

ITAPS Partition Model Interfaces

Onkar Sahni, Ting Xie, Xiaojuan Luo, Kenneth E. Jansen and Mark S. Shephard

October 15, 2007

Contents

1	Introduction to Partition Model	2
2	Definitions	2
2.1	Partition Data Model	3
3	Partition Model Interface Functionality	4
3.1	Type Definitions	4
3.2	iProcParts Level Operations	4
3.2.1	Return Number of Parts on Processor	5
3.2.2	Return Part Ids on Processor	5
3.2.3	Access One Part on Processor	6
3.2.4	Iterator-Based Access (Single Part)	6
3.3	iPart Level Operations	7
3.3.1	Return Part Information	7
3.3.2	Return Neighbor Information of Part	7
3.4	iPartMesh level Operations	8
3.4.1	Iterator-Based Access (Single Part Boundary Entities)	8
3.4.2	Return Remote Copy	9
3.4.3	Check Entity (Part Boundary Entity, Owner Copy, Ghost)	10
3.5	Modify Partition Data	10
3.5.1	Add or Remove Part on Processor	10
3.5.2	Modify Remote Copy of Part Boundary Entity	11

In this document we provide the partition model interface functions required to support parallel mesh database on distributed memory (MIMD) computing environment. We define the concept of a *partition data model* within ITAPS interfaces. First we provide basic definitions and further provide three levels of parallel interface functions:

- Processor level.
- Part level.
- Part mesh level.

1 Introduction to Partition Model

A partition is a collection of *parts*, which for mesh-based applications are typically sub-meshes. It describes the distribution of data over parts. A part has a total amount of computational work associated with it and has defined interactions with other parts. Each part exists wholly within one processor, while multiple parts can reside on one processor¹. Each part has an instance of partition data model that describes its interactions with other parts.

A partition model is a separate data model to represent a mesh partition. It can be viewed as a component of hierarchical domain decomposition, and is developed between the geometric model and the mesh. As a data model, partition model provides the flexibility to represent a partitioned mesh in topology dynamically, and supports interpart boundary links with ease [1, 2], see Figure 1.

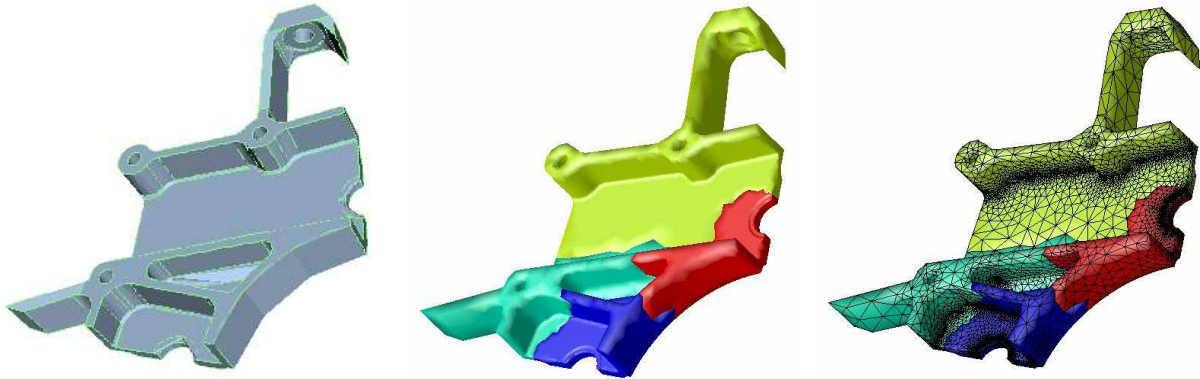


Figure 1: Hierarchy of domain decomposition: geometry model, partition model, and the distributed mesh with four parts.

2 Definitions

We will introduce some definitions to support the concept of partition data model before we define it.

- **Definition: Part Id**

Any part within a partition is assigned a globally unique integer part identifier(Id). The reason we need this is to clearly distinguish between parts.

¹We assume that only one process (or MPI task) exists on one processor.

- **Definition: Object**

The basic unit the partitioning algorithm works with.

