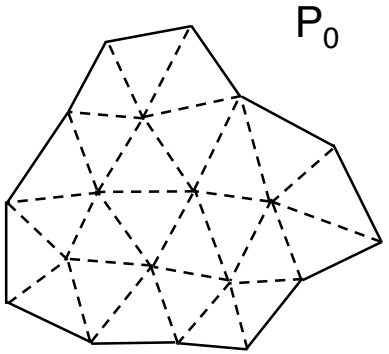
- Communication links - remote copies for partition boundary entities

SCOREC

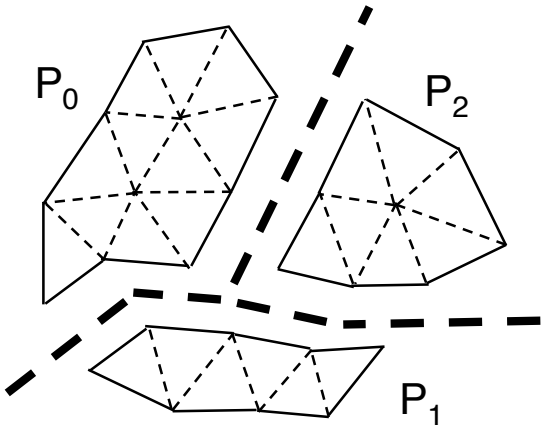
Flexible Distributed Mesh Data Structure

Mesh Migration with reduced complete representation

- STEP A: collect neighboring partition objects (–)
- STEP B: restore downward interior entities (–)
- STEP 1: collect entities to update and clear partition classification and $\mathcal{P}$ of them.
- STEP 2: Determine $\mathcal{P}$
- STEP 3: Update partition classification and collect entities to remove
- STEP 4: create only necessary migrate-in entities in representation and update remote copies
  - Do not send interior entities which will not be on the partition boundary (+)
- STEP 5: remove unnecessary migrate-out entities
- STEP 6: update entity ownership
- STEP C: remove unnecessary interior entities and adjacencies (–)



Serial 2D mesh with MSR



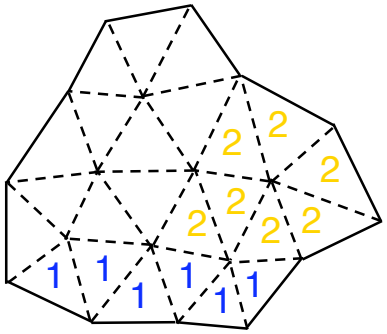
Partitioned 2D mesh with MSR

+: savings in migration time with flexible mesh representation in parallel

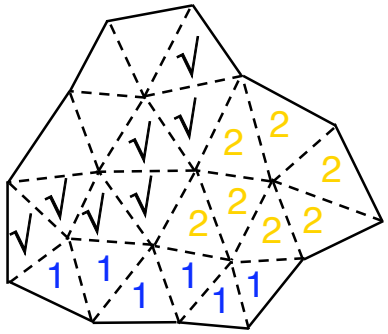
- : losses in migration time with flexible mesh representation in parallel

Flexible Distributed Mesh Data Structure

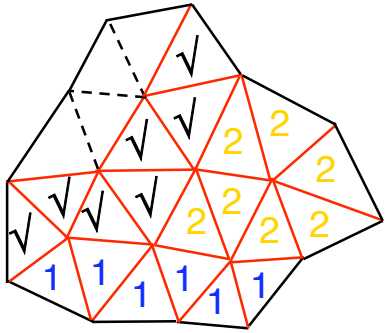
- *Examples*: 2-D mesh migration with the MSR



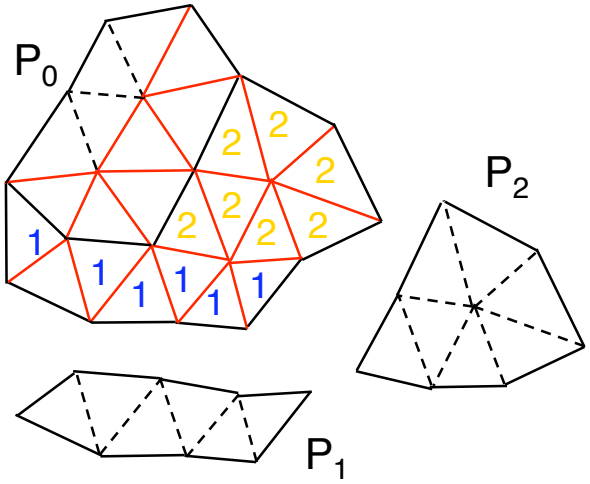
(a) Mark destination pid



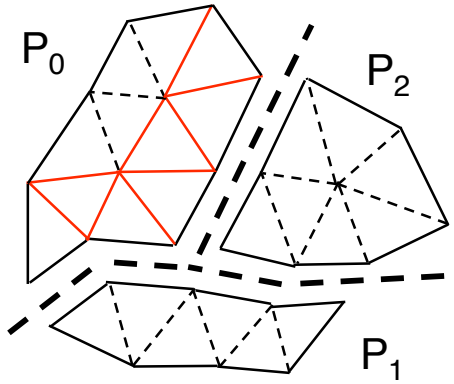
(b) ***step A***: Get neighboring POs



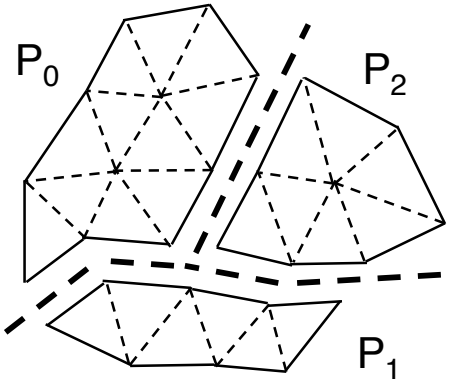
(c) ***Step B***: Restore internal ents



(d) ***Step 4***: Exchange ents and update comm. links



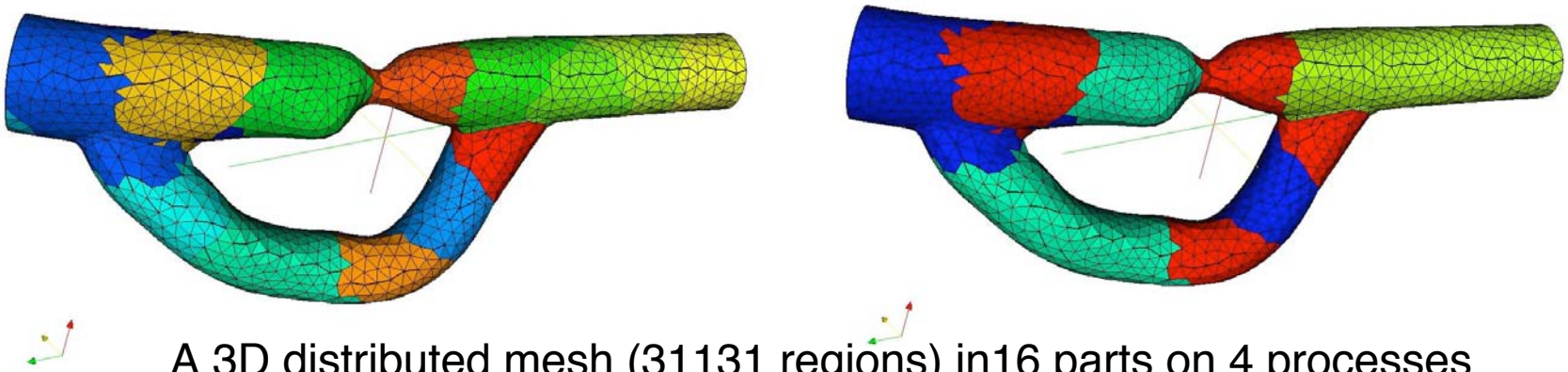
(e) ***Step 5***: Delete migrated ents



(e) ***Step C***: Delete internal ents

Multiple Parts Per Process

- Multiple parts (N>=0) can reside on one process, and one part resides on only one process.
- Change the number of parts on each process dynamically, based on **Zoltan** graph-based partitioning tool and migrating mesh entities among parts.
- Load or write out N parts on M processes from or to N mesh files (N, M > 0), assume one part per mesh file.
- **Test Result**: 8 million mesh exists on 8192 parts on 32 processes.



A 3D distributed mesh (31131 regions) in16 parts on 4 processes (4 parts per process).

