VI International Conference on Adaptive Modeling and Simulation
ADMOS 2013
J. P. Moitinho de Almeida, P. Díez, C. Tiago and N. Parés (Eds)

METHODS AND TOOLS FOR PARALLEL ANISOTROPIC MESH ADAPTATION AND ANALYSIS

MARK S. SHEPHARD[*], E. SEEGYOUNG SEOL[*], CAMERON W. SMITH[*], MISBAH MUBARAK[*], ALEKSANDR OVCHARENKO[*] AND ONKAR SAHNI[*]

[*]Scientific Computation Research Center
Rensselaer Polytechnic Institute
Troy, New York 12180
e-mail: shephard@rpi.edu, smithc11@rpi.edu, seols@rpi.edu, mubarm@rpi.edu,
shurik.asa@gmail.com, sahni@rpi.edu

Key words: Mesh adaptation, boundary layer, parallel adaptation, dynamic load balancing.

Summary. It is well known that adaptive methods provide the most effective means to obtain reliable solutions and control the amount of computation required. However, for many classes of problems the best adaptive method still requires a level of computation that demands massively parallel computing. This paper presents a set of technologies for parallel adaptive simulation that includes a parallel mesh infrastructure, dynamic load balancing procedures and parallel anisotropic mesh adaptation. Examples of anisotropically adapted meshes for real-world fluid flow problems, including boundary layer meshes, are given.

1 INTRODUCTION

Adaptive anisotropic unstructured mesh technologies support the effective analysis of complex physical behaviours modelled by partial differential equations over general three-dimensional domains. Although adaptively defined anisotropic meshes can have two to three orders of magnitude fewer elements than a more uniform mesh for the same level of accuracy, there are many cases where the desired size of the adapted meshes are so large that they can only be solved on parallel computers. The execution of a simulation on a parallel computer requires the mesh to be distributed over the nodes and cores of the parallel computer.

The design of an infrastructure supporting adaptive unstructured meshes on massively parallel computers must consider the management of mesh information, the modification operations to be carried out on the meshes, and the scalability of the algorithms. The most basic functionality of the mesh infrastructure is to support the distribution of the mesh over the cores of the parallel computer. In adaptive simulations additional functionality is needed to adapt the mesh in parallel as the simulation process proceeds. A key requirement of effective parallel simulation is maintaining equal distribution, or load balance, of the computations, especially for the typically dominant analysis-related computations. Since mesh adaptation locally increases and/or decreases the mesh density, methods are needed to redistribute the mesh in order to regain load balance for the subsequent analysis step.

This paper presents a set of three components that are needed to support geometry-based parallel adaptive simulations. The first component is a parallel mesh infrastructure designed to support evolving meshes of any size on massively parallel computers. The second component

supports dynamic load balancing procedures that are capable of regaining the load balance of meshes as they adaptively evolve. The third component is a generalized mesh modification procedure that can execute anisotropic mesh adaptation, including boundary layer meshes, in parallel on distributed meshes.

2 PARALLEL MESH INFRASTRUCTURE

It is clear that simulations on meshes with millions and billions of elements require the mesh to be distributed over the compute cores of the massively parallel computers that execute the simulation. The most common form of mesh decomposition over distributed memory compute cores is to partition into a set of parts where the individual parts are groups of mesh entities. Between neighboring parts, the number of mesh entities on the common boundary is kept as small as possible in order to minimize communications. In addition to the mesh maintaining parallel distribution information, it must also maintain information relating it to the high-level problem domain definition to effectively support general mesh adaptation. This information allows mesh adaptation processes to account for the actual shape of the problem domain as the mesh is adapted and not be restricted to the geometric approximation defined by the initial mesh [12]. Rebalancing the workload as the mesh is adapted requires effective methods to migrate mesh entities between parts and update the inter-part communication links. The Parallel Unstructured Mesh Infrastructure (PUMI) is being developed to support the needs of adaptively evolving meshes on massively parallel computers [22], [23].

The geometric model is the high-level (mesh independent) definition of the domain that consists of a set of topological entities with adjacencies and associated shape information. A general non-manifold boundary representation [26] is needed to support the full range of requirements that often include multiple material domains and reduced dimension entities as elements in the mesh. PUMI interacts with the geometric model through a functional interface that supports the ability to interrogate the geometric model for the adjacencies of the model entities and geometric information about the shape of the entities [2], [24]. The use of such a representation allows the mesh adaptation procedures to interact with the geometric domain through simple topologically driven geometric interrogations to ensure that the mesh modifications are consistent with the actual geometric domain [12]. Other interactions with the geometric domain definition support the proper transformation of the input boundary condition fields onto the mesh and using geometric interrogations to the original geometry to support element integrations for mesh entities bounded by curved domain boundaries [7]. Effective execution of mesh adaptation and field transformation procedure requires a complete mesh representation in which the complexity of any mesh adjacency interrogation is O(1) (i.e., not a function of mesh size [3], [22]). Meeting this requirement does not require the explicit storage of every one of the four levels/orders of mesh entities (vertex, edge, face and region) or all of the 12 possible adjacencies. However, it is critical to use a complete mesh representation [20], [22] so that information on any mesh entity or mesh entity adjacency can be obtained in O(1) time.

