ITAPS
INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

SciDAC
Scientific Discovery through Advanced Computing

ITAPS Parallel Interface v0.1

March 2008

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

Parallel Interface Subcommittee

- Sandia
  - Karen Devine, Vitus Leung
- LLNL
  - Lori Diachin, Mark Miller
- Argonne
  - Tim Tautges, Jason Kraftcheck
- Rensselaer
  - Mark Shephard, Onkar Sahni, Ken Jansen
- U. British Columbia
  - Carl Ollivier-Gooch

Parallel Interface Goals

- Primarily support distributed memory.
  - Accept MPI communicators from application.
  - But allow use of global address space and shared memory paradigms.
- Maintain backward compatibility of serial iMesh.
  - Serial iMesh works as expected within a process.
  - Serial iMesh works as expected for NWGrid.

Terminology

- *Process*: a program executing; MPI process.
  - Number of processes == MPI_Comm_size
  - Process number == MPI_Comm_rank
- *Mesh instance*: mesh database provided by an implementation.
  - Each process has one or more mesh instances.
- *Partition*: describes a parallel mesh.
  - Maps entities to subsets called *parts*.
  - Maps parts to processes.
  - Has a communicator associated with it.
- *Global* operation: an operation with respect to data in all parts in a partition's communicator.
- *Local* operation: an operation with respect to either a part's or process' data.

Partition Characteristics

- Maps entities to parts.
  - Part assignments computed with respect to a set of entities.
  - Computed assignments induces part assignments for adjacent entities.
- Maps parts to processes.
  - Each process may have one or more parts.
  - Each part is wholly contained within a process.
- Has a communicator associated with it.
  - "Global" operations performed with respect to this communicator.
- Accessed via iMeshP_PartitionHandle.

Partition Creation/Destruction

- **Creation/Destruction**
  - **iMeshP_createPartitionAll**
    - **Takes MPI Communicator or NULL.**
  - **iMeshP_destroyPartitionAll**
  - **iMeshP_syncPartitionAll**
    - **After parts are added/updated, computes and stores global information.**
      - **Mapping of parts to processes.**
      - **Number of parts in partition.**
- **Partition Queries**
  - **iMeshP_getPartitions**
    - **Return all partition handles in this mesh instance.**
  - **iMeshP_getPartitionComm**
    - **Returns the MPI Communicator or NULL.**

Partition Queries

- Mapping of parts to processes
  - iMeshP_getNumParts
    - Returns total number of parts in partition.
  - iMeshP_getPartsOnRank
    - Returns part handles for parts on a given process.
  - iMeshP_getRankOfPart
    - Returns process number for a given part.
  - No communication; all values precomputed in iMeshP_syncPartitionAll.
- Global mesh information
  - iMeshP_getNumOfTypeAll
    - Returns total number of entities with given type in a given partition and entity set.
  - iMeshP_getNumOfTopoAll
    - Returns total number of entities with given topology in a given partition and entity set.
  - Require collective communication.

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 via iMeshP_PartHandle.
  - Local part handles identify on-process parts.
  - Remote part handles identify off-process parts.

iMeshP_PartHandle

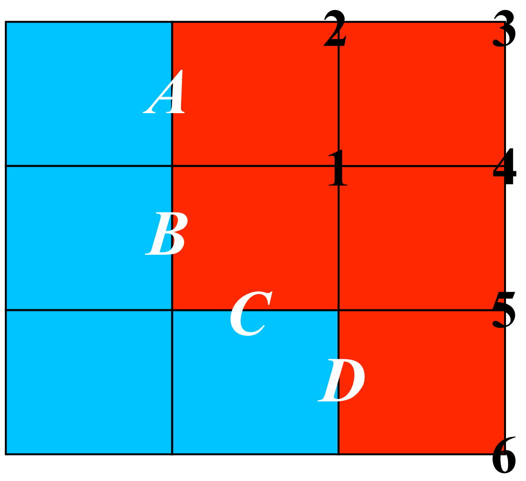
- iMeshP_PartHandle may be substituted for iBase_EntitySetHandle in many iMesh functions to perform local part operations.
  - Get number of local entities in a part with iMesh_getNumOfType, iMesh_getNumOfTopo.
  - Get entities in a part with iMesh_getEntities.
  - Add entity to a local part with iMesh_addEntToSet.
  - Et cetera, et cetera, et cetera.