A part is a collection of objects. In mesh-based applications, an object is a *part mesh entity* (for example, a mesh entity that does not bound any other mesh entities of higher dimension in the case of a mesh of non-manifold domain), or a group of part mesh entities.

- **Definition: Residence Parts**

A set of parts where a mesh entity (not a ghost entity) exists within a partition is called its residence parts.

For any entity M_i^d not on the boundary of any higher order mesh entities, its residence part is determined simply to be the part where it resides. If entity M_i^d is on the boundary of other higher order mesh entities, M_i^d is duplicated on multiple parts depending on the residence parts of its bounding entities, since M_i^d exists on all parts where its bounding mesh entities exists. Therefore, the residence part(s) of M_i^d is the union of residence parts of all entities that it bounds [2].

- **Definition: Entity Ownership**

For mesh entities duplicated at interpart boundaries, a specific copy of the mesh entity on a part within a partition is defined as its *owner*.

A mesh entity residing on any interpart boundary (*part boundary entity*) can exist on multiple part(s), but it has only one owner part at any instant. The ownership of a mesh entity among parts can be changed dynamically.

- **Definition: Remote copies**

Given a mesh entity at interpart boundary, the handles of duplicated copies on other parts along with their corresponding parts Ids are called its *remote copies*. A mesh entity on an interpart boundary must be aware of all its remote copies.

2.1 Partition Data Model

Partition data model extends the mesh data model to include the concept of a partition [3]. It makes use of the concept of entities in the mesh data model, and introduces a new data type *partition model entity*. A partition model consists of partition model entities.

Definition: Partition (Model) Entity

A topological entity in the partition model, P_i^d , which represents a set of mesh entities of dimension d that have the same residence parts. Each partition model entity is uniquely determined by its residence parts.

Each mesh entity has a unique association with a partition model entity within a partition, which is called *partition classification*. For each partition model entity, the set of mesh entities of the same order classified on it defines the *reverse partition classification* for the partition model entity [2].

Figure 2 illustrates a distributed mesh of 3 parts P_0, P_1, P_2 , where mesh entities labeled with arrows indicate the partition classification of the mesh entities onto the partition model entities, P_i^d , and its associated partition model [1, 2]. The partition model consists of 3 partition regions, 3 partition faces and 1 partition edge. The mesh vertices and edges on the thick black lines are classified on partition edge P_1^1 . The mesh vertices, edges and faces on the shaded planes are classified on the partition faces pointed with each arrow. The remaining mesh entities are non-part boundary entities, therefore they are classified on the partition regions. The reverse partition classification of P_1^1 returns mesh edges located on the thick black lines, and the reverse partition classification of partition face P_i^2 returns mesh faces on the shaded planes [2].

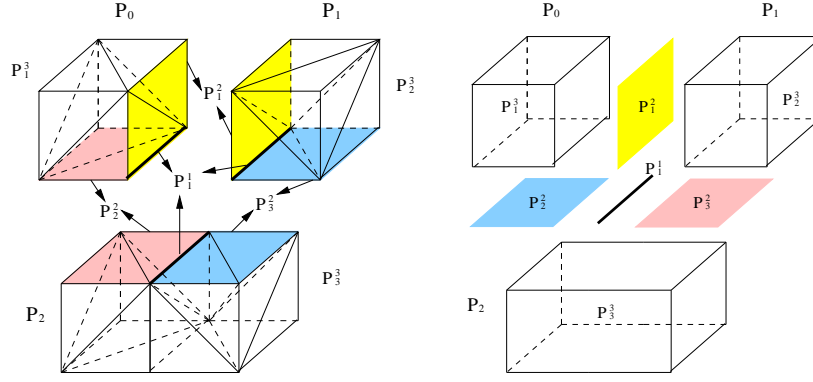


Figure 2: Concept of partition model: Distributed mesh and its association with the partition model via partition classifications.

3 Partition Model Interface Functionality

3.1 Type Definitions

Three levels of interface functions exist:

- processor level
- part level
- part mesh level

3 handles are defined to satisfy these levels of functionality.

The concept of iPart is on the top of the concept of iMesh. An iPart handle is to access a part (a submesh) on one processor within a partition, it includes the information of the part Id, the residing processor rank, and the iMesh handle of the submesh on the part (*part mesh*) along with instance of partition data model on the part.