2.1 Partition Model

When a mesh is distributed to N parts, each part is assigned to a process or processing core. A part is a subset of topological mesh entities of the entire mesh denoted by P_i, $0 \leq i < N$. Figure 1 depicts a 2-D mesh that is distributed to three parts. Each part is treated as a serial mesh with the addition of part boundaries to describe groups of mesh entities that form the links between parts. Mesh entities on part boundaries, called *part boundary entities,* are duplicated on all parts for which they bound other higher order mesh entities. Mesh entities that are not on any part boundaries exist on a single part and are called *interior mesh entities* (with respect to the part). The boundary between the parts in different processes is an inter-process part boundary and the boundary between the parts on the same process is an intra-process part boundary.

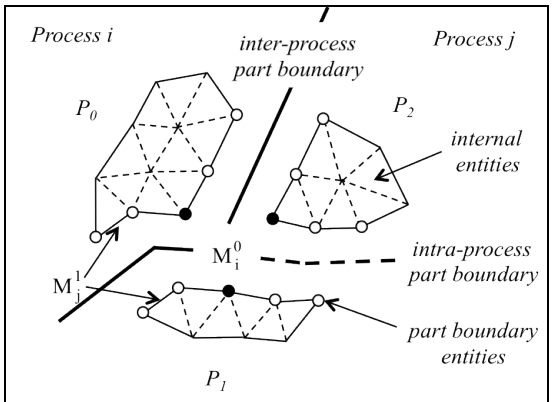


Figure 1: A distributed 2-D mesh.

For each mesh entity, the *residence part set* [22], [23] is a set of part ID(s) where a mesh entity exists based on adjacency information: If mesh entity M_i^d is not adjacent to any higher dimension entities, the residence part set of M_i^d is the ID of the single part where M_i^d exists. Otherwise, the residence part set of M_i^d is the set of part IDs of the higher order mesh entities that are adjacent to M_i^d. Note that part boundary entities share the same residence part if their locations with respect to the part boundaries are the same.

In the 2-D mesh illustrated in Figure 1, the part boundary entities are the vertices and edges that are adjacent to mesh faces on multiple parts. The residence part set of vertex M_i^0 and edge M_j^1 are $\{P_0, P_1, P_2\}$ and $\{P_0, P_1\}$, respectively.

For the purpose of representation of a partitioned mesh and efficient parallel operations, a *partition model* is developed.

- Partition (model) entity: A topological entity, P_i^d, which represents a group of mesh entities of dimension d or less, which have the same residence part set. One part among all parts in residence part set is designated as the owning part.
- Partition classification: Unique association of mesh entities to a partition model entity.

Figure 2 depicts the partition model of the distributed mesh. The mesh vertex M_i^0, depicted in Figure 1, duplicated on three parts, is classified on the partition vertex P_0^0 such that P_0^0 represents mesh vertex duplicated on part $\{P_0, P_1, P_2\}$, P_0^1, P_1^1, and P_2^1 represent mesh edges and vertices duplicated on part $\{P_0, P_1\}$, $\{P_0, P_2\}$, and $\{P_1, P_2\}$, respectively. At the mesh entity level, the proper partition classification is needed to maintaining up-to-date residence part set and owning part information which is key to effective support of an evolving distributed mesh. As illustrated in Figure 3, the partition model can be viewed as a part of hierarchical domain decomposition.

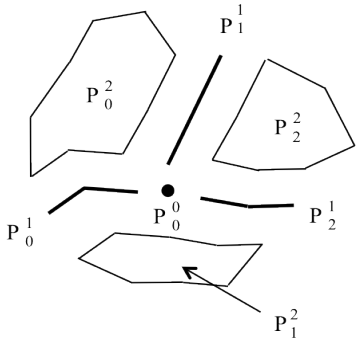
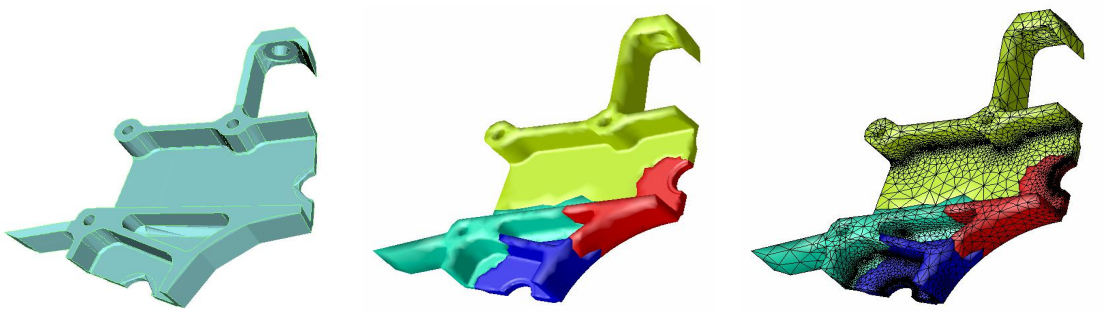


Figure 2: A partition model of the mesh in Figure 1.



(a) Geometric Domain (b) Partition Model (c) Partitioned Mesh

Figure 3: Domain representation hierarchy: (a) geometric model, (b) partition model and (c) partitioned mesh.