Implementation

Programming Elements

- C++: the STL, functors, templates, singletons, generic, etc.
- Parallel: MPI, Autopack[1], Zoltan[2]

Highlights of Functionalities

- STL-like iterators to mesh entities, reverse classification, partition entities, remote copies of an entity
- Arbitrary attachable data to mesh entities of various data types
- General parallel utility services (*ex*) rank(), size()
- Efficient inter-partition communications through automatic message packing via Autopack
- Generic data communicator
- Easy-to-use *Zoltan callbacks* for interfacing with Zoltan and migrating arbitrary user-attached data with mesh entities.
- Parallel mesh I/O

---

1 R. Loy (2000) Argonne National Lab.
2 K. Devine et al (2005) Sandia National Lab.

SCOREC

Performance Test

Memory savings with flexible representation

- Serial
  - 10% - 40% with $\mathcal{R}^2$
  - 11% - 53% with $\mathcal{R}^3$
  - 18% - 77% with $\mathcal{R}^4$
- Parallel
  - 8% - 35% with $\mathcal{R}^2$
  - 8% - 44% with $\mathcal{R}^3$
  - 17% - 72% with $\mathcal{R}^4$

Migration Cost

- STEP 4 (entity exchange) is the most expensive
- With # partitions ≥20-24, # POs ≥300K, migration with $\mathcal{R}^2$, $\mathcal{R}^3$, $\mathcal{R}^4$ outperforms that with $\mathcal{R}^1$ due to less # entity exchanges in STEP 4

$$\mathcal{R}^1 \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \mathcal{R}^2 \begin{bmatrix} 1 & 0 & 0 & 0 \\ - & - & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\mathcal{R}^3 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ - & 0 & - & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \mathcal{R}^4 \begin{bmatrix} 1 & 0 & 0 & 0 \\ - & - & 0 & 0 \\ - & 0 & - & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Example MRM's

$$\mathcal{R}^1 \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \mathcal{R}^2 \begin{bmatrix} 1 & - & 0 & 1 \\ - & - & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

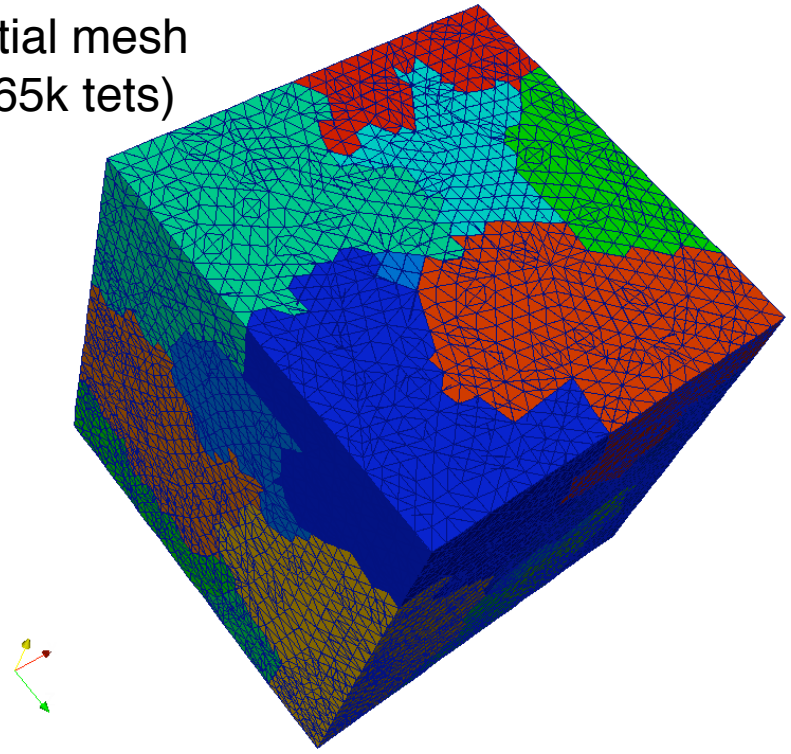
$$\mathcal{R}^3 \begin{bmatrix} 1 & 0 & - & 1 \\ 1 & 1 & 0 & 0 \\ - & 0 & - & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \mathcal{R}^4 \begin{bmatrix} 1 & - & - & 1 \\ - & - & 0 & 0 \\ - & 0 & - & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

MRM's after parallel adjustment

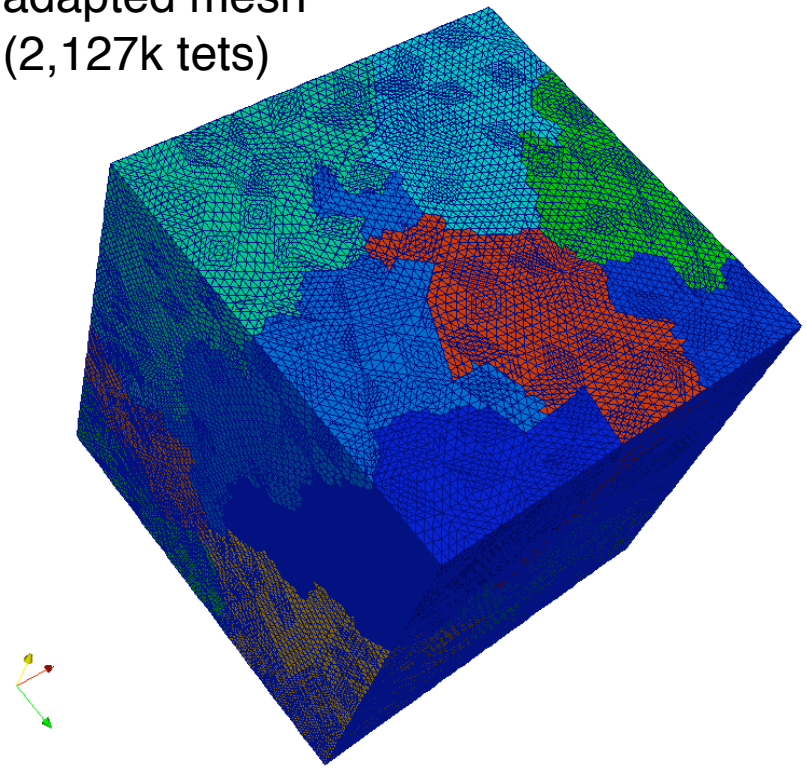
Mesh Adaptation - Refinement

- Parallel Isotropic Mesh Adaptation on Cube Geometry
  - Uniform mesh with uniform size field

initial mesh
(265k tets)



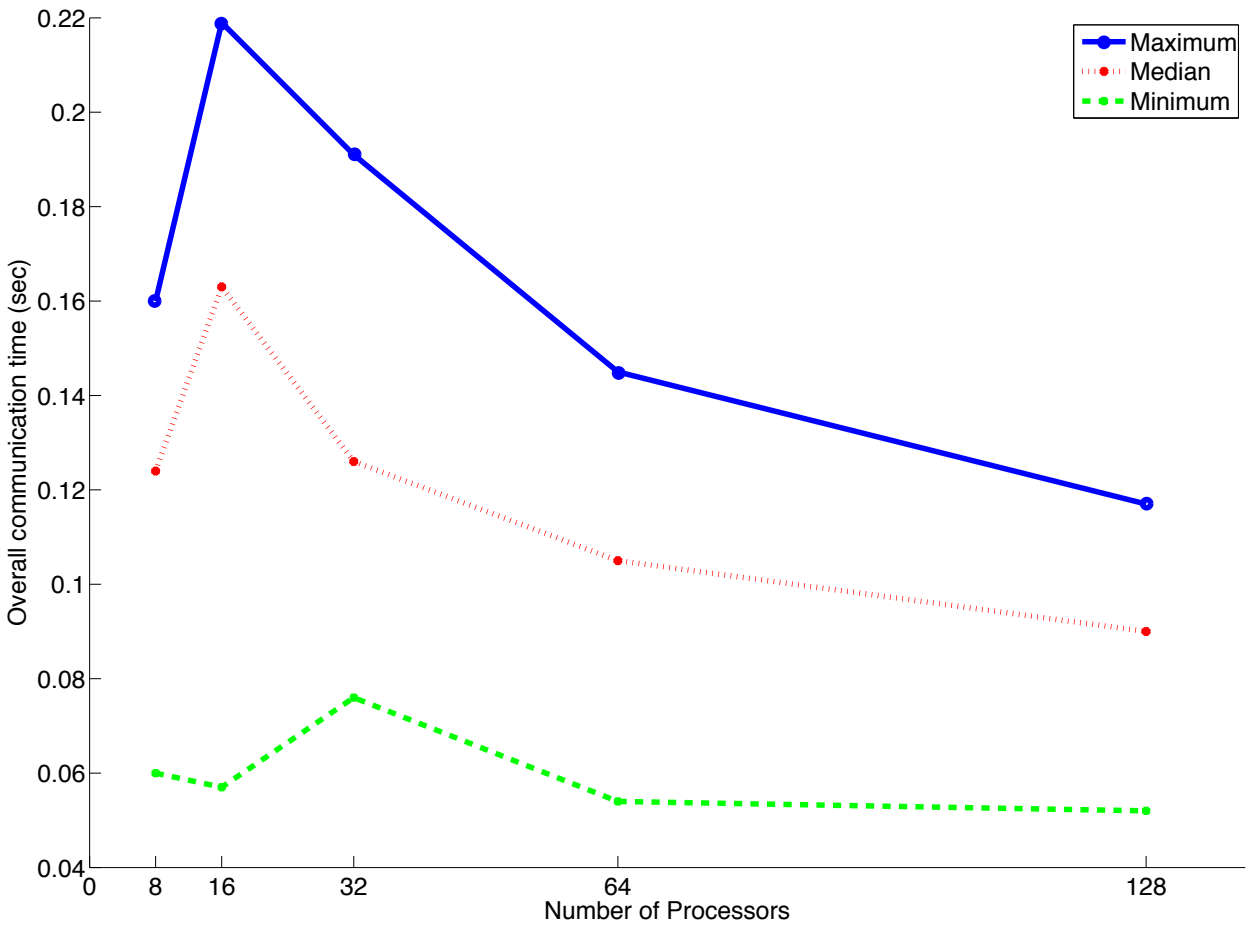
adapted mesh
(2,127k tets)