An iMesh handle is to access a part mesh on a processor, and it is treated as a serial mesh as that in the serial mesh data model. Thus it is suitable for all serial mesh interface functionalities, such as entity iterator, number of entities.

An iProcParts handle is to access a list of part meshes on one processor within a partition. In the case of one part per processor, an iProcParts handle is a part mesh on one processor; while in the case of multiple parts per processor, an iProcParts handle includes all part meshes on one processor.

Figure 3 shows a 2D distributed mesh of 3 parts, where mesh entities on the part boundaries are duplicated on corresponding parts. Assuming that these three parts reside on the same processor, we can access the distributed mesh through the iProcParts handle, which includes a list of 3 parts (iPart handles) with corresponding part meshes (iMesh handles).

3.2 iProcParts Level Operations

Assuming different number of parts k and processors p , i.e., value k may be not equal to value p , multiple parts can reside on a single processor at any instant, and the number of parts on each processor can be different.

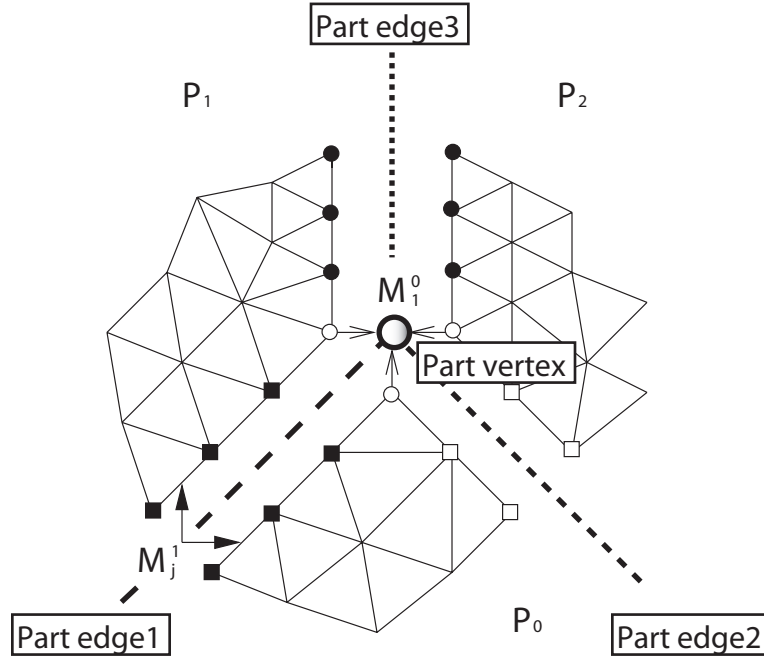


Figure 3: Interpart relationships: 2D illustration of distributed mesh data paradigm.

3.2.1 Return Number of Parts on Processor

- `void iProcParts_getTotNumOfParts(/*in*/ iProcParts_Instance instance,
/*out*/ int* num_part,
int *err);`

Return the total number of parts over all processors within a partition.

- `void iProcParts_getNumOfPartsArr(/*in*/ iProcParts_Instance instance,
/*inout*/ int** numps,
/*inout*/ int* numps_allocated,
/*out*/ int* numps_size,
int* err);`

Return an integer array (collective call) *numps* to indicate the number of parts on each processor, where the array indices correspond to processor ranks. The value *numps_size* equals to the number of processors.

It is useful when the number of parts per processor is different within a partition.

- `void iProcParts_getNumOfPartsOnProc(/*in*/ iProcParts_Instance instance,
/*out*/ int* num_part,
int* err);`

Given an *iProcParts* handle on one processor, return the number of parts on the processor.

For example, for the distributed mesh on one processor in Figure 3, the function returns 3.

3.2.2 Return Part Ids on Processor

- `void iProcParts_getPartIdArrOnProc(/*in*/ iProcParts_Instance instance,`

```

/*inout*/ int** pids,
/*inout*/ int* pids_allocated,
/*out*/ int* pids_size,
int* err);

```

Given an `iProcParts` handle on one processor, return an array of the global part Ids of all local parts that reside on the processor.

