

Examples using ITAPS Parallel Functionalities

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

December 20, 2007

1 Examples

This section demonstrates the usage of some ITAPS parallel functionalities [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 complete information across parts 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 1), dictate the inter-part assembly process. In Figure 1 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 facilitate this process, we can use following ITAPS parallel functionalities.

- **Identify parts that neighbor a given part:** Given a partition instance and a part handle, return the part IDs of all parts in the partition that shares part boundary with the given part (i.e., neighboring parts) through some entities of a given type; to create communication traces/links between neighboring parts in the partition.
- **Part boundary iterator:** Given a partition instance and a part handle (along with specific entity types, topologies, and neighboring part IDs), get an iterator over entities along the part boundary shared with the neighboring parts; to determine communication volume between neighboring parts and fill-in communication arrays.
- **Provide information about copies of entities and identify the owner of the entity copies:** Given a partition instance and an entity handle, get the ownership of the entity (owner entity); to set-up control relationships (also referred to as master-slave control relationships) for governing the inter-part assembly process, where the owner/master copy is the incharge.

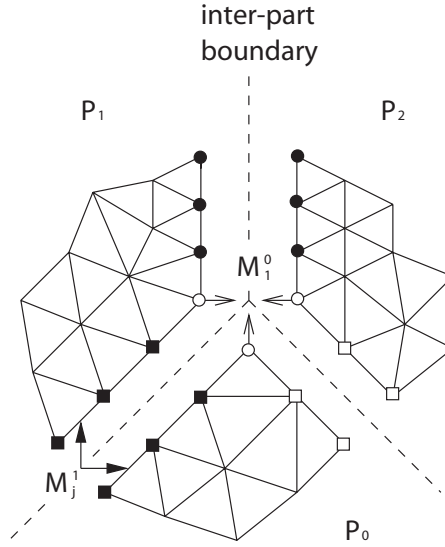


Figure 1: Inter-part relationships: 2D illustration of distributed mesh data paradigm.

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 2 illustrates the process of subdivision on distributed mesh for a 2D case. The utilities needed to perform such an operation includes:

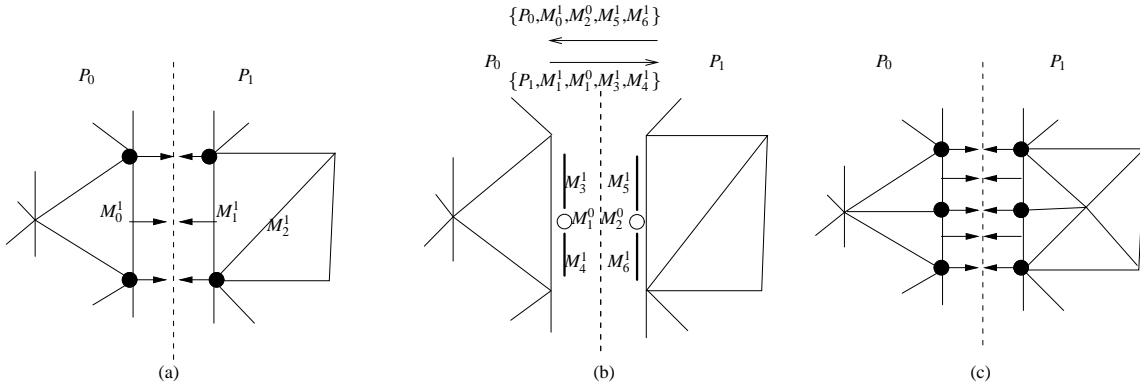


Figure 2: Dynamic inter-part links: schematic of distributed subdivision of mesh entities for a 2D example.

- **Provide entity categorization within a part and information about copies of entities:** Given a partition instance, a part handle, and an entity handle, check if the 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.
- **Update copies of entities from mesh modification¹:** Given a partition instance, an entity handle, add or clear remote copy information for a given mesh entity; to repair/update inter-part links.

¹This required function does not exist in the current ITAPS parallel document [1].

Example 3: local mesh modification using 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 3 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, utilized for such operators include:

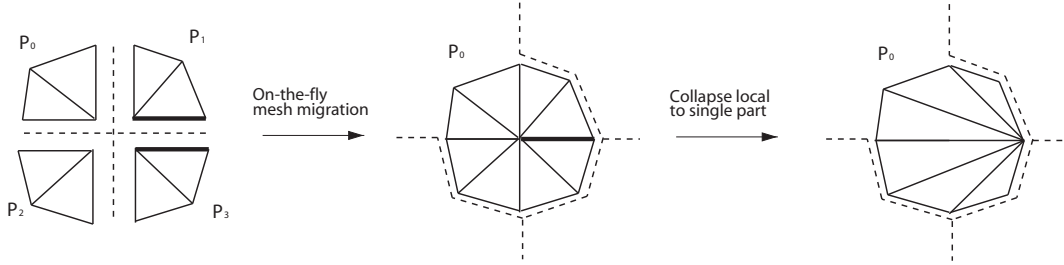


Figure 3: On-the-fly mesh migration: schematic of distributed edge collapse operation for a 2D case.

- **Provide entity categorization within a part:** Given a partition handle, a part handle and an entity handle, check if the mesh entity involved in collapse operation is on part boundary; to invoke migration of local cavity affected by the operator.
- **Add/remove (migrate) entities to/from on-process and/or off-process parts, sendEntArrToParts(MIGRATE):** migrate mesh entities affected by coarsening locally to parts; to allow for serial execution of collapse operator.

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 4. To support dynamic load balancing, we use following operations:

- **Add/remove (migrate) entities to/from on-process and/or off-process parts):** Given a partition handle, entity-handles to be migrated and destination parts IDs, perform migration based on the partitioner library (like Zoltan) to achieve load balance.

References

- [1] CombinedModelByKaren.txt. Dec,2007.