Tests run on IBM Blue Gene/L

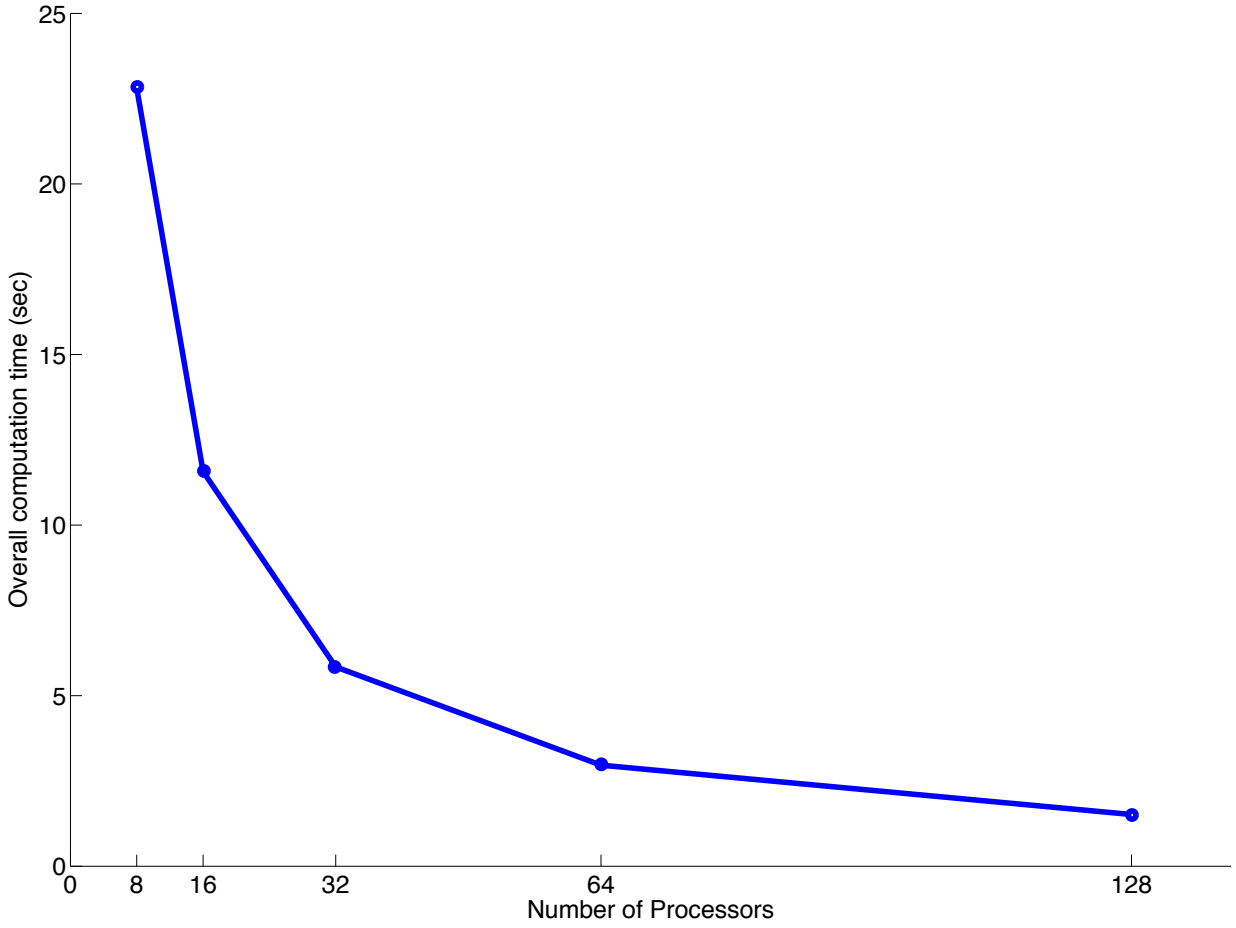
- Slow processors
- Fast interprocessor communication

Mesh Adaptation - Refinement



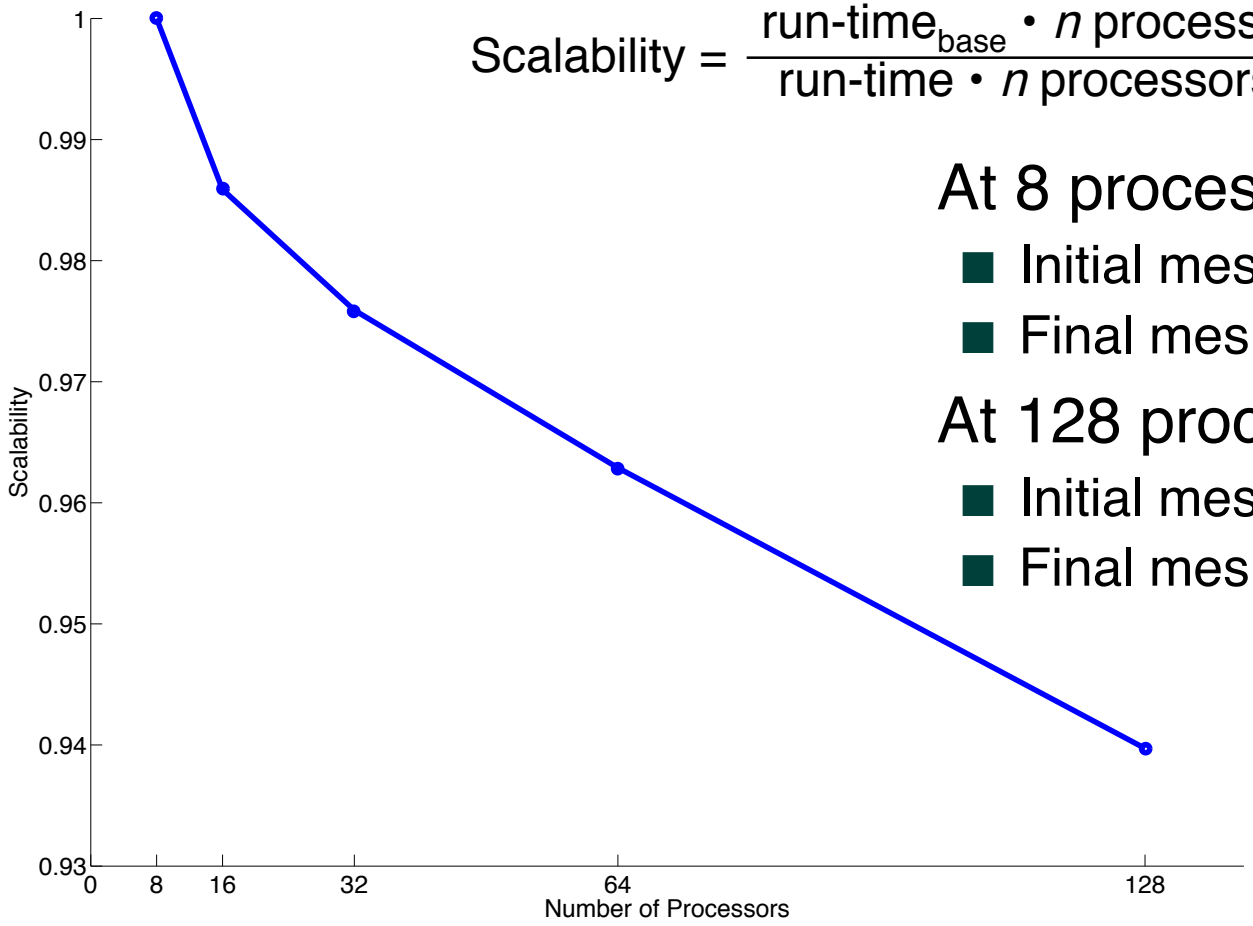
Communication time for
one iteration of mesh adaptation

Mesh Adaptation - Refinement



Total time for
one iteration of mesh adaptation

Mesh Adaptation - Refinement



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

At 8 processors

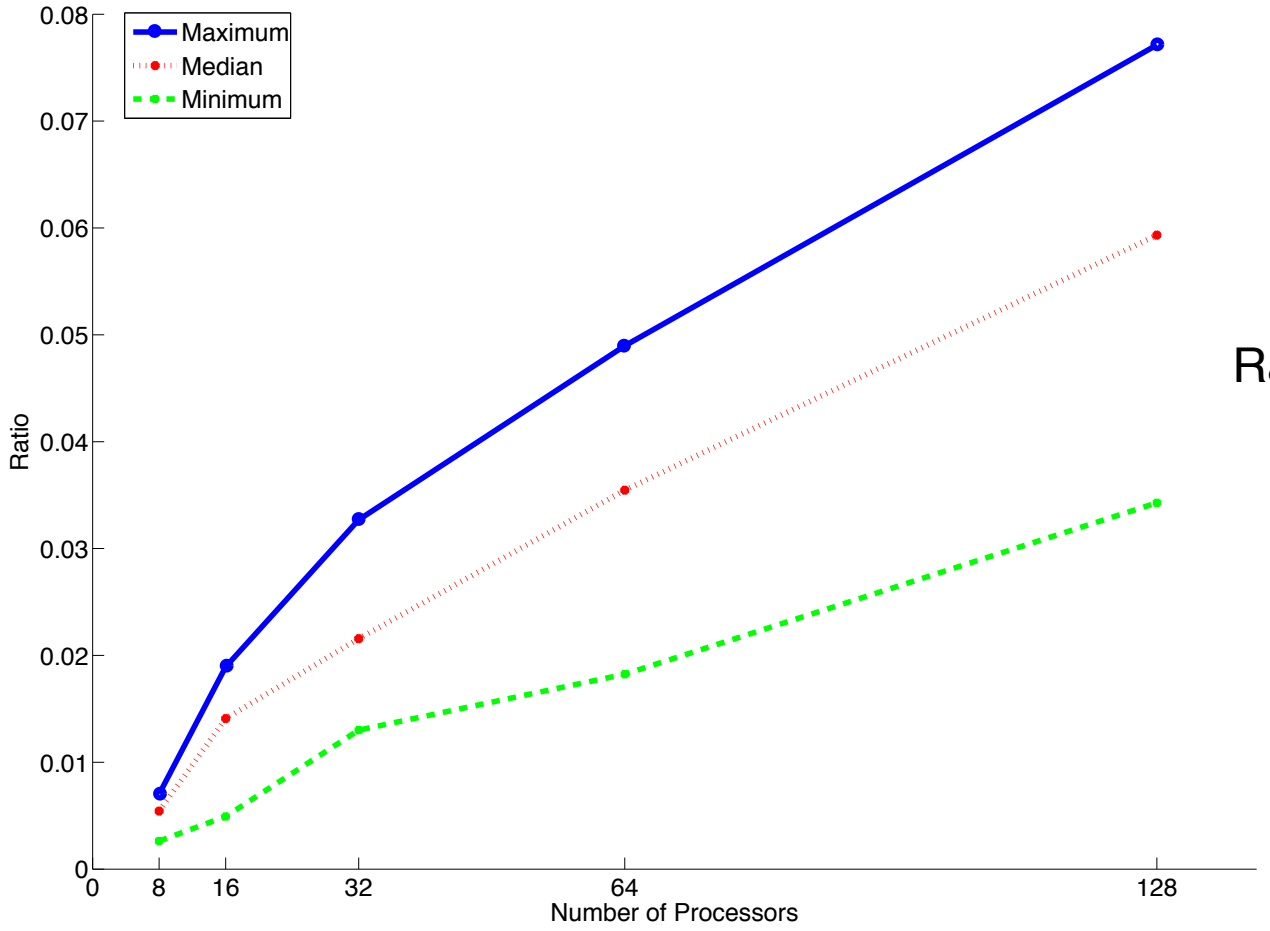
- Initial mesh - 33.1K/processor
- Final mesh - 226K/processor

