WORKING DRAFT : Services and features provided by FMDB to support parallel computing

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

September 25, 2007

Contents

1 Abstract

This document describes the services and features provided by FMDB [1] to support distributed-memory computing. FMDB utilizes the concept of partition model to represent a partition, and creates inter-part relationships with the help of partition model entities. The next five sections describe the concept of partition model and provide a set of representative parallel operations for various services. Examples are presented in Section 7 to illustrate the usage of these operations in finite element analysis and mesh-based adaptive simulations.

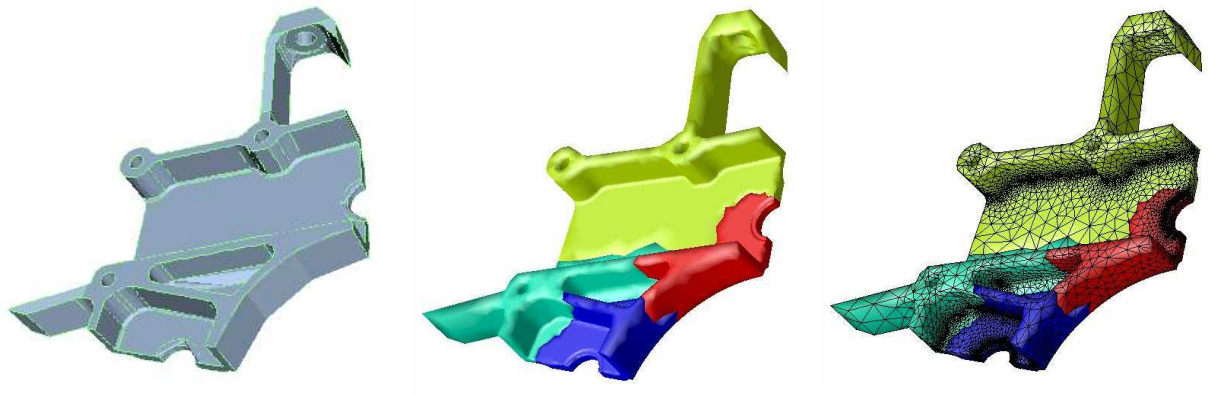

Figure 1: Hierarchy of domain decomposition: geometry model, partition model, and the distributed mesh with four parts.

2 Partitioning and Inter-part Relationships

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 work associated with it and has defined interactions with other parts. Each part exists wholly within one process, while multiple parts can reside on one process.

A partition model is a separate data model created in FMDB 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 [1, 2], see Figure 1.

2.1 Concept of partition model

- A part is a collection of objects, where *objects* are the basic unit the partitioning algorithm works with. An object in mesh-based applications 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. Each object is assigned to a unique part of the partition, where each part has a global part identifier(Id).
- A partition model consists of *partition model entities*, where each partition model entity is a set of mesh entities with the same bounding parts. Figure 2 illustrates a distributed mesh, where mesh entities labeled with arrows indicate the partition classification of the mesh entities onto the partition model entities, and its associated partition model [1, 2].
- 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].
- A mesh entity residing on a part boundary (*part boundary entity*) can exist on multiple part(s), but it has only one owner part at any instant. Ownership of mesh entities is derived from its partition classification. Given a mesh entity on a part boundary, the memory locations of duplicated copies on other parts along with residence part Ids are called *remote copies*.

2.2 Operations

1. **Partitioned Mesh Level Operations**

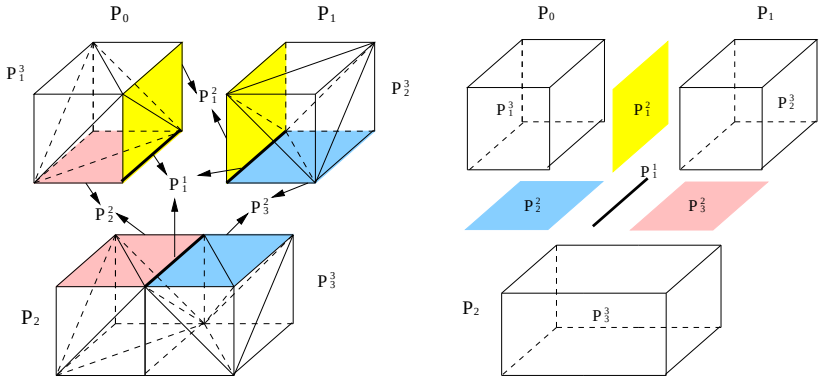

Figure 2: Concept of partition model: Distributed mesh and its association with the partition model via partition classifications.