Part Creation/Destruction

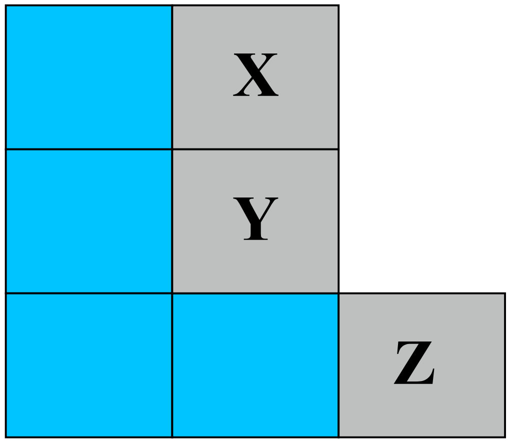
- Create/Destroy a part.
  - iMeshP_createPart
    - Creates a part and adds it to a partition.
  - iMeshP_destroyPart
    - Removes a part and invalidates the part handle.
  - After all parts are created and populated, application must call iMeshP_syncPartitionAll to precompute partition data.

More Terminology

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



- *Ghost entity*: Any non-owned entity that is not a part-boundary entity.
  - E.g., Regions X, Y, and Z are ghost regions for the blue part.
- *Copies*: ghost entities + non-owned part-boundary entities.



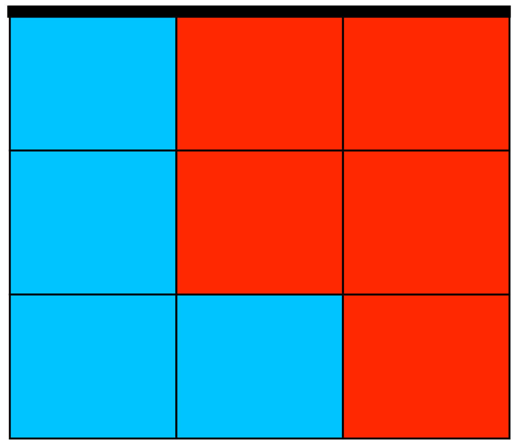
Part Neighbors

- Parts A and B are *neighbors* if Part A has copies of entities owned by Part B or vice versa.
- Part neighbors
  - iMesh_getNumPartNbors
    - Return number of parts neighboring a given part.
  - iMesh_getPartNbors
    - Return remote part handles for part neighbors.
- Entities on part boundaries
  - iMeshP_getNumPartBdryEnt
    - Return number of boundary entities shared with a given part.
  - iMeshP_getPartBdryEnts
    - Return boundary entities shared with a given part neighbor.
  - iMeshP_initPartBdryEntIter
    - Iterator over boundary entities shared with a given part.

Parts and Entity Sets

- Part handles may be passed to iMesh EntitySet functions for local operations.
- But also need functions accepting both part handle and EntitySet handle.
  - E.g., Boundary conditions.
    - Store entities with the boundary condition in an EntitySet.
    - Iterate over entities in both a given boundary condition EntitySet and a given part.
  - E.g., Multiple meshes with a single partition.
    - Store meshes as separate entity sets in iMesh instance.
    - Generate a single partition of both meshes.
    - Iterate over entities in both a given mesh EntitySet and a given part.

Edges in BC EntitySet



Allow query of entities in both red part and BC EntitySet.

Parts and Entity Sets

- Return data with respect to both local part handle AND entity set handle.
  - Functions mimic subset of iMesh functions.
  - iMeshP_getNumOfType
  - iMeshP_getNumOfTopo
  - iMeshP_getAllVtxCoords
  - iMeshP_getVtxCoordIndex
  - iMeshP_getEntities
  - iMeshP_getAdjEntities
  - iMeshP_initEntIter
  - iMeshP_initEntArrIter