2.2 Migration

Mesh migration is a procedure that moves mesh entities from one part to another. Migration supports: (i) partitioning, (ii) dynamic load balancing, and (iii) obtaining mesh entities needed for mesh modification operations. In the mesh migration procedure, a partition object is the basic unit to assign a destination part id. It can be either a mesh entity not on the boundary of any higher dimension mesh entities or a group of mesh entities contained in a single part called p-set [27]. Figure 4 presents the pseudo code of migration algorithm while figure 5 shows the steps involved.

```
INPUT: a list of partition objects and destination parts. See Figure 5(a).
OUTPUT: a mesh with new partitioning

Step 1. Collect all mesh entities that will be effected by migration.
Step 2. For entities collected in step 1, determine residence part set and
update partition classification based on new partitioning. See Figure 5(b).
Step 3: Among entities collected in step 1, collect entities to remove
based on residence part set.
Step 4. Exchange p-set and entities along with partition classification.
See Figure 5(c).
Step 5. For entities collected in step 1, and newly created entities on
destination parts, update part boundary links.
Step 6: Delete entities collected in step 3. See Figure 5(d).
```

Figure 4: Pseudo code of migration algorithm.

2.3 Ghosting

A class of operations on mesh entities near a part boundary such as error estimation and element shape optimization often require data from adjacent mesh entities that are internal to neighboring parts. This information could be obtained by the repeated communication requests between parts. However, such communications adversely affect scalability and performance. Thus it is desirable to minimize them and localize the data for part boundary computations. In cases where this needed data is static for the entire operation, such as solution fields used in error estimation, or change slowly, such as coordinates in element shape optimization, an alternative is to provide all the needed information through local

copies of the remote data. The common approach creates one or more layers of *ghost* copy of the needed un-owned mesh and field data near part boundaries [8], [25].

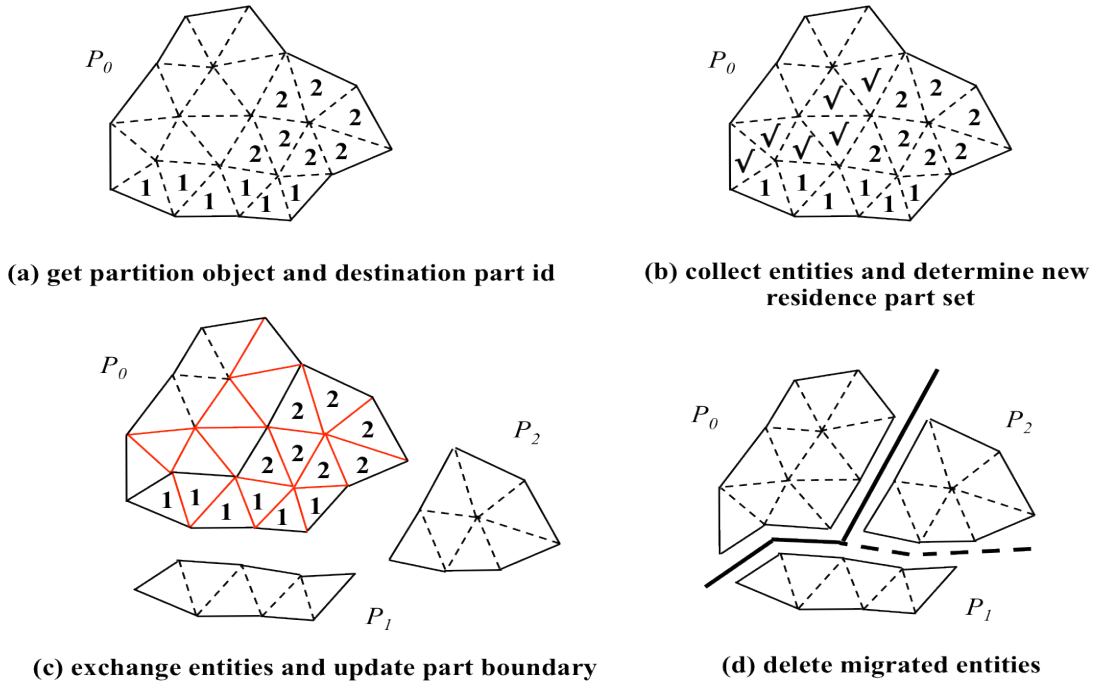
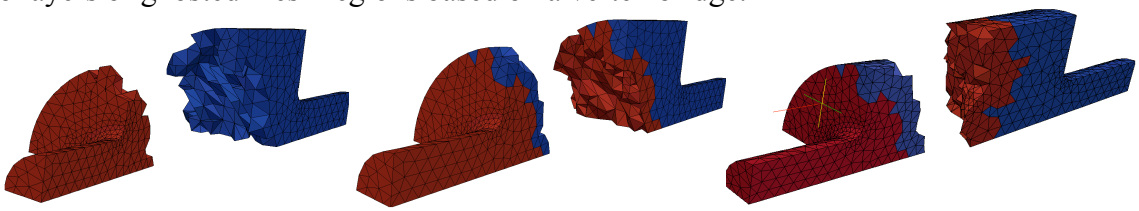


Figure 5: Migration steps demonstrated on a 2-D mesh.