For example, for the distributed mesh on one processor in Figure 3, the function returns an array of part Ids, including P_0, P_1, P_2 .

It is useful because we do not always assume the global part Ids of all local parts on one processor are continuous.

3.2.3 Access One Part on Processor

- `void iProcParts_isPartOnProc(/*in*/ iProcParts_Instance instance, /*in*/ const int part_id, /*out*/ int* has_data, int* err);`

Given an `iProcParts` handle on one processor and a global part Id *part_id*, check if the part with part Id *part_id* resides on the processor. The integer value *has_data* is true if it resides on the processor, otherwise it returns false.

- `void iProcParts_getGlobalPartOnProc(/*in*/ iProcParts_Instance instance, /*in*/ const int part_id, /*out*/ iPart_Instance* part_instance, int* err);`

Given an `iProcParts` handle on one processor and a global part Id *part_id*, return an `iPart` handle of the part that corresponds to the part Id *part_id* on the processor. If no part with such global part Id exists on the processor, an *error* is generated.

For example, we assume P_i specifies the global part Id for each part in the distributed mesh in Figure 3 given the global part Id I , the function returns the `iPart` handle of the part P_1 .

- `void iProcParts_getLocalPartOnProc(/*in*/ iProcParts_Instance instance, /*in*/ const int part_order, /*out*/ iPart_Instance* part_instance, int* err);`

Given an `iProcParts` handle on one processor and a local part order *part_order*, return an `iPart` handle of the part that corresponds to the local part order on the processor. If the integer part value *part_order* over the range of the local part number on the processor, an *error* is generated.

For example, for the distributed mesh in Figure 3, we assume the 3 parts are on the same processor, and they are in an ascending local order based on their global part Ids. The local order starts from 0 in the C language convention. Given the local order 1, the function returns the `iPart` handle of part P_1 .

3.2.4 Iterator-Based Access (Single Part)

- `void iProcParts_initPartIter(/*in*/ iProcParts_Instance instance, /*out*/ iPart_PartIterator* part_iterator, int* err);`

Create a part iterator of `iProcParts_Instance` handle to access all local parts on the processor.

- `void iProcParts_getNextPartIter(/*in*/ iProcParts_Instance instance,
/*in*/ iPart_PartIterator part_iterator,
/*out*/ iPart_Instance* part_instance,
/*out*/ int *has_data,
int* err);`

Get the next part in the iterator of the processor. The integer value *has_data* is true if the iterator has more parts to return, and false when there are no more parts to return. When the value is false, the data in the iPart handle argument is not guaranteed to be a valid handle.

- `void iProcParts_resetPartIter(/*in*/ iProcParts_Instance instance,
/*in*/ iPart_PartIterator part_iterator,
int* err);`

Reset the part iterator.

- `void iProcParts_endPartIter(/*in*/ iProcParts_Instance instance,
/*in*/ iPart_PartIterator part_iterator,
int* err);`

Destory the part iterator.

3.3 iPart Level Operations

3.3.1 Return Part Information

- `void iPart_getPartId(/*in*/ iPart_Instance instance,
/*out*/ int* part_id,
int* err);`

Given a part, return its globally unique integer part Id.

- `void iPart_getProcRank(/*in*/ iPart_Instance instance,
/*out*/ int* proc_rank,
int* err);`

Given a part, return the processor rank of the processor on which it resides.

Each part is required to be stored wholly within only one processor, i.e., a part may not span across processors [3]. For example, the mapping between processor rank and part Id is useful for part communication in the case of multiple parts per process.

- `void iPart_getPartMesh(/*in*/ iPart_Instance instance,
/*out*/ iMesh_Instance* mesh_instance,
int* err);`

Given a part, return its corresponding iMesh handle of the submesh.

3.3.2 Return Neighbor Information of Part

- `void iPart_getNghbProcRanks(/*in*/ iPart_Instance instance,
/*inout*/ int** nghbs,
/*inout*/ int* nghbs_allocated,`

```

/*out*/ int* nghbs_size,
int* err);

```

Given an iPart handle, return a list of processor ranks of processors that neighbor the processor where the current part resides within a partition.