At 128 processors

- Initial mesh 2.0K/processor
- Final mesh 16.7K/processor

Scalability for one iteration of mesh adaptation

Mesh Adaptation - Refinement

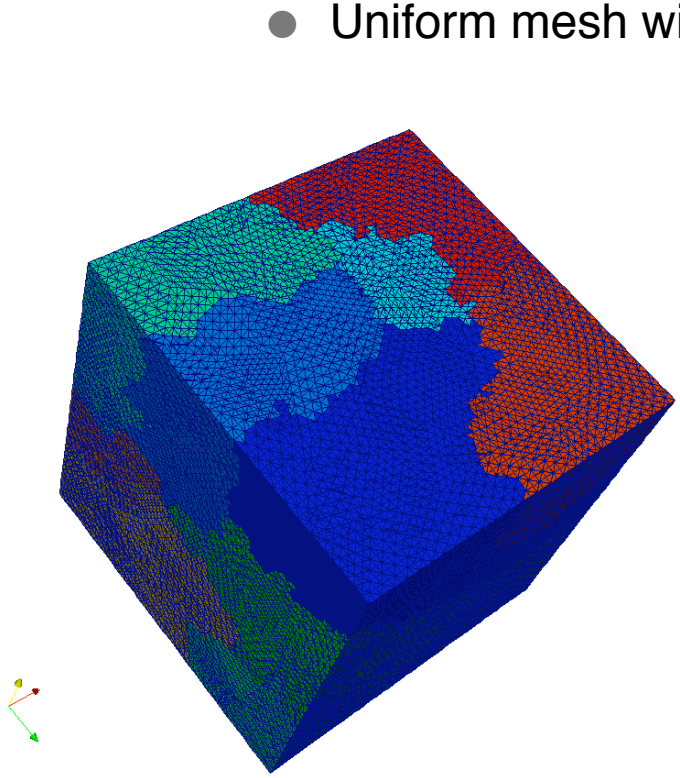


$$\text{Ratio} = \frac{\text{Communication time}}{\text{Total time}}$$

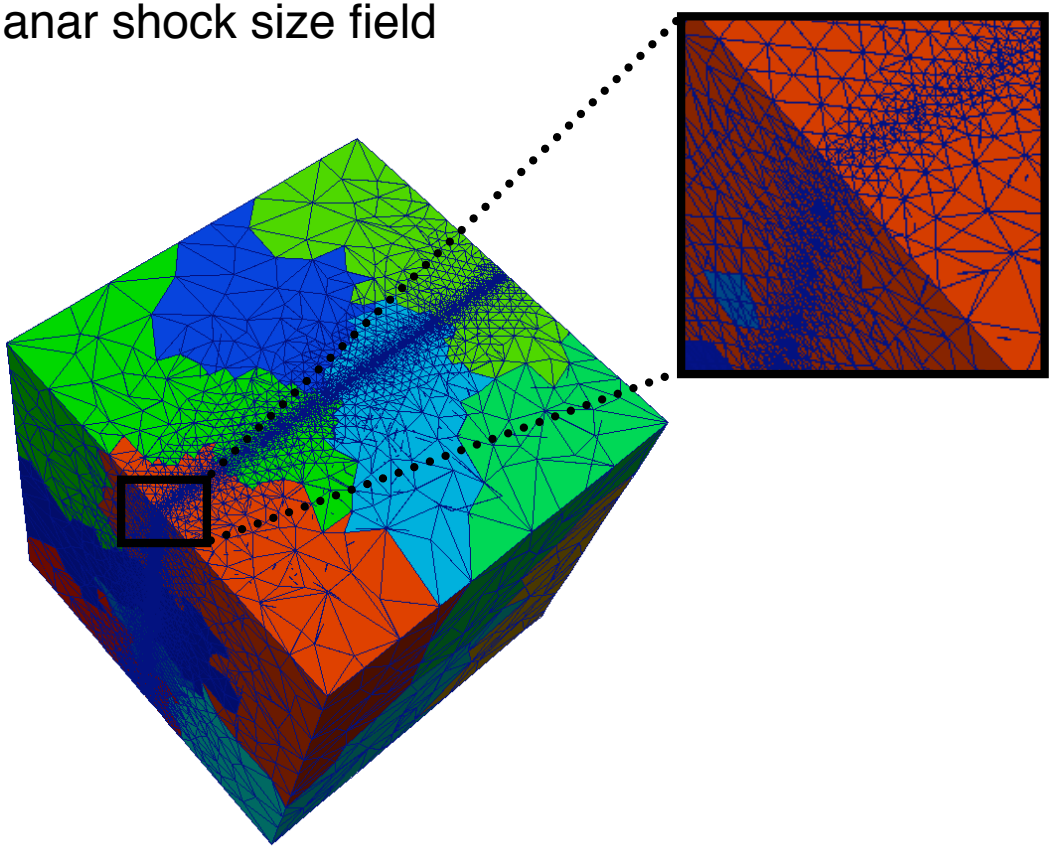
Communication to total time ratio for one iteration of mesh adaptation

Mesh Adaptation - Refinement and Coarsening

- Parallel Isotropic Mesh Adaptation on Cube Geometry
  - Uniform mesh with planar shock size field

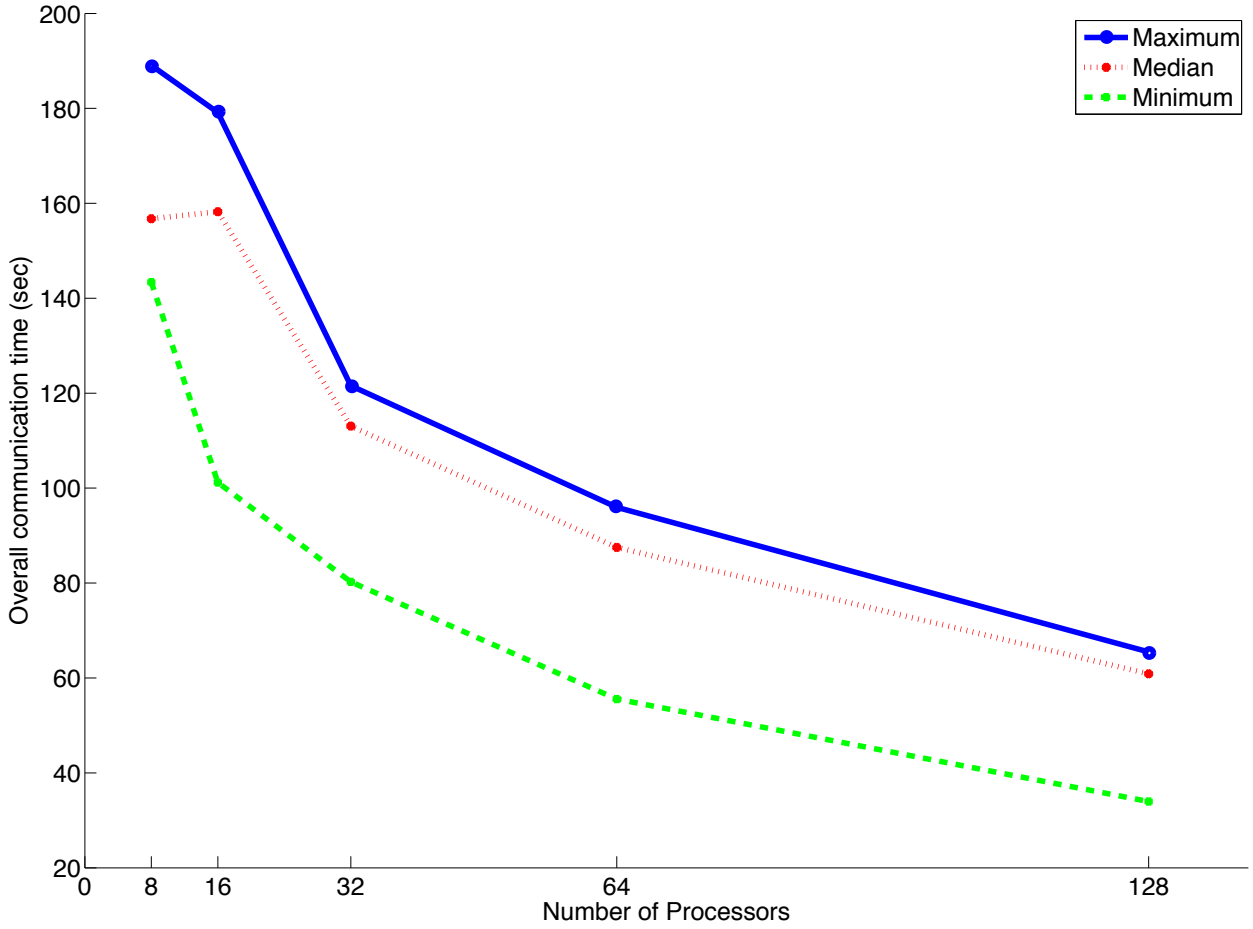


initial mesh
(1,528k tets)