To support a full range of ghost requirements for unstructured meshes, a generalized N-layer ghosting parallel ghosting algorithm has been developed and implemented [15] in PUMI. The algorithm supports the creation of layers of copied mesh entities of desired order and their bounding entities based on specification of a bridge dimension using mesh adjacencies. The key components of the ghost creation process are:

- Ghost dimension: Permissible options in a topological mesh representation can be regions, faces or edges. As ghost entities are specified through a bridge dimension, the lowest possible dimension of a ghost entity can be an edge since the minimum bridge is a vertex. Vertices are ghosted if they are part of higher dimension ghost entities. For example, in a 1-D mesh, the only possible ghost dimension is an edge and the vertices that are on the boundary of ghost edges can be ghosted to create the ghost edges.
- Bridge dimension: Ghost entities are specified similar to second order adjacencies using a bridge entity. The bridge entity must be of lower topological order than the ghost and can be a face, edge or a vertex. Two common examples of ghosted entities are: (i) the mesh regions that are bounded by faces classified on a partition model face, and (ii) the mesh regions that are bounded by vertices classified on partition model vertices, edges or faces. A less common, but supported case would be the mesh edges that are bounded by vertices classified on partition model vertices, edges or faces.
- Number of layers: Number of layers of ghost entities. Layers are measured from the part boundary.

For example, to get two ghost layers of regions, bridged by bounding vertices, the ghost dimension is set to region, the bridge dimension set to vertices and number of layers is 2. When multiple ghost layers are requested by an application, the ghosting process starts with the first (innermost) layer of ghosts adjacent to part boundary. Figure 6 shows zero, one and two layers of ghosted mesh regions based on a vertex bridge.



(a) partition with no ghosting (b) one layer (c) two layers

Figure 6: Application of region ghosting based on a vertex-based bridge.

Figure 7 illustrates the ghosting procedure with ghost dimension 2, bridge dimension 1 and the number of layers 1: (a) initial mesh (b) collect mesh faces to ghost which are bounded by partition model edges (c) for mesh faces and their downward adjacent entities collected in step b, determine the destination part id(s) to migrate to based on residence part set (d) exchange entities (e) at the original copy, update ghost copy information.

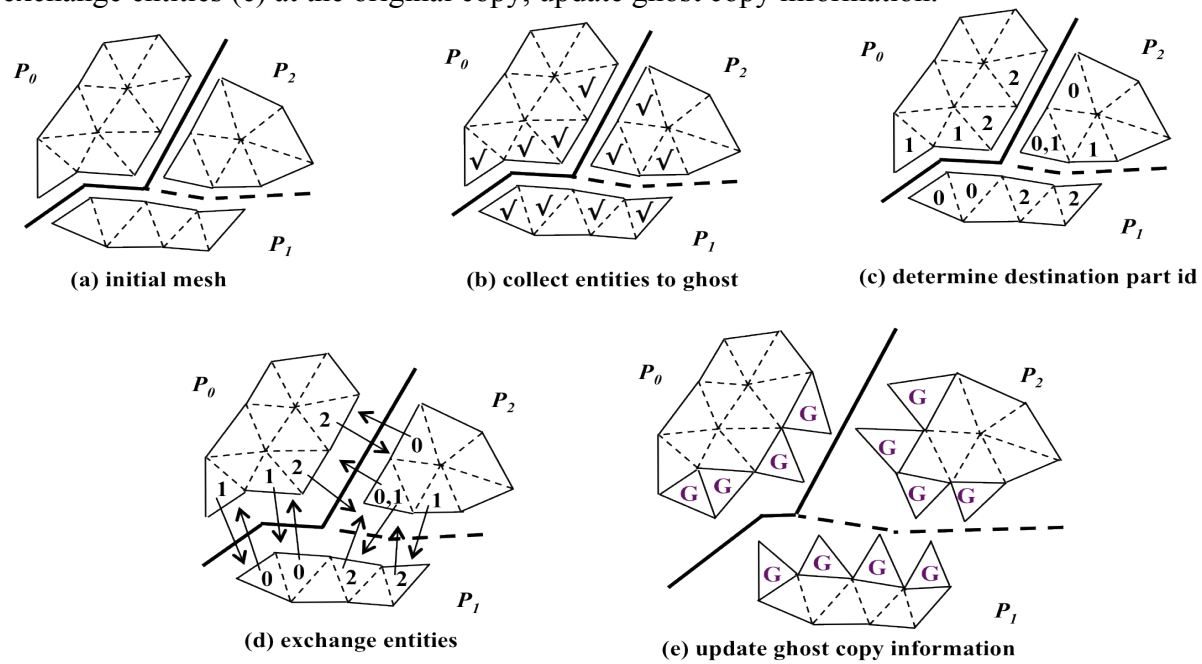


Figure 7: 2-D Ghosting steps.

A ghost entity stores information about its owner entity and the part where the owner entity exists. At a minimum, a ghost entity's owner must also store information about its ghost copies that exist. This synchronizes the ghost copies synchronized with their owner entities and eliminates the need for inter-part communication if there are any queries about ghost entity ownership. The inter-part communication required in the ghost creation process is optimized by utilizing a general-purpose package that sends messages within a fixed process

neighborhood by packing small MPI messages and avoiding unnecessary calls to MPI collective operations [18].

3 DYNAMIC LOAD BALANCING

Another key component of supporting unstructured mesh simulation workflows is dynamic load balancing. At a minimum, the mesh needs to be partitioned such that resulting adapted mesh fits within the memory of each node onto which the mesh is partitioned. PDE analysis additionally requires consideration of the workload by accounting for the different entity types defining the computation load of the phases of the analysis. In both cases peak imbalance determine performance since one heavily loaded processor can force all the others to sit idle while it completes. Small valleys (with load below average) leave a few processes idle having a minimum affect of scaling. Therefore, the reduction of peaks for each workflow step is critical for parallel performance and scaling. A dynamic partitioning algorithm must also account for the connectivity of the unstructured mesh such that the part boundaries are optimized to minimize the amount of communications across neighboring parts.