Figure 3 shows a partitioned mesh, where mesh entities on the part boundaries are duplicated on corresponding parts.

- void **M_boundingPids**(OUT array<int> bps)
  Get a list of part Ids of all neighboring parts to the current part.
  For example, in Figure 3, the bounding parts of part P_1 are parts P_0 and P_2.
- pPartBdryEntIter **M_partBdryEntIter**(IN pMesh mesh, IN int dim)
  Given a part and an integer *dim*, get an iterator over mesh entities of dimension *dim* (0 for mesh vertices, 1 for mesh edges and 2 for mesh faces) on the part boundary. It is used with: pEntity **PartBdryEntItex_next**(INOUT pPartBdryEntIter).

2. **Partition Model Level Operations**

- void **PModel_update**(INOUT pPModel, IN pMesh mesh)
  Given a partitioned mesh (collective call), update the ownership of partition model entities according to poor-to-rich rule based on the number of objects residing on each part and create new partition model entities if necessary. This utility is typically required during dynamic (re-)partitioning or on-the-fly mesh migration.
- pPModelEntIter **PModel_entityIter**(IN pPModel pmodel)
  Provides an iterator over partition model entities. It is used with: pPModelEntity **PModelEntItex_next**(INOUT pPModelEntIter).

3. **Entity Level Operations**

- bool **EN_onPartBdry**(IN pENtity entity)
  Given a mesh entity, check if it is on any part boundary.
- bool **EN_isOwner**(IN pEntity entity)
  pRemoteCopy **EN_ownerCopy**(IN pEntity entity)

  Check if the given mesh entity (copy on current part) is the owner copy. Each mesh entity residing on any part boundary exists on multiple parts, but only a single copy among all duplicate copies on neighboring parts is assigned as owner. For a mesh entity on part boundary, also get its remote copy information relating to owner part (i.e., part Id of owner part and memory location on it).

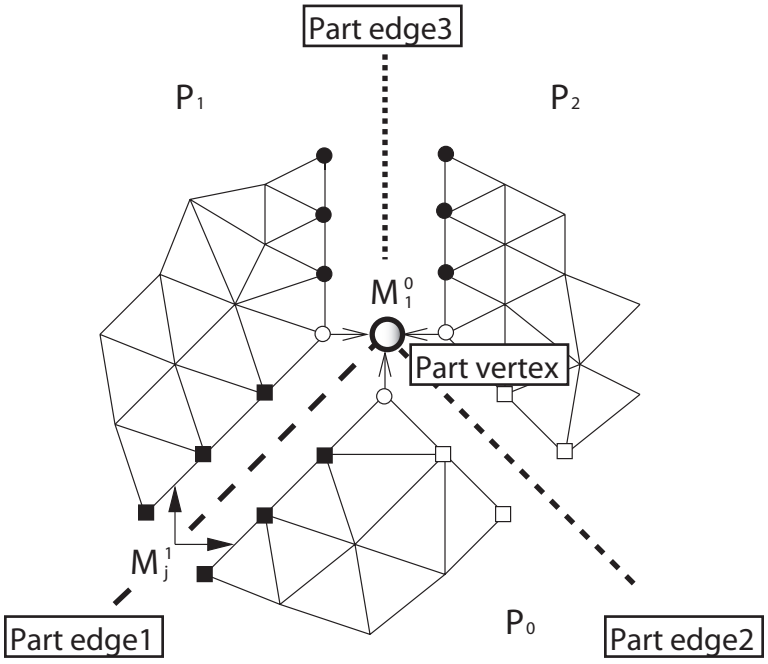

Figure 3: Inter-part relationships: 2D illustration of distributed mesh data paradigm.

- pEntity **EN_getRemoteCopy**(IN pEntity entity, IN int pid)
  void **EN_getRemoteCopies**(IN pEntity entity, OUT array<pRemoteCopy> > remoteCopies)
  void **EN_addRemoteCopy**(INOUT pEntity entity, IN pRemoteCopy remoteCopy)
  and void **EN_clearRemoteCopy**(INOUT pEntity entity, IN pRemoteCopy)

  Given a mesh entity, get its remote copy relating to part with part Id *pid*, or get list of all remote copies. Also add or clear a remote copy for a given mesh entity.
- pPModelEntity **EN_getPClassification**(IN pEntity entity)
  void **EN_setPClassification**(INOUT pEntity entity, IN pPModelEntity pmentity)

  Given a mesh entity, get or set its partition classification.