Entity Characteristics

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

Entity Functions

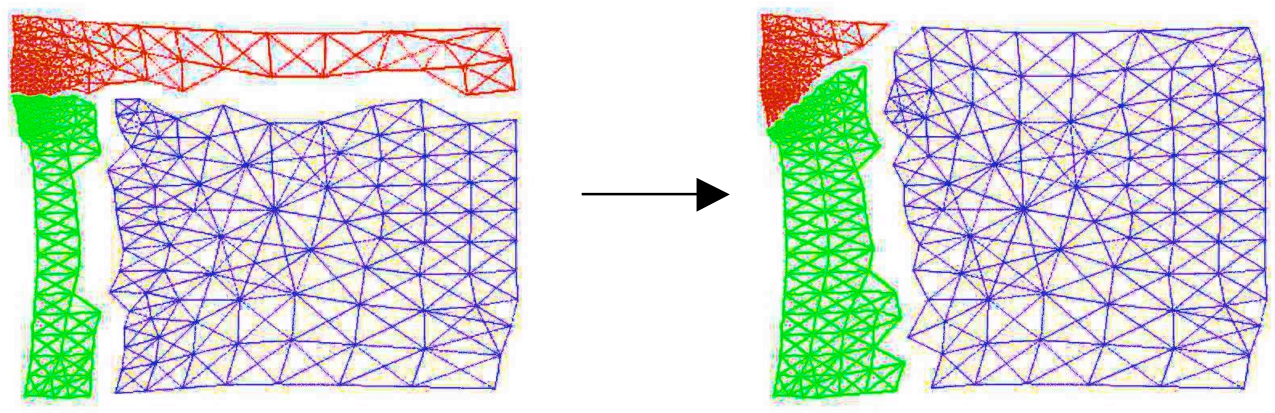
- Ownership of entity.
  - iMeshP_getEntOwnerPart
    - Return (possibly remote) part handle of an entity’s owner part.
  - iMeshP_isEntOwner
    - Return flag indicating whether a given part handle is an entity’s owner.
  - iMeshP_getEntStatus
    - Return flag indicating whether entity is internal, boundary, or ghost.

Entity Functions

- Entity copies
  - iMeshP_getNumCopies
    - Return number of copies of an entity.
  - iMeshP_getCopyParts
    - Return the remote part handles of copies of an entity.
  - iMeshP_getCopies
    - Return the remote part handles and remote entity handles of copies of an entity.
  - iMeshP_getCopyOnPart
    - Return the remote entity handle for an entity copy on a given part.
  - iMeshP_getOwnerCopy
    - Return the part handle and entity handle from an entity's owning part.
- Reminder: These functions do not require communication.

Inter-part Mesh Operations

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



Inter-part Mesh Operation Requests

- Inter-part mesh operations are coordinated via iMeshP_RequestHandles.
  - More than an MPI_Request!
  - Indicates status of a given mesh operation.
    - E.g., Migrate entity; Update vertex coordinates; Update part-boundary entities; Exchange tag data.
  - Contents are implementation dependent.
    - MPI_Requests
    - Flags/functions indicating what data to send, what to do with data once received.
    - May involve more than one round of communication (e.g., mesh migration).
  - iMeshP_RequestHandle completes when entire mesh operation is completed.
- iMeshP_RequestHandle enables…
  - Overlapping communication/computation
  - Asynchronous communication

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.

Waiting for Requests

- Similar to MPI_Wait functions, except waiting for *mesh operation* to complete.
- Block until requests are complete.
  - iMeshP_Wait
    - Returns when a given request completes.
  - iMeshP_WaitAny
    - Returns when any given request completes.
  - iMeshP_WaitAll
    - Returns when all given requests complete.
  - iMeshP_WaitEnt
    - Waits for a given request to complete; returns entity handles modified by the request.
- Check completion status of requests.
  - iMeshP_Test
    - Tests for a given request's completion.

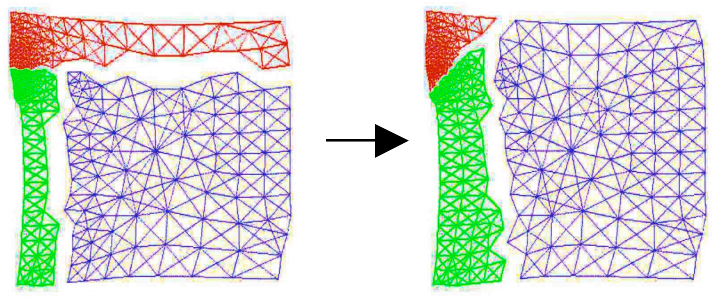
Request Polling

- During mesh modification, parts sometimes do NOT know how many requests they'll receive or from which processors they'll receive requests.

- Need to occasionally check for and handle outstanding requests.
  - iMeshP_pollForRequests
    - Determine whether any requests are pending and, if so, handle them.