The most powerful partitioning procedures for meshes are the graph and hypergraph-based methods as they can explicitly account for application defined balance criteria via graph node weights, and one piece of the mesh connectivity information via the definition of graph edges. Hypergraph-based methods can further optimize the mesh partition at the cost of increased run-time over the graph-based methods [4]. Graph based methods balance the weighted values of the graph nodes while trying to minimize the number of graph edges between parts. When partitioning an unstructured mesh, the graph nodes are selected to be the appropriate set of mesh entities, where in most cases the set of graph nodes are all the mesh entities of the highest order (mesh regions in 3D and mesh faces in 2D). The graph edges are defined by the mesh adjacencies that happen to be of importance to the simulation step for which the partition is being constructed. For example in the case of linear finite elements where the unknowns are at mesh vertices, creating a graph edge between each mesh regions that shares a vertex is important, while in a face-based finite volume procedure graph edges should be defined between the pair of mesh regions that share a face.

In addition to the use of standard graph-based procedures, consideration is being given to drive selected dynamic load balancing operations directly from the adjacencies of the mesh entities since it is the selected adjacency information that defines the graph edges in standard graph-based partitioners. Two advantages of a tool that performs parallel Partitioning using Mesh Adjacencies (ParMA) are: (i) it can more easily account for the balancing of mesh entities of different dimensions at the same time, and (ii) could potentially be more computationally and memory efficient since, by working with the existing mesh topological information, it avoids the need to construct a separate partition graph.

The ability to use ParMA to effectively improve mesh partitions in-turn to improve the scalability of finite element solvers has been clearly demonstrated [28]-[31], where consideration was given to the balance of mesh regions, critical for equation formation, as well as mesh vertices, critical for equation solution of linear finite elements. Recent extensions to ParMA have generalized these procedures such that mesh entities of all orders, with assigned priority, can be considered [23]. For example, in the case of quadratic finite elements, there are unknowns at both the mesh vertices and mesh edges. Thus in that case

there is a high priority given to balancing the mesh edges and vertices, since equation solution is the dominating computational step, while the mesh regions are given a lower priority, since their balance controls the scaling of the equation formation step.

A second area of application of ParMA currently being investigated is repartitioning before a mesh adaptation step. Typically the mesh before a mesh adaptation step is well balanced, however, since the mesh adaptation procedures are going to refine the elements is some areas and coarsen them in others, the adapted mesh would be dramatically out of balance after adaptation, to the point that exceeding available memory becomes quite likely. Thus before the mesh is adapted, the new mesh size information is used to assign weights to the current mesh entities (>1 in areas where the mesh will get finer, <1 in areas where the mesh will get coarser) and the mesh in rebalanced. The execution of this process yields a mesh that is very close to being well balanced after mesh adaptation [29]. A full graph-based predictive partitioning [29] is used before mesh adaptation to ensure it will execute without problems and again at the end to refine the balance for the next analysis step. Noting that the only goal of the current predictive load balancing is to load balance the subsequent analysis after the mesh is adapted, it is not considering the scaling of the adaptive process itself, and that even with its current goals, the load balance must be improved between mesh adaptation and the next analysis step. This indicates that there is potential for improving the process. The primary idea under current consideration, potentially as the mesh is being adapted, is to merge neighboring parts in which the number of mesh entities after adaptation will be less than, or equal to, that of a balanced part, and to split parts that will be heavily refined into a number of parts such that each has about the average number of mesh entities after adaptation.

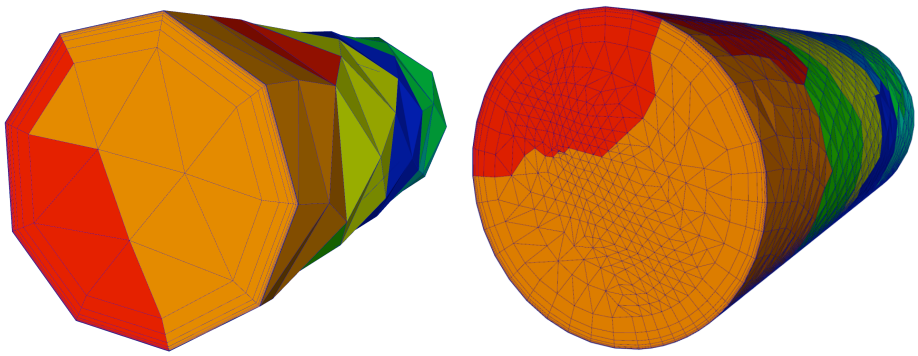
4 PARALLEL ANISOTROPIC MESH ADAPTATION

Many physical problems of interest involve directional solution features. To address such cases adaptive mesh control methods are designed to match an anisotropic mesh size field defined through the application of a posteriori correction indicators [1], [5], [9], [19]. In the case of viscous flow problems is it is important to supplement the general anisotropic mesh adaptation procedures with ones that can maintain a semi-structured boundary layer mesh on selected boundaries [5], [10], [11], [17], [21]. The two approaches to creating the adaptive anisotropic meshes given an adaptively defined anisotropic mesh size field, including adapted boundary layers, are complete domain re-meshing methods, and methods that use local mesh modification. Adaptive re-meshing accounts well for curved domains and mesh resolution. However, this is at the cost of re-meshing the entire domain. A more serious concern of the use of global re-meshing, especially for problems where accurate transfer of the solution fields to the new mesh is required, is both the cost and accuracy of general mesh-to-mesh solution transfers. Conversely, mesh adaptation based on local mesh modification can be a faster method that when coupled with local solution transfer methods, can provide more accurate solution transfer. However, the set of local mesh modification operators must be rich enough to be able to produce the desired anisotropic mesh configurations, while accounting for the curved domain geometry (e.g., as defined by the CAD). A local mesh modification-based procedure that meets these requirements builds off a complete set of mesh modification operations that include compound operators [13] and that maintains semi-structured boundary layers (if any) [17], [21], while local operations also ensure that the adapted mesh conforms to

the curved domain geometry [12], [13].