3 Load Balancing and Data Migration

FMDB supports both static and dynamic load balancing through Zoltan library, including both initial partition and re-partition. It supports weighted partitioning with user-specified, non-uniform object weights along with communications weights.

- void **EN_setWeight**(INOUT pEntity entity, IN double b)
  double **EN_getWeight**(IN pEntity entity)

  Given a mesh entity, set or get its weight.

- void **PM_loadbalance**(INOUT pMesh mesh, IN pmMigrationCallbacks cb)

  Perform (re-)partitioning and load balancing (collective call). Zoltan library is used to determine the part assignment for all the objects and FMDB performs mesh migration to construct the partitioned mesh. Callback mechanism is required to perform migration of user-specific data, for example, solution fields over mesh.

- void **PM_migration**(INOUT pMesh mesh, IN array<pair<pEntity,pid> > objectsToMove, IN pmMigrationCallbacks cb)

  Given a partitioned mesh (collective call) and an associative list (*objectsToMove*) of objects and their destination part Id, migrate the desired mesh entities to construct or modify the partitioned mesh.

4 File I/O

FMDB supports reading and writing of a partitioned mesh into files with the ability to store partition information (currently each part has its independent file).

- void **PM_load**(IN pMesh mesh, IN char *filename)

  Load the partitioned mesh (collective call) from files.

- void **PM_write**(IN pMesh mesh, IN char *filename)

  Write the partitioned mesh (collective call) into files.

5 Ghost Entities

FMDB supports ghosting of mesh entities that are in proximity to part boundaries (this is an ongoing effort).

- void **PM_createGhost**(INOUT pMesh mesh, IN pmMigrationCallbacks cb)

  Given a partitioned mesh (collective call), create information about ghost entities.

- void **PM_expandGhost**(INOUT pMesh mesh, IN pmMigrationCallbacks cb, IN array<pEntity> vtsToExpand, IN int dimToExpand)

  Given a partitioned mesh (collective call), expand the layer of ghost entities according to *vtsToExpand*.

- void **PM_removeGhost**(INOUT pMesh)

  Given a partitioned mesh (collective call), remove the ghost entities.

6 Multi-Parts per Process

There is ongoing effort to allow FMDB to support multiple parts per process.

- void **PM_setNumParts**(IN array<int> int_arr)

  Set the number of parts over all processes (collective call). User can specify multiple parts per process through this operator.

- void **PM_getNumParts**(IN array<int> int_arr)

  Get the number of parts over all processes (collective call).

- void **PM_loadbalance**(IN array<pMesh> mesh, IN pmMigrationCallbacks cb)

  Perform partitioning and load balancing (collective call) with multiple parts per process. And user-specific data can be handled with the help of given callback object *cb*.

7 Examples

This section demonstrates the services and features provided by FMDB [1], to support distributed-memory computing, with the help of examples. Four types of examples are presented that cover a wide range of applications under parallel adaptive mesh-based simulations:

1. **Inter-part relationships**: example - parallel mesh-based analysis like finite-element solver.
2. **Dynamic inter-part links**: example - subdivision of mesh entities.
3. **On-the-fly mesh migration**: example - execution of local mesh modification operator of edge collapse.
4. **Dynamic partitions**: example - re-partitioning for dynamic load balance, for example, after mesh adaptation.

Example 1: parallel mesh-based analysis (inter-part relationships)

Typically in a distributed (parallel) finite element solver, an initial step of computations is performed on each part of the partition to assemble local (part) information which is followed by a communication step to gather global information through inter-part assembly process. In the communication phase, mesh entities (representing degree of freedoms) that reside on part boundaries, and are duplicated over multiple parts (see Figure 3), dictate the inter-part assembly process. In Figure 3 mesh vertex M_1^0 resides on boundary of three parts whereas mesh edges M_j^1 (and their bounding mesh vertices) reside on two. To apply distributed mesh data paradigm the concept of partition model is utilized which not only provides information about part boundary but also distinguishes between each set of mesh entities formed by ones with the same bounding parts (each set represents a partition model entity), see Figure 2. Inter-part relationships (as supported by FMDB) required to facilitate this process includes:

- **M_boundingPids()**: Other parts sharing part boundary with a given part (i.e., neighboring parts); to create communication traces between neighboring pair of parts in the partition.
- **M_partBdryEntIter()**: Iterator over mesh entities residing on part boundaries; to determine communication volume between neighboring pair of parts and fill-in communication arrays.
- **EN_isOwner()** and **EN_ownerCopy()**: Mesh entity ownership information (a single copy among all duplicates on neighboring parts is assigned as owner); to set-up control relationships for governing the inter-part assembly process, it is also referred to as master-slave control relationships.
- **EN_getRemoteCopies()**: Remote copies (memory location of a mesh entity duplicated on neighboring parts along with their residence part Ids); to execute inter-part assembly process for mesh entities on part boundary.