- `void iPart_getNghbPartIds(/*in*/ iPart_Instance instance,
/*inout*/ int** nghbs,
/*inout*/ int* nghbs_allocated,
/*out*/ int* nghbs_size,
int* err);`

Given an iPart handle, return a list of part Ids of parts that neighbor the current part within a partition. The neighboring parts can reside on the same processor as the current part, or other processors.

For example, in Figure 3, the neighboring parts of part P_1 are parts P_0 and P_2 , wherever processors they reside.

3.4 iPartMesh level Operations

All interface functions in the serial mesh data model are enabled to accept either Root Set handles, Entity Set handles or iPart handles [3]. And following functions are added to access partition information for an individual mesh entity.

3.4.1 Iterator-Based Access (Single Part Boundary Entities)

- `void iPartMesh_initPartBdryEntIter(/*in*/ iPart_Instance instance,
/*in*/ const int entity_type,
/*in*/ const int entity_topology,
/*in*/ const int requested_nghb_part_id,
/*inout*/ iMesh_EntityIterator* entity_iterator,
int* err);`

Create a part boundary entity iterator of part for a given entity type or topology on a given interpart boundary shared by the current part and requested part of *requested_nghb_part_id* within a partition. If both type and topology are specified, they must be consistent and topology takes precedence. If the integer value of *requested_nghb_part_id* equals -1, the function returns a part boundary entity iterator for a given entity type or topology along the whole interpart boundary of the current part.

This function is useful to construct neighboring part communication whatever the interpart boundary is internal of a processor, or between processors.

- `void iPartMesh_getNextPartBdryEntIter(/*in*/ iPart_Instance instance,
/*in*/ iMesh_EntityIterator entity_iterator,
/*out*/ iBase_EntityHandle* entity_handle,
/*out*/ int *has_data,
int* err);`

Get the next entity in the iterator. The integer value *has_data* is true if the iterator has more entities to return, and false when there are no more entities to return. When the value is false, the data in the entity handle argument is not guaranteed to be a valid handle.

- `void iPartMesh_resetPartBdryEntIter(/*in*/ iPart_Instance instance,
/*in*/ iMesh_EntityIterator entity_iterator,
int* err);`

Reset the part boundary entity iterator.

- `void iPartMesh_endPartBdryEntIter(/*in*/ iPart_Instance instance,
/*in*/ iMesh_EntityIterator entity_iterator,
int* err);`

Destroy the part boundary entity iterator.

- `void iPartMesh_getNumOfPartBdryEntities(/*in*/ iPart_Instance instance,
/*in*/ const int entity_type,
/*in*/ const int entity_topology,
/*in*/ const int requested_nghb_part_id,
/*out*/ int* entity_num,
int* err);`

Return the number of part boundary entities for a given entity type or topology on a given interpart boundary shared by the current part and the request part of *requested_nghb_part_id* within a partition. If no interpart boundary is shared by the current part and the requested part, the integer value *entity_num* returns zero. If the integer value of *requested_nghb_part_id* equals -1, the function returns the number of entities for a given entity type or topology along the whole interpart boundary of the current part.

3.4.2 Return Remote Copy

- `void iPartMesh_getOwnerOfEnt(/*in*/ iPart_Instance instance,
/*in*/ const iBase_EntityHandle entity_handle,
/*out*/ int* part_id,
/*out*/ iBase_EntityHandle* requested_entity_handle,
int *err);`

Given a part mesh and an entity, return its remote copy *requested_entity_handle* on its owner part and the part Id of its owner part *part_id*. If the entity has no remote copy, i.e., it is not on any interpart boundary, an error is generated.

- `void iPartMesh_getCopiesOfEnt(/*in*/ iPart_Instance instance,
/*in*/ const iBase_EntityHandle entity_handle,
/*inout*/ int** pids,
/*inout*/ int* pids_allocated,
/*out*/ int* pids_size,
/*inout*/ iBase_EntityHandle** requested_entity_handle,
/*inout*/ int* requested_entity_handle_allocated,
/*out*/ int* requested_entity_handle_size,
int *err);`

Given a part mesh and an entity, return an array of its remote copies *requested_entity_handle*, and an array of part Ids *pids*, whose elements are corresponding part Ids for the elements in *requested_entity_handle* with the same array indices. If the entity has no remote copy, i.e., it is not on any interpart boundary, an error is generated.