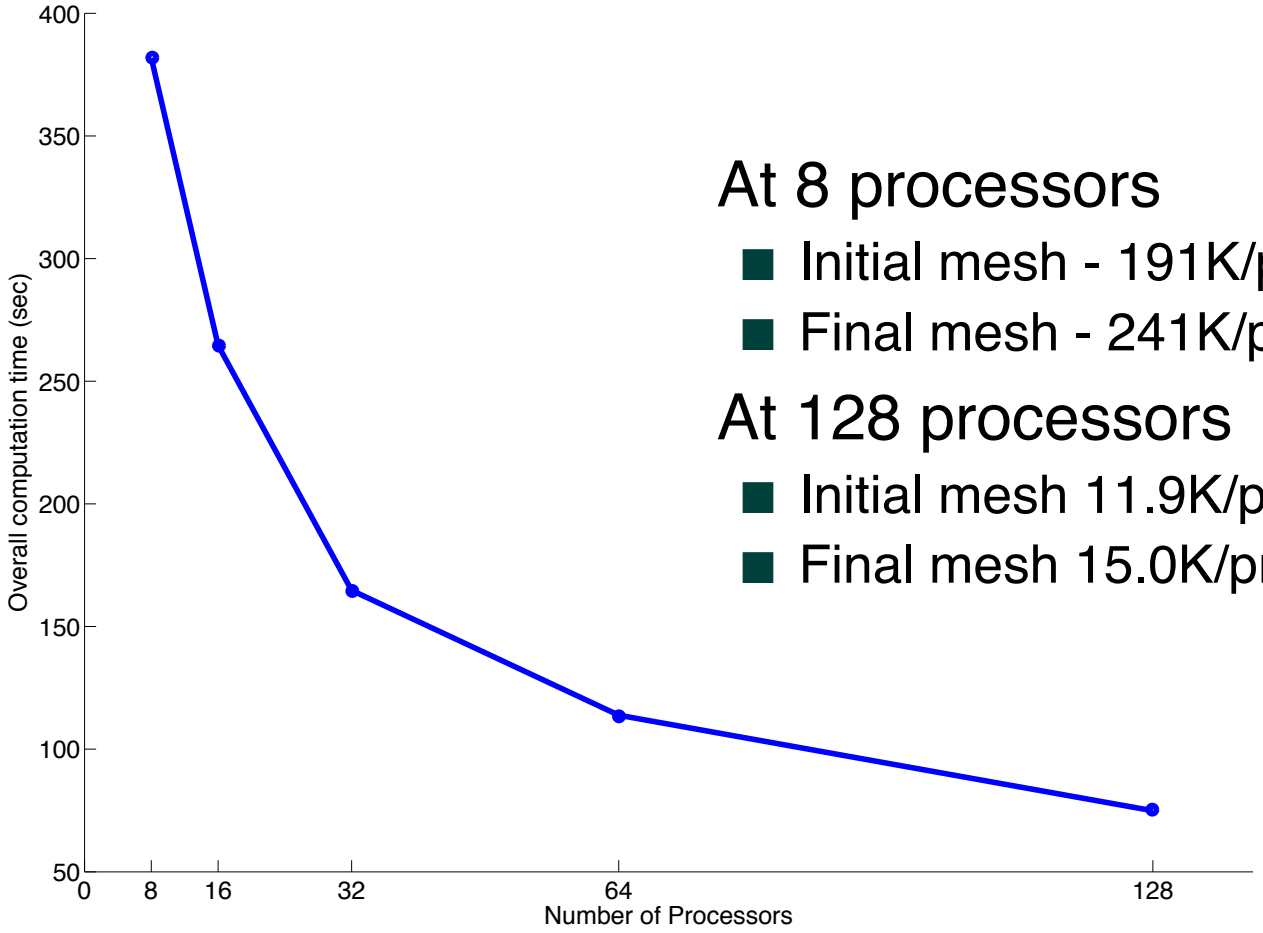
adapted mesh
(1,926k tets)