Example 2: subdivision of mesh entities (dynamic inter-part links)

Mesh refinement usually relies on subdivision templates where marked mesh edges are split/divided along with mesh entities surrounding them. To apply such an operation on distributed mesh requires the flexibility of dynamic inter-part

links such that each part can apply subdivision templates locally (regardless of breaking inter-part links) and then carry a communication step to repair the broken inter-part links. Figure 4 illustrates the process of subdivision on distributed mesh for a 2D case. The utilities (as provided by FMDB) needed to perform such an operation includes:

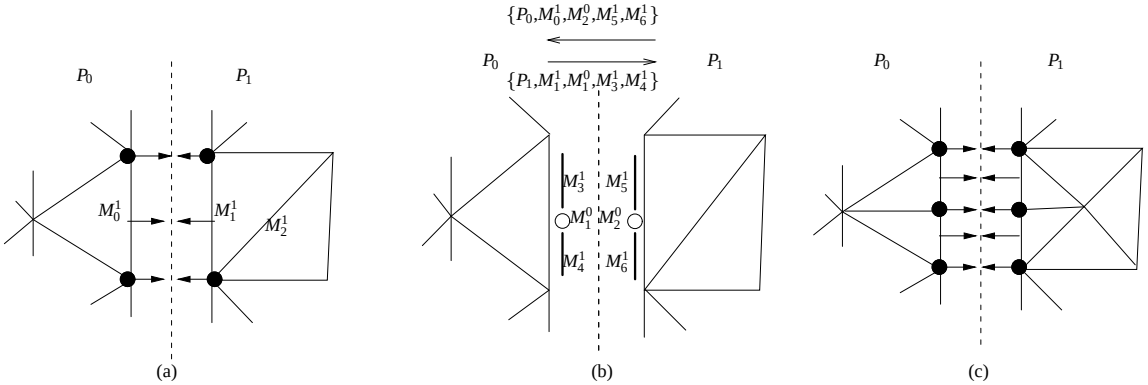

Figure 4: Dynamic inter-part links: schematic of distributed subdivision of mesh entities for a 2D example.

- **EN_onPartBdry()** and **EN_getRemoteCopies()**: Check if a mesh entity to be subdivided is on part boundary and get remote copies for ones on part boundary; to create communication messages to repair inter-part links broken due to subdivision.
- **EN_addRemoteCopy()** and **EN_clearRemoteCopy()**: add or clear remote copy information for a given mesh entity; to repair/update inter-part links.
- **EN_getPClassification()** and **EN_setPClassification()**: get or set partition classification of a mesh entity; to assign partition model entity association/classification of child mesh entities derived from parent mesh entities.

Example 3: local mesh modification of edge collapse (on-the-fly mesh migration)

Mesh coarsening based on local mesh modifications relies on repeated evaluation and execution of mesh modification operators, majorly edge collapse, that change both the local topology and geometry. Since the direct evaluation and/or execution of such operators on mesh entities on any part boundary for distributed meshes are complex as well as inefficient due to complicated communication pattern; a on-the-fly local (cavity level) mesh migration-based parallelization approach is applied. In such a situation, mesh entities affected by the operation are first migrated into one single part and then the operation is executed locally on the single part. Figure 5 demonstrates the concept using a edge collapse operator for a 2D case (where bold line representing mesh edge residing on parts P_1 and P_3 needs to be collapsed). The distributed mesh support, provided by FMDB, utilized for such operators include:

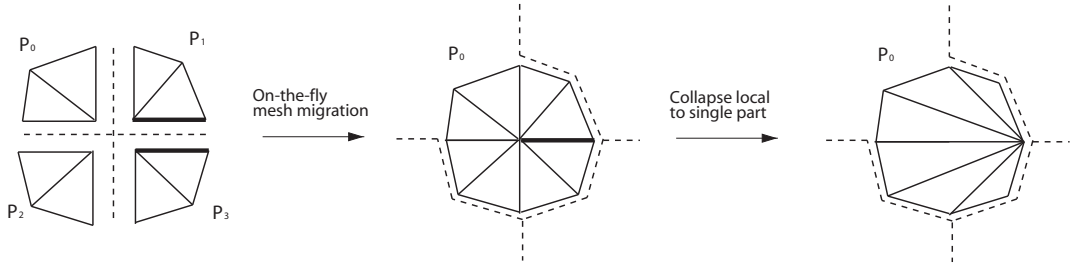

Figure 5: On-the-fly mesh migration: schematic of distributed edge collapse operation for a 2D case.