The parallel implementation of the general mesh modification operators work directly with the partitioned mesh by querying the PUMI provided partition model to coordinate operations during mesh refinement. Mesh coarsening and swapping operations are supported by PUMI mesh migration functions to move required mesh entities between parts [6]. The parallel migration procedures have been extended to include mesh sets that require stacks of semi-structured mesh entities to be migrated together thus supporting the parallel execution of semi-structured boundary layer adaptation [17].

Figure 8 shows a simple example of parallel mesh adaptation including a boundary layer. Figure 8(a) shows an initial coarse mesh that includes a boundary layer while Figure 8(b) shows an adapted mesh on the same simple geometry. In more general cases, the adaptation of the boundary layer can locally reduce the anisotropy to the point where it is desirable to convert the top portions of the boundary layer to be a regular unstructured mesh such that more general unstructured mesh modification operations can be applied. To support this functionality the mesh adaptation procedures need additional extensions to deal with the pyramid elements that are introduced by local trimming of the boundary layer mesh (Figure 9) [17].



(a) Initial Mesh (b) Adapted Mesh

Figure 8: 2-D Boundary layer adaptation.

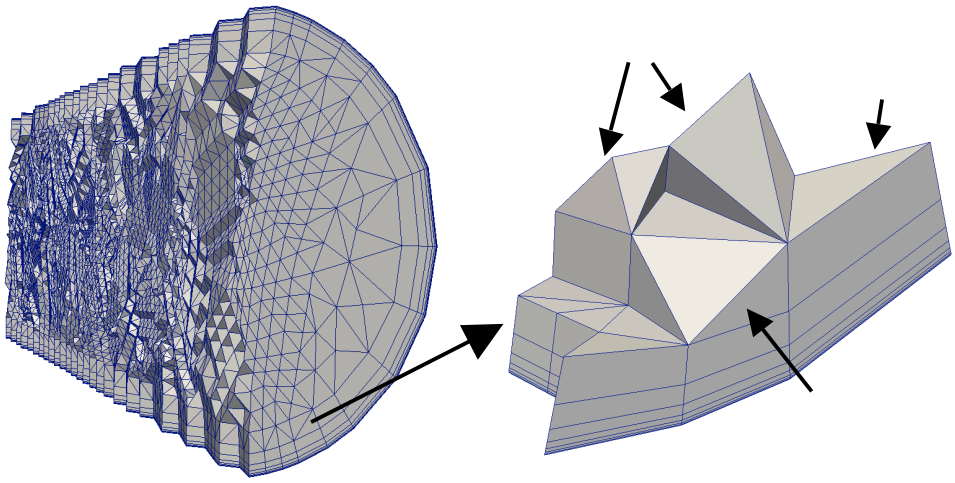


Figure 9: An example requiring the introduction of pyramid elements to allow the trimming boundary layers.

5 PARALLEL ADAPTIVE EXAMPLES

The first example is a viscous flow simulation of a NASA CIAM scramjet case run with a free stream Mach number of 6.2, and a free stream reference temperature of 203.5 Kelvin. The initial boundary layer mesh has 2.9M regions with a mid-section cut-away view of the boundary layer mesh, including close-up of the inlet, is shown in the top two images in Figure 10. The adapted boundary layer mesh has 43M regions and is shown in the bottom two images in Figure 10.