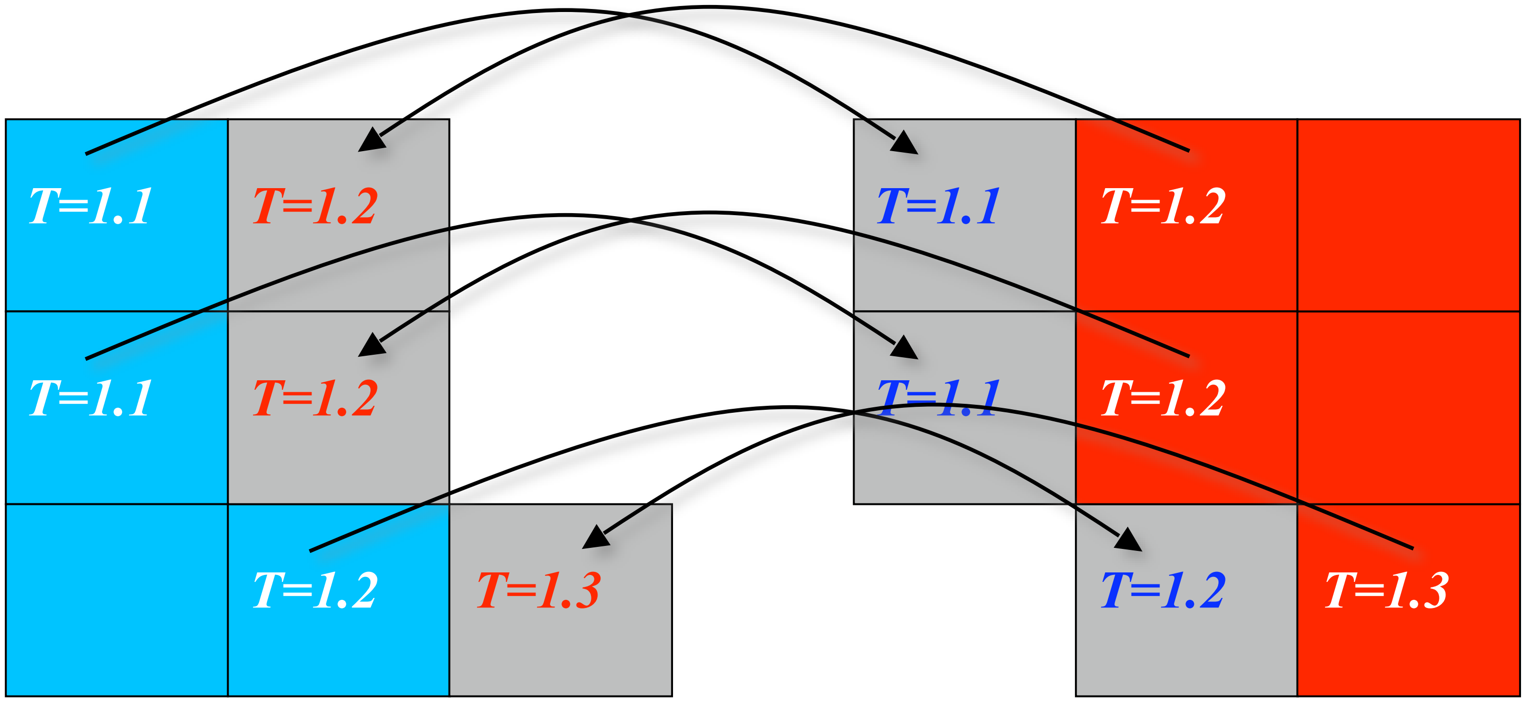
Large-Scale Migration

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



Exchange Entity Tag Data

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

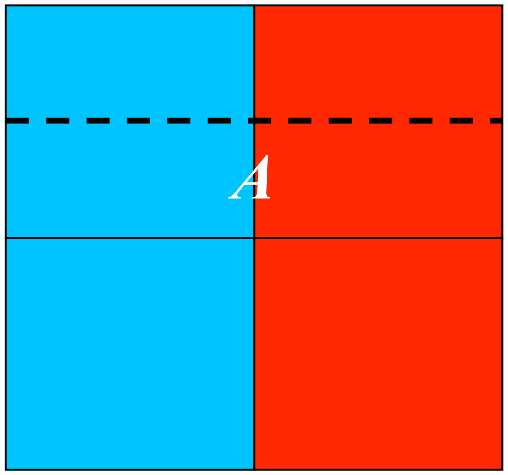


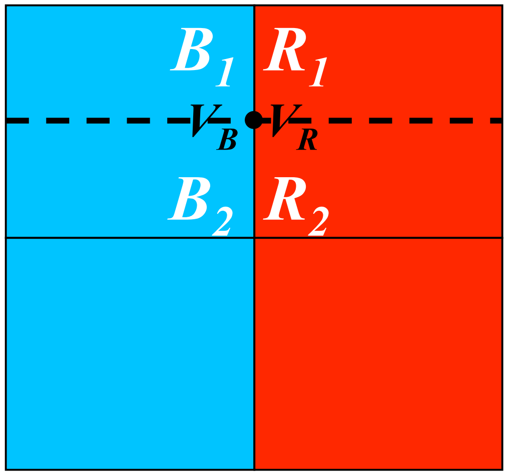
Non-blocking Tag Exchange

- Application calls asynchronous tag exchange function.
  - iMeshP_IExchTags
    - Sends tag data from owner to neighbors; posts receives for tag data for copies.
    - Returns iMeshP_RequestHandle.
  - iMeshP_IExchTagsEnt
    - Sends tag data from owner to neighbors; posts receives for tag data.
    - Returns iMeshP_RequestHandle.
  - Call must be made by all participating parts.
    - Parts know which neighbors they will communicate with.
- Application does something else for awhile.
- Application calls appropriate wait function with the iMeshP_RequestHandle returned by exchange.