Mesh Adaptation - Refinement and Coarsening



Communication time for
five iterations of mesh adaptation

Mesh Adaptation - Refinement and Coarsening



At 8 processors

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

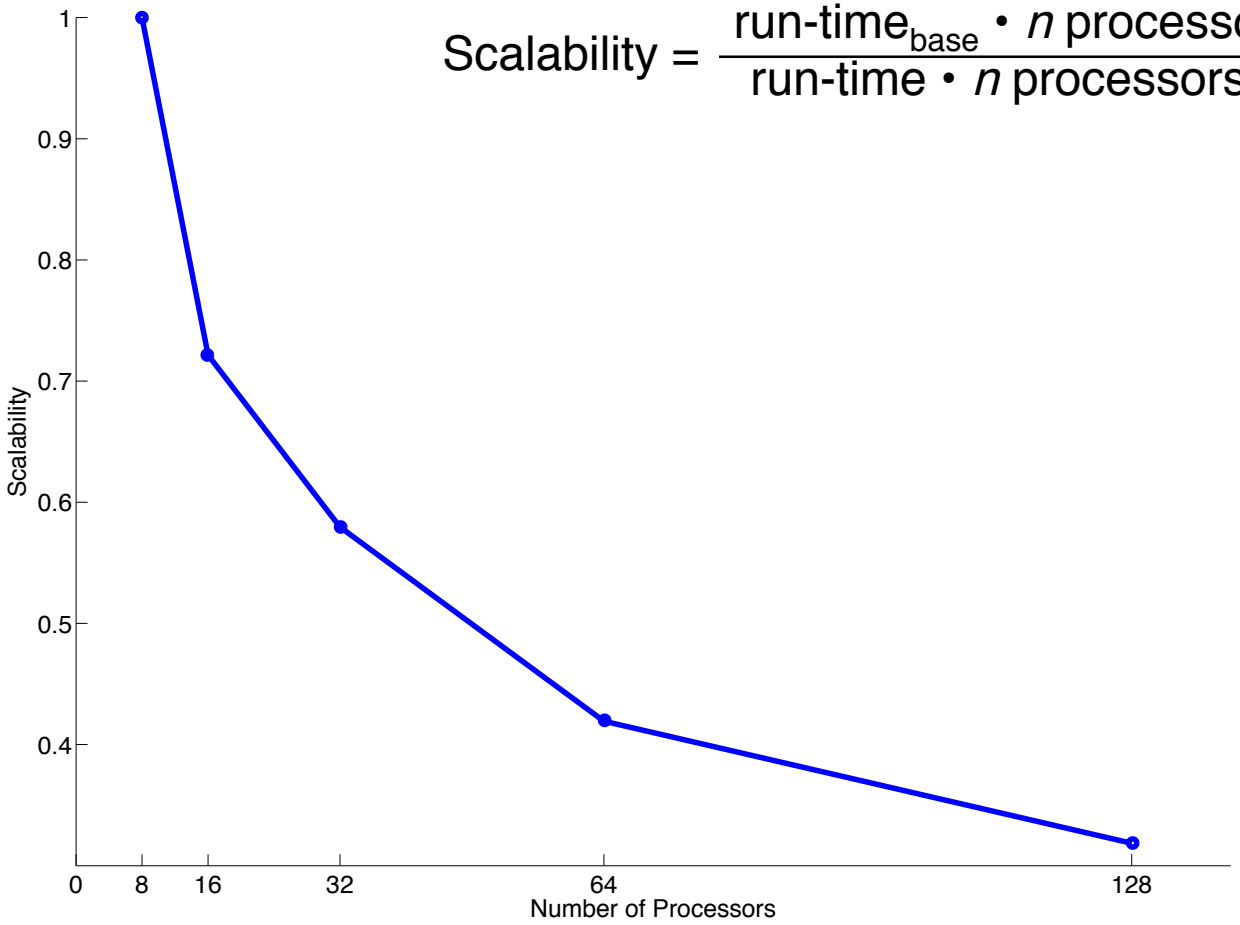
At 128 processors

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

Total time for
five iterations of mesh adaptation

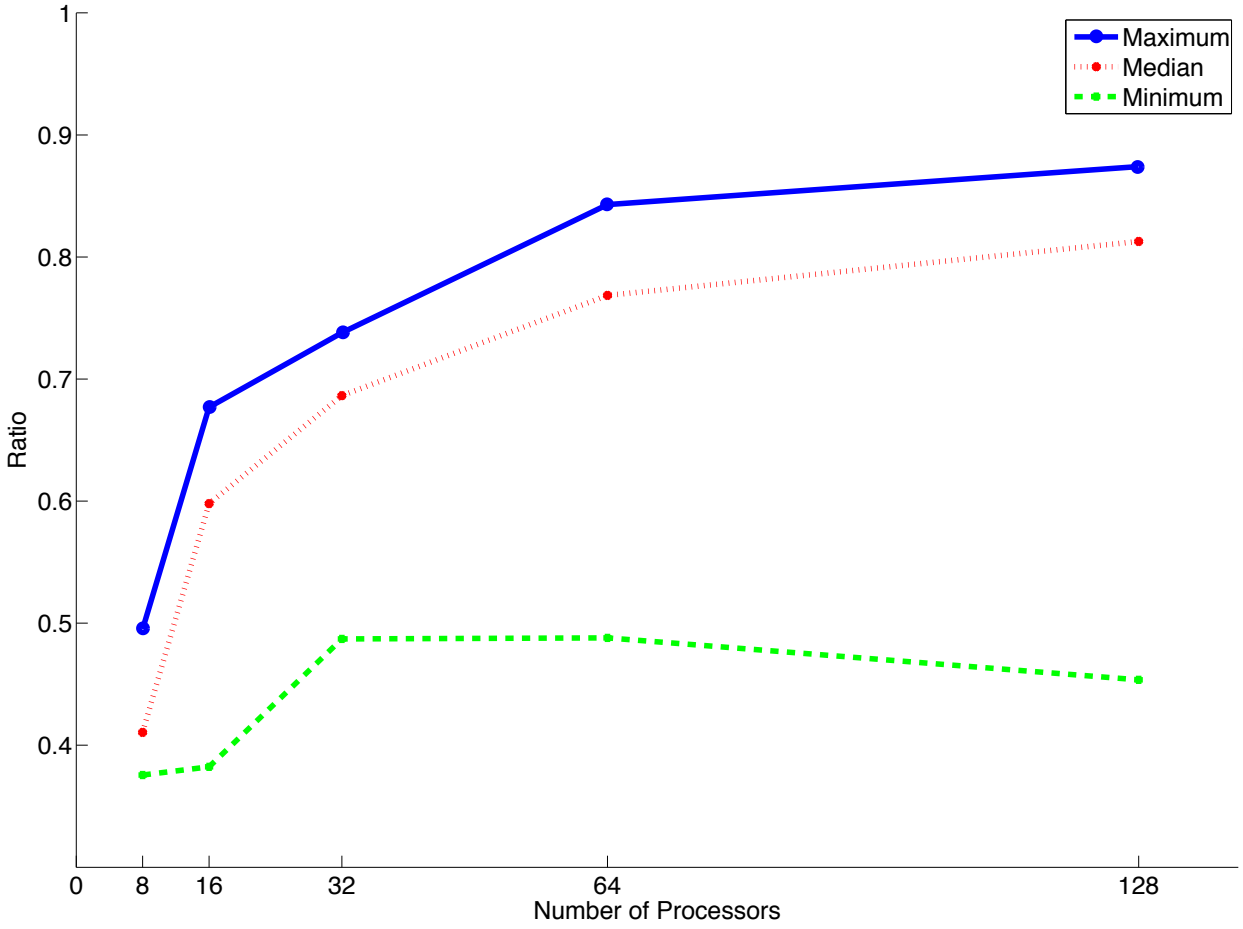
Mesh Adaptation - Refinement and Coarsening

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



Scalability for five iterations
of mesh adaptation

Mesh Adaptation - Refinement and Coarsening



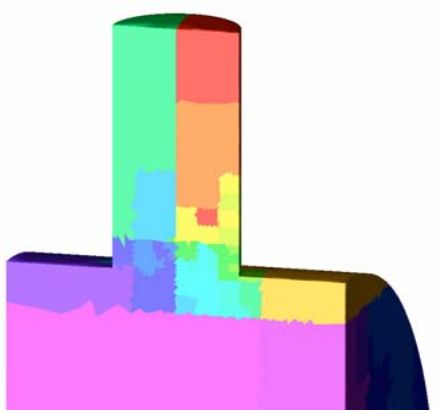
$$\text{Ratio} = \frac{\text{Communication time}}{\text{Total time}}$$

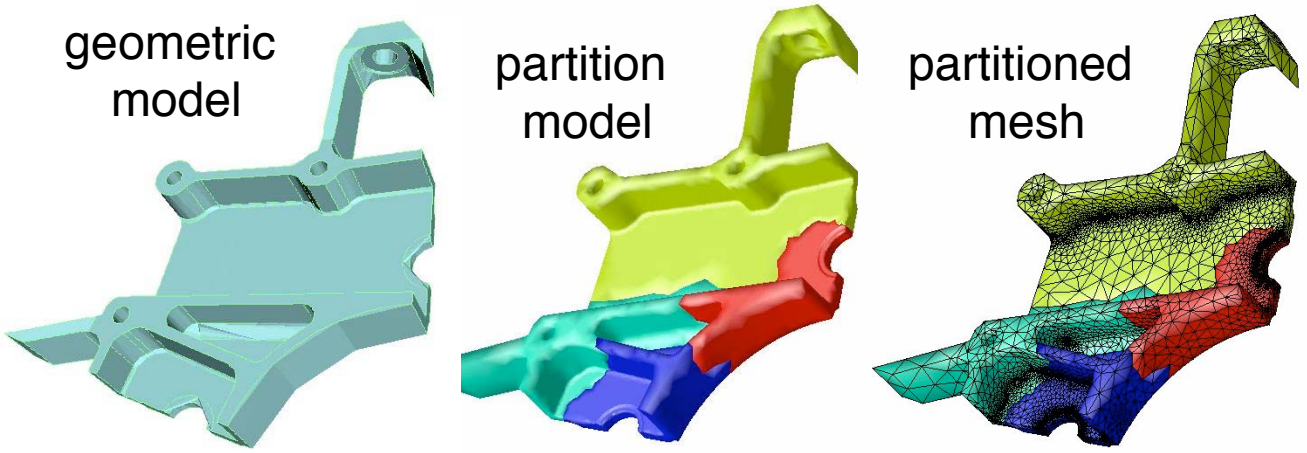
Communication to total time ratio for five iterations of mesh adaptation

Parallel Adaptive Analysis

Functions available to support parallel adaptive

- Distributed mesh representation capable of providing needed mesh information
- Ability to distribute the mesh and support mesh level communications between processors
  - Partition model for inner processor communications
  - Mesh migration
- Parallel mesh modification
- Dynamic load balancing (Zoltan)