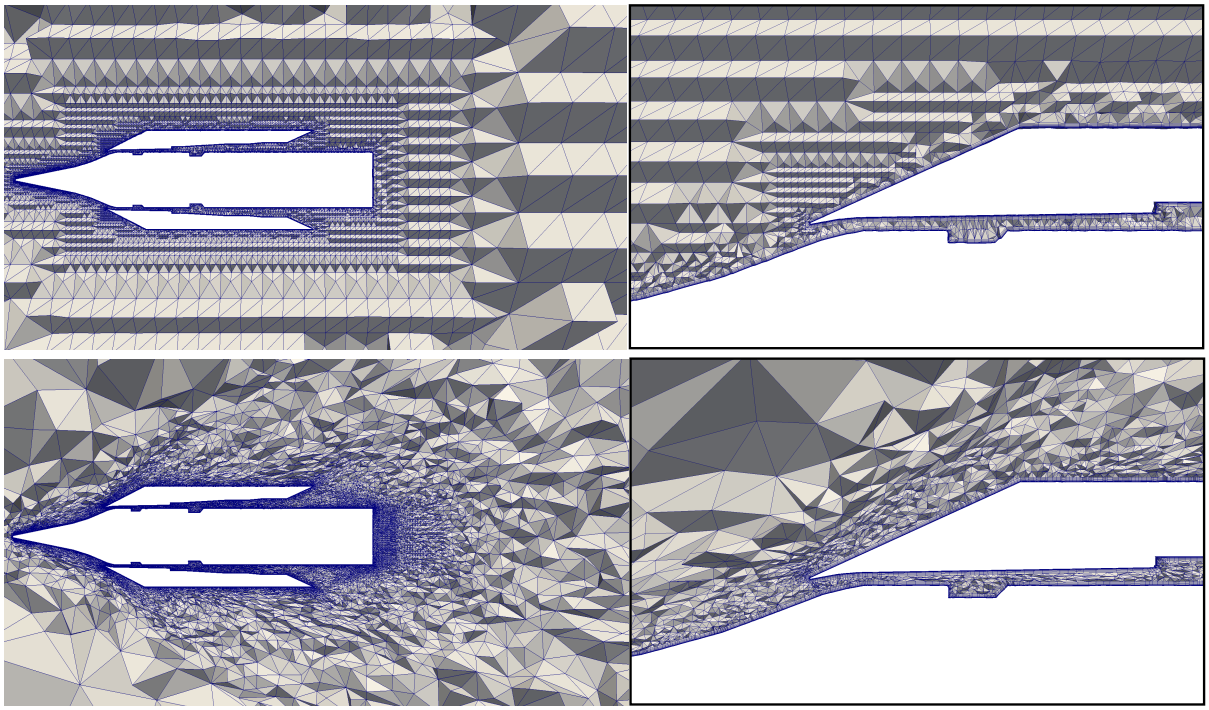


Figure 10: Initial and anisotropic adapted meshes for a scramjet engine.

The second example is a multiphase flow in which a fluid in being injected into air. In this example the interfaces between the fluid and air is modeled using a level set method [16]. The mesh adaptation procedure is keyed to perform anisotropic mesh adaptation at the zero level set that represents the dynamic two-fluid interface. Figure 11 shows the anisotropically adapted mesh at three different time steps in the simulation.

6 CLOSING REMARKS

This paper provides an overview of a set of procedures to perform parallel anisotropic mesh adaptation of unstructured meshes that can include semi-structured boundary layer meshes. The parallel mesh infrastructure, PUMI, parallel partitioning using mesh adjacencies, ParMA and mesh adaptation procedures, MeshAdapt, have been implemented using a component-based approach in which all interactions are controlled through functional interfaces. This approach allows these tools to be efficiently coupled with various unstructured mesh analysis codes and other mesh related components such as mesh generators, dynamic load balancers, etc. See http://www.scorec.rpi.edu/software.php for

information on, and access to, these components.

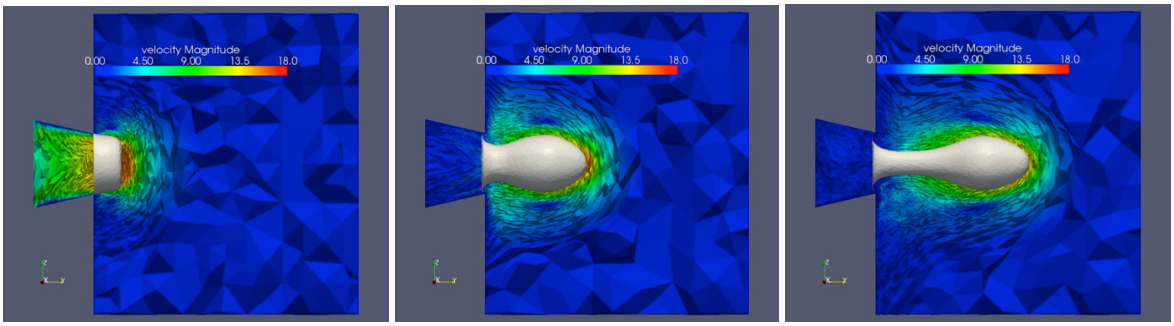


Figure 11: Anisotropic adapted meshes for a two-phase flow problem.

REFERENCES

[1] Alauzet, F., Li, X., Seol, E.S. and Shephard, M.S., "Parallel anisotropic 3D mesh adaptation by mesh modification", *Engineering with Computers*, 21 (2006) 247–258.

[2] Beall, M.W., Walsh, J. and Shephard, M.S., "A comparison of techniques for geometry access related to mesh generation," *Engineering with Computers,* 20(3):210-221, 2004.

[3] Beall, M.W. and Shephard, M.S., "A general topology-based mesh data structure," *Int. J. Numerical Methods in Engineering*, 40(9):1573–1596, 1997.

[4] Boman, E.G., Devine, K.D., Fisk, L.A., Heaphy, R., Hendrickson, B., Leung, V., Vaughan, C., Catalyurek, U., Bozdag, D. and Mitchell, W., "Zoltan home page," September 2011, http://www.cs.sandia.gov/Zoltan.

[5] Botasso, C.L., "Anisotropic mesh adaption by metric-driven optimization", *Int. J. Numer. Meth. Engng.*, 60 (2004) 597–639.

[6] de Cougny, H.L. and Shephard, M.S., "Parallel Refinement and Coarsening of Tetrahedral Meshes", *Int. J. Numer. Meth. Engng.*, 46:1101-1125, 1999.

[7] Dey, S., Shephard M.S. and Flaherty, J.E., "Geometry Representation Issues Associated with p-Version Finite Element Computations", *Comp. Meth. App. Mech. and Eng.*, 150(1-4):29-55, 1997.

[8] Dreher, J. and R. Grauer, R., "Racoon: A parallel mesh-adaptive framework for hyperbolic conservation laws", *Parallel Computing*, 31(8-9):913–932, 2005.