Edge Splitting with Non-Blocking Update

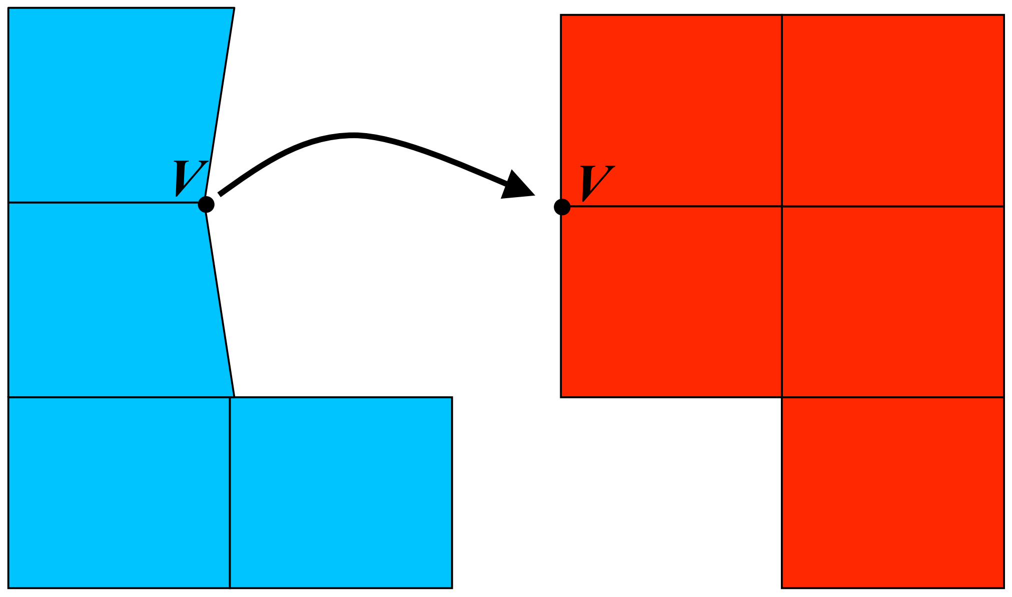
- Blue and red parts decide to split edge A.
- Red part creates edges R_1, R_2 and vertex V_R.
- Blue part creates edges B_1, B_2 and vertex V_B.
- Blue and red parts call iMeshP_replaceOnPartBdry to request replacement of A with new edges and vertices on opposite part.
- Blue and red parts call iMeshP_pollForRequests;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





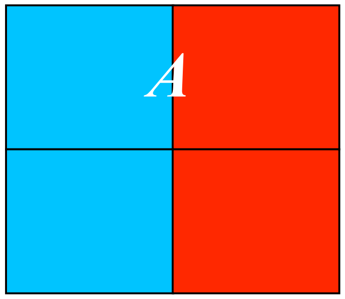
Mesh Smoothing with Non-Blocking Update

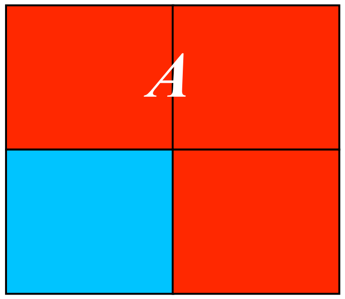
- Blue part decides to move vertex *V*.
- Blue part calls iMeshP_updateVtxCoords to request update of *V*'s vertex coordinates on red part.
- Red part calls iMeshP_pollForRequests; iMeshP implementation receives request and updates *V*'s coordinates.