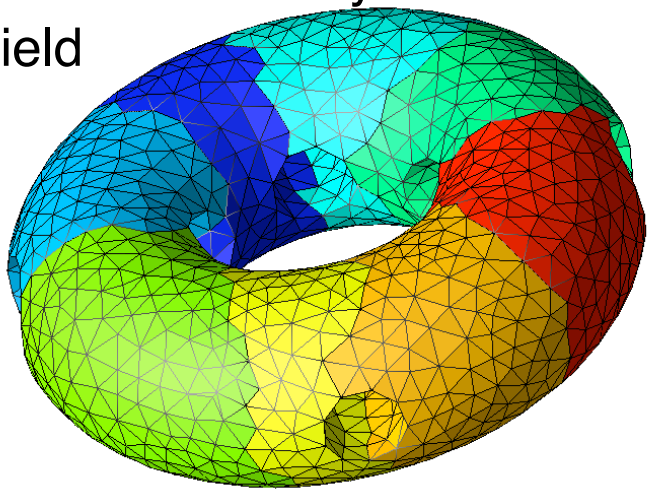




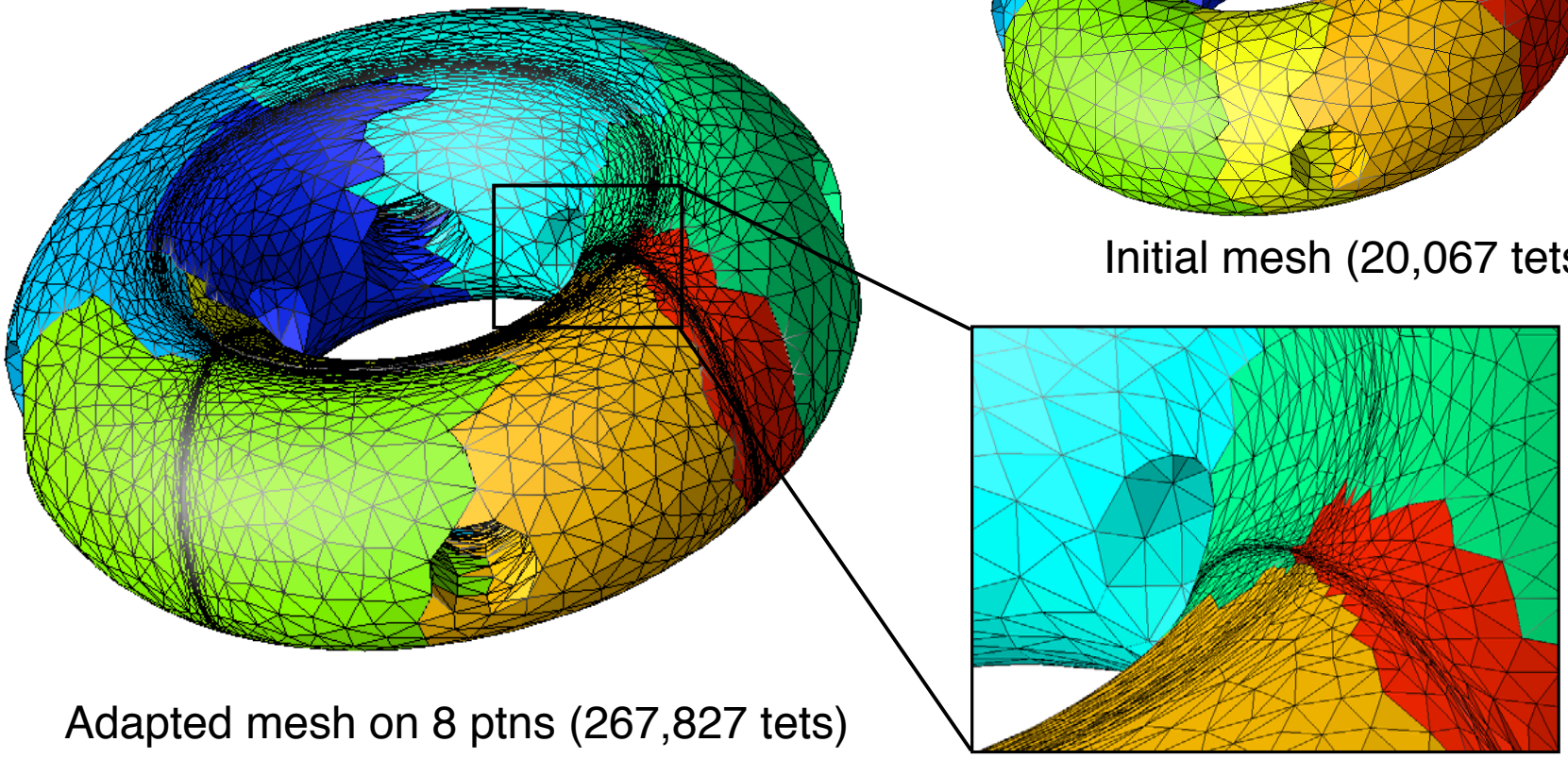
SCOREC

Applications

- Parallel Anisotropic Mesh Adaptation on Torus Geometry
  - Two spherical shocks analytical size field



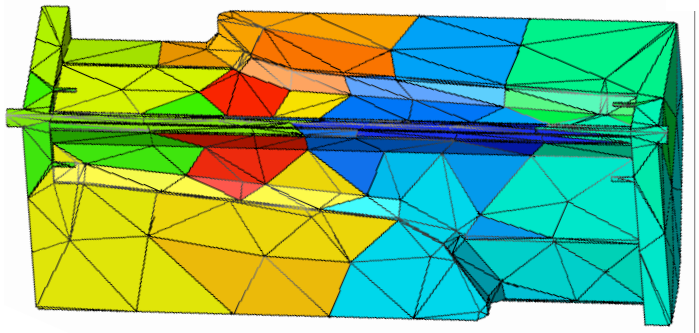
Initial mesh (20,067 tets)



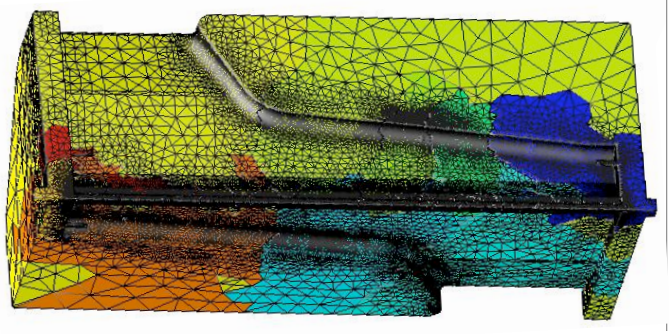
Adapted mesh on 8 ptns (267,827 tets)

Adaptive Loop for Accelerator Design

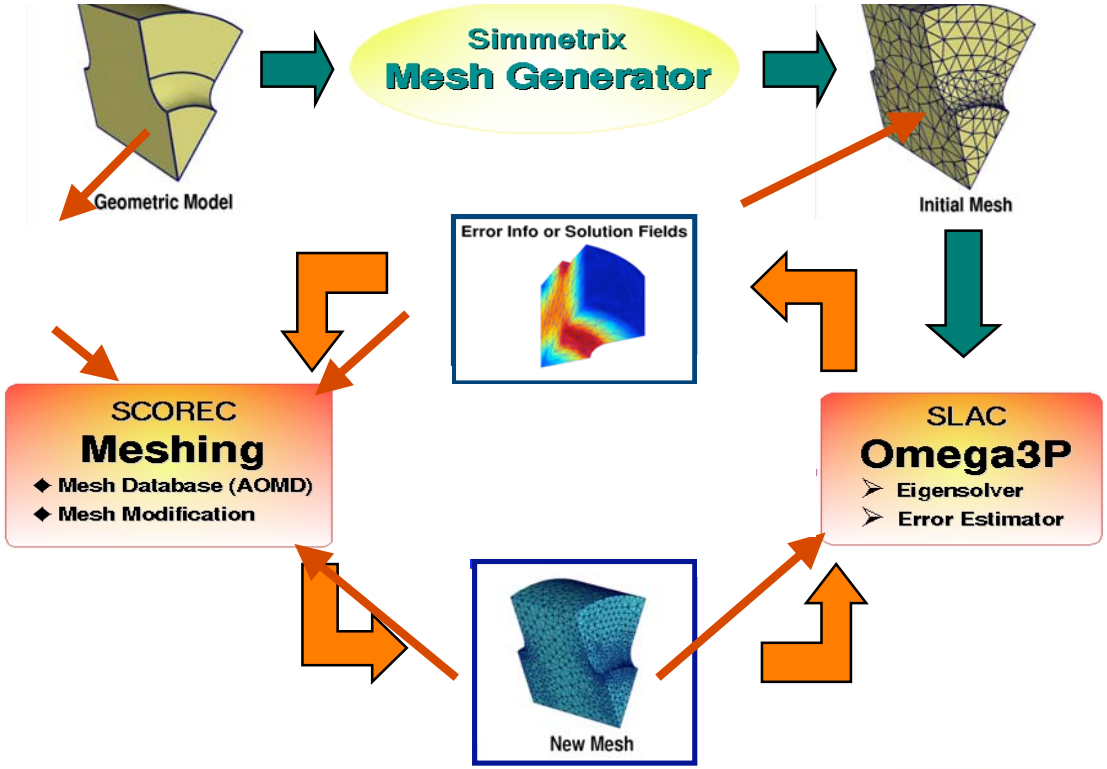
- Complex CAD geometry
- Physics modeling by the SLAC Omega3P
- High level modeling accuracy needed
  - E.g., 0.1% error in frequency predictions
- Parallel adaptive mesh control needed to provide accuracy needed



Initial mesh (1,595 tets)

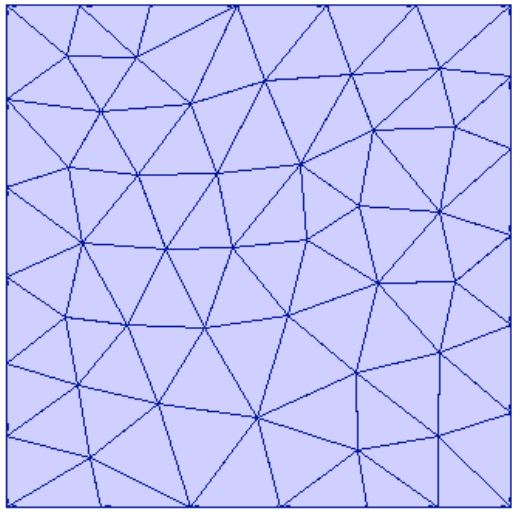


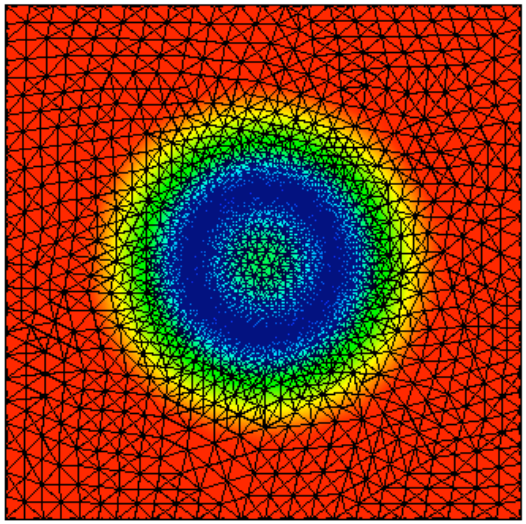
Adapted mesh (23,082,517 tets)



SCOREC

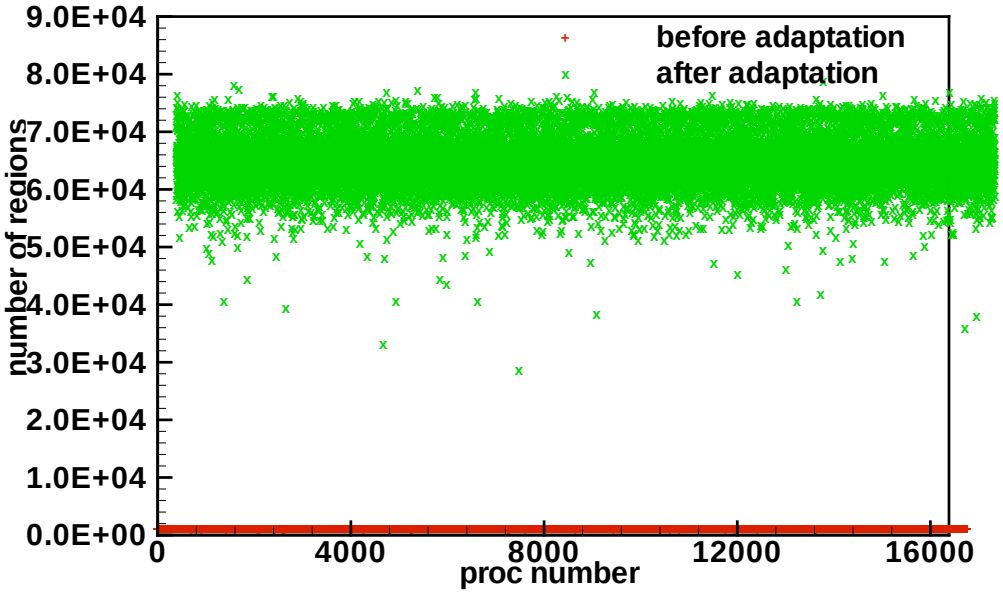
Mesh Adaptation for 1 Billion Element Mesh