- `void iPartMesh_getCopyOfEnt(/*in*/ iPart_Instance instance,
/*in*/ const iBase_EntityHandle entity_handle,`

```

/*in*/ int part_id,
/*out*/ iBase_EntityHandle* requested_entity_handle,
int *err);

```

Given a part mesh and an entity, return its remote copy *requested_entity_handle* on the part with part Id *part_id*. If the entity has no remote copy on a part with such part Id, an error is generated.

- ```
void iPartMesh_getNumOfCopiesOfEnt(/*in*/ iPart_Instance instance,
/*in*/ const iBase_EntityHandle entity_handle,
/*out*/ int* entity_num,
int *err);
```

Given a part mesh and an entity, return the number of its remote copies. If the entity has no remote copy, the integer value *entity\_num* returns zero.

### 3.4.3 Check Entity (Part Boundary Entity, Owner Copy, Ghost)

- ```
void iPartMesh_isEntOnPartBdry(/*in*/ iPart_Instance instance,
/*in*/ const iBase_EntityHandle entity_handle,
/*out*/ int* is_on,
int *err);
```

Given a part mesh and an entity, check if it is on any interpart boundary. If it is, the integer value *is_on* returns true, otherwise, it returns false.

- ```
void iPartMesh_isEntOwner(/*in*/ iPart_Instance instance,
/*in*/ const iBase_EntityHandle entity_handle,
/*out*/ int* is_owner,
int *err);
```

Given a part mesh and an entity, check if it is on the owner part. If it is, the integer value *is\_owner* returns true, otherwise, it returns false.

- ```
void iPartMesh_isEntGhost(/*in*/ iPart_Instance instance,
/*in*/ const iBase_EntityHandle entity_handle,
/*out*/ int* is_ghost,
int *err);
```

Given a part mesh and an entity, check if it is a ghost entity. If it is, the integer value *is_ghost* returns true, otherwise, it returns false.

3.5 Modify Partition Data

This section is still under progress.

3.5.1 Add or Remove Part on Processor

- ```
void iProcParts_addPartOnProc(/*inout*/ iProcParts_Instance instance,
```

```

/*in*/ iPart_Instance part_instance,
int* err);

```

Add an iPart handle to an iProcParts handle. If such iPart handle already exists on the processor, an error is generated.

- `void iProcParts_rmvPartOnProc(/*inout*/ iProcParts_Instance instance,
/*in*/ iPart_Instance part_instance,
int* err);`

Remove an iPart handle from an iProcParts handle. If no such iPart handle exists on the given processor, an error is generated.

### 3.5.2 Modify Remote Copy of Part Boundary Entity

- `void iPartMesh_addCopyOfEnt(/*in*/ iPart_Instance instance,
/*inout*/ iBase_EntityHandle entity_handle,
/*in*/ int part_id,
/*in*/ iBase_EntityHandle requested_entity_handle,
int *err);`

Given a part mesh and an entity, add one entity *requested\_entity\_handle* on the part with part Id *part\_id* as one of its remote copies. If the entity has no such remote copy on a part with such part Id, an error is generated.

- `void iPartMesh_rmvCopyOfEnt(/*in*/ iPart_Instance instance,
/*inout*/ iBase_EntityHandle entity_handle,
/*in*/ int part_id,
/*in*/ iBase_EntityHandle requested_entity_handle,
int *err);`

Given a part mesh and an entity, remove one of its remote copies: entity *requested\_entity\_handle* on the part with part Id *part\_id*. If the entity has no such remote copy on a part with such part Id, an error is generated.

## References

- [1] E. S. Seol and M. S. Shephard (2006) *Efficient distributed mesh data structure for parallel automated adaptive analysis*. Engineering with Computers 22:197-213.
- [2] E. S. Seol. *FMDB:Flexible Distributed Mesh Database for Parallel Automated Adaptive Analysis*. PhD dissertation, RPI, 2005.
- [3] ITAPS. *ITAPS Parallel Interface: Requirements Document*, Oct 2, 2007.