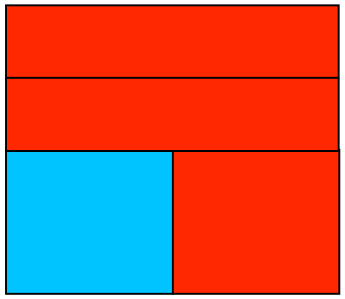


Micro-migration for Mesh Modification

- Blue part owns edge *A*.
- Red part needs edge *A* to do edge swapping.
- Red part calls iMeshP_migrateEntity to request edge *A* from Blue part.
- Blue part calls iMeshP_pollForRequests; iMeshP implementation receives request and sends *A* and its higher-order adjacencies to Red part.
- Red part calls iMeshP_Wait to wait for its migrate request to complete.
- Red part performs edge swapping.







More Mesh Modification Functions

- Add/remove copies of selected entities.
  - iMeshP_addGhostOf
    - Request creation of a ghost entity on a given part.
    - Returns iMesh_RequestHandle.
  - iMeshP_rmvGhostOf
    - Requests removal of a ghost entity on a given part.

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.

Ghost entities

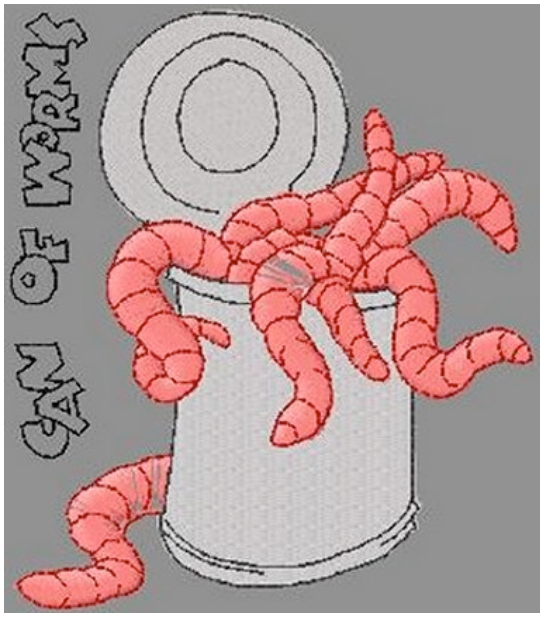
- Ghost entities: copies of entities that not on a part boundary.
- Ghost entities are not required to have remote handles of all copies of the entities.
- Next task of parallel interface committee: ghosting interface.
  - Common ghosting patterns based on mesh adjacencies will be easy to specify.
  - Unusual ghosting patterns will likely be more difficult to specify.

Support for Multiple Partitions

- Multiple partitions of the same mesh are often desired.
  - E.g., Crash simulations need …
    - Partition of volumetric mesh for force calculations.
    - Geometric partition of surface mesh for contact detection.
- Do not want to store/maintain both partitions always.
- Designate one partition the “primary” partition.
- Move entities to “secondary” partitions as needed.
- To do:
  - Define functions to designate a partition as the “primary” partition.
  - Define functions for mapping from primary to secondary partitions, and back again.

File I/O

- **Needs:**
  - **Load: Populate a mesh instance AND a partition. Return the partition handle.**
  - **Save: Store partition information in files.**
  - **Support for parallel file I/O:**
    - **Single file and *P* processes.**
    - ***N* << *P* files distributed to *P* processes.**
    - ***P* parallel files.**
  - **Provide initial distribution of serial file data to *P>1* processes.**



http://www.ceannmor.com/images/CanOfWorms.jpg

For More Information

- DraftInterface.h -- v0.1 syntax for functions
- requirements.pdf -- requirements document
- Bootcamp_March2008.pdf -- this presentation
- Itaps-parallel@mcs.anl.gov -- archive of subcommittee discussions
- Karen Devine -- kddevin@sandia.gov