[9] Frey, P.L. and Alauzet, F., "Anisotropic mesh adaptation for CFD computations", *Computer Meth. Applied Mechanics and Engineering*, 194:5068–5082, 2005.

[10] Kallinderis Y. and Kavouklis, C., "A dynamic adaptation scheme for general 3-D hybrid meshes", *Comput. Methods Appl. Mech. Engrg*. 194:5019–5050, 2005.

[11] Khawaja, A., Minyard, T. and Kallinderis, Y., "Adaptive hybrid grid methods", *Comput. Methods Appl. Mech. Engrg.*, 189:1231–1245, 2000.

[12] Li, X., Shephard, M.S. and Beall, M.W., "Accounting for curved domains in mesh adaptation," *Int. J. Numerical Methods in Engineering*, vol. 58, no. 2, pp. 247–276, 2003.

[13] Li, X., Shephard, M.S. and Beall, M.W., "3-D Anisotropic Mesh Adaptation by Mesh Modifications", *Comp. Meth. Appl. Mech. Engng.*, 194(48-49):4915-4950, 2005.

[14] Lu, Q., Shephard, M.S., Tendulkar, S. and M.W. Beall, M.W., "Parallel Curved Mesh Adaptation for Large Scale High-Order Finite Element Simulations", *Proc. 21st International Meshing Roundtable, Springer*, NY, pp. 419-436, 2012.

[15] Mubarak, M., "A parallel ghosting algorithm for the flexible distributed mesh database (FMDB), Scientific Computation Research Center, RPI, Troy, NY, 2011, http://www.scorec.rpi.edu/reports/view_report.php?id=548.

[16] Nagrath, S., Jansen, K.E., and Lahey, R.T., Jr., "Computation of incompressible bubble dynamics with a stabilized finite element level set method", *Computer Methods in Applied Mechanics and Engineering*, 194:4565-4587, 2005.

[17] Ovcharenko, A., Chitale, K., Sahni, O., Jansen, K.E., Shephard, M.S., Tendulkar, S. and Beall, M.W., "Parallel Adaptive Boundary Layer Meshing for CFD Analysis", *Proc. 21st International Meshing Roundtable*, Springer, NY, pp. 437-455, 2012.

[18] Ovcharenko, A., Ibanez, D., Delalondre, F., Sahni, O., Jansen, K.E., Carothers, C.D. and Shephard, MS. "Neighborhood Communication Paradigm to Increase Scalability in Large-Scale Scientific Applications" *Parallel Computing*, 38(3):140-156, 2012.

[19] Park, M.A., "Parallel Anisotropic Tetrahedral Adaptation", 46th AIAA Aerospace Sciences Meeting and Exhibit, AIAA 208-917, 2008.

[20] Remacle, J.F., and Shephard, M.S., "An algorithm oriented mesh database," *Int. J. Num. Methods in Engineering*, 58(2):349–374, Sep. 2003.

[21] Sahni, O., Jansen, K.E., Shephard, M.S., Taylor, C.A. and Beall, M.W., "Adaptive boundary layer meshing for viscous flow simulations", *Engineering with Computers*, 24:267–285, 2008.

[22] Seol, E.S. and Shephard, M.S., "Efficient distributed mesh datastructure for parallel automated adaptive analysis," Eng. with Computers, 22(3-4):197–213, Nov. 2006.

[23] Seol, S., Smith, C.W., Ibanez, D.A. and Shephard, M.S., "A Parallel Unstructured Mesh Infrastructure", http://www.scorec.rpi.edu/reports/view_report.php?id=591, 2012.

[24] Shephard, M.S., "Meshing environment for geometry-based analysis", *Int. J. Numerical Methods in Engineering*, 47(1-3):169-190, 2000.

[25] Stewart, J.R. and H.C. Edwards, H.C, "A framework approach for developing parallel adaptive multiphysics applications", *Finite Elements Analysis and Design*, 40(12):1599–1617, 2004.

[26] Weiler, K.J., "The radial-edge structure: a topological representation for non-manifold geometric boundary representations," *Geometric Modeling CAD Appl.*, pp. 3–36, 1988.

[27] Xie, X, Seol, S., Shephard. M.S., "Generic Components for Petascale Adaptive Unstructured Mesh Simulations", *Eng. with Computers,* DOI 10.1007/s00366-012-0288-4, Accepted Sep. 2012.

[28] Zhou, M., Sahni, O., Xie, T., Shephard, M.S. and Jansen, K.E., "Unstructured Mesh Partition Improvement for Implicit Finite Element at Extreme Scale", *Journal of Supercomputing*, 59(3): 1218-1228, 2012.

[29] Zhou, M., Xie, T., Seol, S., Shephard, M.S. Sahni, O. and Jansen, K.E. "Tools to Support Mesh Adaptation on Massively Parallel Computers", *Engineering with Computers*, 28(3):287-301, 2012

[30] Zhou, M., Sahni, O., Shephard, M.S., Devine, K.E. and Jansen, K.E., "Controlling unstructured mesh partitions for massively parallel simulations", *SIAM J. Sci. Comp.*, 32(6):3201-3227, 2010.

[31] Zhou, M., Sahni, O., Shephard, M.S., Carothers,C.D. and K.E. Jansen, J.E., "Adjacency based reordering algorithm for acceleration of finite element computations", *Scientific Programming*, 18(2):107-123, 2010.