- **EN_onPartBdry()**: Check if a mesh entity involved in collapse operation is on part boundary; to invoke migration of local cavity affected by the operator.
- **PM_migration()**: migrate mesh entities affected by coarsening locally to parts; to allow for serial execution of collapse operator. It is invoked once coarsening of the interior mesh (excluding part boundary) has been carried out.
- **EN_getRemoteCopies()**, **EN_addRemoteCopy()** and **EN_clearRemoteCopy()**: get remote copies and also add or clear remote copies of mesh entities on part boundaries; to perform on-the-fly mesh migration and update inter-part boundaries.
- **PModel_update()**: update the ownership of partition model entities and/or create new partition model entities if necessary; to update partition model due to mesh migration.
- **EN_getPClassification()** and **EN_setPClassification()**: get or set partition classification of a mesh entity; to assign partition model entity association/classification of migrated mesh entities.

Example 4: re-partitioning for load balance (dynamic partitions)

In mesh-based adaptive simulations, as the partitioned mesh is modified the computational load (dependent on mesh entities) on each part is altered. To be able to balance the load to effectively progress the simulations dynamic partitions are required, see Figure 6. To support dynamic load balancing, FMDB provides following operations:

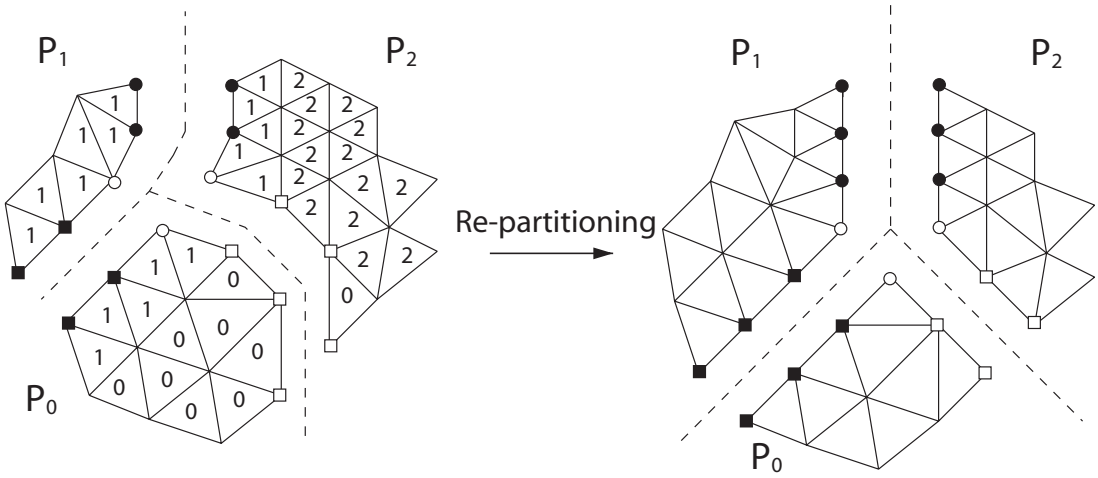

Figure 6: Dynamic partitions: re-partitioning to support dynamic load balancing; (left) mesh faces showing destination parts and (right) after mesh migration.

- **EN_setWeight()**: Set a weight representing computational load for a given mesh entity; to allow for re-partitioning of a mesh with non-uniform load distribution.
- **PM_loadBalance()** (which calls Zoltan library and **PM_migration()**): Perform load balance using Zoltan library and migrate mesh entities; to construct the partitioned mesh.
- **EN_getRemoteCopies()**, **EN_addRemoteCopy()** and **EN_clearRemoteCopy()**: get remote copies and also add or clear remote copies of mesh entities on part boundaries; to update inter-part boundaries due to re-partitioning.
- **PModel_update()**: update the ownership of partition model entities and/or create new partition model entities if necessary; to update partition model due to re-partitioning.

- **EN_getPClassification()** and **EN_setPClassification()**: get or set partition classification of a mesh entity; to assign partition model entity association/classification of migrated mesh entities.

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.