Initial and adapted mesh (zoomed for 1/2 bubble), colored by the magnitude of mesh size field

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

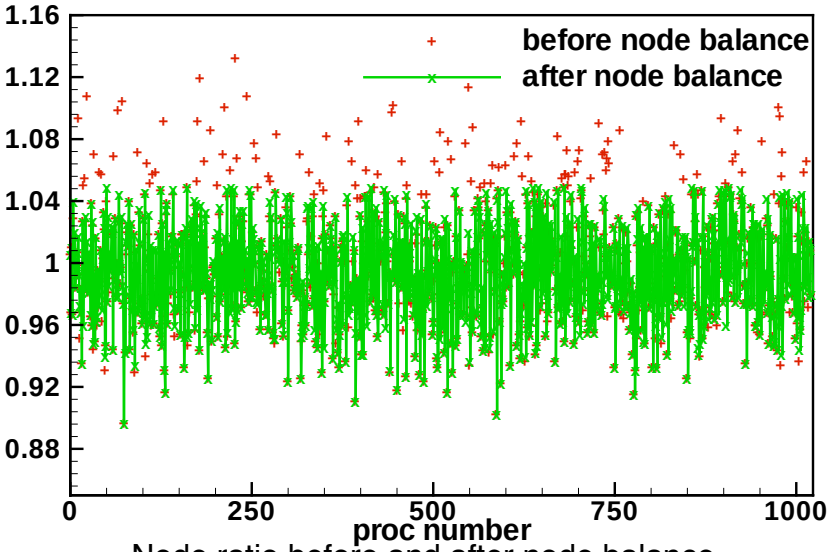


Number of regions of adapted mesh among 16k parts

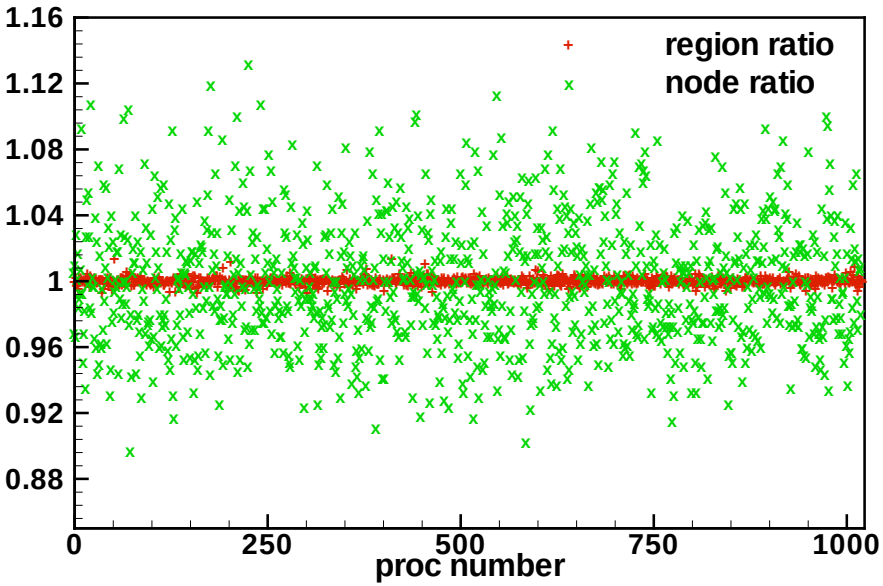
- Initial mesh: uniform, 17,179,836 mesh regions
- Adapted mesh: 160 air bubbles 1,064,284,042 mesh regions
- Multiple predictive load balance are used to make the adaptation possible
- Larger meshes possible (not out of memory) but this element count is appropriate for solver

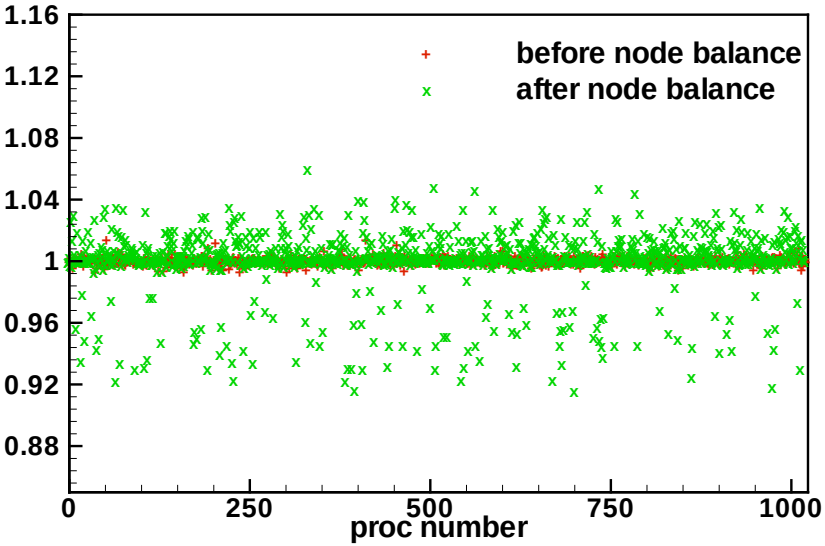
Nodal Balance by Local Modification

- For light loaded mesh (small number of regions for each process), well distributed mesh (based on the number of regions) could have bad nodal balance.
- Local modification method is used to balance the number of nodes on each part.
- Region (node) ratio = number of region (node)/average number of region (node)
- Average number of regions for the test: 2434
- Number of parts:1024



Node ratio before and after node balance



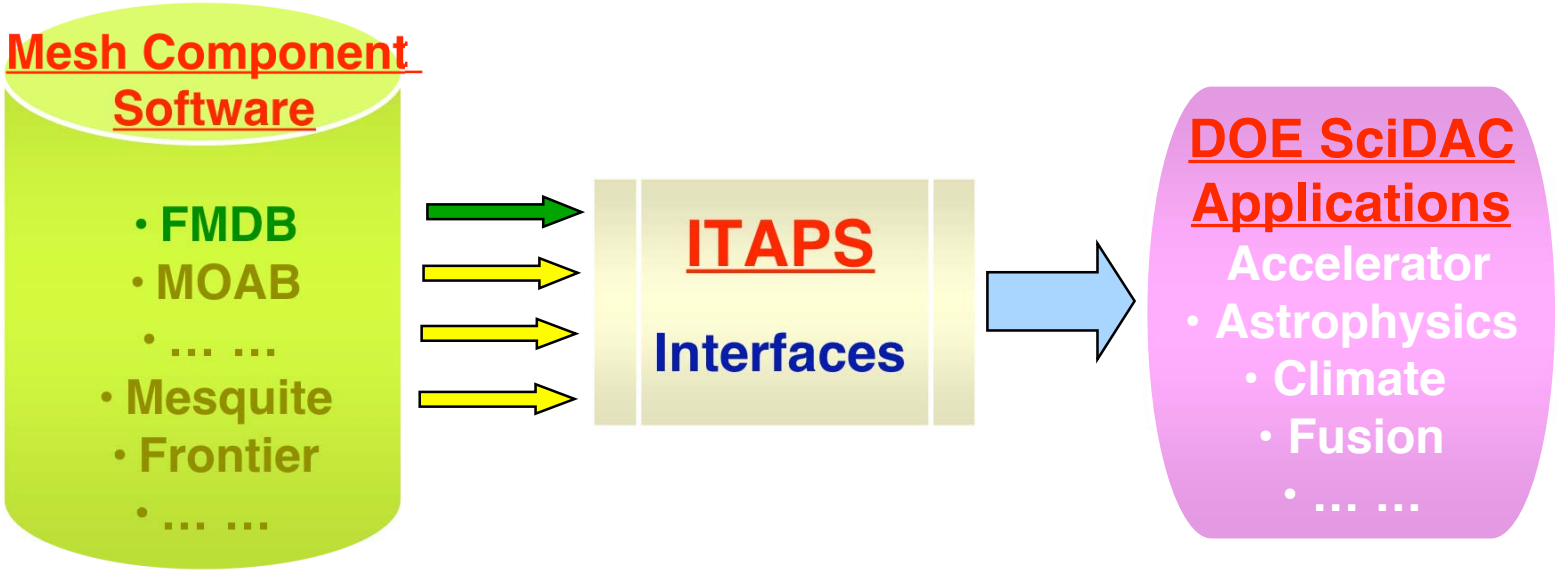


Region ratio before and after node balance

SCOREC

FMDB is Part of ITAPS

- ITAPS tools (*http://www.itaps-scidac.org*)
  - Provide a core functionality of the Interoperable Technologies for Advanced Petascale Simulations (**ITAPS**) meshing tools
  - The only ITAPS component thus far that supports
    - Geometry-based adaptive analysis
    - Distributed mesh operations in parallel
    - Flexible mesh representation



Closing Remarks

Highlights of the FMDB

- Flexible mesh data structure construct and maintain the user-requested mesh representations
- Implemented in simulation packages supporting parallel adaptive simulation
  - Serial/parallel mesh adaptation
  - Serial/parallel error estimation
  - Easily linked with external analysis codes
- Open source available at http://www.scorec.rpi.edu/FMDB